Migración

Javier Bustos Jiménez

Procesos Servidores

- Se implementan para proporcionar servicios a un conjunto de clientes.
- Hay varios aspectos de diseño que deben considerarse al implementar servidores:
  - Concurrencia,
  - localización de los servicios,
  - control sobre los servidores,
  - estado de los servidores.

Tipos de servidores

- Respecto a su **concurrencia**:
  - Servidores **iterativos**: un solo hilo espera peticiones y las procesa.
  - Servidores **concurrentes**: un hilo (o proceso) espera peticiones y ante cada petición, le pasa la petición a otro hilo (o proceso).

- Respecto a su **localización** (punto de entrada):
  - **Punto de entrada fijo y conocido**: los clientes conocen de antemano la ubicación (p.ej: ftp, http, etc)
  - **Punto de entrada desconocido**: resoluble mediante un servicio de nombres (debe conocerse la ubicación del servicio de nombres)

Tipos de Servidores

- Respecto al **control** que se puede ejercer sobre ellos:
  - **Sin control**: los clientes que quieren desconectar, simplemente abortan la conexión, por ejemplo terminando la propia aplicación cliente
  - **Con control**: se establece un canal de comunicación adicional entre cliente y servidor, de forma que el cliente puede actuar sobre el servidor: para detenerlo, pararlo, reanudarlo, etc.
- Respecto a su **estado interno**:
  - **Sin estado**: los servidores no guardan información respecto al estado de los clientes ni respecto a su conversación (p.ej: un servidor http)
  - **Con estado**: el servidor guarda información de cada cliente. Ante caídas y recuperaciones del servidor, se debe recuperar la información de cada cliente.

… pero

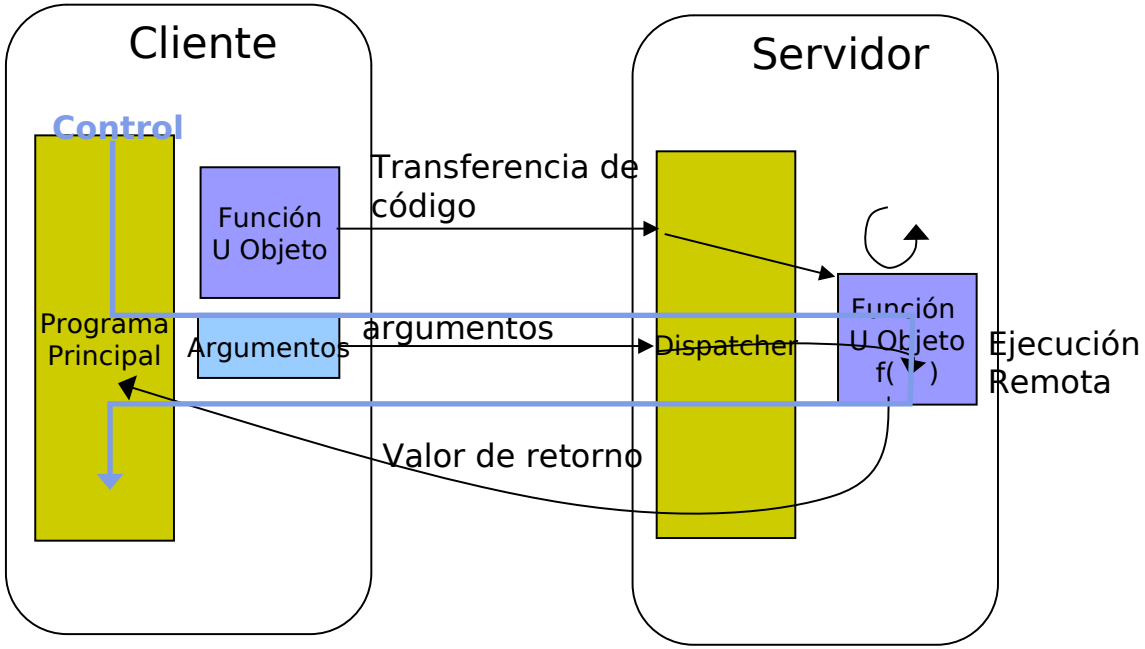
- El esquema anterior es útil cuando se realiza paso de datos.

