

# Tecnologías Grid

## Condor

Master en Sistemas y Servicios Informáticos para Internet  
Área de Arquitectura y Tecnología de Computadores  
Universidad de Oviedo



# Condor

Conceptos previos

# Conceptos previos

## □ Antes de empezar...

### ▣ ¿Se entiende el concepto de Grid?

- La definición no es sencilla (distintas personas definen Grid de formas distintas)

### ▣ Definición comúnmente aceptada:

- Un sistema distribuido formado por un conjunto de recursos geográficamente dispersos y poco acoplados (loosely coupled) que actúan de forma conjunta para realizar grandes tareas

# Conceptos previos

- Todo es un grid...
  - ▣ Marketing hype
    - Demasiado bueno para ser cierto...
  - ▣ Buzzword
    - Ciertos conceptos se ponen de moda y la gente los empieza a utilizar de forma indiscriminada
    - Típico ejemplo: framework
    - Nuevo ejemplo: cloud

# Conceptos previos

- Todo es un grid...
  - ▣ Un cluster no es un grid
    - En un cluster los recursos son homogéneos, estáticos, dedicados y muy acoplados (tightly coupled). La gestión de recursos en un cluster se encuentra centralizada.
    - La tecnología grid, entre otras muchas cosas, permite interconectar clusters y compartir sus recursos (computacionales y de almacenamiento)

# Conceptos previos

## □ Computación distribuida

### ▣ Objetivo:

- Permitir a una comunidad de usuarios ejecutar tareas sobre un conjunto de recursos distribuidos compartidos

### ▣ Problema:

- En general el número de tareas es mayor que el número de recursos disponibles

### ▣ Solución:

- Mecanismo que gestione de forma eficiente los recursos (la idea de un sistemas operativo y los recursos de ordenador pero distribuidos)

# Conceptos previos

- Tipo de tareas que se ejecutan en un grid
  - ▣ Principalmente los denominados *Embarrassingly parallel*
    - No existen dependencias o comunicaciones entre las tareas
    - Fácil de paralelizar
  - ▣ Paralelización
    - Descomposición de las tareas
    - Descomposición de los datos

## □ High Throughput Computing (HTC)

### ▣ Definición:

- Sistema de gestión de gran cantidad de potencia computacional tolerante a fallos sobre largos periodos de tiempo mediante la utilización eficiente de los recursos disponibles
- HTC es HPC (High Performance Computing) diseñado para trabajos que se ejecutan durante mucho tiempo





# Condor

## Introducción

# Introducción a Condor

## □ Condor:

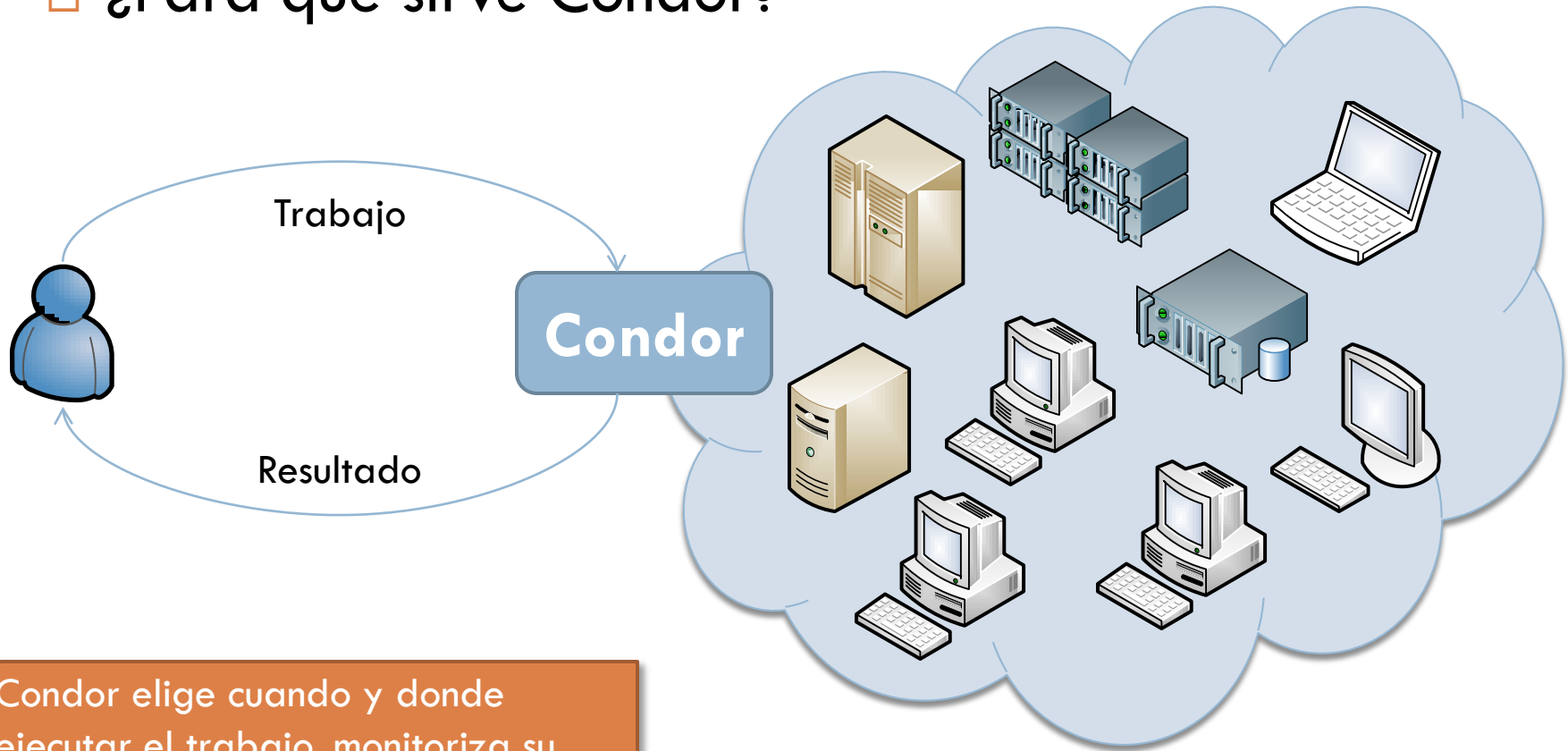
- Un sistema especializado de gestión de carga computacional
  - Se puede utilizar como *batch system* o *job scheduler*

## ▣ Funcionalidad que ofrece:

- Gestión de recursos
- Gestión de trabajos
- Políticas de planificación
- Sistema de prioridades
- Monitorización de recursos y trabajos

# Introducción a Condor

## □ ¿Para qué sirve Condor?



Condor elige cuando y donde ejecutar el trabajo, monitoriza su progreso y, finalmente, informa al usuario de su terminación

# Introducción a Condor

- ▣ Condor = HTC + *opportunistic computing*:
  - ClassAd: encaje de los requisitos de los trabajos con las características de los recursos disponibles
  - Checkpoint y migración: los trabajos se puede mover de máquina durante su se ejecución
  - Las llamadas al sistema de E/S se redirigen a la máquina que lanzó el trabajo

# Introducción a Condor

## □ Terminología utilizada por Condor

### □ Cluster de ordenadores:

- Un conjunto de ordenadores dedicados para uso no interactivo

### □ Pool

- Un conjunto de ordenadores donde se ejecuta Condor.

**Pueden ser dedicados o no**

### □ Flock

- Un conjunto de Pools

### □ Trabajo (Job)

- La representación de Condor de una tarea

# Introducción a Condor

- Gestión de la computación distribuida en un Grid
  - ▣ La planificación no se puede hacer de forma centralizada: hay muchos dueños
    - Planning:
      - Adquisición de recursos de los usuarios
    - Scheduling:
      - Gestión de los recursos
  - ▣ Condor utiliza un mecanismo llamado Matchmaking sobre ClassAds

# Introducción a Condor

## □ Conceptos sobre Matchmaking

### ▣ Objetivo:

- Encajar los trabajos sobre las maquinas disponibles

### ▣ Ejemplo:

#### ■ Trabajo

- Se debe ejecutar sobre Linux con 2 GB de RAM

#### ■ Máquina

- Windows con 512 MB de RAM
- Sólo ejecutar trabajos del Departamento de Informática

### ▣ También se permiten preferencias

- Para ejecutar un trabajo se prefieren máquinas con mucha memoria

# Introducción a Condor

## □ Preferencias de cada tipo de usuario

### ▣ Los usuarios de Condor se dividen en tres grupos

- Usuario que envía trabajo
- Dueños de máquinas
- Administradores del Pool

Pueden ser o no las  
mismas personas

### ▣ Cada grupo tiene sus preferencias

#### ■ Preferencias del dueño de una máquina

- Prefiero ejecutar tareas del Departamento de Informática
- En esta máquina sólo se ejecutarán tareas de 22:00 a 06:00

#### ■ Preferencias del administrador de un Pool

- Prioridad de usuarios
- Cuando un trabajo puede desalojar a otro trabajo

Tipos de usuario



Tipos de máquinas



# Introducción a Condor

- ClassAds (Classified Advertisements)
  - ▣ Mecanismo para encajar tareas con máquinas:
    - Es la forma que utiliza Condor para que los trabajos y los recursos anuncien sus características y requisitos

ClassAD de un trabajo

```
MyType      = "Job"
TargetType  = "Machine"
Owner       = "ruf"
Requirements =
    (Arch == "INTEL")
    && (OpSys == "LINUX")
    && (Disk >= DiskUsage)
    && ((Memory * 1024) >= ImageSize)
    ...
```

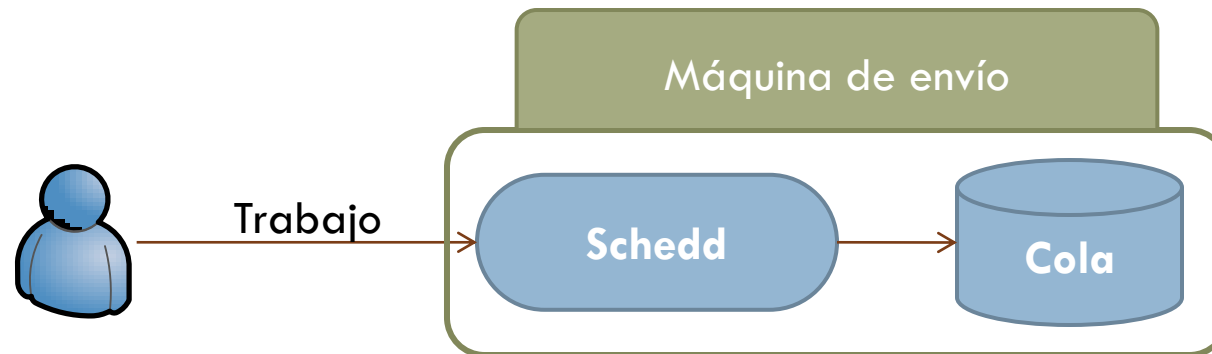
ClassAD de una máquina

```
MyType = "Machine"
TargetType = "Job"
Name = "servidor.uniovi.es"
Machine = "servidor.uniovi.es"
Rank = 0.000000
OpSys = "LINUX"
Arch = "X86_64"
Activity = "Idle"
...
```

# Introducción a Condor

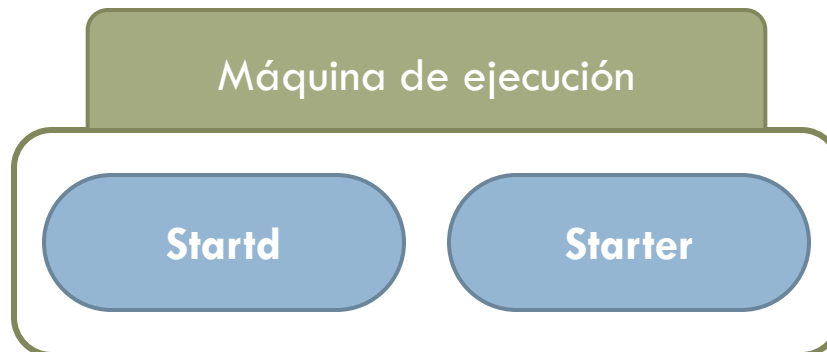
- Envío de un trabajo (*job submission*):
  - ▣ Un trabajo se describe mediante un ClassAd
  - ▣ El trabajo se envía desde una máquina que lo permita (máquina de envío o *submission machine*)
  - ▣ Cada máquina de envío, una o varias, tiene su propia cola trabajos (colas distribuidas)
  - ▣ El envío lo gestiona el servicio Schedd

Hay un servicio Schedd en cada máquina de envío



# Introducción a Condor

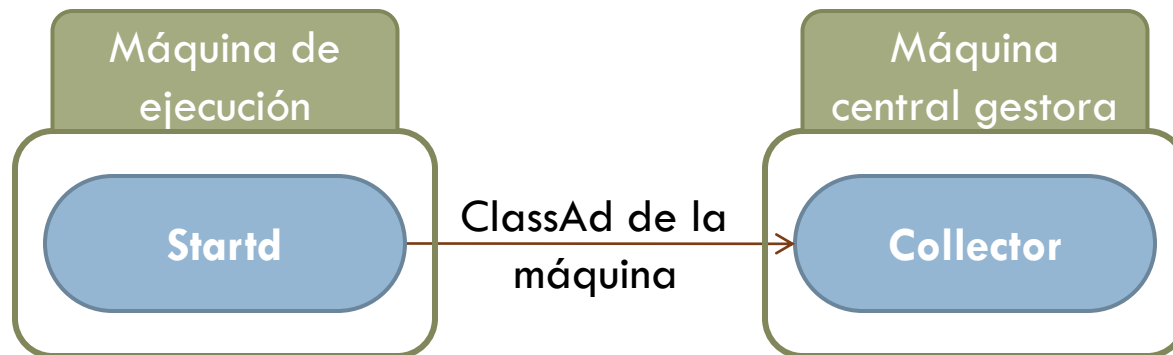
- Ejecución de trabajos:
  - ▣ La ejecución la realizan las máquinas de ejecución
  - ▣ Se responsabilizan de arrancar, pausar, y reanudar los trabajos
  - ▣ Aseguran el cumplimiento de las políticas de sus dueños
  - ▣ La ejecución se gestiona a través del servicio Startd
    - Arranca el servicio Starter para cada trabajo que ejecuta



Hay un servicio Startd en cada máquina de ejecución

# Introducción a Condor

- Hay una máquina central gestora
  - ▣ Una por pool que se encarga de realizar la gestión de trabajos y recursos
- Las máquinas de ejecución se anuncian a la máquina central gestora:
  - ▣ Información estática y dinámica:
    - Fichero de configuración con características y preferencias
    - Uso de la CPU, memoria libre...



