

Lección 11: Coordinación mediante espacios de tuplas

- Introducción
- El modelo de coordinación Linda
- Ejemplos
- Aproximación conceptual a los monitores
- Ejercicios

Introducción

- Comunicación síncrona implica fuerte acoplamiento
 - tiempo y espacio
 - ambos procesos han de existir
 - la estructura de red ha de ser conocida por los procesos
- Modelo de coordinación Linda
 - desacoplar procesos
 - almacenamiento persistente de datos

El modelo de coordinación Linda

- David Gelernter, 1985
Generative communication in Linda
ACM Transactions on Programming Languages and Systems
Volume 7, Issue 1 (January 1985), pp. 80-112
- Linda es un lenguaje/ sistema de coordinación
- Basado en un espacio “lógico” de memoria compartida: el *espacio de tuplas*
- Los procesos se coordinan:
 - introduciendo tuplas en el espacio
 - retirando, *de manera selectiva*, tuplas del espacio
 - consultando la existencia de tuplas en el espacio
- ¿Qué son tuplas?
 - versión general: una especie de listas anidadas (árboles)
 - listas con elementos que pueden ser listas
 - versión original: una especie de listas planas

JavaSpaces
GigaSpaces
TSpaces
JXTASpaces
XMLSpaces
C-Linda

El modelo de coordinación Linda

- Ejemplos de tuplas:

["v", 9, 4] ["v", 9, 7] ["w", 2, 4]

["María", "Luis"] ["s", 9]

- Ejemplos de patrones:

["v", x, y]

Process P::

 string s

["v", x, 4]

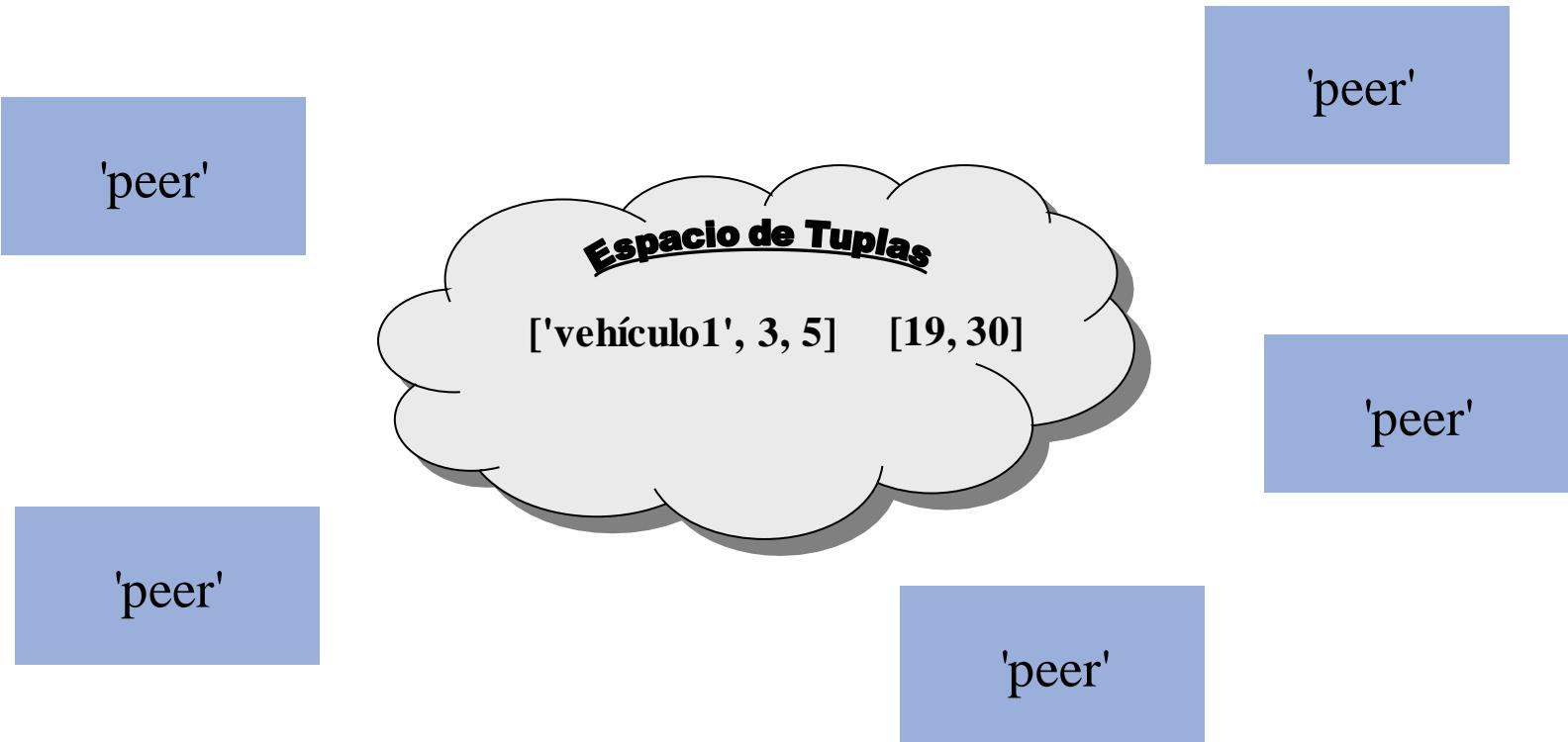
["s", x, 3+1]