- Sin embargo, hay veces en que es mejor realizar un paso de programas, que deben desarrollar distintas tareas según donde se ejecuten. Esto se conoce como **migración de código**.

Esquemas de ejecución

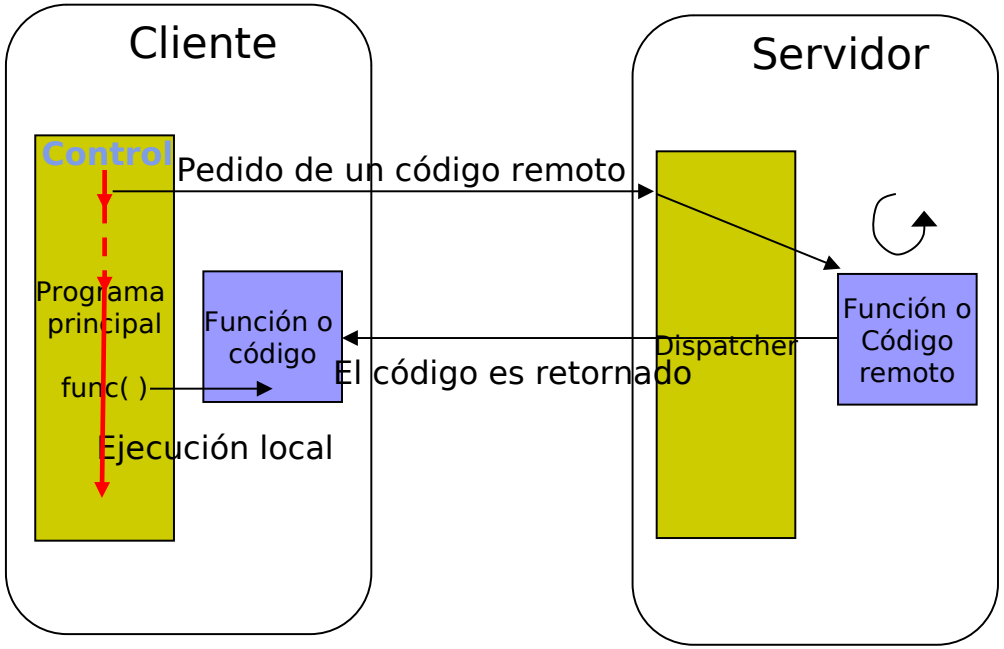
- Ejecución remota (RPC)
- Code-on-demand (Javascript)
- Migración de procesos

Ejecución remota



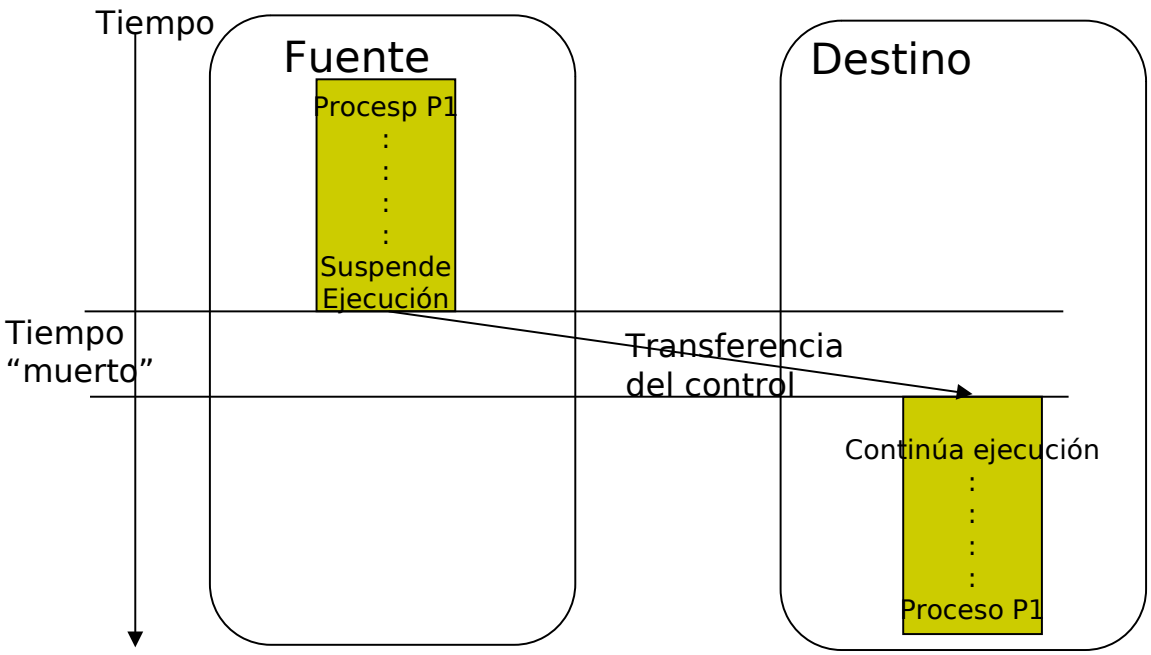
- El código es enviado junto con los argumentos
- El servidor actúa como un típico servidor cíclico