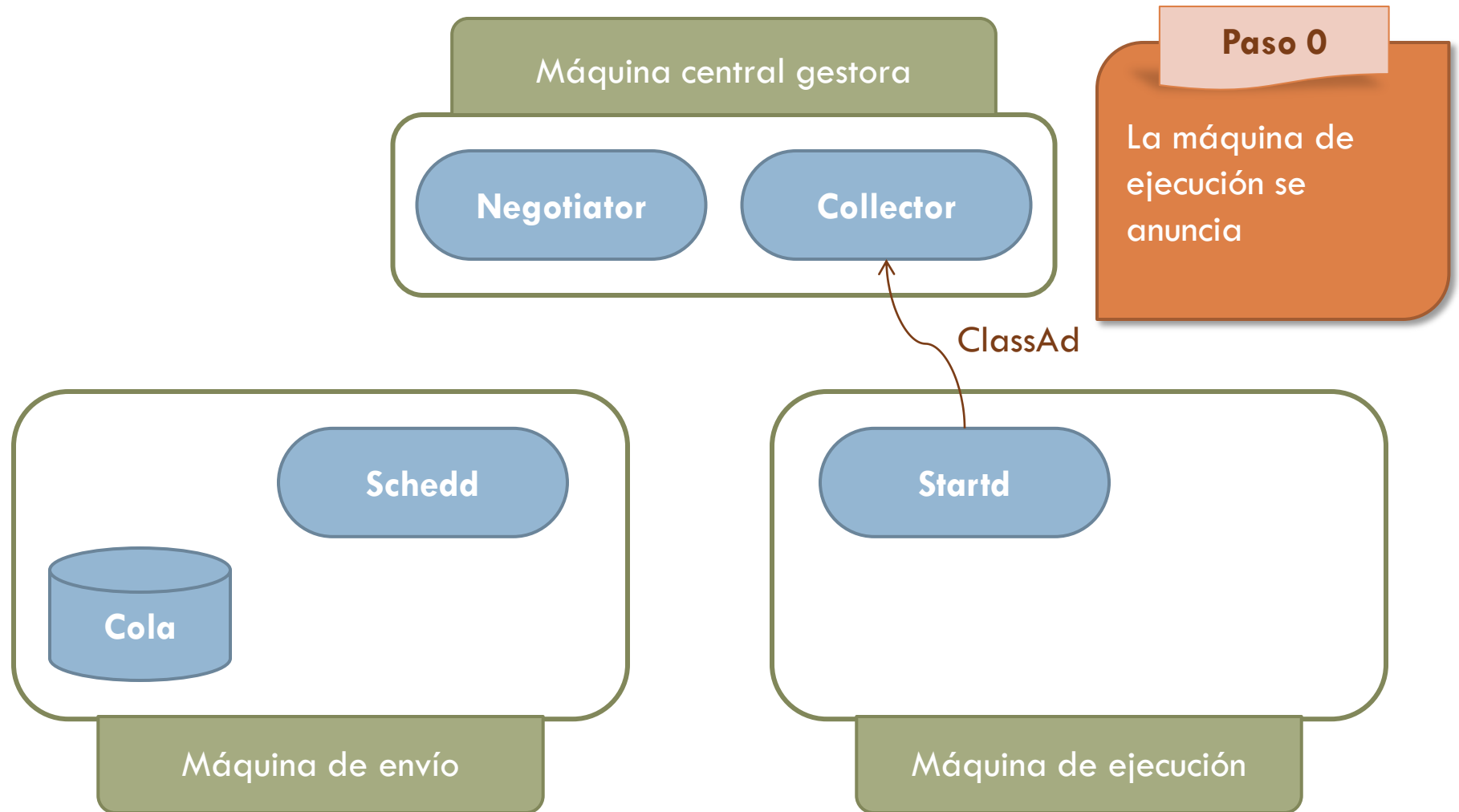
Hay un servicio Collector en la máquina central gestora

# Introducción a Condor

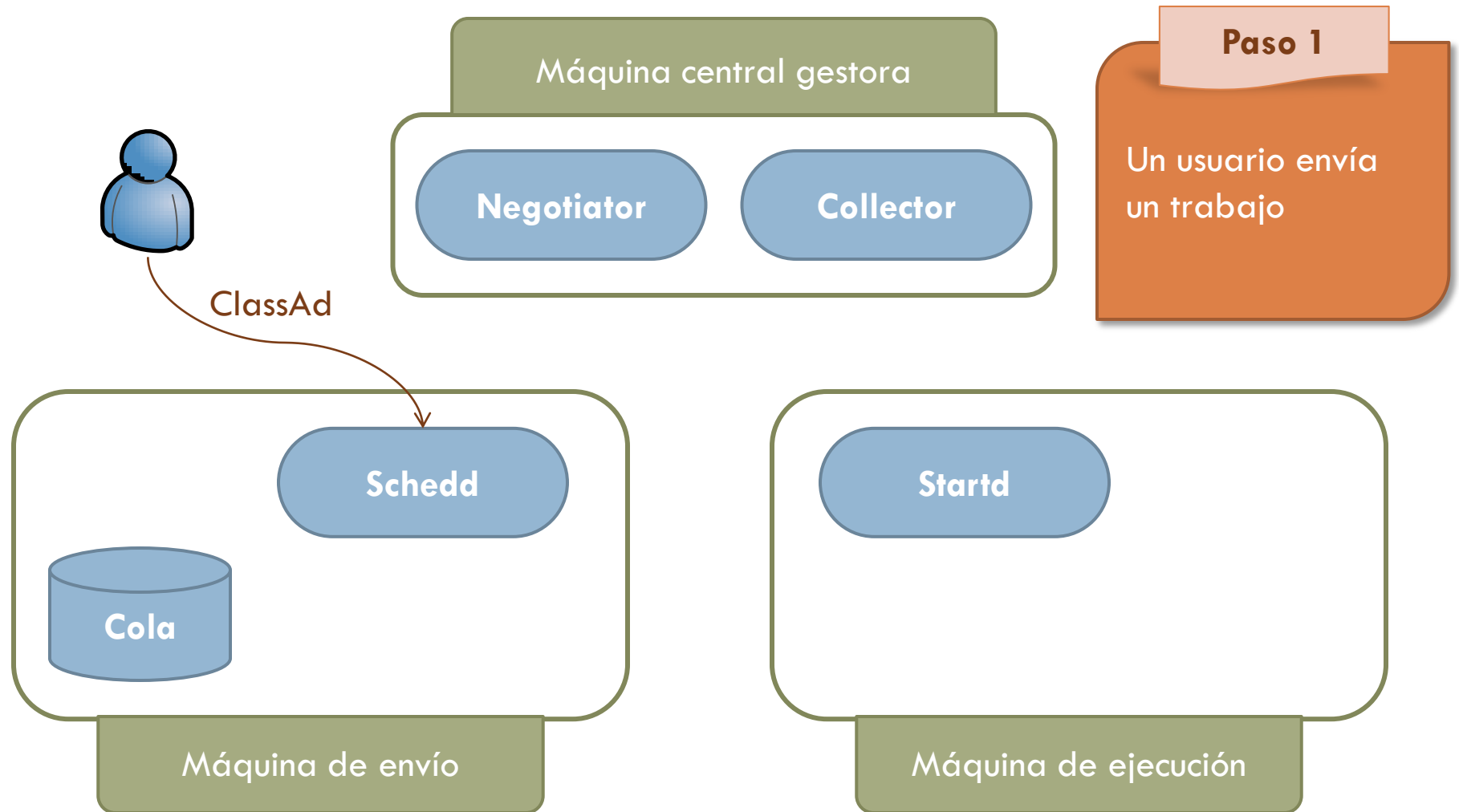
- Funcionamiento del Matchmaking:
  - ▣ Lo realiza el servicio Negotiator en la máquina central gestora
  - ▣ En cada ciclo de negociación (*Negotiation Cycle*)
    - El Negotiator contacta el servicio Collector
      - Pregunta por la información de las máquinas
    - El Negotiator contacta con cada Schedd
      - Pregunta por las características del trabajo
    - El Negotiator compara trabajos y máquinas:
      - Evalúa requisitos de ambos
      - Busca un matching