 int x,y

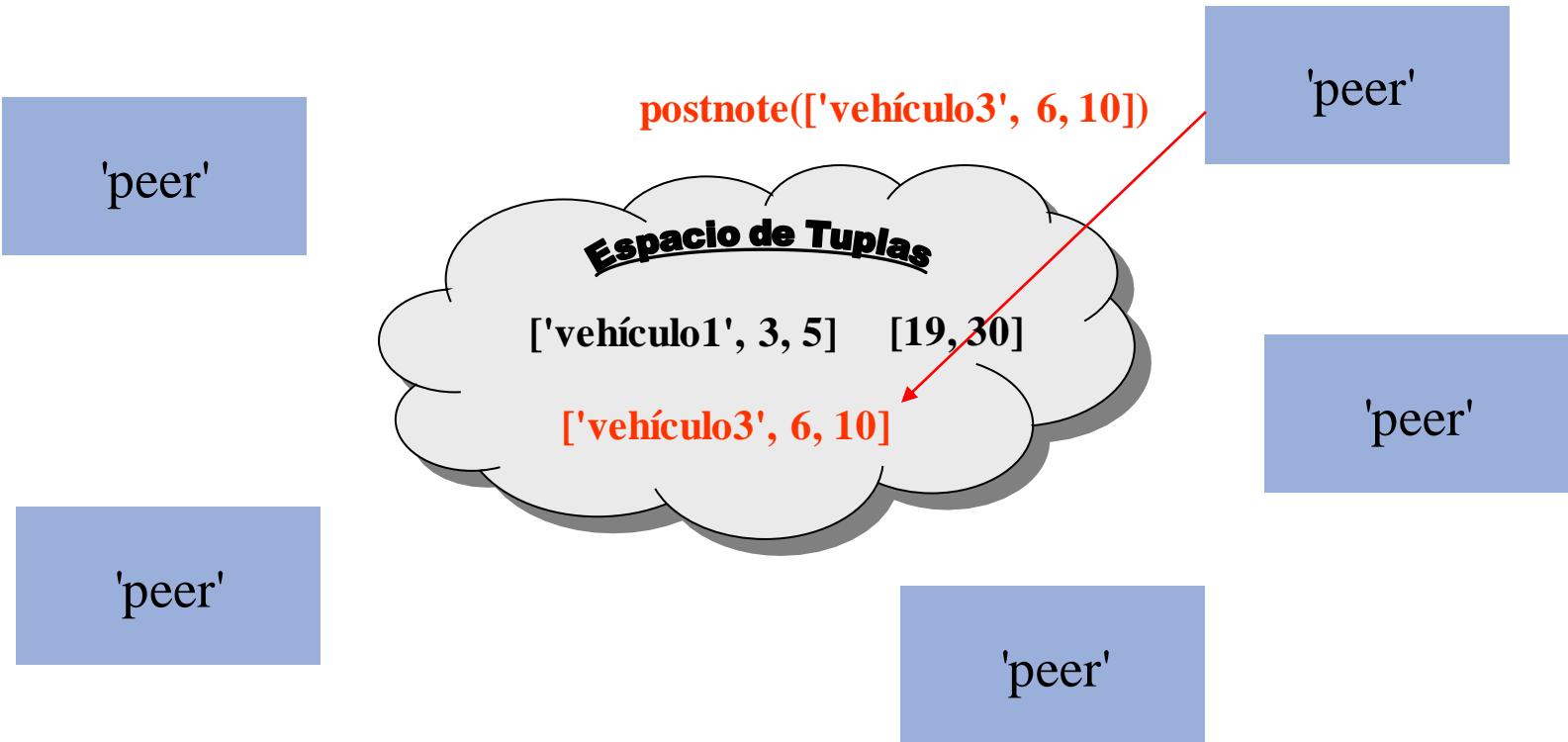
[s, x, 4]

[s, x=, 4]