Code-on-Demand



- El servidor actúa como un servidor de códigos remotos
- El código remoto es llevado al cliente
- El cliente lo ejecuta localmente
- El control de ejecución permanece en el cliente mientras está suspendido por la descarga del código.

Migración de procesos



Motivos para migrar código

- **Aumentar la eficiencia**:
  - **Repartir carga computacional**: si una máquina tiene mucha carga, se observará una mejora global del rendimiento del sistema distribuido, si se migra su código a otra máquina con menos carga.
  - **Disminuir carga de la red de comunicaciones**: Si se observa mucho tráfico entre dos máquinas debida a la interacción entre dos “procesos”, puede ser conveniente migrar uno de ellos a la máquina del otro.
  - **Aumentar la velocidad de procesamiento**: distribuyendo el procesamiento entre máquinas para aumentar el grado de paralelismo.

Motivos para migrar código

- **Permitir la carga dinámica de código**:
  - **Uso de partes del código no conocidas a priori**: Por ejemplo podemos construir clientes que hagan uso de protocolos genéricos de comunicación con los servidores, pero que carguen del servidor el protocolo específico que se utiliza en cada momento.
  - **Mejorar la distribución/instalación del código en sistemas grandes**: muchas veces no es posible instalar una aplicación en todo un sistema, de forma que la mejor opción consiste en que los clientes descarguen el código que deben ejecutar conforme lo usen.

Un programa es:

- **Segmento de código**: contiene únicamente instrucciones

- **Segmento de recursos**: contiene referencias a recursos externos, tales como archivos, impresoras, mecanismos de sincronización, etc.

- **Segmento de ejecución**: contiene el estado de la ejecución del proceso: variables, pila y contador de programa.

Modelos de migración

- Respecto a los segmentos que migran:
  - **Movilidad débil**: solo migra el segmento de código (p.ej: los Applets de Java)
  - **Movilidad fuerte**: migran el segmento de código y el segmento de ejecución.

… considerando

- Migrar el segmento de recursos muchas veces puede ser problemático. Por ejemplo migrar la referencia a un socket abierto TCP de una máquina a otra.

- La clave para entender las posibilidades de migración del segmento de recursos radica en el enlace que exista entre el **recurso y el proceso** y el **recurso y la máquina**.

Tipo de enlace entre recursos y procesos

- **Enlace por identificador**: un proceso accede a los recursos por un identificador. Por ejemplo: URL, dirección TCP, etc. En este caso el recurso es único y debe mantenerse la asociación.

- **Enlace por valor**: lo que necesita el proceso es el valor de un recurso, no su unicidad. Por ejemplo, cierta biblioteca de código.

- **Enlace por tipo**: El proceso necesita únicamente cierto tipo de recurso. Por ejemplo: memoria, disco, etc.

