Lección 14Bis: El modelo de coordinación Linda

- Introducción
- Características de Linda
- Presentación informal
- Acceso al espacio de tuplas
- Tuplas y funciones de "matching"
- Uso de Linda
- Linda como generalización de PAM
- Ejemplo: de nuevo, el servidor de ficheros
- Ejercicios

Introducción

- En el modelo "clásico" de comunicación asíncrona no hay recepción condicionada
  - si leo de un canal, me trago lo que llegue
- Esto representa un pequeño inconveniente cuando
  - sólo me interesan determinados mensajes
    - porque ahora sólo quiero ésos
    - porque comparto el canal y sólo he de coger los que sean mios
    - etc.
- Linda incluye estas posibilidades

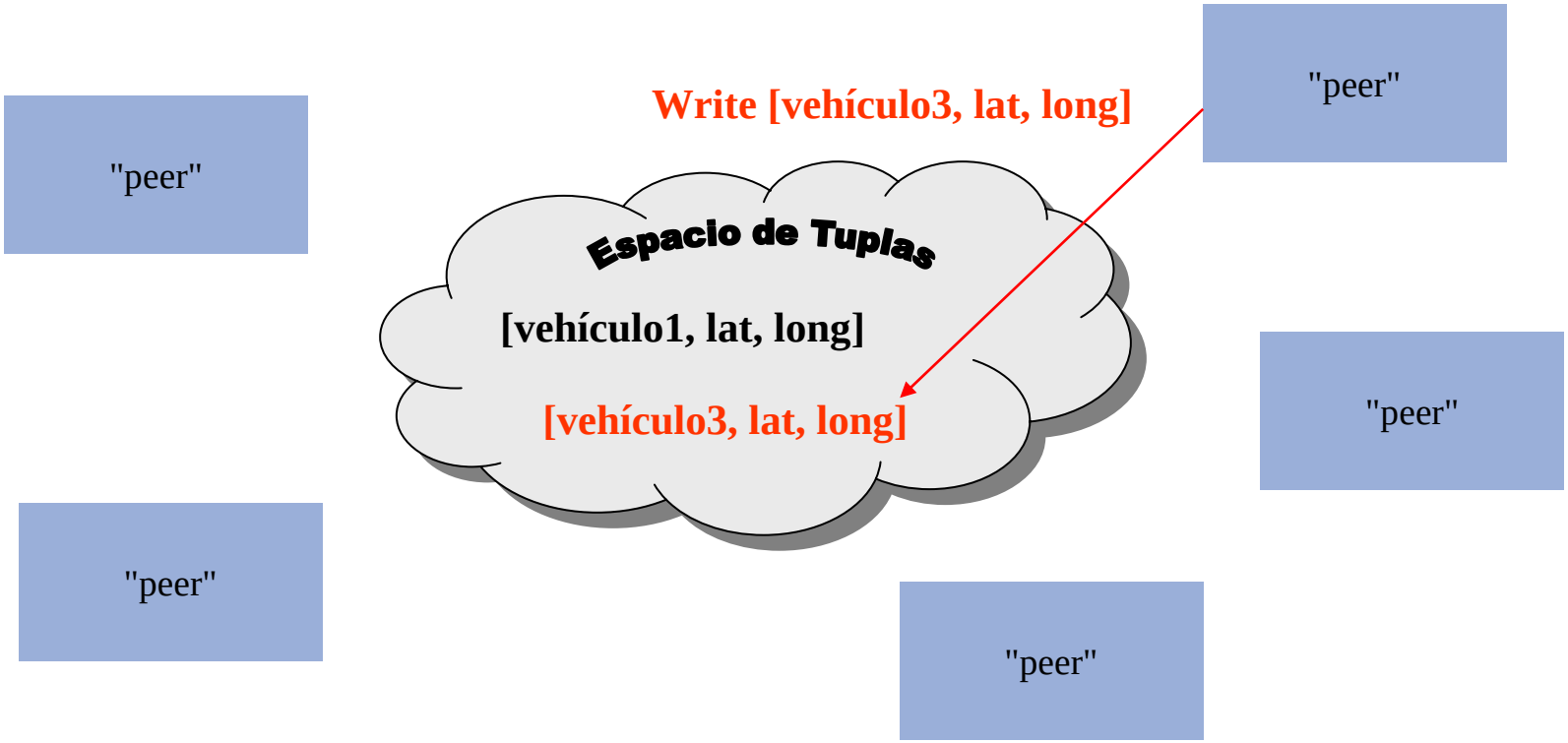
Características de Linda

- 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 Lisp (árboles)
  - versión original: una especie de listas Lisp sin anidamiento