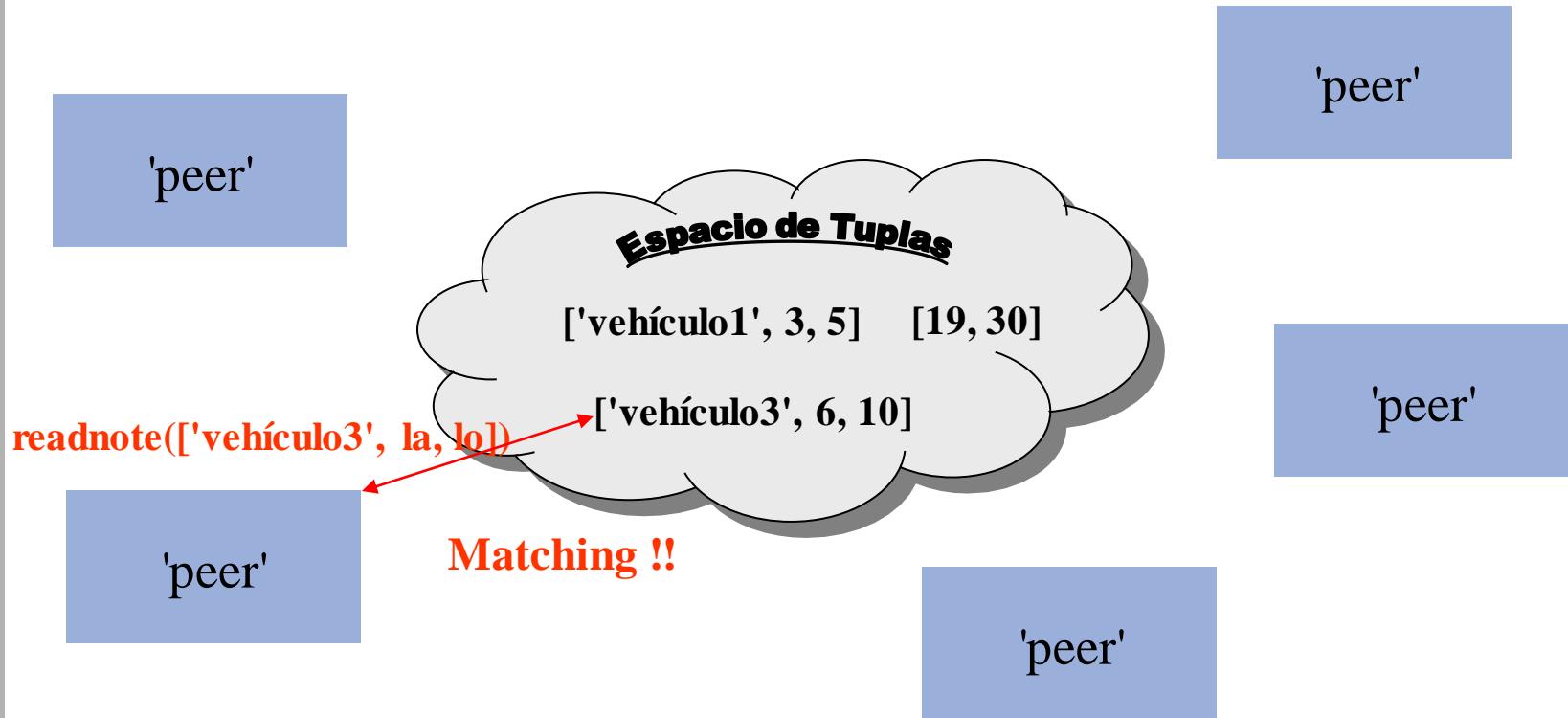
El modelo de coordinación Linda



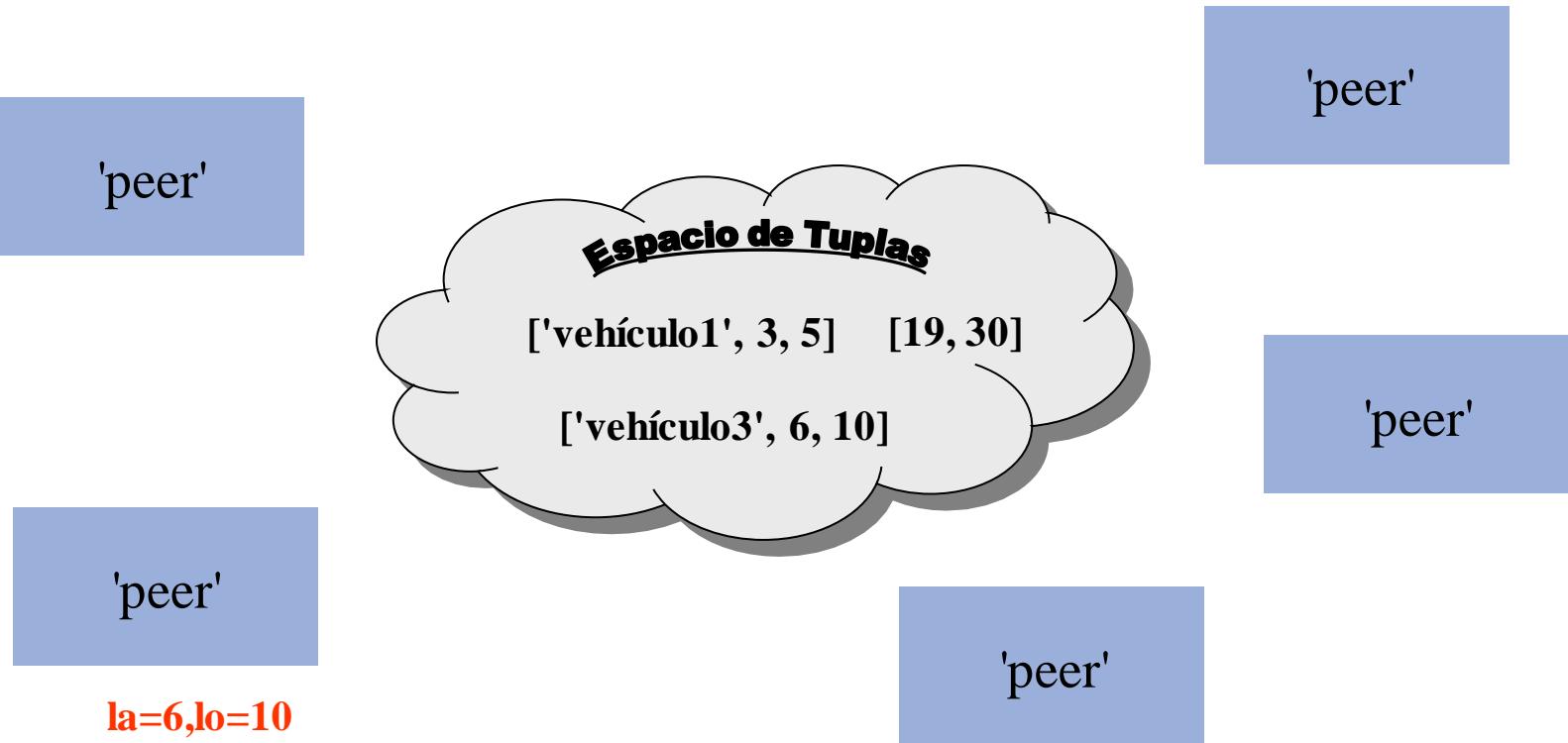
El modelo de coordinación Linda



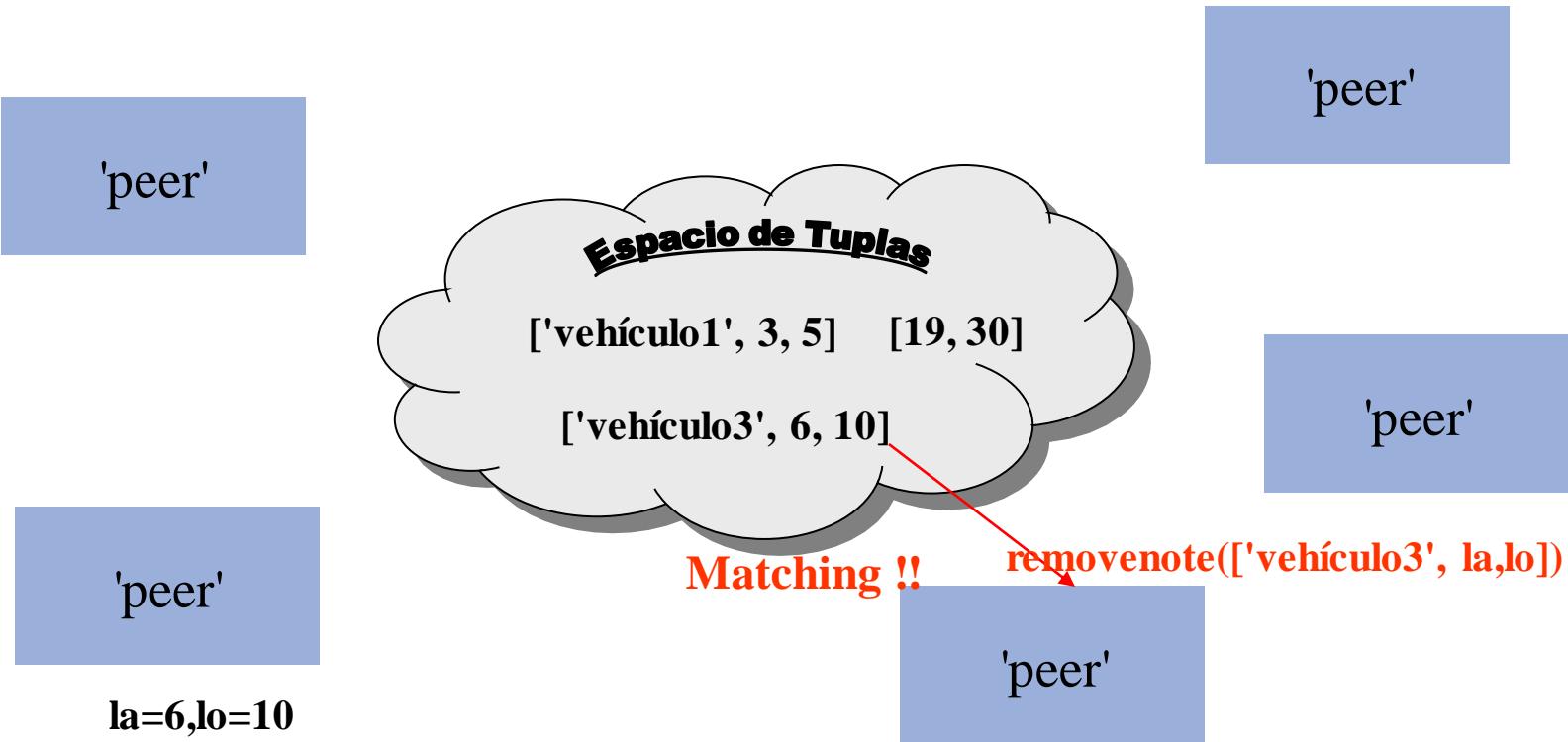
El modelo de coordinación Linda



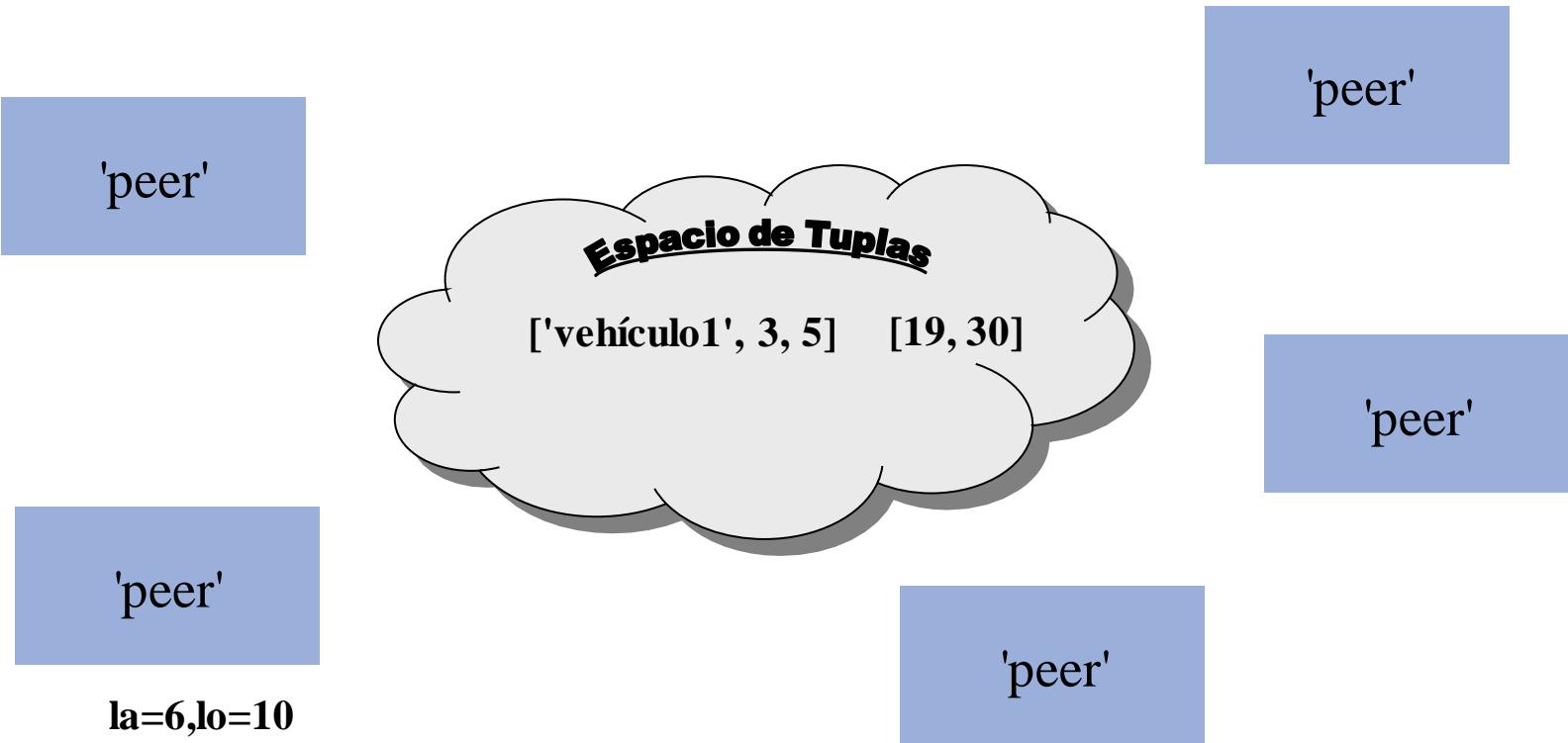
El modelo de coordinación Linda



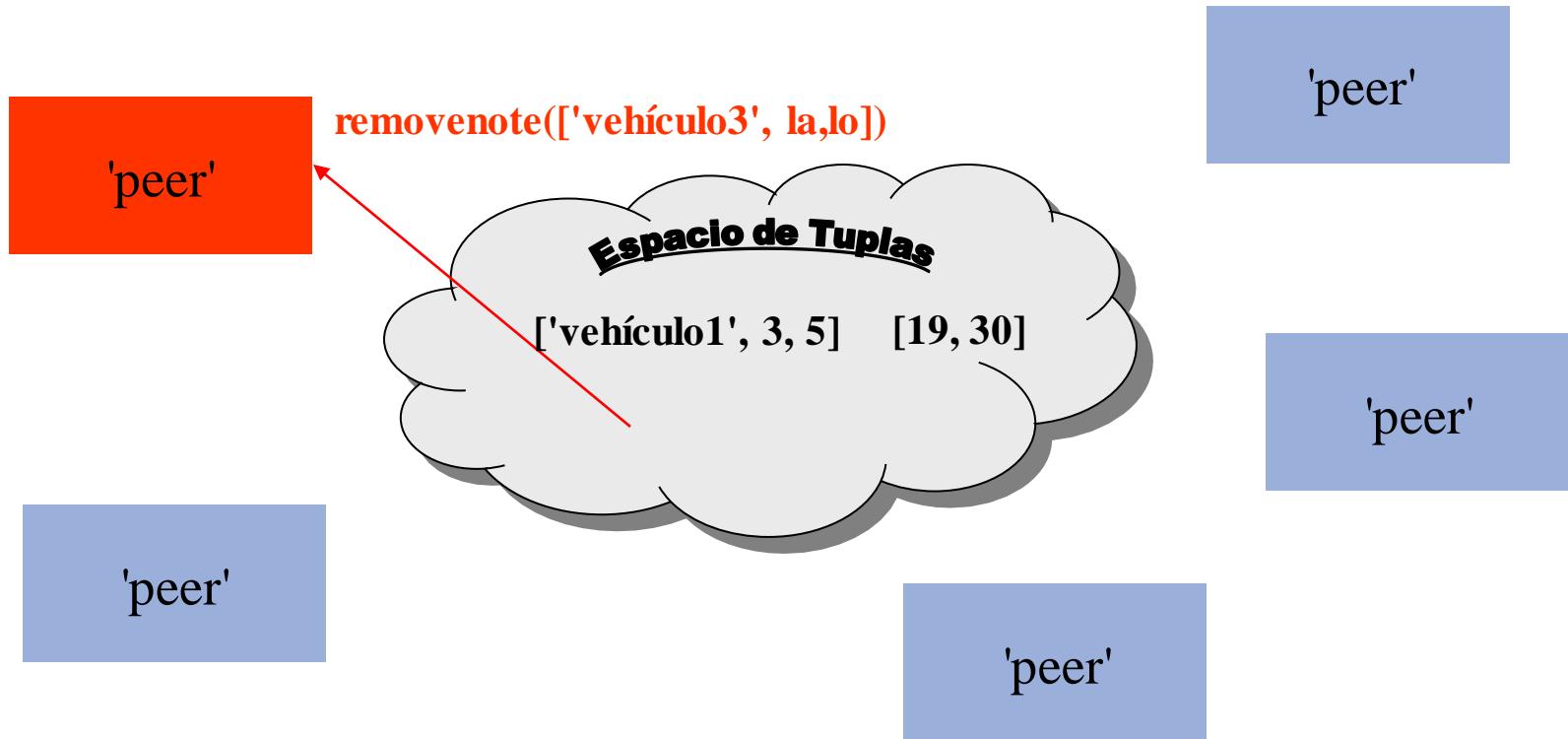
El modelo de coordinación Linda



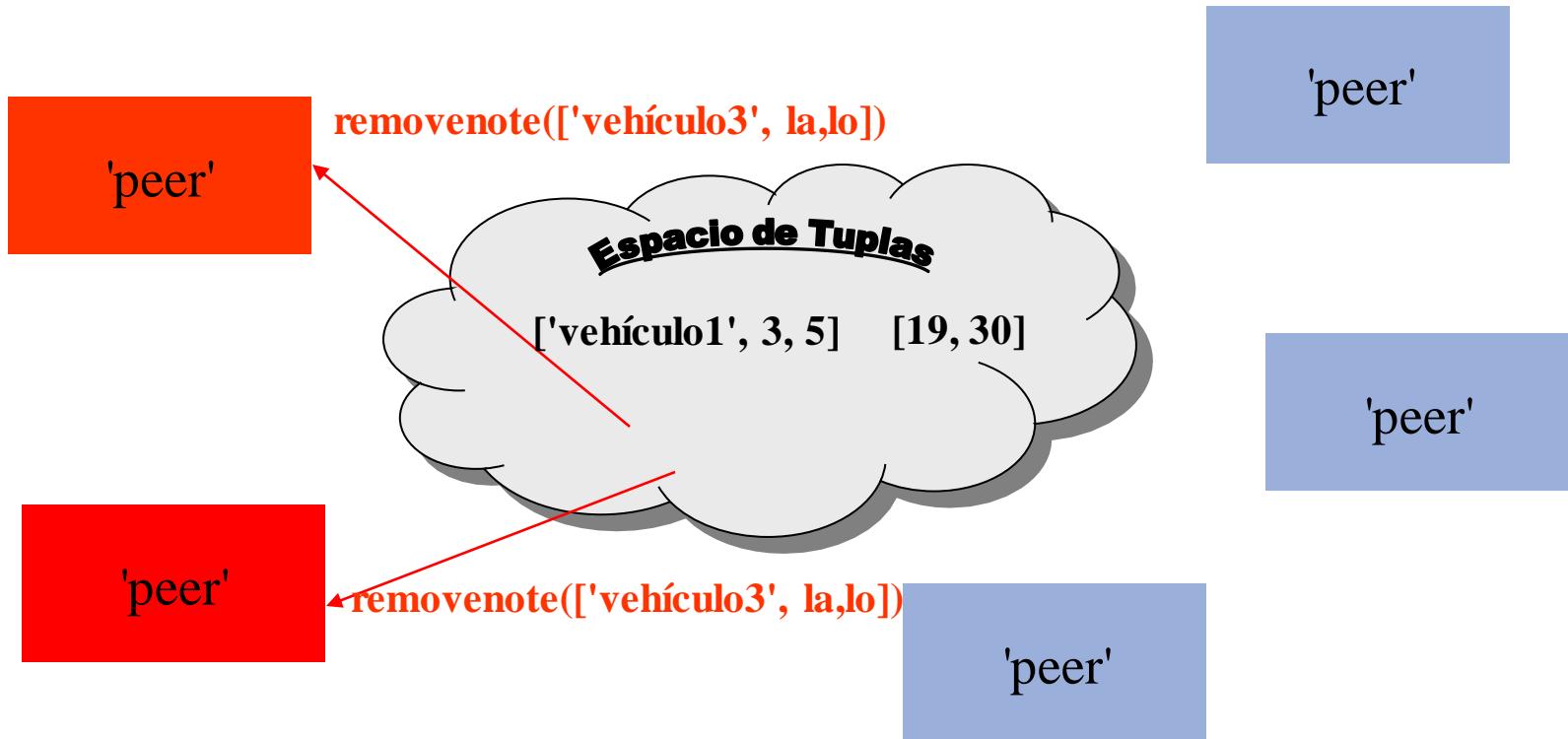
El modelo de coordinación Linda



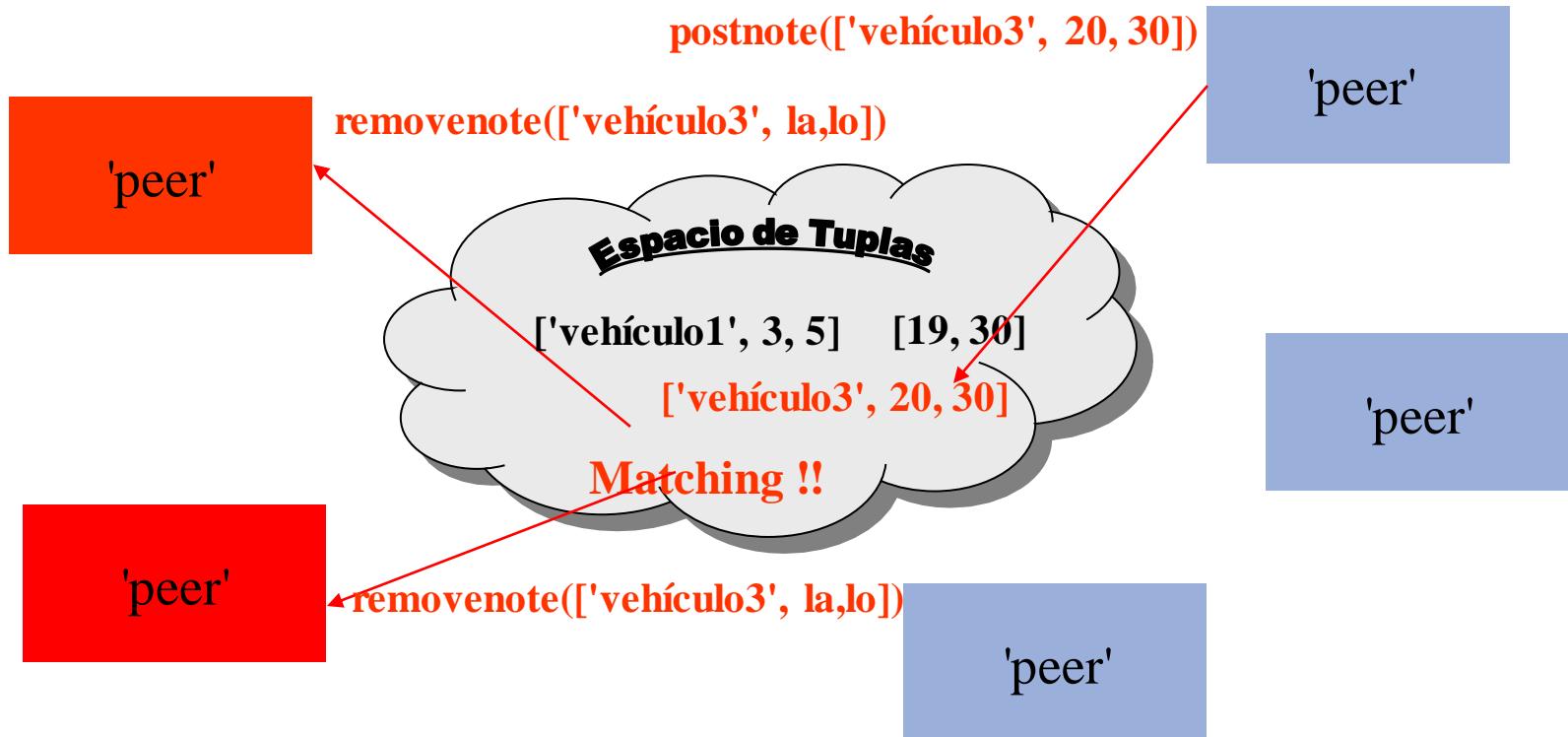
El modelo de coordinación Linda



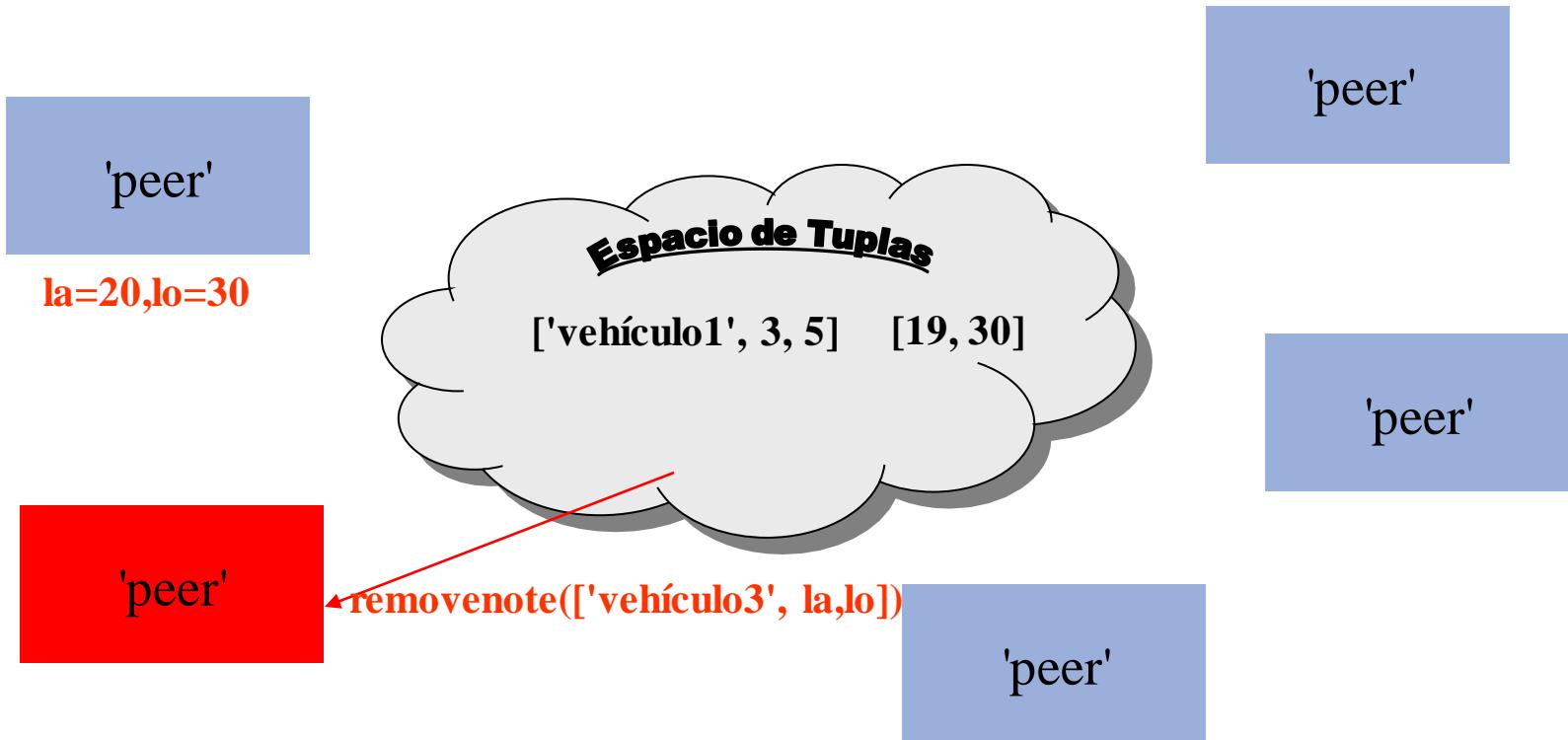
El modelo de coordinación Linda



El modelo de coordinación Linda



El modelo de coordinación Linda



El modelo de coordinación Linda