Tipo de enlace entre recursos y máquinas

- **Recursos no enlazados**: son fáciles de migrar pues no existe esta asociación. Por ejemplo archivos de datos únicamente utilizados por los procesos que migran (el proceso se lleva el archivo).

- **Recursos no enlazados, pero interrelacionados con otros**. Por ejemplo: bases de datos.

- **Recursos fijos**: recusos que sólo existen en una máquina y no se pueden mover. Por ejemplo un socket TCP.

Entonces, ¿qué hacemos?

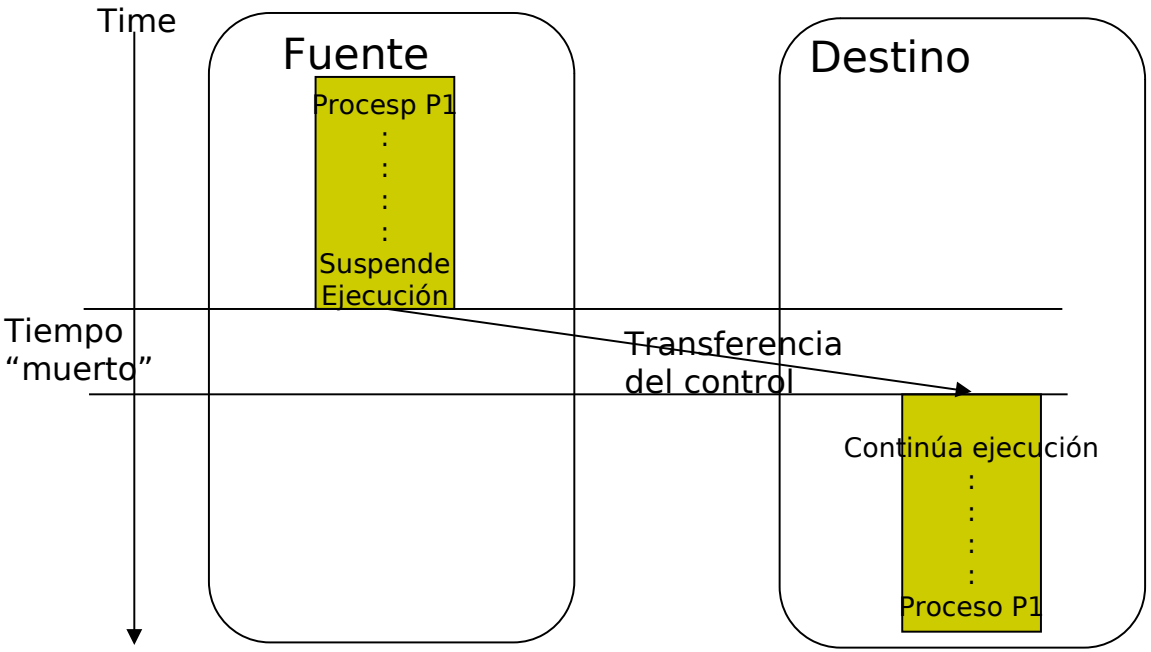
- Una vez clasificados los tipos de recursos atendiendo a su enlace con los procesos y máquinas donde residen, podemos detallar las acciones a realizar para migrar los segmentos de recursos:
  1. **Mover** el recurso a la máquina destino. (MV)
  2. Establecer una **referencia global** al recurso, dejando el recurso fijo, el proceso migrado remotamente lo continuará referenciando. (RG)
  3. **Copiar** el valor del recurso a la máquina destino. (CP)
  4. **Re-enlazar** con un recurso similar en la máquina destino. (RE)

En resumen

	No enlazados	Interrelacionados	Fijos
Por identificador	MV, RG	RG, MV	RG
Por Valor	CP, MV, RG	RG, CP	RG
Por Tipo	RE, MV, CP	RE, RG, CP	RE, RG