JavaSpaces
GigaSpaces
TSpaces
JXTASpaces
XMLSpaces
C-Linda

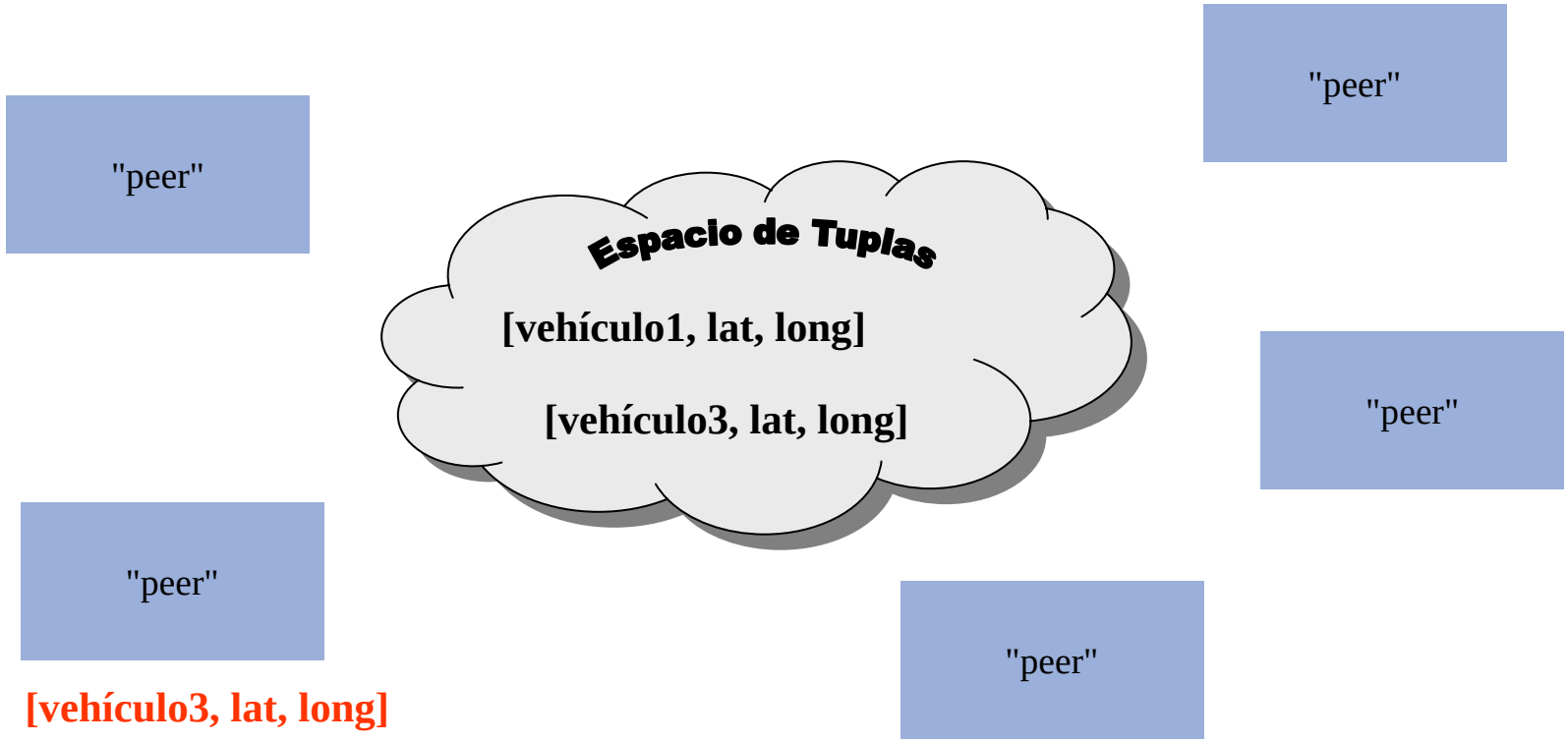
Presentación informal

Presentación informal

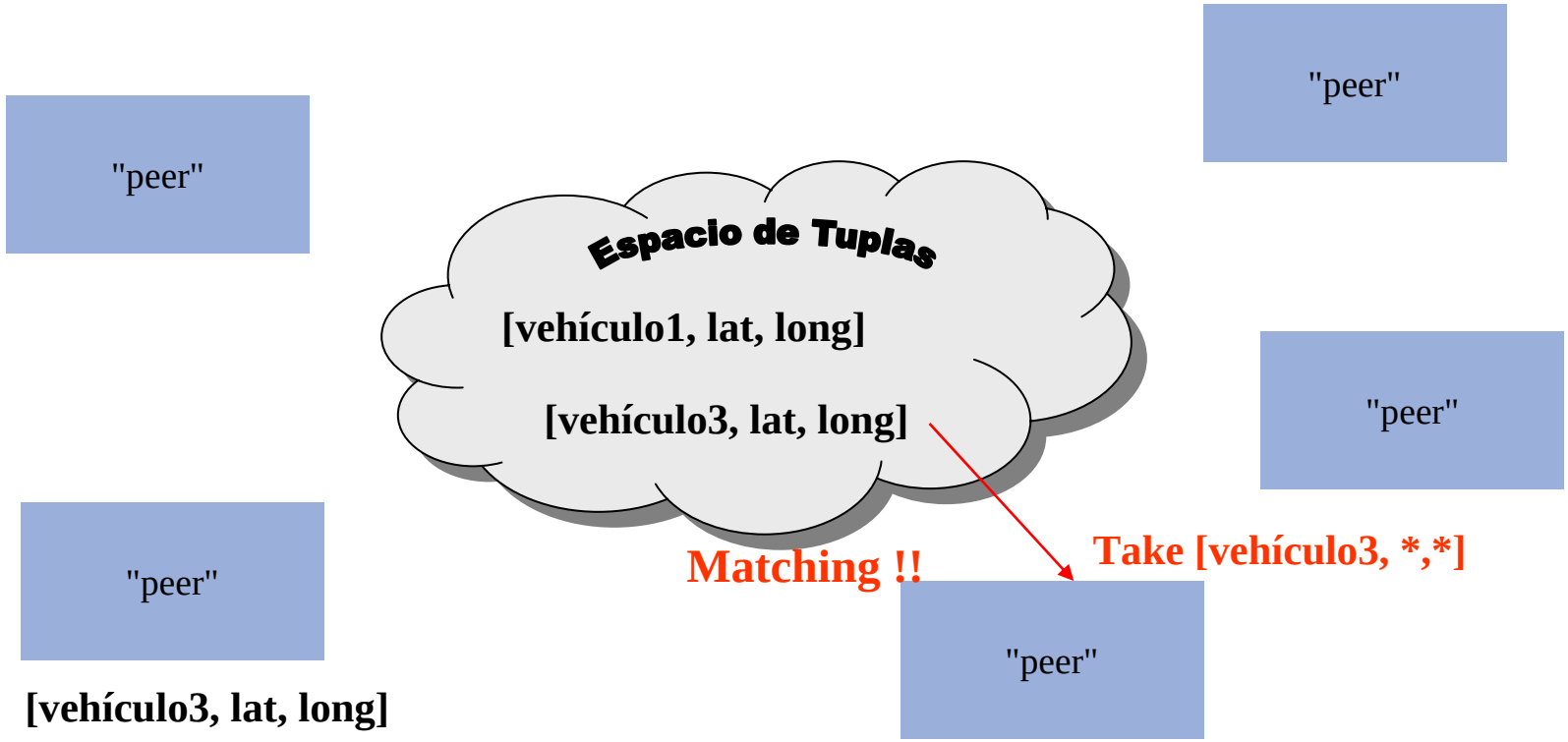


Presentación informal

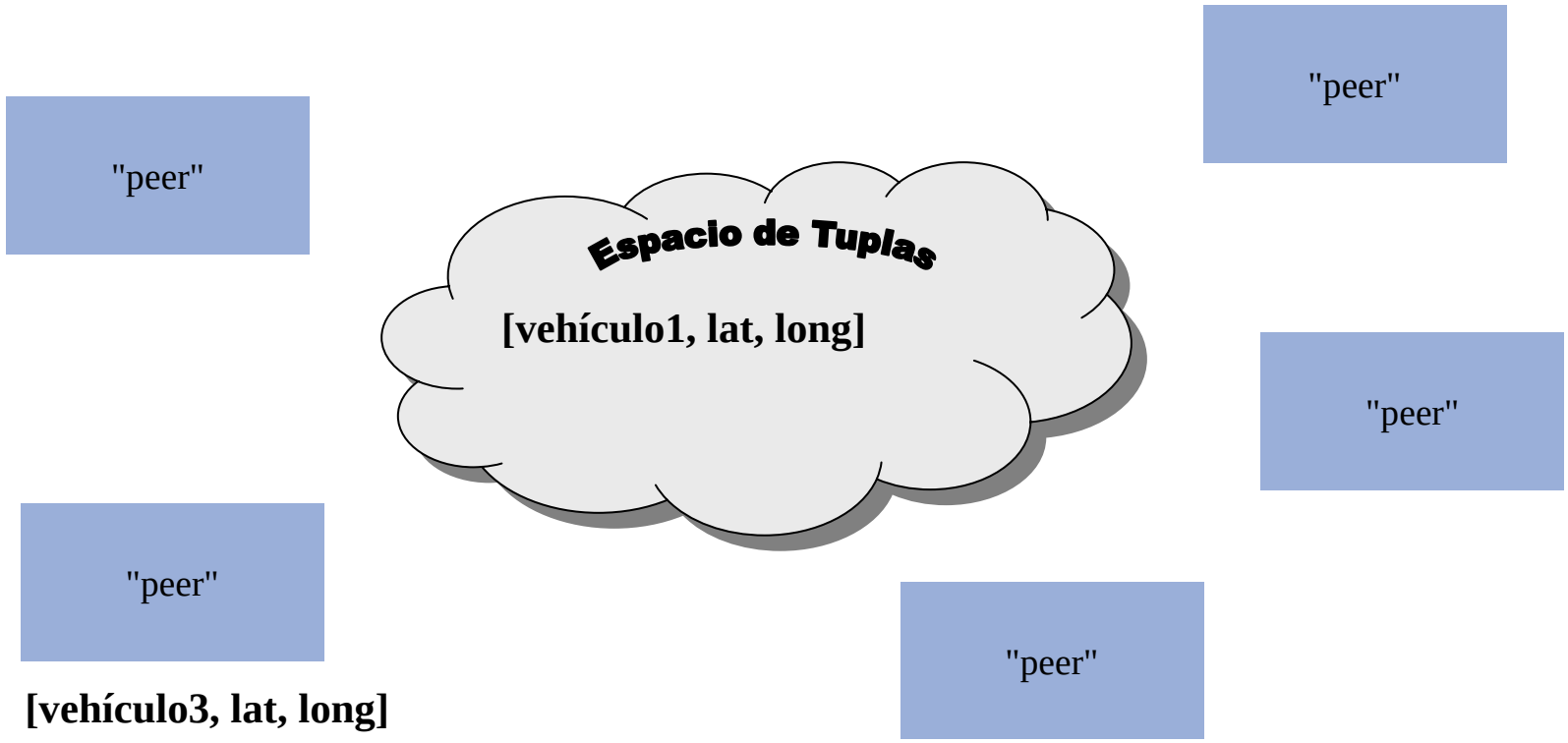
Presentación informal



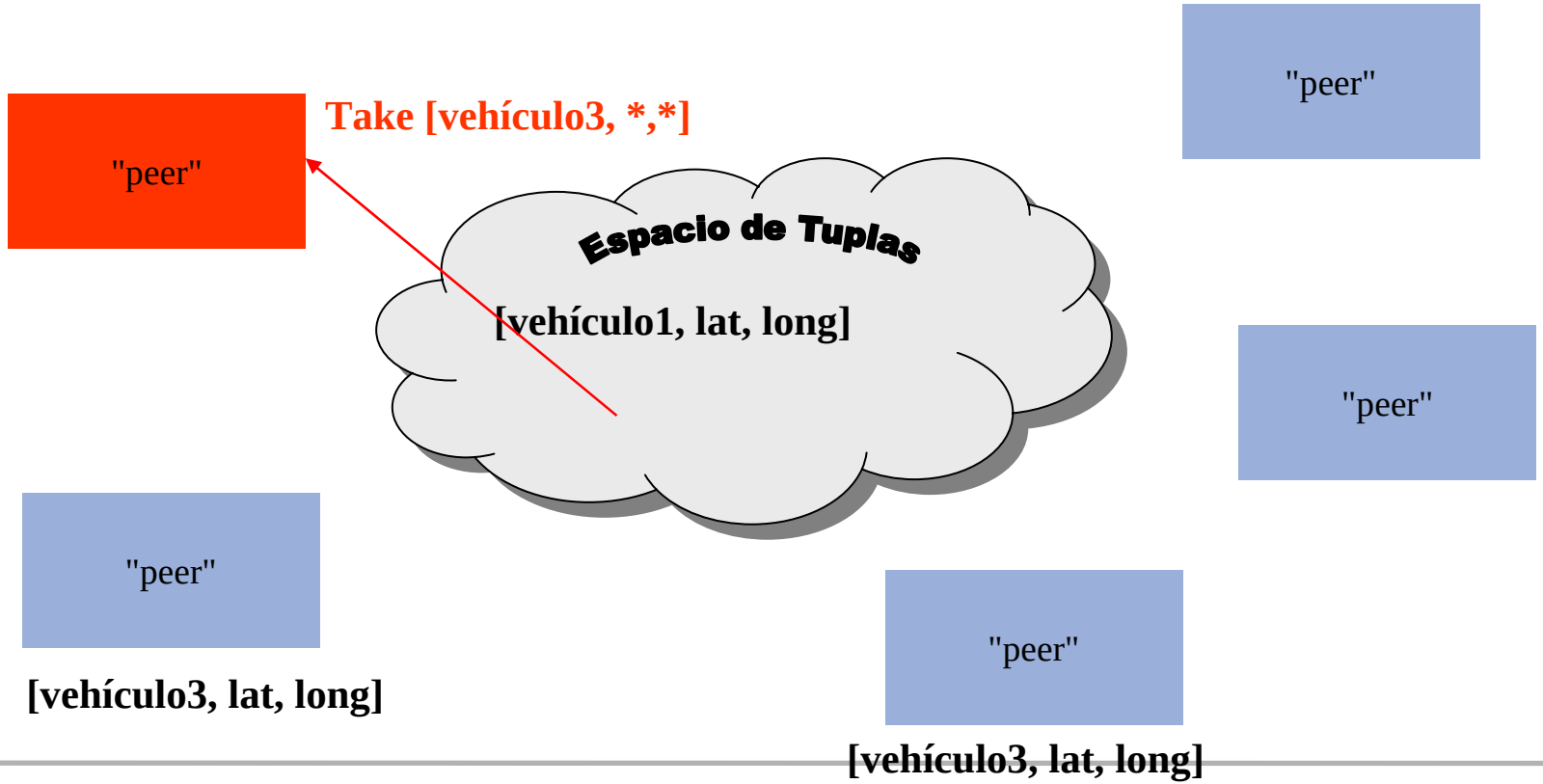
Presentación informal



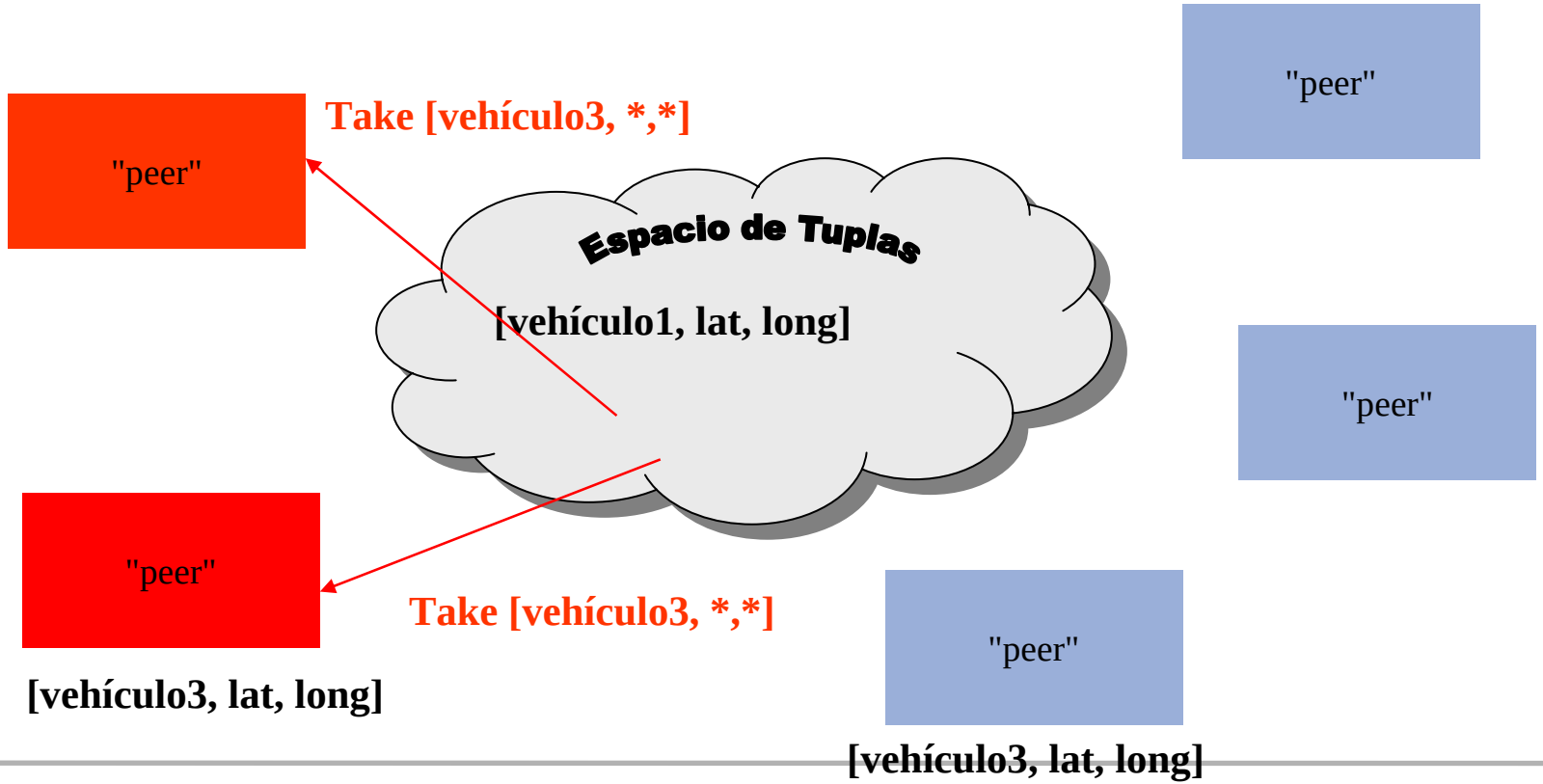
Presentación informal



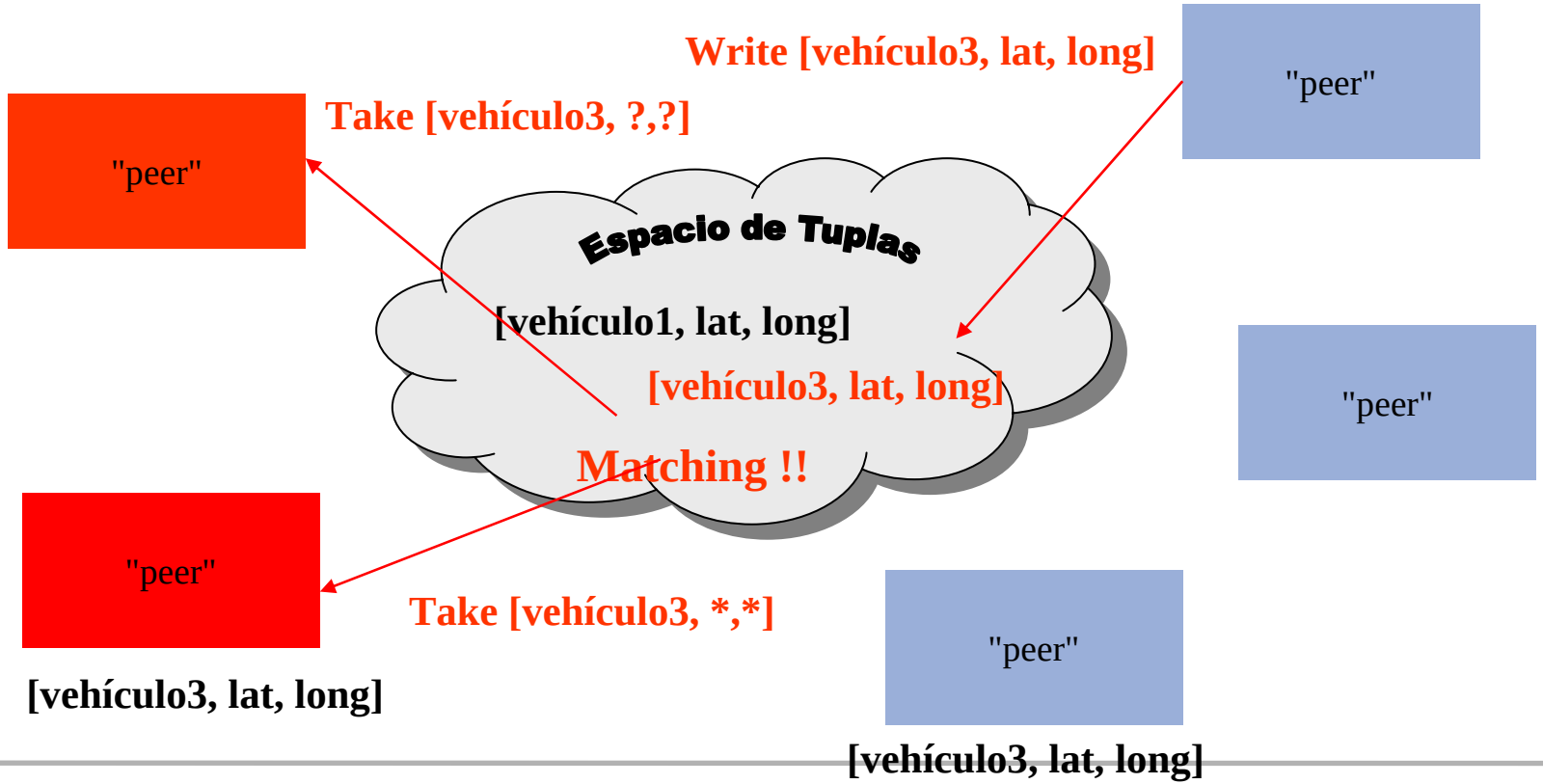
Presentación informal



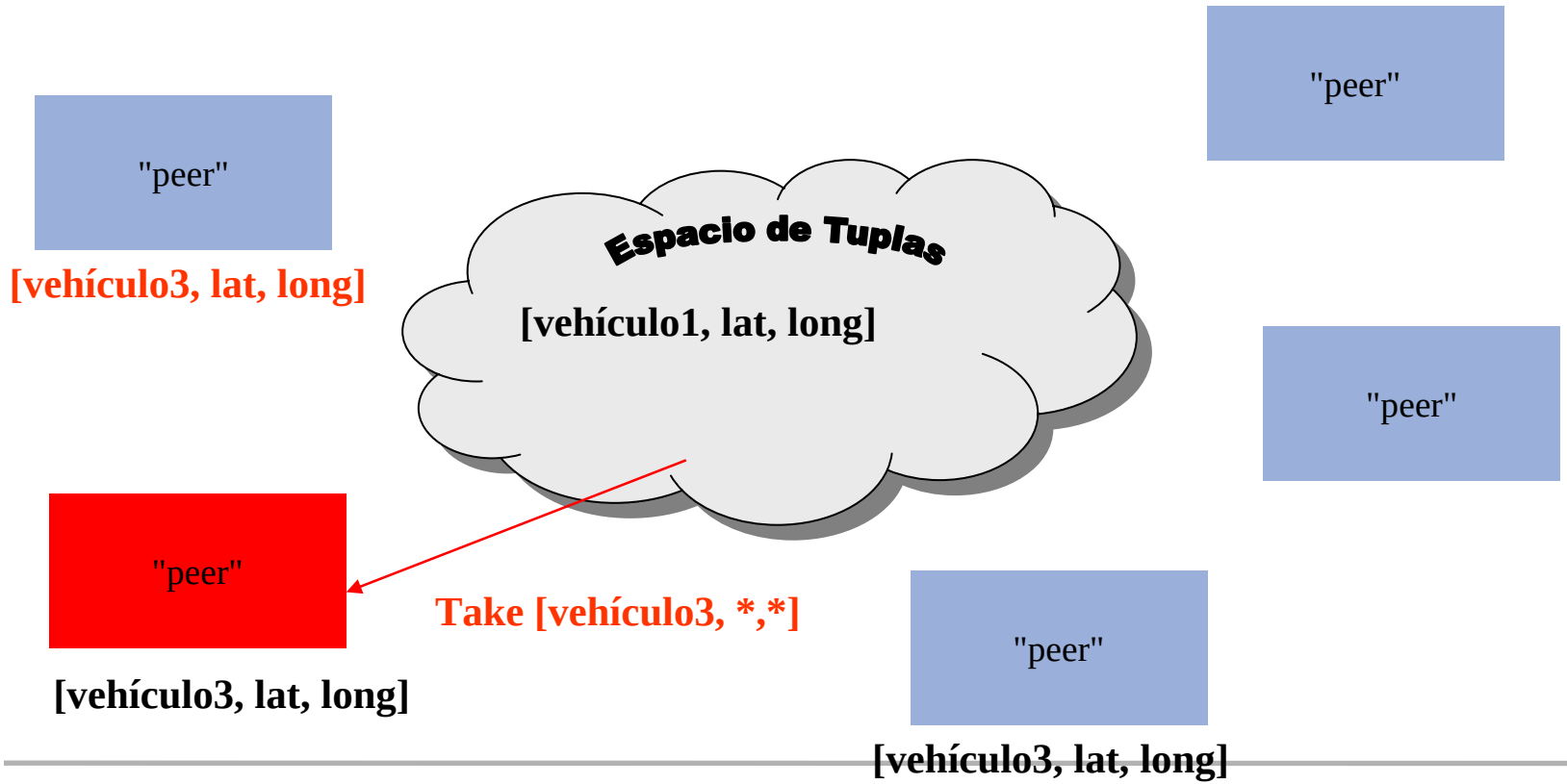
Presentación informal



Presentación informal



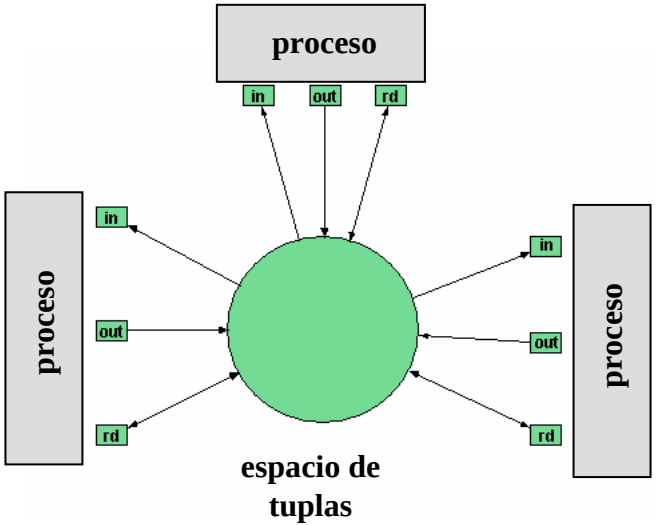
Presentación informal



Acceso al espacio de tuplas