- Habitualmente se definen tres operaciones sobre un espacio de tuplas:
 - **postnote([exp1,exp2,...])**
 - t es una tupla; el proceso deposita la tupla y sigue (**no bloqueante**)
 - **removenote([v_v1,v_v2,...])**
 - v_v* es un valor, una variable o un parámetro formal
 - el proceso se **bloquea** hasta que el espacio de tuplas le asigna una que corresponda al patrón
 - la tupla es quitada del espacio de tuplas
 - **readnote([v_v1,v_v2,...])**
 - versión del “removenote” en que la tupla que le es asignada no se elimina del espacio de tuplas
 - puede haber varios read simultáneos

out, write, PN

in, take, RN

Read, RD

¡atómicas!

Ejemplo: Un semáforo

- Implementación de las operaciones de semáforo

```
semaphore s := 0  
semaphore s2 := 3
```

```
Process P  
  wait(s2)  
  wait(s2)  
  wait(s)
```

```
Process Q  
  wait(s2)  
  signal(s)
```

Ejemplo: Un semáforo

- Implementación de las operaciones de semáforo

```
semaphore s := 0  
semaphore s2 := 3
```

```
Process P  
wait(s2)  
wait(s2)  
wait(s)
```

```
Process Q  
wait(s2)  
signal(s)
```

```
Process init  
for i:=1..3  
postnote(['s2'])
```

```
Process P  
removenote(['s2'])  
removenote(['s2'])  
removenote(['s'])
```

```
Process Q  
removenote(['s2'])  
postnote(['s'])
```

Ejemplo: Un semáforo

- Implementación de las operaciones de semáforo

```
semaphore s := 0  
semaphore s2 := 3
```

```
Process P  
wait(s2)  
wait(s2)  
wait(s)
```

```
Process Q  
wait(s2)  
signal(s)
```

```