Hay un servicio Negotiator en la máquina central gestora

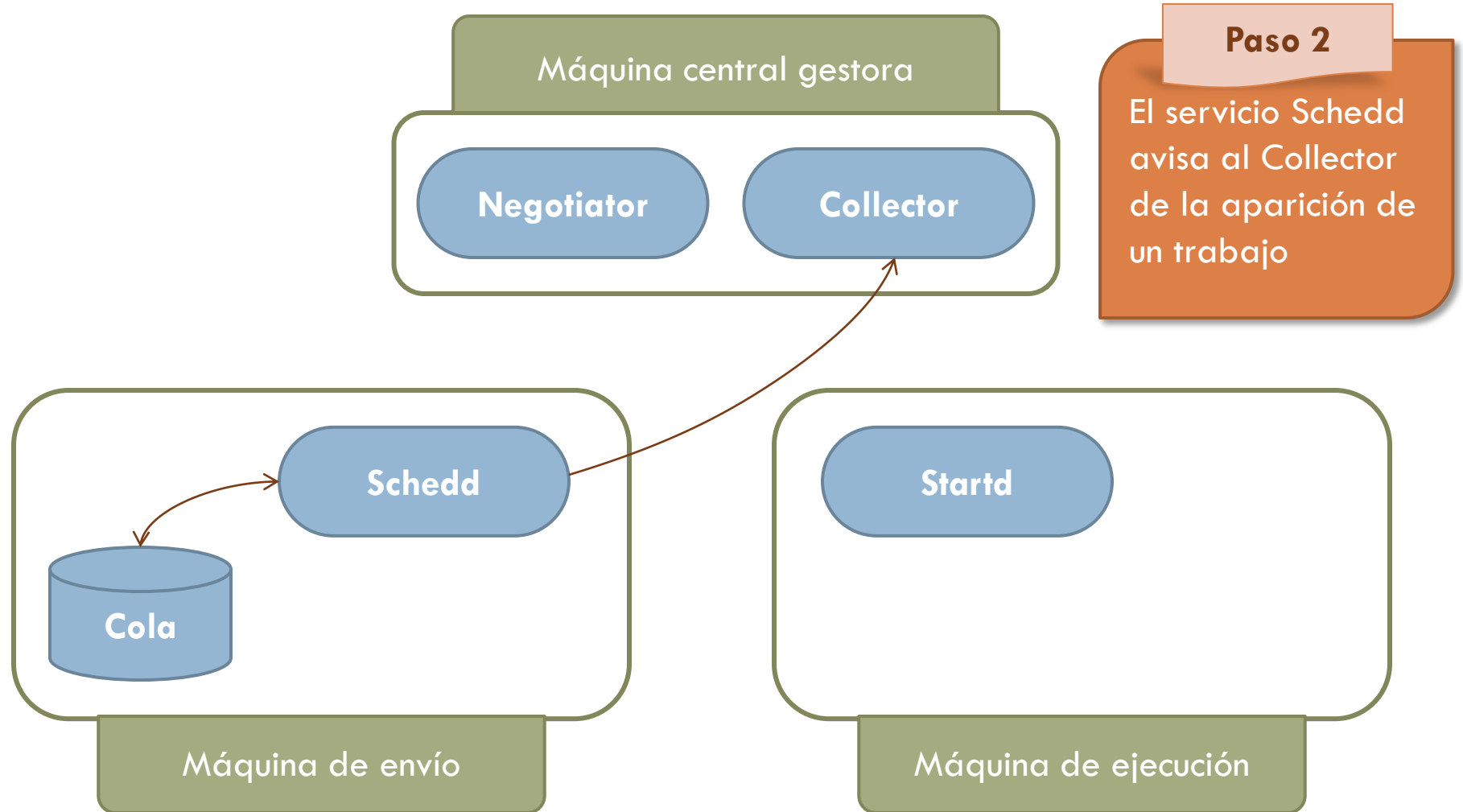
# Introducción a Condor



# Introducción a Condor

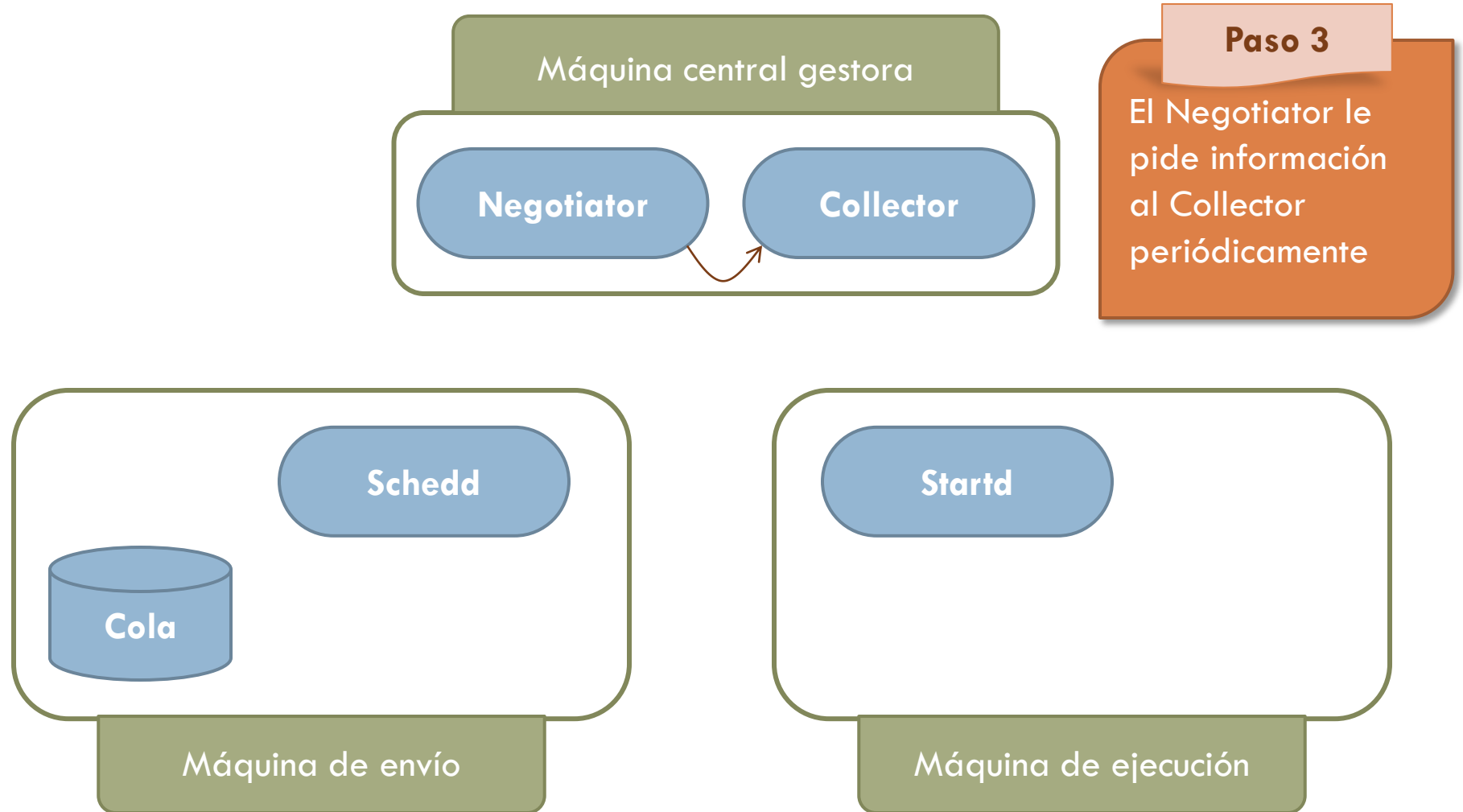


# Introducción a Condor

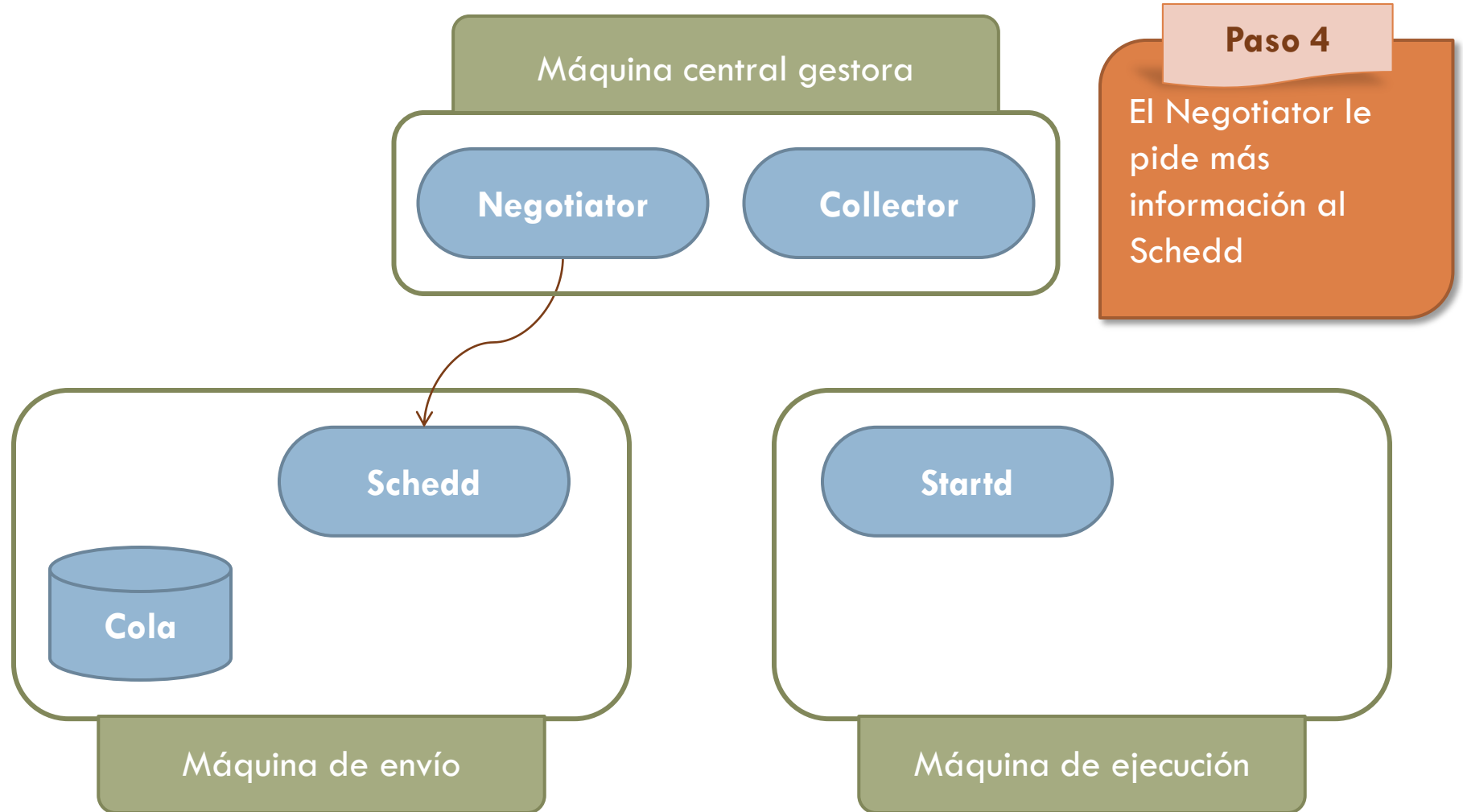




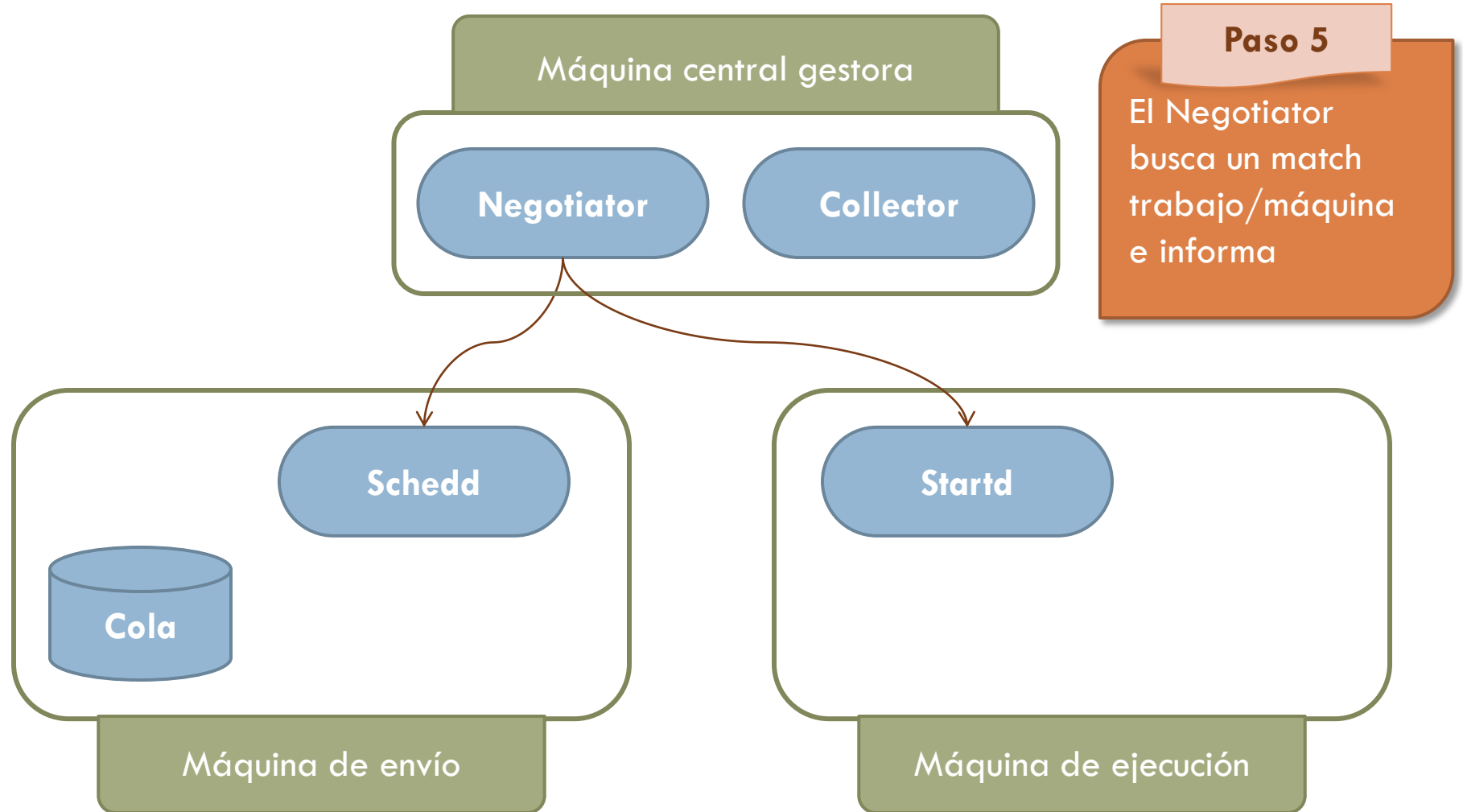
# Introducción a Condor



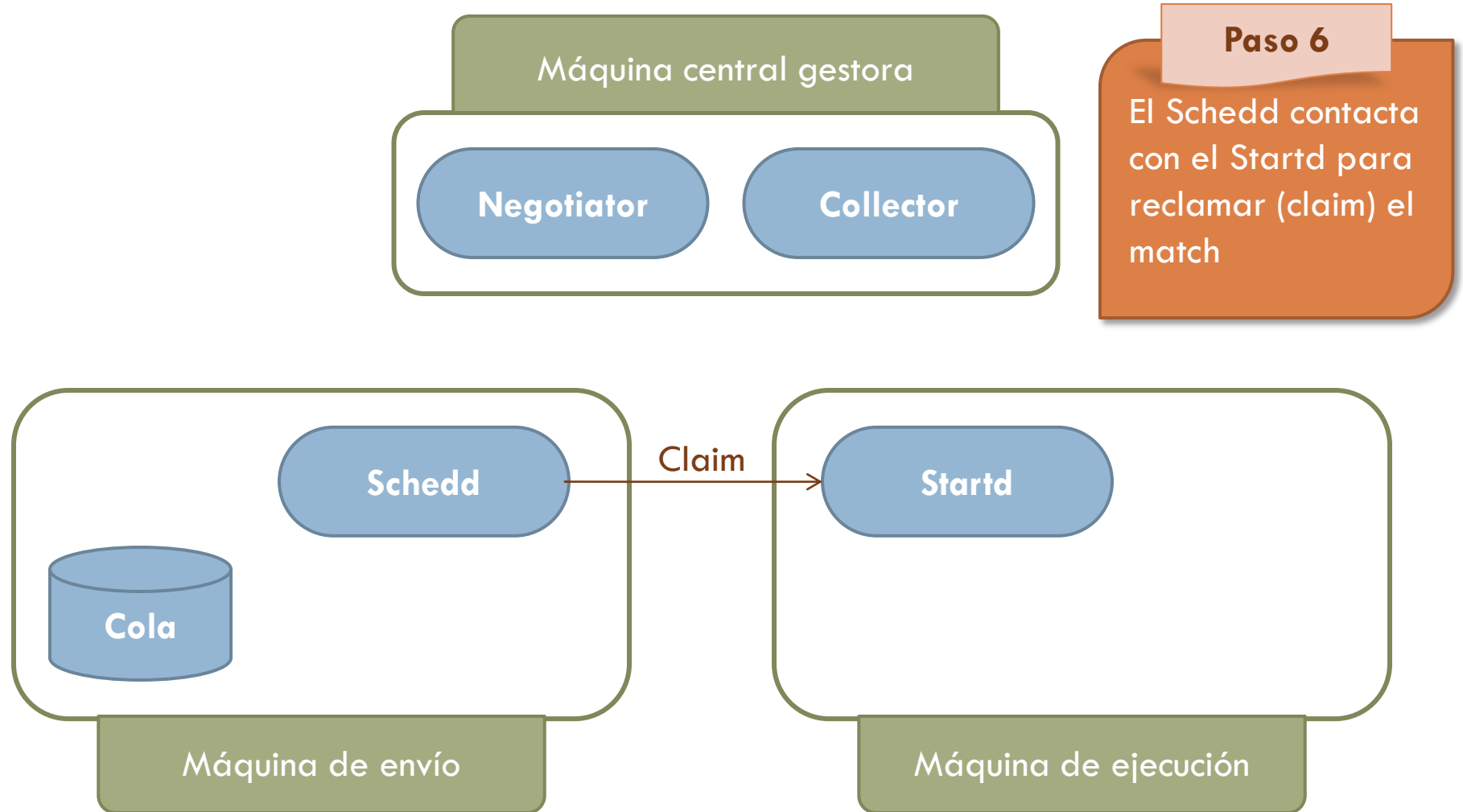
# Introducción a Condor



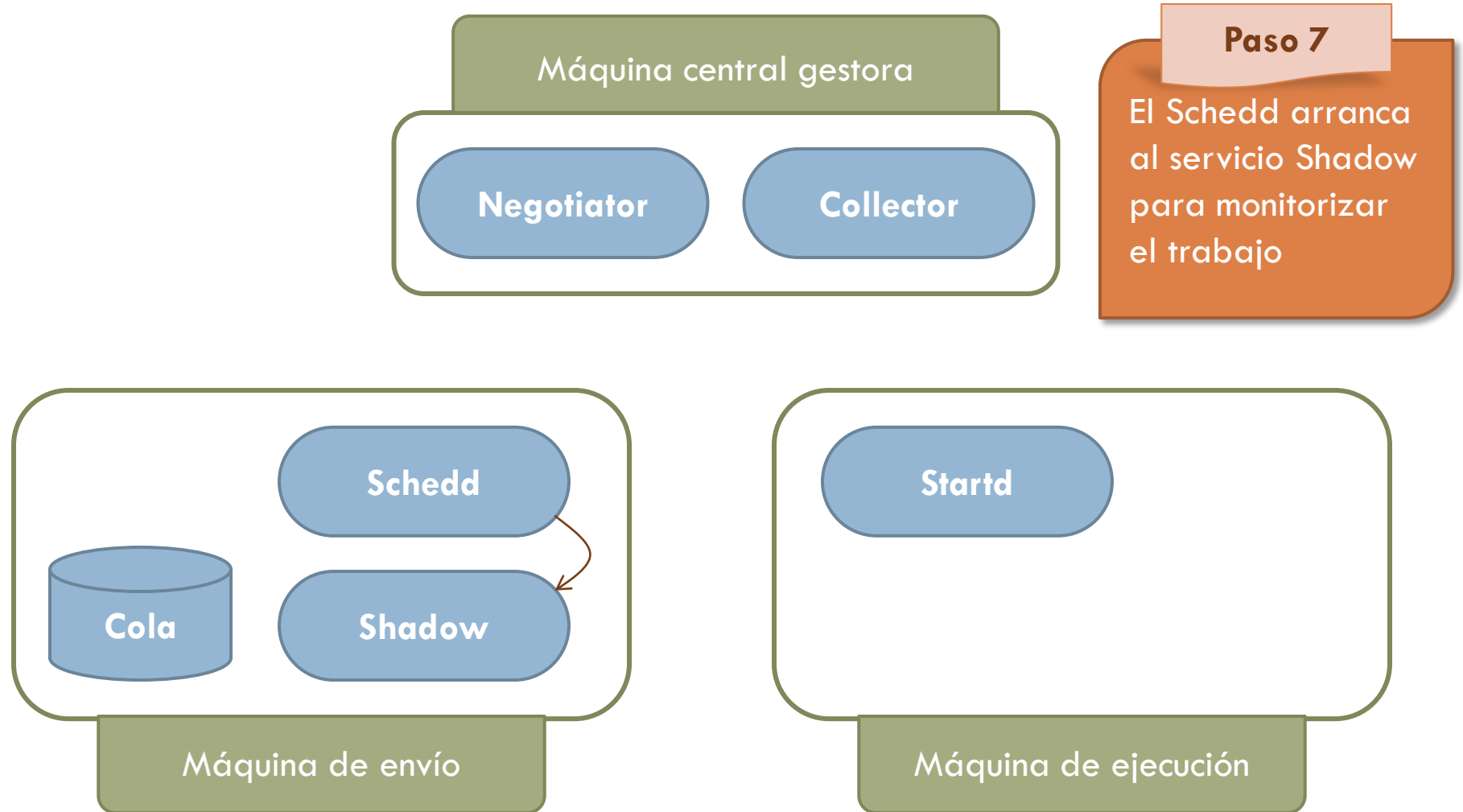
# Introducción a Condor



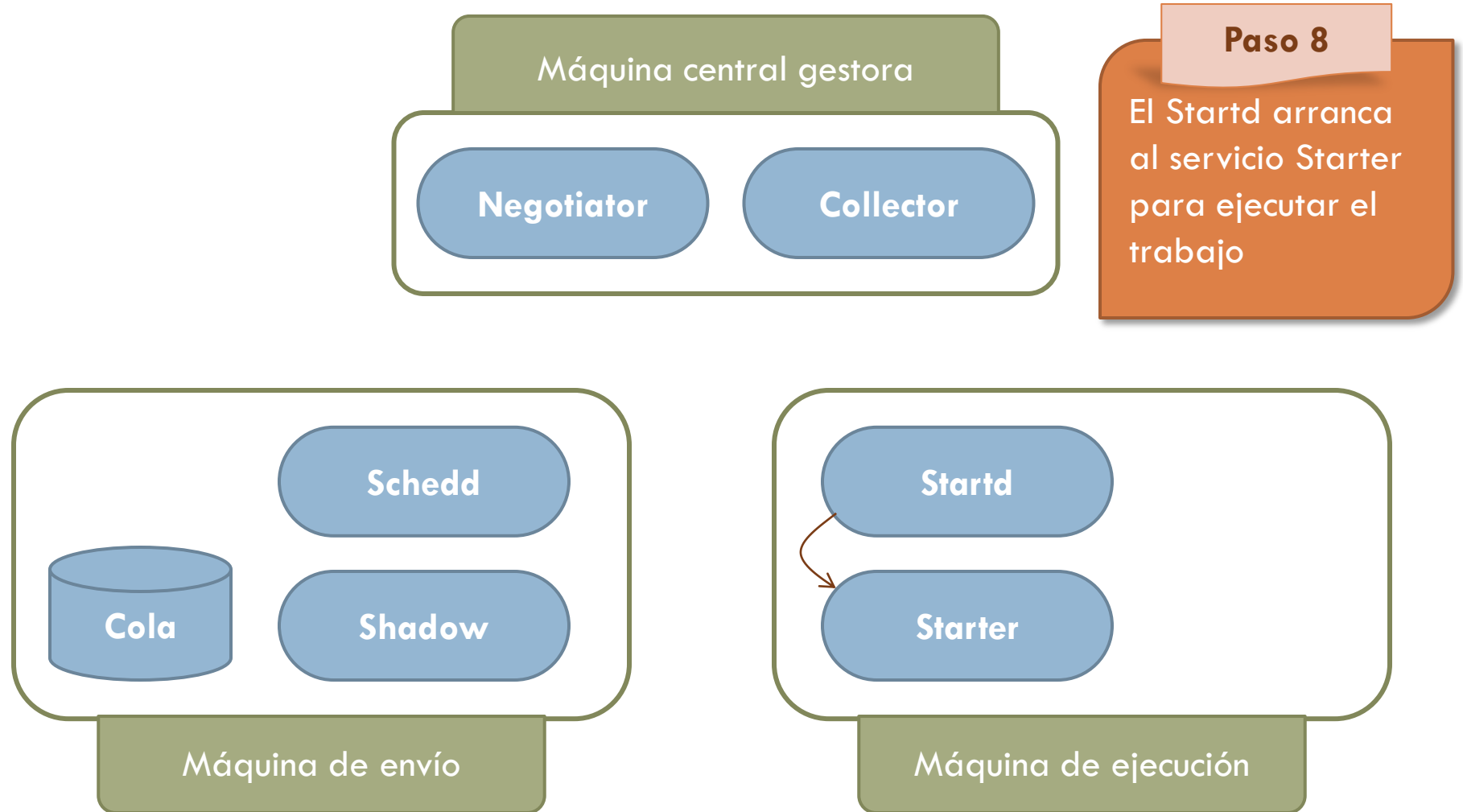
# Introducción a Condor



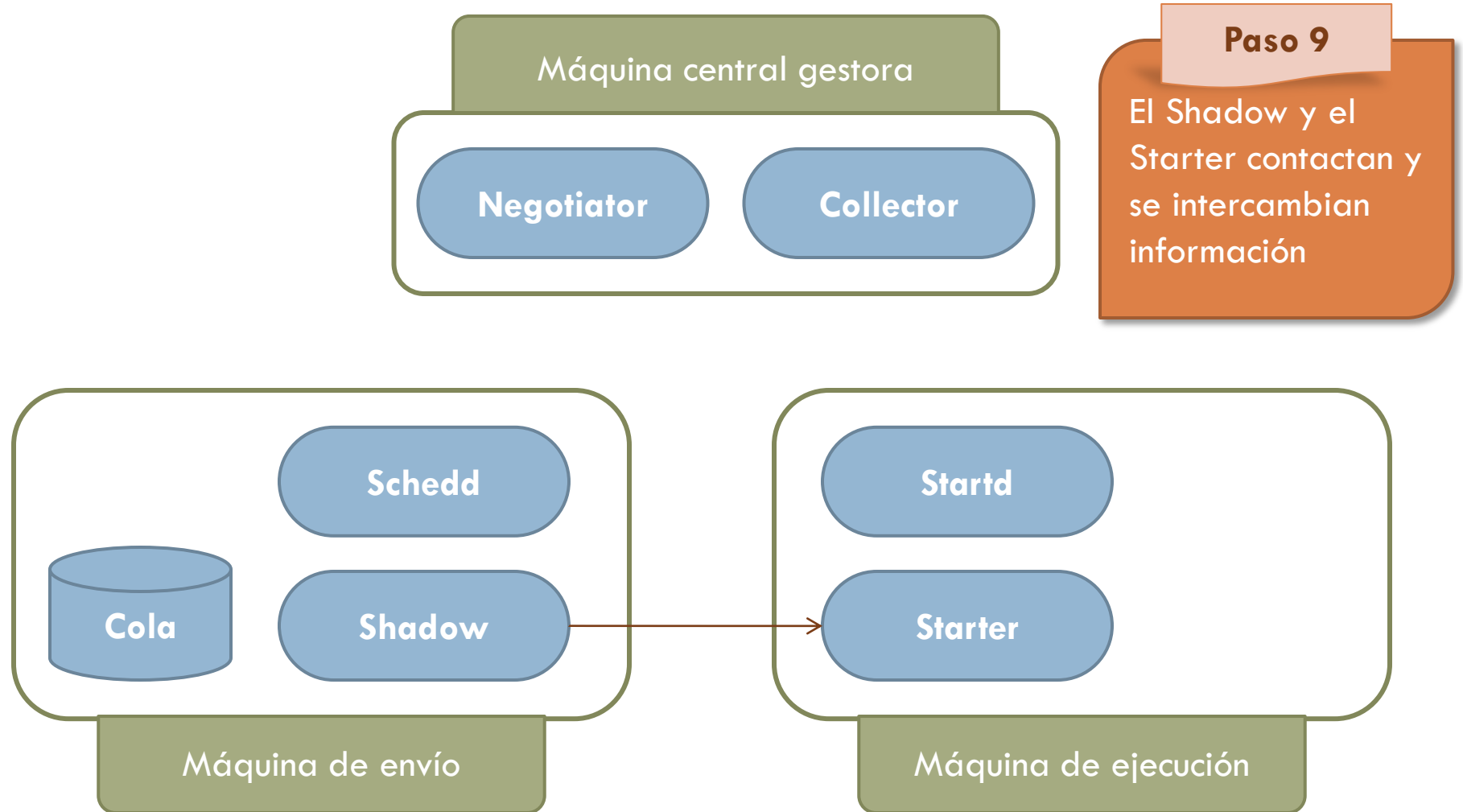
# Introducción a Condor



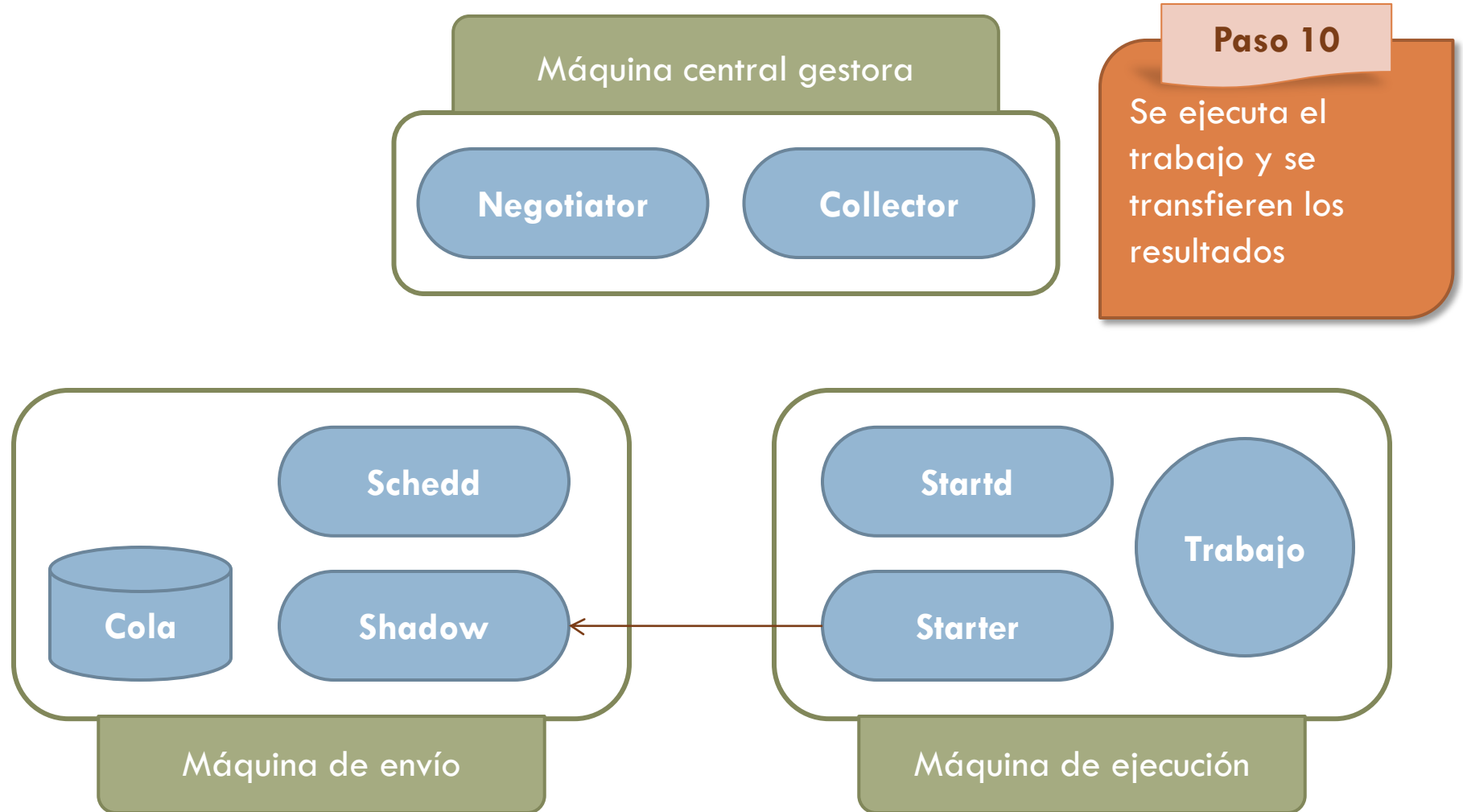
# Introducción a Condor



# Introducción a Condor



# Introducción a Condor





# Introducción a Condor

## □ Procesos de Condor:

### ▣ Master:

- Se encarga de arrancar los procesos Condor de esa máquina. Depende del rol de la máquina

### ▣ Collector:

- Almacena los ClassAds

### ▣ Negotiator:

- Realiza el matchmaking

# Introducción a Condor

## □ Procesos de Condor:

### □ Schedd:

- Gestiona la cola de trabajos

### □ Shadow:

- Monitoriza la ejecución de un trabajo

### □ Startd:

- Gestiona la ejecución de trabajos sobre una máquina

### □ Starter:

- Gestiona la ejecución de un trabajo

# Introducción a Condor

## □ Procesos Condor:

### ▣ Necesarios en un pool:

- Un Collector/Negotiator por pool
- Uno o muchos Schedd
- Unos o muchos Startd

### ▣ Algunas posibilidades de infraestructura:

- Todos los procesos en la misma máquina: pool de Condor personal
- Una máquina central gestora y muchas máquinas que sirven para enviar y para ejecutar
- Una máquina central gestora, muchas máquinas de envío y muchas máquinas de ejecución



# Condor

Ejecución de trabajos

# Ejecución de trabajos en Condor

- Pasos para ejecutar un trabajo en Condor:
  1. Elegir un universo
  2. Hacer el trabajo *batch-ready*
  3. Crear el fichero de descripción del trabajo
  4. Enviar el trabajo

# Ejecución de trabajos en Condor

## 1. Elección del universo:

- El universo indica la forma en la que Condor debe gestionar el trabajo
- Tipos:
  - *Vanilla: para casi cualquier tipo de trabajo secuencial*
  - *Standard: para trabajos con checkpoints y migración*
  - *Java: para trabajos programados en Java*
  - *VM: para máquinas virtuales*
  - *Grid: para trabajos que se ejecutan en un grid externo*
  - *Parallel: para trabajos paralelos (MPI)*
  - ...

# Ejecución de trabajos en Condor

## 2. Hacer el trabajo *batch-ready*:

- ▣ Tiene que ejecutarse en background:
  - Sin E/S interactiva, interfaz, etc.
- ▣ Se puede usar STDIN, STDOUT y STDERR:
  - Con redirección a ficheros
- ▣ Se puede acceder a otros ficheros de datos

```
$ programa < entrada.txt > salida.txt 2>errores.txt &
```

batch-ready

≈

programa clásico de shell