- Habitualmente se definen tres operaciones sobre un espacio de tuplas:
  - **write(t) (out)**
    - t es una tupla; el proceso deposita la tupla y sigue (no bloqueante)
  - **take(p) (in)**
    - p es un patrón; 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
  - **read(p) (rd)**
    - versión del "take" en que la tupla que le es asignada no se elimina del espacio de tuplas
    - puede haber varios read simultáneos

Acceso al espacio de tuplas



Tuplas y funciones de "matching"

- En el modelo básico las tuplas son bastante simples
- Veamos una definición más general

Let A be a set of typed elements, called *atoms*, for which a mapping

$$EQ_{\mathcal{A}} : \quad \mathcal{A} \times \mathcal{A} \longrightarrow \{0, 1\}$$

is assumed to be defined. The set of tuples $\mathcal{T}_A$ is defined as follows:

1. for any $a \in A$, $[a]$ belongs to $\mathcal{T}_{\mathcal{A}}$
2. for any $[a] \in \mathcal{T}_A$, $[[a]]$ belongs to $\mathcal{T}_{\mathcal{A}}$
3. if $[e_1]$ and $[e_2]$ belong to $\mathcal{T}_{\mathcal{A}}$, then $[e_1\ e_2]$ also belongs to $\mathcal{T}_{\mathcal{A}}$

Tuplas y funciones de "matching"

- Let $\mathcal{T}_{\mathcal{A}}$ be a set of tuples. The *extended* tuple set, $\mathcal{T}_{\mathcal{A}^*}$ is defined as the tuple set $\mathcal{T}_{A\cup\{*\}}$