Process init  
for i:=1..3  
  postnote(['s2'])  
  postnote(['P'])  
  postnote(['Q'])
```

```
Process P  
removenote(['P'])  
removenote(['s2'])  
removenote(['s2'])  
removenote(['s'])
```

```
Process Q  
removenote(['Q'])  
removenote(['s2'])  
postnote(['s'])
```

Ejemplo: Un semáforo

- Implementación de las operaciones de semáforo

```
semaphore s := 0  
semaphore s2 := 3
```

Process P
wait(s2)
wait(s2)
wait(s)

Process Q
wait(s2)
signal(s)

Process init
for i:=1..3
postnote(['s2'])
postnote(['init'])

Process P
readnote(['init'])
removenote(['s2'])
removenote(['s2'])
removenote(['s'])

Process Q
readnote(['init'])
removenote(['s2'])
postnote(['s'])

Primeros ejercicios

- Ejercicio 1: Resolver el problema de la “2-exclusión mutua”: no más de dos procesos en la sección crítica simultáneamente

```
Process P(i:1..N)::  
    while true  
  
        //SC  
  
    end
```

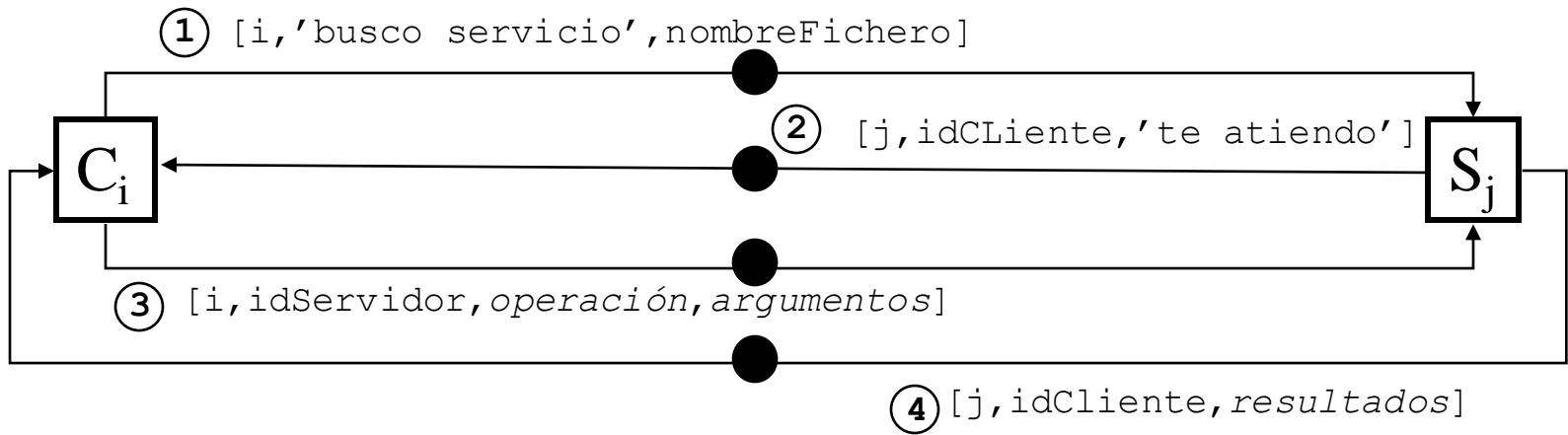
- Ejercicio 2: resolver el mismo problema, pero usando únicamente una tupla

Ejemplo: Servidor de ficheros

- Es un ejemplo de arquitectura cliente-servidor
- Sistema que puede mantener hasta n ficheros abiertos simultáneamente
- Compuesto por:
 - n servidores
 - m clientes
- Tal que:
 - cliente pide atención por parte de un servidor (cualquiera) para acceder a un fichero
 - un servidor libre puede atender las peticiones de un cliente
 - las operaciones sobre el fichero son pedidas por el cliente y atendidas por el servidor