# Ejecución de trabajos en Condor

## 3. Crear el fichero de descripción del trabajo:

- Fichero de texto con la información sobre el trabajo:
  - Nombre del programa ejecutable
  - Argumentos del programa
  - Universo
  - Ficheros de entrada/salida
  - Variables de entorno
  - Requisitos
  - Preferencias

Fichero de descripción  
≠  
ClassAd

```
Executable = programa  
Universe = vanilla  
Input = entrada.txt  
Output = salida.txt  
Error = errores.txt  
Log = log.txt  
Queue
```

**programa.sub**



## 4. Enviar el trabajo:

- ▣ El trabajo se envía utilizando el comando `condor_submit`
- ▣ El comando `condor_submit`:
  - Procesa el fichero de descripción y crea el ClassAd
  - Envía el ClassAd al Schedd
- ▣ El Schedd almacena el trabajo en la cola
  - El comando `condor_q` permite ver el estado de la cola

# Ejecución de trabajos en Condor

- Antes del enviar un trabajo se puede comprobar el estado del pool:

```
[ruf@glite-tutor ruf]$ condor_status
```

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
slot1@iceage-wn-01	LINUX	INTEL	Unclaimed	Idle	0.000	1012	0+00:03:29
slot2@iceage-wn-01	LINUX	INTEL	Unclaimed	Idle	0.000	1012	0+01:48:25
slot3@iceage-wn-01	LINUX	INTEL	Unclaimed	Idle	0.000	1012191	16+48:32
slot4@iceage-wn-01	LINUX	INTEL	Unclaimed	Idle	0.000	1012	0+02:05:07
...							
slot3@iceage-wn-14	LINUX	INTEL	Unclaimed	Idle	0.000	1012	16+21:19:09
slot4@iceage-wn-14	LINUX	INTEL	Unclaimed	Idle	0.000	1012	16+21:19:10
Total Owner Claimed Unclaimed Matched Preempting Backfill							
INTEL/LINUX	56	0	0	56	0	0	0
Total	56	0	0	56	0	0	0

# Ejecución de trabajos en Condor

## □ Ejemplo de envío:

```
[ruf@glite-tutor ruf]$ condor_submit programa.sub
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 111.
```

```
[ruf@glite-tutor ruf]$ condor_q
```

```
-- Submitter: glite-tutor.ct.infn.it : <193.206.208.141:9674> : glite-tutor.ct.infn.it
  ID          OWNER          SUBMITTED      RUN_TIME ST PRI SIZE CMD
 111.0       ruf              1/24 13:48    0+00:00:00 R  0   0.0  programa
```

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

```
[ruf@glite-tutor ruf]$ condor_q
```

```
-- Submitter: glite-tutor.ct.infn.it : <193.206.208.141:9674> : glite-tutor.ct.infn.it
  ID          OWNER          SUBMITTED      RUN_TIME ST PRI SIZE CMD
```

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

# Ejecución de trabajos en Condor

## □ Log del trabajo:

```
[ruf@glite-tutor ruf]$ more log.txt
000 (112.000.000) 01/24 13:55:38 Job submitted from host: <193.206.208.141:9674>
...
001 (112.000.000) 01/24 13:55:41 Job executing on host: <193.206.208.205:9680>
...
005 (112.000.000) 01/24 13:55:45 Job terminated.
    (1) Normal termination (return value 0)
        Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
        Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
        Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
        Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage
56 - Run Bytes Sent By Job
5101 - Run Bytes Received By Job
56 - Total Bytes Sent By Job
5101 - Total Bytes Received By Job
...
```

El log describe la vida del trabajo

# Ejecución de trabajos en Condor

## □ Clusters y procesos:

- Al conjunto de trabajos que se describen en el fichero de envío se le denomina cluster
  - Un cluster tiene un identificador único
  - Cada trabajo en un cluster se le denomina proceso
    - Se empiezan a numerar en cero
- El identificador de un trabajo en Condor es el número de cluster seguido del número de proceso
  - ID: 10.0 (cluster 10, proceso 0)
  - ID: 10.0, 10.1, 10.2 (cluster 10, procesos 0, 1 y 2)

# Ejecución de trabajos en Condor

## □ Fichero de envío de dos trabajos:

```
Executable = programa  
Universe = vanilla
```

```
Input = entrada_0.txt  
Output = salida_0.txt  
Error = errores_0.txt  
Log = log_0.txt  
Arguments = -p 0  
Queue
```

Cluster 10, proceso 0

```
Input = entrada_1.txt  
Output = salida_1.txt  
Error = errores_1.txt  
Log = log_1.txt  
Arguments = -p 1  
Queue
```

Cluster 10, proceso 1

Suponiendo que se le  
asigna el cluster número 10

# Ejecución de trabajos en Condor

## □ Fichero de envío de dos trabajos:

```
Executable = programa
Universe = vanilla

Input = entrada_$(Process).txt
Output = salida_$(Process).txt
Error = errores_$(Process).txt
Log = log_$(Process).txt
Arguments = -p $(Process)
InitialDir = prueba_$(Process)
Queue 600
```

Se puede parametrizar

# Ejecución de trabajos en Condor

□ Envío de los trabajos:

```
griduser@C024222006:~/condor-test$ condor_submit programa600.sub
```

## Submitting

```
job(s) .....
.....
.....
.....
.....
.....
.....
.....
600 job(s) submitted to cluster 11.
```



# Ejecución de trabajos en Condor

## □ Envío de los trabajos:

```
griduser@C024222006:~/condor-test$ condor_q
```

```
-- Submitter: C024222006.ipop : <242.24.222.6:9501> : C024222006.ipop
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
11.0	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 0
11.1	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 1
11.2	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 2
11.3	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 3
11.4	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 4
11.5	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 5
...							
11.595	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 595
11.596	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 596
11.597	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 597
11.598	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 598
11.599	griduser	1/26 08:01	0+00:00:00	I	0	0.0	programa -p 599