MV = Mover, RG = Referencia Global, CP = Copiar, RE = Re-enlazar

Migración de procesos



- Seleccionar el proceso a migrar
- Seleccionar el nodo de destino
- Suspender el proceso
- **Capturar el estado** del proceso
- Enviar el estado al destino
- Continuar la ejecución
- ***Forward*** de las referencias futuras al nuevo destino

Modelos de migración

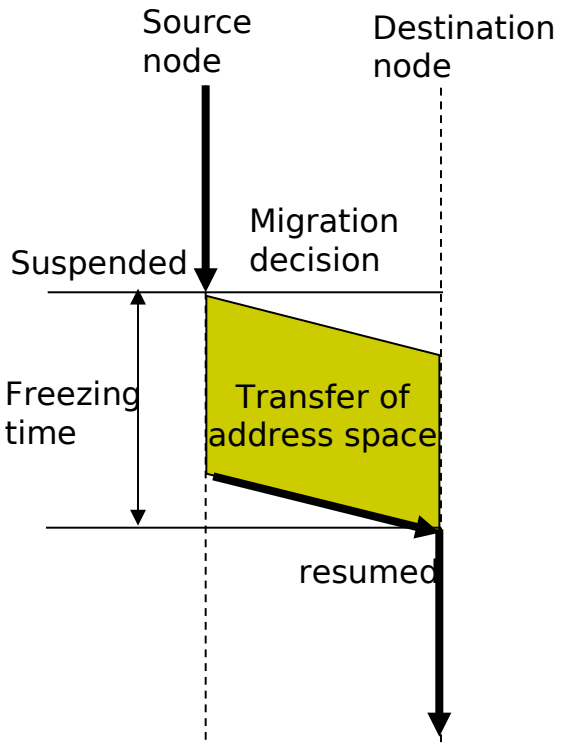
- Respecto a quien inicia el proceso:
  - **Migración iniciada por el emisor**: un programa decide enviar su propio código a máquinas remotas. Por ejemplo, un buscador Web que envía subprogramas buscadores a otras máquinas.
  - **Migración iniciada por el receptor**: un programa decide pedir código para continuar con su ejecución. Por ejemplo los Applets de Java

Captura del estado

- **Registro de la CPU: justo antes de detener la ejecución (¿es parte de la ejecución?)**
- **Espacio de direcciones (¿punteros? ¿hacia donde?)**
- **Entrada / Salida:**
  - **Rápidas:** hacerlas antes de migrar
  - **Permanentes (por ejemplo archivos):**
    - ¿puedo llevarme el archivo?
    - ¿puedo mantenerme conectado con el recurso?
    - ¿estará el archivo en el destino?