- An element of the extended tuple set is called *pattern* (also *template* or *query template*)
- A *matching function* M is a mapping

$$M: \quad \mathcal{T}_{\mathcal{A}} \times \mathcal{T}_{\mathcal{A}^*} \longrightarrow \{0,1\}$$

- Given a tuple `t` and a pattern `p` defined over the same set, they are said to *match* in M if, and only if, $M(t,p)=1$.
- The symbol $*$ plays the role of a wildcard for the matching function.

Tuplas y funciones de "matching"

- Se pueden definir diferentes formas de "matching"
- Veamos algunas interesantes:

The ***strong matching*** function $SM_{\mathcal{A}}$, is defined as follows:

$$SM_{\mathcal{A}}: \quad \mathcal{T}_{\mathcal{A}} \times \mathcal{T}_{\mathcal{A}^*} \longrightarrow \{0,1\}$$

$$[e_1 \ldots e_n], [v_1 \ldots v_m] \hookrightarrow (n = m) \wedge$$

$$\left(\bigwedge_{\alpha=1}^{n} \begin{cases} If \;\; e_\alpha \in \mathcal{A} \wedge v_\alpha \in \mathcal{A} & then \;\; EQ_{\mathcal{A}}(e_\alpha, v_\alpha) \\ If \;\; e_\alpha \in \mathcal{A} \wedge v_\alpha = \; * & then \;\; 1 \\ If \;\; e_\alpha \in \mathcal{T}_{\mathcal{A}} \wedge v_\alpha \in \mathcal{T}_{\mathcal{A}^*} & then \;\; SM_{\mathcal{A}}(e_\alpha, v_\alpha) \\ Else \;\; 0 \end{cases} \right)$$

Tuplas y funciones de "matching"

The ***weak matching***, $WM_{\mathcal{A}}$

$$WM_{\mathcal{A}}: \quad \mathcal{T}_{\mathcal{A}} \times \mathcal{T}_{\mathcal{A}^*} \longrightarrow \{0,1\}$$

$$[e_1 \ldots e_n], [v_1 \ldots v_m] \hookrightarrow (n = m) \wedge$$

$$\left(\bigwedge_{\alpha=1}^{n} \begin{cases} If \;\; e_\alpha \in \mathcal{A} \wedge v_\alpha \in \mathcal{A} & then \;\; EQ_{\mathcal{A}}(e_\alpha, v_\alpha) \\ If \;\; e_\alpha \in \mathcal{T}_{\mathcal{A}} \cup \mathcal{A} \wedge v_\alpha = \; * & then \;\; 1 \\ If \;\; e_\alpha \in \mathcal{T}_{\mathcal{A}} \wedge v_\alpha \in \mathcal{T}_{\mathcal{A}^*} & then \;\; WM_{\mathcal{A}}(e_\alpha, v_\alpha) \\ Else \;\; 0 \end{cases} \right)$$

Tuplas y funciones de "matching"