```
600 jobs; 600 idle, 0 running, 0 held
```

# Ejecución de trabajos en Condor

## □ Eliminar trabajos:

- ▣ Para eliminar trabajos se puede utilizar el comando `condor_rm`:

- Sólo aquellos que te pertenezcan

## ▣ Ejemplos:

- `condor_rm 1 1`

- Elimina todo el cluster 1 1

- `condor_rm 1 1.1`

- Elimina el proceso 1 del cluster 1 1

- `condor_rm -all`

- Los elimina todos

# Ejecución de trabajos en Condor

- Acceso a ficheros durante la ejecución:
  - ▣ Opción 1: usar un sistema de ficheros distribuidos:
    - No forma parte de Condor
  - ▣ Opción 2: dejar que Condor transfiera los ficheros:
    - Puede transferir ficheros de entrada y de salida de forma explícita
    - En el universo standard las llamadas al sistema de E/S se redirigen, no es necesario indicar la transferencia

# Ejecución de trabajos en Condor

- Acceso a ficheros durante la ejecución:
  - ▣ Parámetro del fichero de descripción del trabajo:
    - ShouldTransferFiles:
      - YES: transferir ficheros a la máquina de ejecución
      - NO: se supone que la máquina de ejecución y la de envío comparten un sistema de ficheros
      - IF\_NEEDED: transferir sólo si es necesario

```
Universe      = vanilla
Executable    = histo.py
Arguments     = libro.txt
Output        = histo.txt
should_transfer_files    = YES
when_to_transfer_output = ON_EXIT
transfer_input_files     = libro.txt
Queue
```

# Ejecución de trabajos en Condor

## □ Más opciones del fichero de descripción:

### ▣ Requisitos:

- `Memory >= 256 OpSys == "LINUX" && Arch == "INTEL"`

### ▣ Preferencias:

- `Rank = KFLOPS + Memory`

Por defecto, Condor asume que se requiere la misma arquitectura que la máquina de envío

```
Executable = programa
Universe = vanilla
Input = entrada_$(Process).txt
Output = salida_$(Process).txt
Error = errores_$(Process).txt
Log = log_$(Process).txt
Arguments = -parametro $(Process)
InitialDir = prueba_$(Process)
Requirements = Memory >= 256 OpSys == "LINUX" && Arch == "INTEL"
Rank = KFLOPS + Memory
Queue 600
```

# Ejecución de trabajos en Condor

## □ Envío de trabajos a un pool heterogéneo:

```
universe      = vanilla
Executable    = povray.$$ (OpSys) .$$ (Arch)
Log           = povray.log
Output        = povray.out.$ (Process)
Error         = povray.err.$ (Process)

Requirements  = (Arch == "INTEL" && OpSys == "LINUX") || \
                (Arch == "INTEL" && OpSys == "SOLARIS26") || \
                (Arch == "SUN4u" && OpSys == "SOLARIS28")

Arguments     = +W1024 +H768 +Iimage1.pov
Queue

Arguments     = +W1024 +H768 +Iimage2.pov
Queue

Arguments     = +W1024 +H768 +Iimage3.pov
Queue
```

# Ejecución de trabajos en Condor

## □ Universo Standard:

- ▣ Proporciona los mecanismos necesarios para crear checkpoints, para pausar y para reanudar un proceso
- ▣ Un checkpoint almacena el estado de un proceso:
  - Memoria, CPU, E/S
- ▣ Dado un checkpoint un proceso puede ser reanudado desde el instante en el que el checkpoint se creó
- ▣ Es necesario enlazar el programa con librerías de Condor

```
$ condor_compile gcc -o programa programa.c
```

No está disponible  
sobre windows

# Ejecución de trabajos en Condor

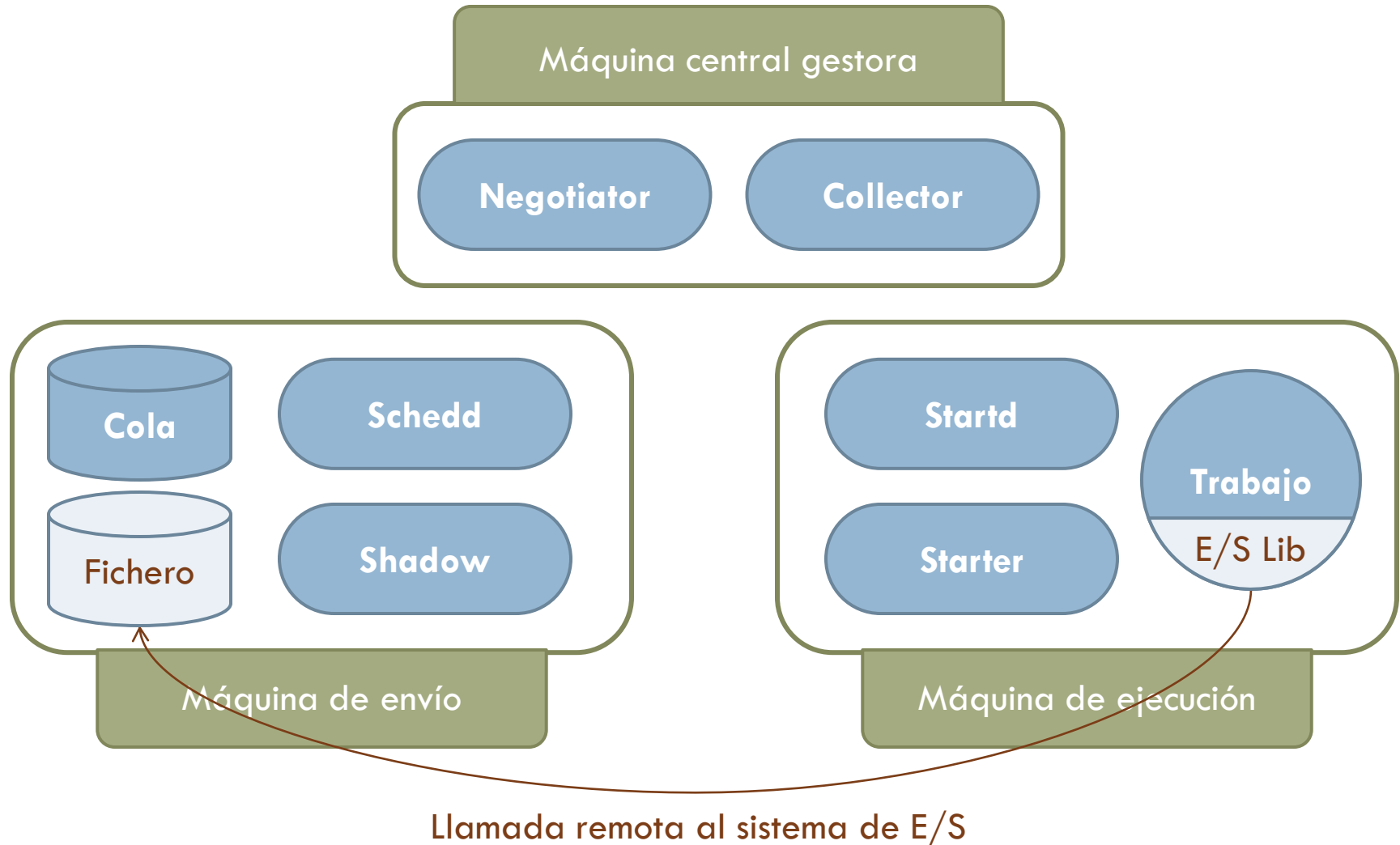
- Limitaciones de los checkpoints:
  - ▣ Un proceso no puede crear subprocesos (fork)
  - ▣ Un proceso no puede crear hilos
  - ▣ No se pueden utilizar mecanismos de IPC: pipes, memoria compartida
- ¿Cuándo se hace un checkpoint?
  - ▣ Periódicamente, si se desea
  - ▣ Cuando el trabajo es desalojado por otro de prioridad superior
  - ▣ Cuando la máquina de ejecución deja de estar libre
  - ▣ Con comandos:
    - `condor_checkpoint`, `condor_vacate`, `condor_off`, `condor_restart`



# Ejecución de trabajos en Condor

- Llamadas remotas al sistema:
  - ▣ Las llamadas al sistema de E/S son redirigidas de forma transparente desde la máquina de ejecución a la máquina de envío
    - Se pueden leer o escribir ficheros como si fueran locales
  - ▣ Permite la migración de trabajos de forma transparente:
    - Se hace un checkpoint cuando el trabajo se ejecuta en la máquina A y se continua la ejecución en la máquina B

# Ejecución de trabajos en Condor





# Condor

## Administración

# Administración de Condor

## □ Instalación en Unix:

### ■ VDT:

- `pacman -get http://vdt.cs.wisc.edu/vdt_1101_cache:Condor`

### ■ Fuentes:

- `./configure`

- `make`

- `make install`

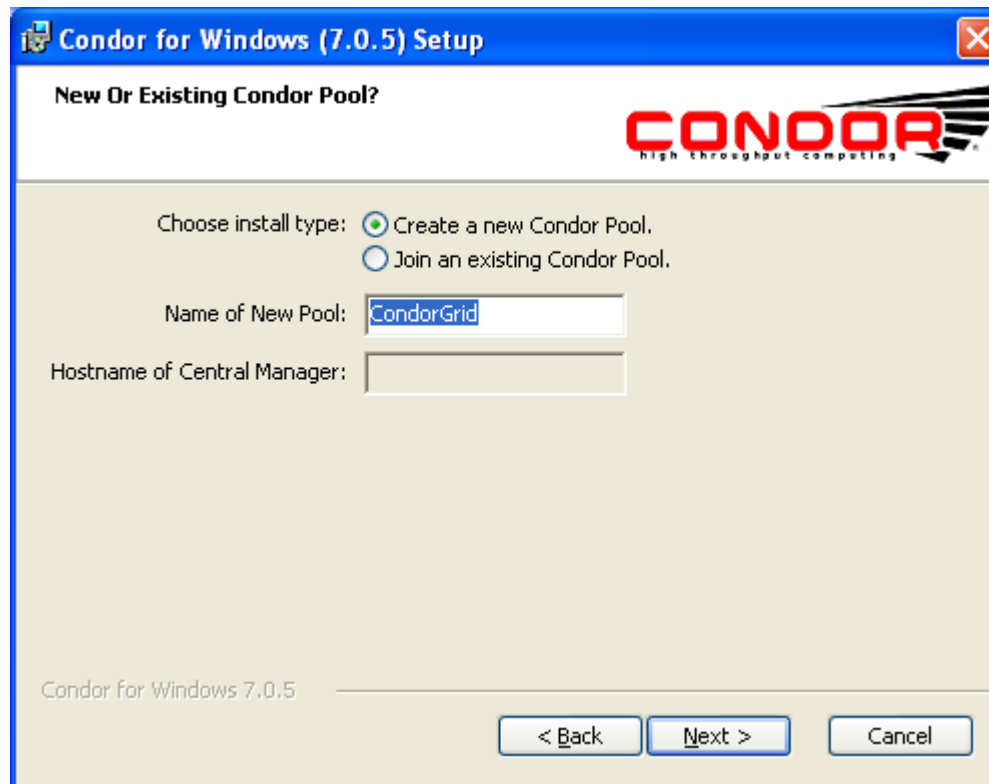
### ■ Binarios

- `rpm -i condor...`

VDT: Virtual Data  
Toolkit

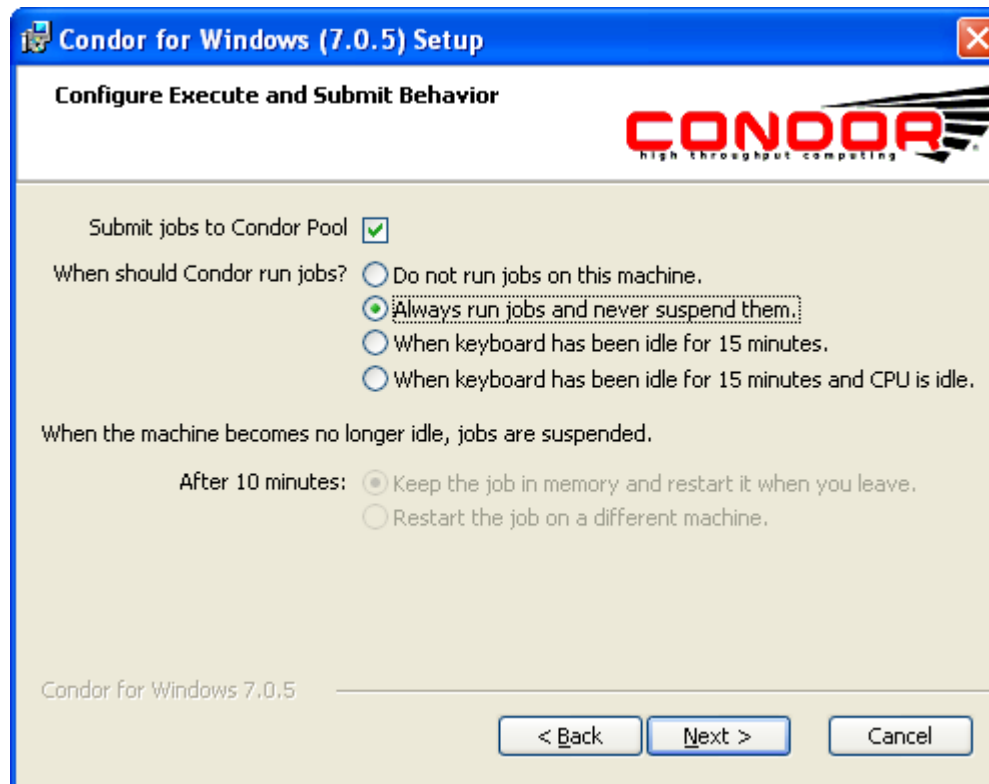
# Administración de Condor

## □ Instalación en Windows:



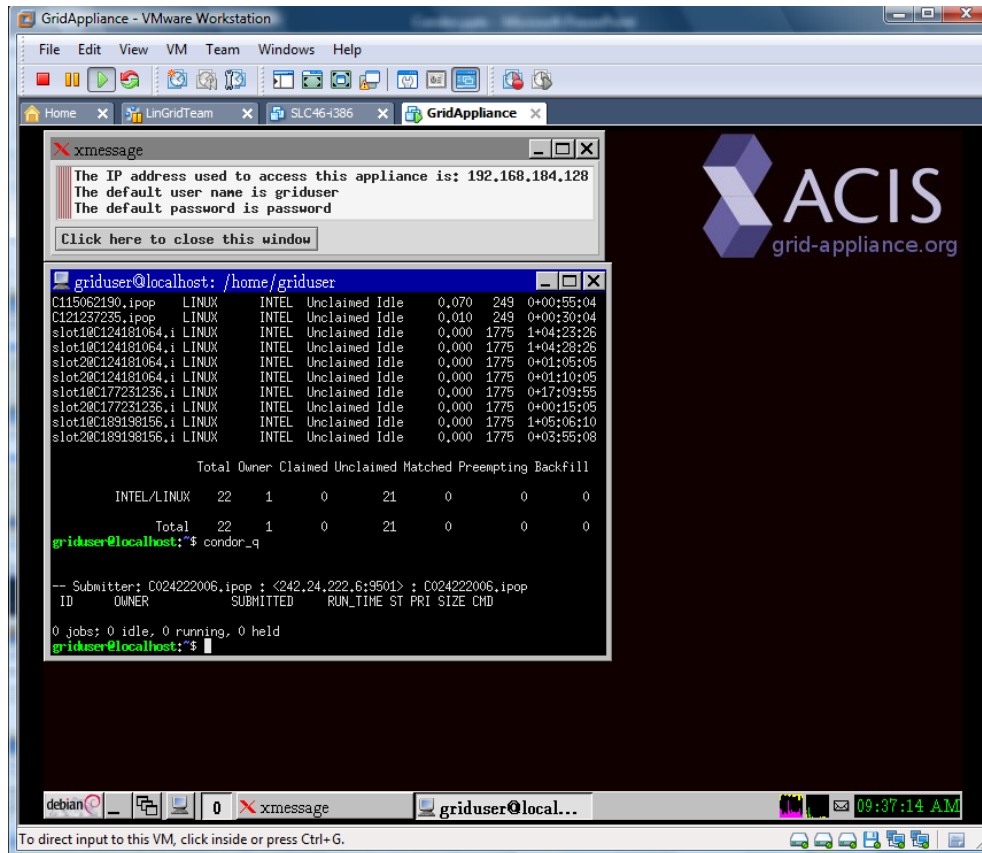
# Administración de Condor

## □ Instalación en Windows:



# Administración de Condor

## ❑ Instalación con Grid Appliance:



Listo para usar sin  
instalación

# Administración de Condor

## □ Configuración:

- Se realiza mediante ficheros de texto

- Dos tipos:

- Globales

- `/opt/condor/etc/condor_config`

- Locales

- `/var/condor/condor_config.local`

La configuración  
local tiene prioridad  
sobre la global



## □ Algunas variables configurables:

### ■ DAEMON\_LIST

- Máquina de envío:

  - DAEMON\_LIST=MASTER, SCHEDD

- Máquina de ejecución:

  - DAEMON\_LIST=MASTER, STARTD

- Máquina central gestora:

  - DAEMON\_LIST=MASTER, COLLECTOR, NEGOTIATOR

### ■ CONDOR\_HOST

- Nombre de la máquina central gestora

- Algunas variables configurables:
  - HOSTALLOW\_ADMINISTRATOR
    - Nombre de las máquinas con permisos de administración
  - HOSTALLOW\_READ
    - Nombre de las máquinas con permisos para comprobar el estado del pool
  - HOSTALLOW\_WRITE
    - Nombre de las máquinas con permisos para unirse al pool

# Administración de Condor

## □ Algunas variables configurables:

### ■ START

- Cuando ejecutar trabajos

### ■ PERIODIC\_CHECKPOINT

- Cada cuanto crear checkpoints

### ■ WANT\_VACATE

- Cuando desalojar

### ■ CONTINUE

- Cuando reanudar trabajos

Forma de indicar la creación de checkpoints cada tres horas  $\pm$  30 min:

```
(LastCkpt) > (3 * $(HOUR) + $RANDOM_INTEGER(-30,30,1) * $(MINUTE) )
```

# Administración de Condor

## □ Ejemplos de configuración:

```
...
MINUTE                = 60
ActivityTimer         = (CurrentTime - EnteredCurrentActivity)
ContinueIdleTime      = 5 * $(MINUTE)

# Only start jobs if:
# 1) the keyboard has been idle long enough, AND
# 2) the load average is low enough OR the machine is currently
#    running a Condor job
START      = ( (KeyboardIdle > $(StartIdleTime)) \
               && ( $(CPUIidle) || (State != "Unclaimed" && State != "Owner")) )

# Continue jobs if:
# 1) the cpu is idle, AND
# 2) we've been suspended more than 10 seconds, AND
# 3) the keyboard hasn't been touched in a while
CONTINUE = ( $(CPUIidle) && ($(ActivityTimer) > 10) \
             && (KeyboardIdle > $(ContinueIdleTime)) )

...
```



# Condor

## Virtualización

## □ Universos en Condor

- *Vanilla: para casi cualquier tipo de trabajo secuencial*
- *Standard: para trabajos con checkpoints y migración*
- *Java: para trabajos programados en Java*
- *VM: para máquinas virtuales*
- *Grid: para trabajos que se ejecutan en un grid externo*
- *Parallel: para trabajos paralelos (MPI)*
- ...

## □ Máquinas virtuales

- *Simulan el hardware y el software cree que se ejecuta sobre una máquina real*

## □ Ventajas

### ■ Sandboxing

- *Protegen a las máquinas de los trabajos*

### ■ Checkpoints y migración

- *Se guarda el estado completo de la máquina*

### ■ Elevación de privilegios

- *Un trabajo puede ejecutarse como root*

### ■ Independencia de plataforma

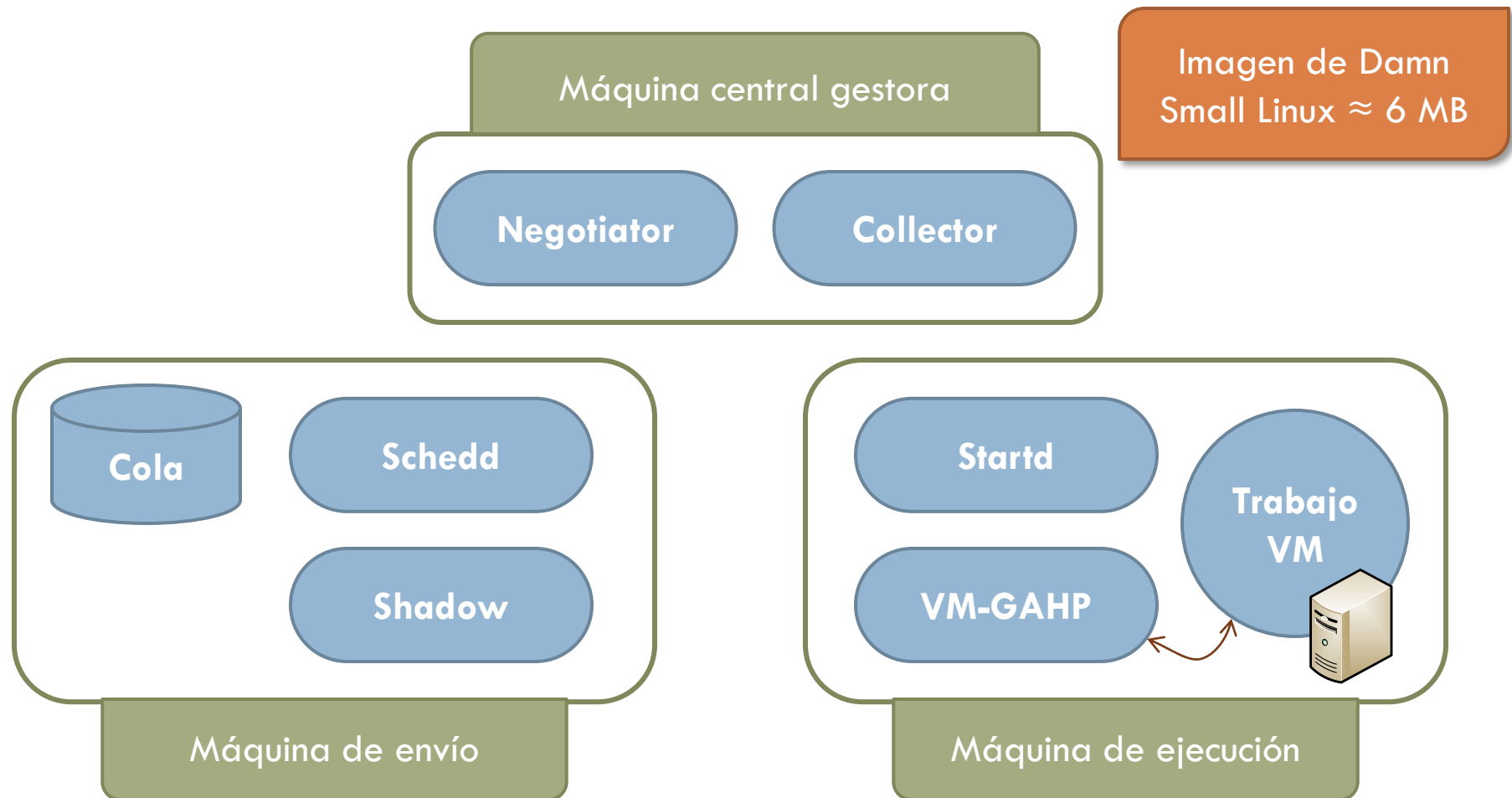
- *Un programa windows se puede ejecutar sobre linux y viceversa*

## □ Universo VM

- ▣ La imagen de la máquina virtual es el trabajo
- ▣ La salida del trabajo es la imagen modificada
- ▣ En la máquina de ejecución, el Stard inicia el servicio VM-GAHP
  - VM-GAHP interactua con la máquina virtual (vmware, xen)



# Virtualización



- Checkpoints y migración
  - ▣ No es necesario enlazar con las librerías de Condor
  - ▣ Tiene menos limitaciones que el universo Standard
    - Se pueden crear hilos y procesos
  - ▣ No hay E/S remota
- Configuración del SO
  - ▣ La aplicación se debe ejecutar cuando al SO arranque
  - ▣ El SO debe terminar cuando termine la aplicación



# Condor

Trabajos con dependencias

# Trabajos con dependencias

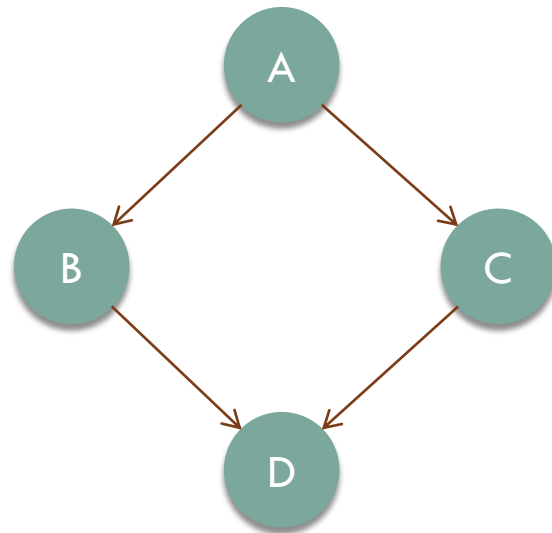
- DAG (Directed Acyclic Graph)
  - ▣ Un DAG se utiliza para representar un conjunto de trabajos mediante un esquema lógico donde la entrada, salida o ejecución de un trabajo dependen de otros trabajos
- DAGMan (DAG Manager)
  - ▣ Mecanismo que implementa Condor para gestionar las dependencias entre trabajos

DAGMan es un sistema de gestión de flujo de carga computacional (workflow system)

# Trabajos con dependencias

## □ Ejemplo de DAG

- ▣ No ejecutar B hasta que termine A
- ▣ No ejecutar C hasta que termine A
- ▣ No ejecutar D hasta que termine B y C



Cada nodo del DAG  
es un trabajo

# Trabajos con dependencias

## □ Descripción del DAG

- ▣ Se describe mediante un fichero donde se indican las dependencias de cada trabajo