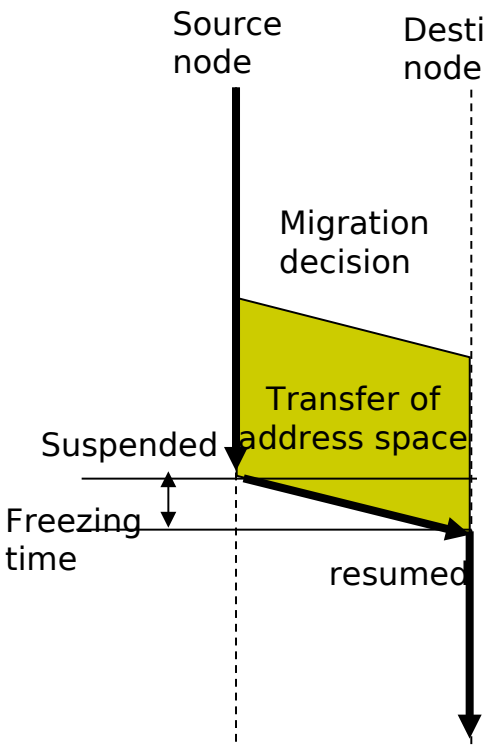
Transferencia de direcciones

Total Freezing



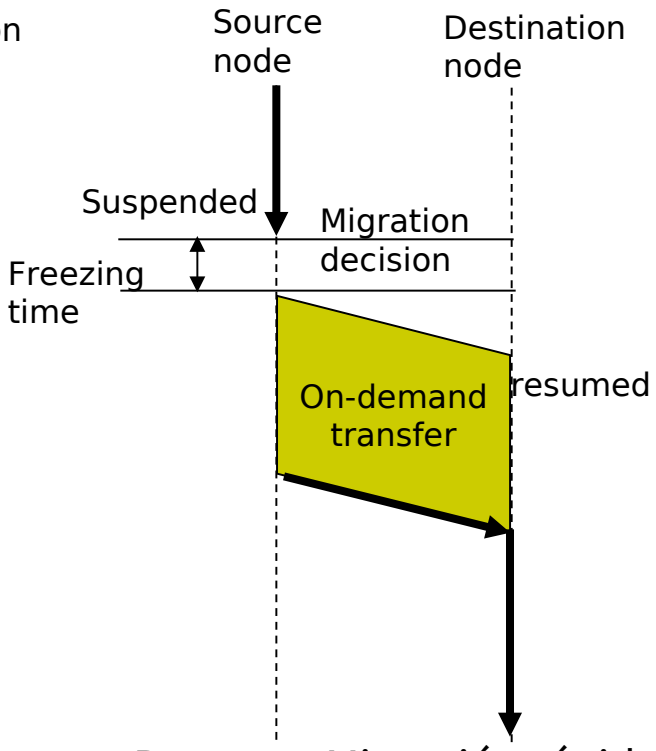
Pros: fácil de implementar
Contras: suspensión