Let us assume that a set of attribute *names* $\mathcal{N} \subseteq \mathcal{A}$ is defined.
The ***attribute matching*** mapping, $AM_{\mathcal{A},\mathcal{N}}$, is defined as follows:

$$AM_{\mathcal{A},\mathcal{N}}: \quad \mathcal{T}_{\mathcal{A}} \times \mathcal{N} \longrightarrow \{0,1\}$$

$$[e_1 \ldots e_n], a \hookrightarrow \bigvee_{\alpha=1}^{n} \begin{cases} If \;\; e_\alpha \in \mathcal{A} \;\; then & 0 \\ If \;\; e_\alpha \in \mathcal{T}_{\mathcal{A}} \wedge WM_{\mathcal{A}}(e_\alpha, [a\,*]) \;\; then & 1 \\ Else \;\; AM_{\mathcal{A},\mathcal{N}}(e_\alpha, a) & \end{cases}$$

Tuplas y funciones de "matching"

Let us assume that given $\mathcal{A}$, $\bar{\mathcal{T}}_{\mathcal{N}}$ denotes the set of *plain* tuples
The ***general attribute matching*** is defined as follows

$$GAM_{\mathcal{A},\mathcal{N}}: \quad \mathcal{T}_{\mathcal{A}} \times \bar{\mathcal{T}}_{\mathcal{N}} \longrightarrow \{0,1\}$$

such that

$$t, [a_1 \ldots a_m] \hookrightarrow \bigwedge_{\alpha=1}^{m} AM_{\mathcal{A},\mathcal{N}}(t, a_\alpha)$$

Uso de Linda

- Vamos a suponer definido el siguiente entorno de datos:

```
Tipos
 tupla= .....  --tuplas de lo que sea
 modosMatching =(strong,weak,attribute,generalAttribute)
```

- Los tipos tupla y patron son TAD con los operadores

```
•:=,=,/=
•[] es la tupla vacía
•t1+t2 es la concatenación
•si t1 es prefijo de t2, t2-t1 es la tupla resta
•long(t) es el número de componentes
•t(i), con i en el rango 1..long(t), da la componente
•en un patrón, NULL representa un comodín
•....
```

Uso de Linda

- Declaración de un canal Linda

```
nombreCanal: canalLinda
```

- Asumiendo los tipos de datos ***tupla*** y ***patrón,*** las operaciones se invocan:

```
Vars t: tupla
     p: patron
     m: modosMatching := ....
     c: canalLinda
t = c.take(m,p)
t = c.read(m,p)
c.write(t)
```

Linda como generalización de PAM

```
Vars c1: Canal(T1)
     c2: Canal(T2)
     v: T2

...
send c1(t1)
receive c2(v)
...
```

```
Vars c: canalLinda
     t: tupla
     v: T2

...
c.write(["c1",t1])
t:=c.take(strong,["c2",null])
v:= t(2)
...
```

Ejemplo: de nuevo, el servidor de ficheros

- Otro 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

```
Tipos tipoOper = Enum(READ,WRITE,CLOSE)
      c: canalLinda

Cliente(i:1..m)::
  Vars idServidor: Ent;  t: tupla
  ...
  c.write([i,"buscoServicio",nombreFichero])
  t := c.take(strong,[NULL,i,"teAtiendo"]) --espera respuesta
  idServidor := t(1)
  Mq meDeLaGana
     c.write([i,idServidor,operación,argumentos])
     t := c.take(strong,[idServidor,i,NULL])
     --t(3) tiene los resultados
     ...
  FMq
```

Ejemplo: servidor de ficheros

```
ServidorDeFicheros(j:1..n)::
  Vars nombreF: Cadena; idCliente: Ent
       seguir: booleano := Falso
         --"buffer" local, direcciones de disco, etc.
  Mq Verdad
    t := c.take(strong,[NULL,"buscoServicio",NULL])
    <idCliente,nombreF> := <t(1),t(3)>
    c.write([idCliente,j,"teAtiendo"])
    seguir:=Verdad
    Mq seguir
      t := c.take(strong,[idCliente,j,NULL,NULL])
      Sel t(3)=READ:  ...
          t(3)=WRITE: ...
          t(3)=CLOSE: ...
      FSel
      c.write([j,idCliente,resultados])
    FMq
  FMq
```

Ejercicios

- **Ejercicio 1**: Dos tipos de procesos, T1 y T2 pueden entrar y salir de una habitación común. Un proceso T1 no puede salir de la habitación hasta que haya coincido en ella, simultáneamente, con dos T2, mientras que un T2 no puede salir hasta que haya coincido con al menos un T1. Se pide desarrollar el controlador de acceso a la habitación, así como el código de los procesos de tipo T1 y T2. La comunicación de los procesos con el controlador debe hacerse mediante Linda. Asumiendo que simpre hay procesos de ambos tipos queriendo entrar, comentar aspectos como posibilidades de bloqueo, de equidad, etc. en la solución propuesta.

Ejercicios

- **Ejercicio 2**: 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. La solución debe ser equitativa, asumiendo que un proceso que toma una impresora la libera.

Ejercicios

- **Ejercicio 3**: El problema de la montaña rusa. Considerar un sistema con P posibles pasajeros y una montaña rusa, que puede llevar nP pasajeros (nP<P). Los pasajeros, viciosos ellos, están siempre esperando para montar en la montaña rusa y, cuando acaban, vuelven a querer montar. Por cicatería de los dueños, la montaña sólo se pone en marcha cuando tiene todas las plazas ocupadas. Desarrollar el código de un procesoo pasajero y del controlador de la montaña. Usar Linda como medio de coordinación.