```
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub

Parent A Child B C
Parent B C Child D
```

**trabajo.dag**

Cada nodo del DAG es un trabajo que tiene su propio fichero de descripción

# Trabajos con dependencias

## □ Envío del DAG

- ▣ Para enviar el DAG se utiliza el comando `condor_submit_dag`

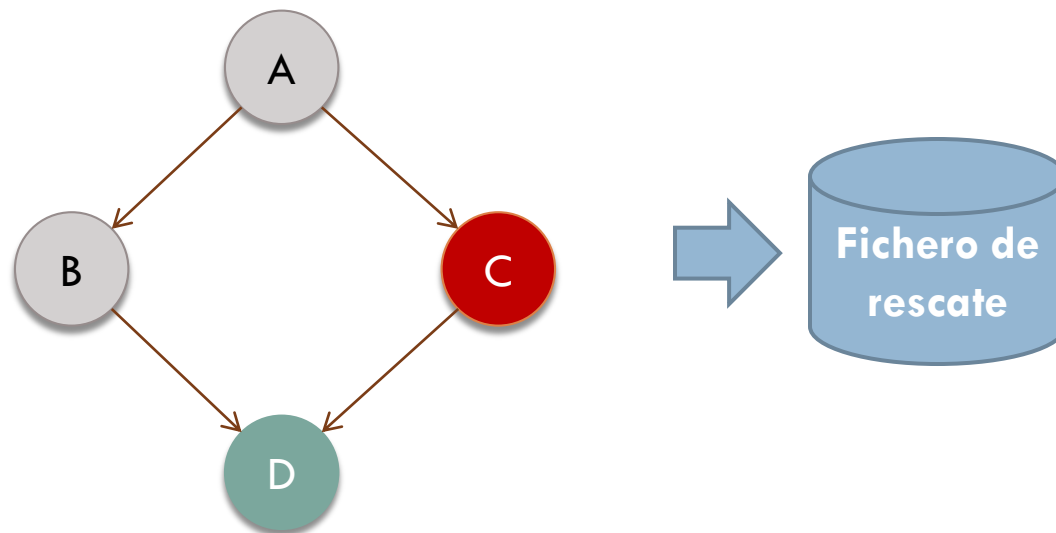
```
$ condor_submit_dag trabajo.dag
```

- ▣ `condor_submit_dag` envía el ejecutable DAGMan al universo Scheduler
- ▣ DAGMan se encarga de enviar los trabajos indicados en el fichero `trabajo.dag` en el momento adecuado según las dependencias descritas

Los trabajos en el universo Scheduler se ejecutan sobre la máquina de envío

# Trabajos con dependencias

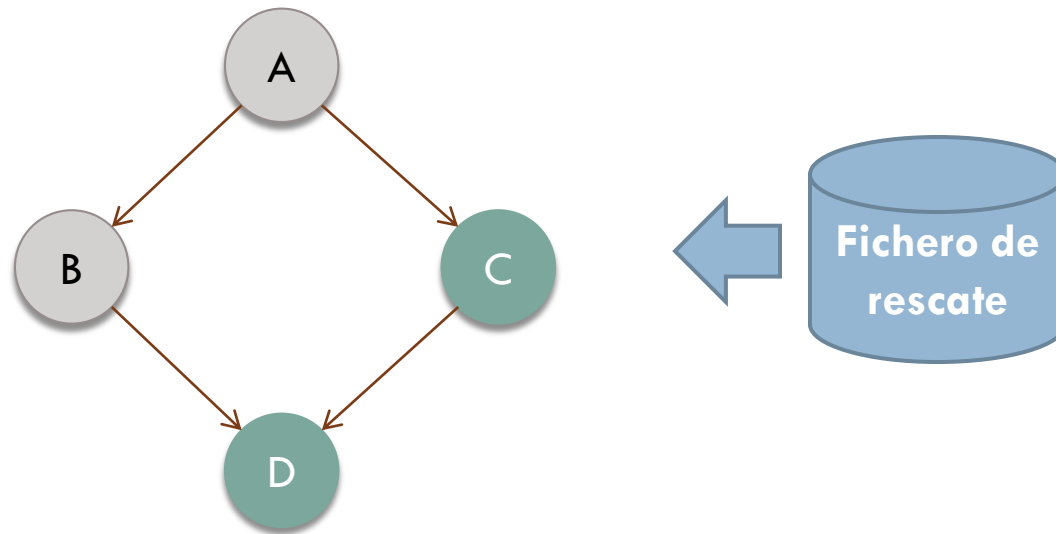
- Gestión de errores en el DAG
  - ▣ En caso de errores en un trabajo, DAGMan continua hasta que no pueda progresar más y luego crea un fichero de rescate con el estado del DAG





# Trabajos con dependencias

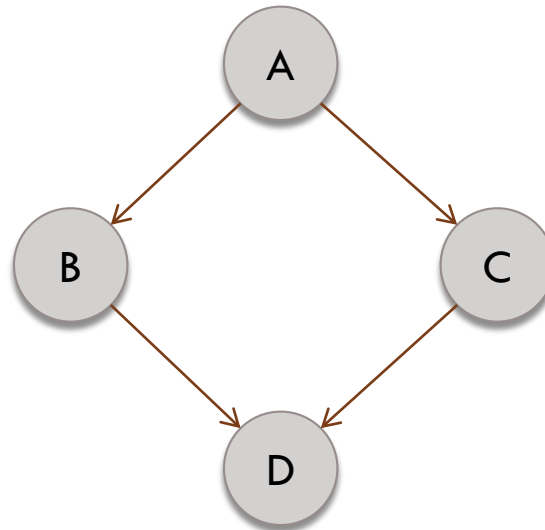
- Gestión de errores en el DAG
  - ▣ Cuando el trabajo con errores está listo para volver a ejecutarse, se restaura el estado del DAG gracias al fichero de rescate



# Trabajos con dependencias

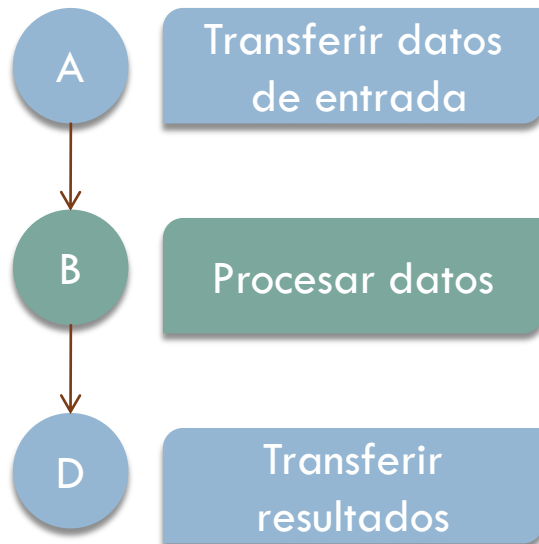
## □ Finalización del DAG

- Cuando el DAG es completado, el trabajo DAGMan termina



# Trabajos con dependencias

- Stork (cigüeña)
  - ▣ Sistema de planificación de transferencia de datos
  - ▣ Se integra con DAGMan



```
Data Input  transfer_input.stork
Job Proceso trabajo.sub
Data Output transfer_output.stork
```

```
Parent Input  Child Proceso
Parent Proceso Child Output
```

**trabajo.dag**

```
[
  dap_type = transfer;
  src_url = "ftp://servidor/fich.dat";
  dest_url = "file:/tmp/fich.dat";
]
```

**transfer\_input.stork**



# Condor

## Grid

## □ Grid con Condor:

### □ Pool Condor distribuido

- Conjunto de máquinas conectadas a través de internet
- Requiere coordinación entre las máquinas

### □ Condor flocking

### □ Universo grid

Un pool de Condor también se puede ver como un cluster

## □ Condor flocking

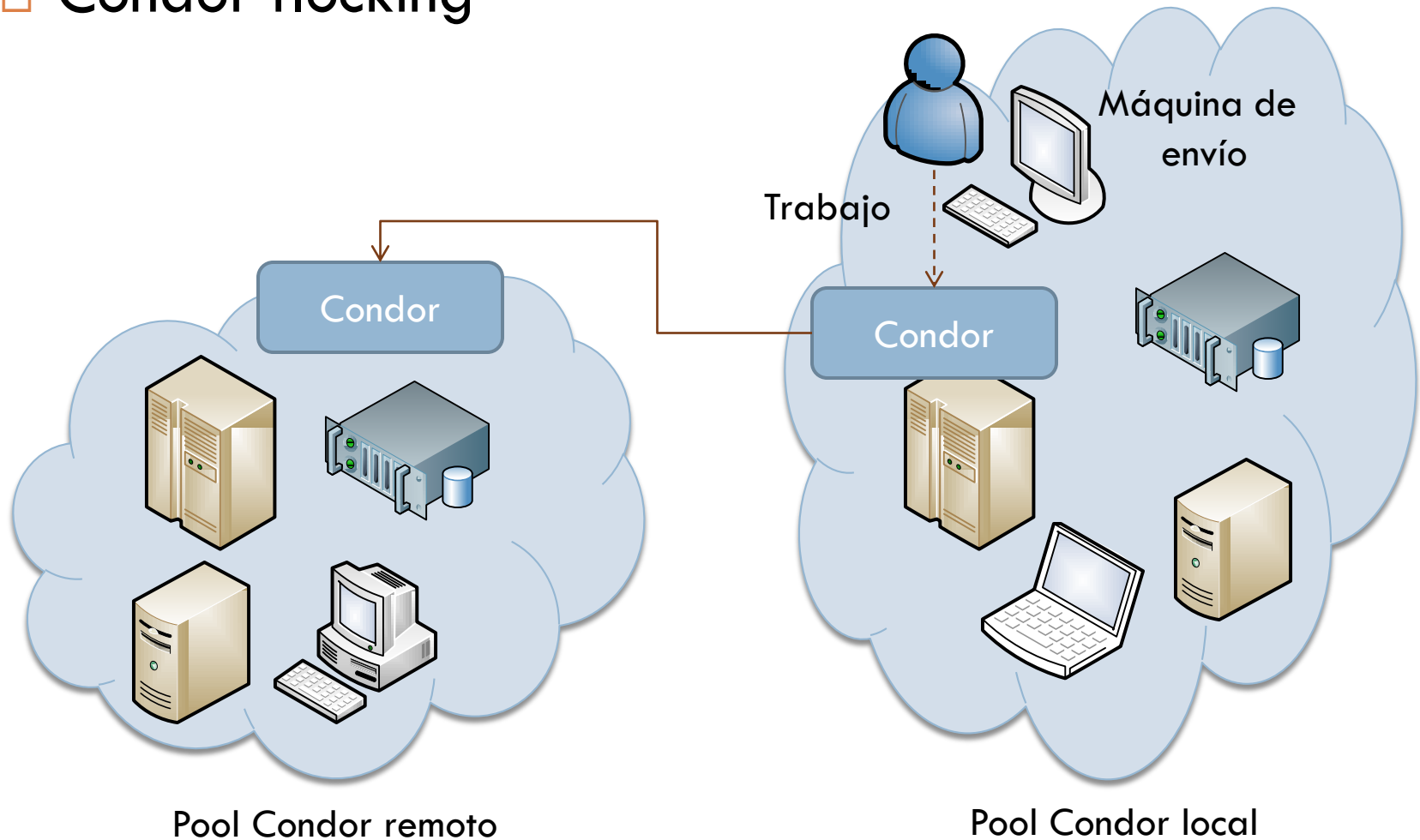
- Una máquina de envío pertenece a un pool
- Se puede configurar la máquina de envío para que envíe trabajo a otro pool en caso de que su pool no pueda atender sus necesidades
- Un trabajo se puede ejecutar sobre el pool propio o sobre otro en el que tenga permisos el usuario

```
...  
FLOCK_HOSTS = Pool12, Pool13  
...
```

condor\_config

# Grid

## □ Condor flocking



## □ Condor flocking

- La lista de pools remotos es propiedad del Schedd que se ejecuta en la máquina de envío
  - Cada usuario puede tener acceso a sus pools remotos
  - Cada pool le permite el acceso a un usuario concreto
- Los pools remotos se contactan en el orden especificado
  - Los usuarios locales del pool tiene prioridad sobre los usuarios que utilizan flocking