Ejemplo: Servidor de ficheros

- Esquema de intercambio de mensajes a través de Linda



Ejemplo: Servidor de ficheros

```
Process cliente(i:1..m)
    integer idServidor
    string nombreFichero := ... --el que sea
    ...
    postnote([i,'busco servicio',nombreFichero])
    removenote([idServidor,i,'te atiendo'])
    --esquema [from,to,...]
while me_da_la_gana
    postnote([i,idServidor,operación,argumentos])
    removenote([idServidor=,i=,resultados])
    ...
```

Ejemplo: Servidor de ficheros

```
Process servidorDeFicheros(j:1..n)
    string nomFich
    integer idCliente

    loop
        removenote([idCliente,'busco servicio',nomFich])
        postnote([j,idCliente,'te atiendo'])
        while me_da_la_gana
            removenote([idCliente=,j=,operacion,argumentos])
            --procesar operación y obtener resultados
            postnote([j,idCliente,resultados])
```

Ejercicios

- **Ejercicio :** Dar una solución al problema de los 5 filósofos utilizando Linda como medio para la sincronización
 - Versión 1: cada proceso coge los tenedores de uno en uno
 - Versión 2: cada proceso coge ambos tenedores a la vez
 - Pista: representar el estado de los tenedores con una única tupla
 - Versión 2: hay un **proceso servidor** de tenedores, que concede los dos simultáneamente

7.2 Problem: the **wavefront computation** of a matrix

N. Carriero and D. Gelernter

"How to write parallel programs. A first course "

The MIT Press, 1990

Given a function $h(x, y)$, we need to compute a matrix H such that the value in the i th row, j th column of H —that is, the value of $H[i, j]$ (in standard programming language notation) or $H_{i,j}$ (in mathematical notation)—is the function h applied to i and j , in other words $h(i, j)$.

The value of $h(i, j)$ depends on three other values: the values of $h(i - 1, j)$, of $h(i, j - 1)$ and of $h(i - 1, j - 1)$. To start off with, we're given the value of $h(0, j)$ for all values of j , and of $h(i, 0)$ for all values of i . These two sets of values are simply the *two strings that we want to compare*. In other words, we can understand the computation as follows: draw a two-dimensional matrix. Write one of the two comparands across the top and write the other down the left side. Now fill in the matrix; to fill in each element, you need to check the element on top, the element to the left, and the element on top and to the left.

Ejercicios

- **Ejercicio:**
 - Dos tipos de procesos, A y B pueden entrar y salir de una habitación común. Un proceso A no puede salir de la habitación hasta que haya coincidido en ella, simultáneamente, con dos B, mientras que un B no puede salir hasta que haya coincido con al menos un A
 - Se pide desarrollar el controlador de acceso a la habitación, así como el código de los procesos de tipo A y B. La comunicación de los procesos con el controlador debe hacerse mediante Linda.

Ejercicios

- **Ejercicio:** Nuestro sistema informático dispone de dos impresoras, I1 e I2, parecidas pero no idénticas. Éstas son usadas por tres tipos de procesos cliente. Los del primer tipo requieren la impresora I1, los del segundo la I2, mientras que los del tercero cualquiera de las dos. Usando Linda como medio de coordinación, desarrollar el código de los tres tipos de impresoras y del controlador de las impresoras.

Aproximación conceptual a los monitores

- Es fácil implementar mediante el espacio de tuplas una gestión de la concurrencia análoga a la hecha con monitores

*variables permanentes
declaración proc.
llamadas a procs.
entrada a proc.
terminación proc.
wait
signal
cuerpo proc.*

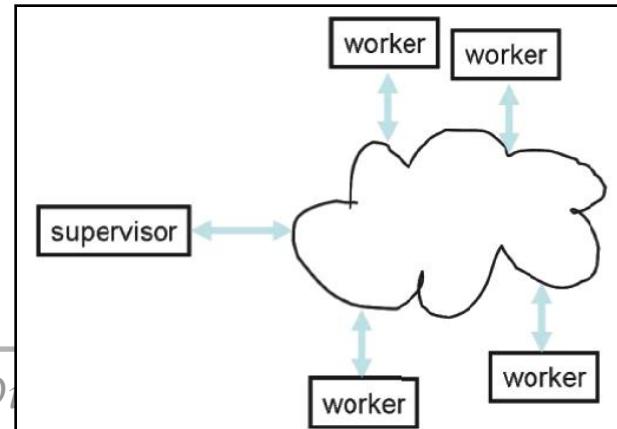
*variables locales servidor
disponibilidad de servicio
envío tupla solicitud de servicio
"removenote" de la solicitud
"postnote" de la respuesta
encolar peticiones como pendiente
procesa petición pendiente en cola
"Switch" sobre solicitudes*

monitor

proceso
servidor

La figura 1 representa de manera esquemática el patrón de diseño *supervisor-worker*. Este esquema propone una organización sencilla en la que un proceso supervisor suministra tareas a los trabajadores a través de un medio común para el intercambio de información. Los procesos trabajadores van tomando las solicitudes, obtienen los resultados y los envía al supervisor, quedando a la espera de nuevas tareas encargadas por el proceso supervisor, o hasta que éste le indique que puede terminar. El supervisor toma los resultados y actúa en función a cuál sea su objetivo.

Utilizando Linda como medio de comunicación, se pide resolver el siguiente problema. Se trata de ordenar 100 ficheros de datos, de manera que un proceso supervisor suministra los nombres de los ficheros, y cuatro procesos trabajadores van tomando las órdenes depositadas por el supervisor y ordenan el fichero cuyo nombre se les ha suministrado. Cada trabajador va iterando ese proceso hasta que recibe del supervisor la orden de terminar. Dependiendo del entorno de ejecución, es posible que distintos procesos trabajador tengan distintas velocidades.



```

process supervisor
    constant natural N := 100
    string array [1..N] ficheros := (1..N, ...)
        --Nombres de ficheros a ordenar. Ya inicializada
        --completar de acuerdo al enunciado del ejercicio
end process

process worker(i:1..4):::
    operation ordena_fichero(string nom_fich)
        --Pre: el fichero en <nom_fich> existe
        --Post: el fichero en <nom_fich> está ordenado
        --Com: Asumimos que está implementada
        ...
    end operation
    --completar de acuerdo al enunciado de
end process

```