Pre-transferring



Pros: reduce suspensión
Contras: Aumenta tiempo total

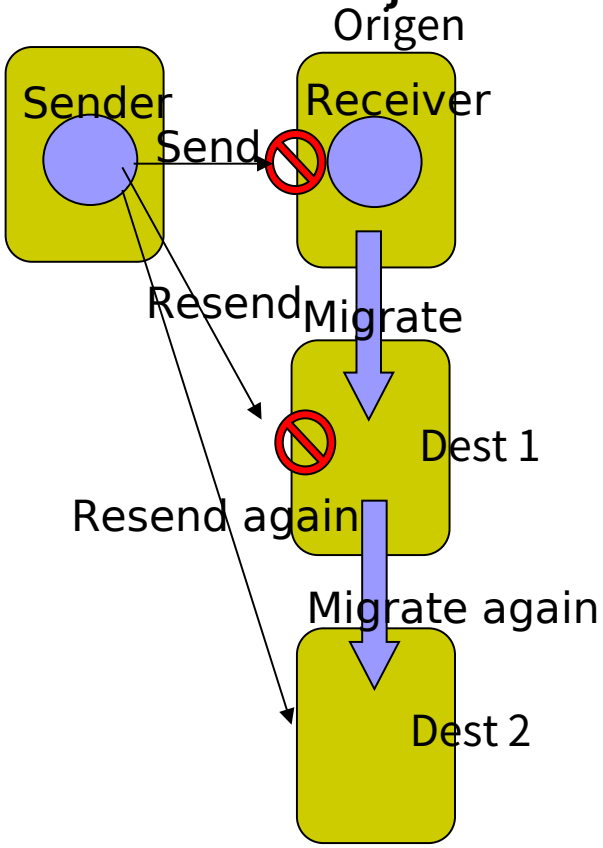
Transfer-on-reference



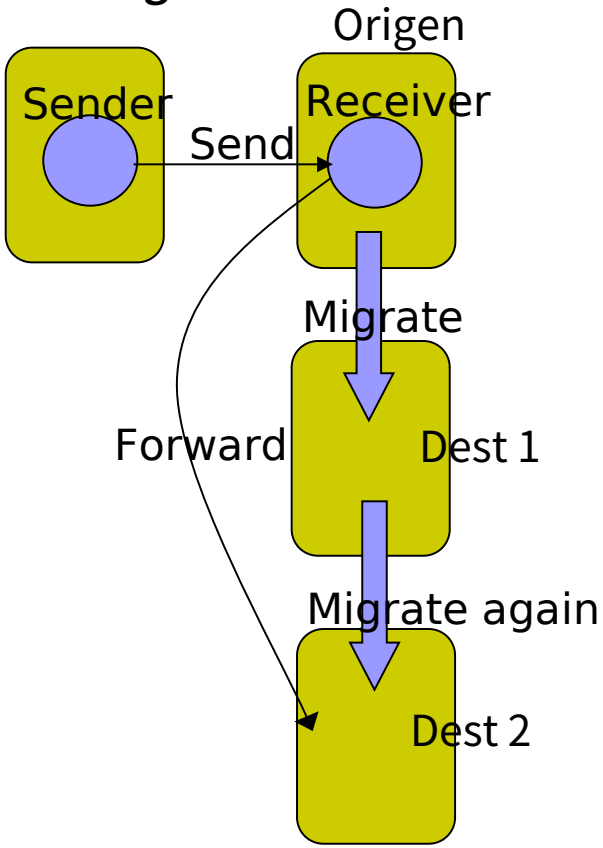
Pros: Migración rápida
Contras: Latencia de memoria

¿Forward?

Reenvío de mensajes

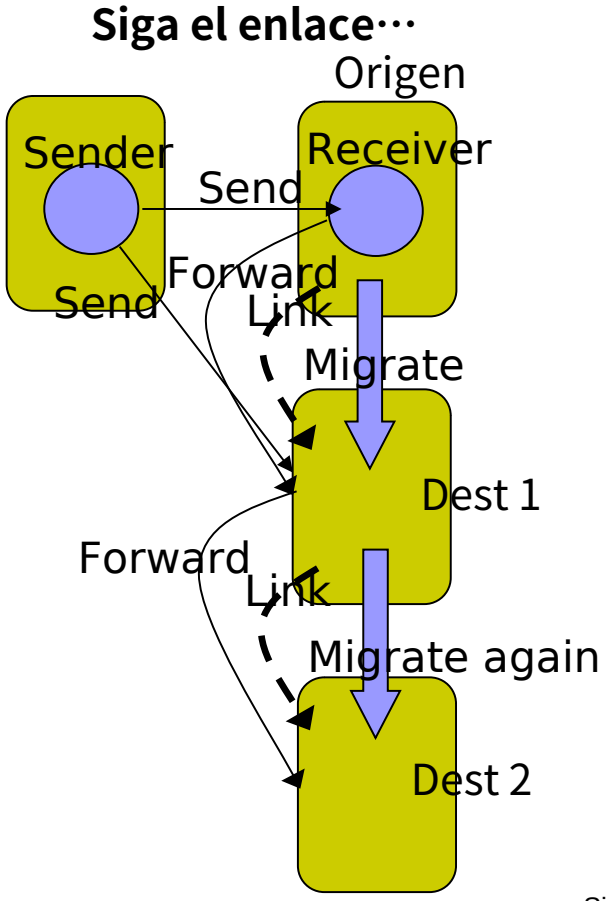


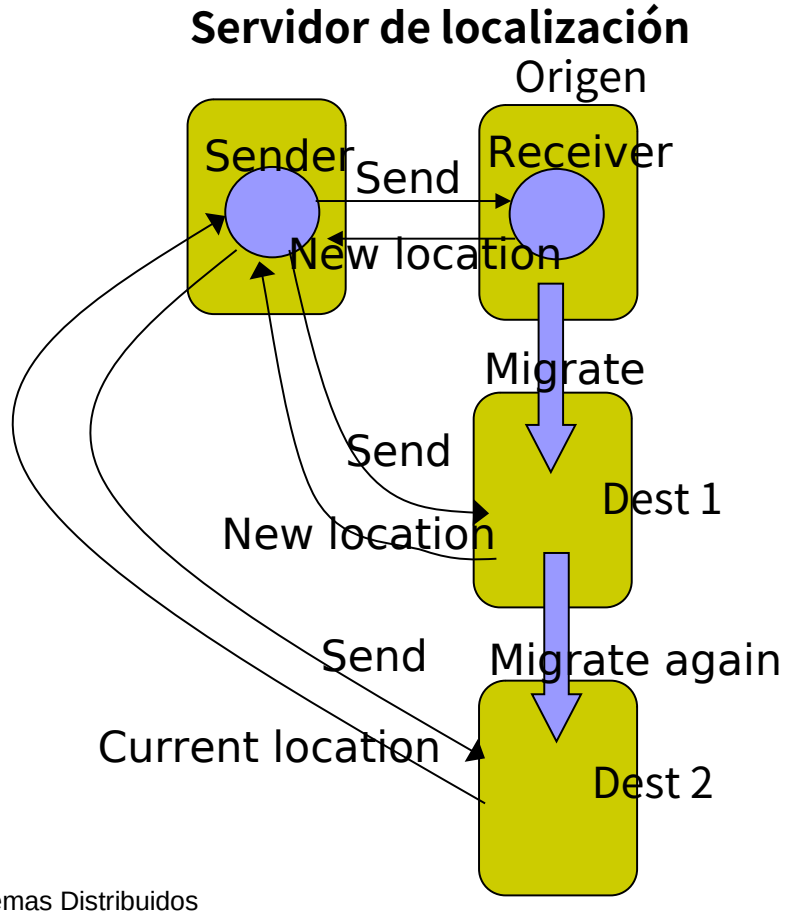
El sitio original mantiene referencia




MIGRATION
IS NOT A CRIME

Ah! Forward!


Siga el enlace…
Origen
Sender
Receiver
Send
Send
Forward
Link
Migrate
Dest 1
Forward
Link
Migrate again
Dest 2


Servidor de localización
Origen
Sender
Send
Receiver
New location
Migrate
Send
Dest 1
New location
Send
Migrate again
Current location
Dest 2

Migración en Sistemas Heterogeneos

- Representación de Datos (¿arquitectura?)
- Precisión de datos (¿punto flotante?)

→ En general son muy caros

→ Java lo ha hecho más barato.

Agentes (inteligentes)

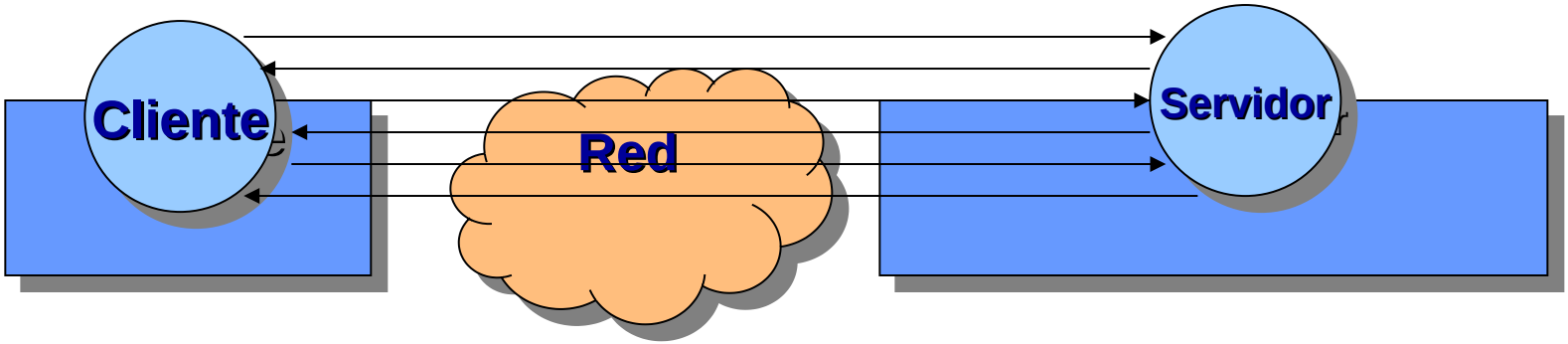
- **Agente:**
  - **Proceso autónomo, capaz de reaccionar ante cambios de su entorno y de iniciar cambios en su entorno, posiblemente en colaboración con usuarios y otros agentes.**

Tipos de Agentes

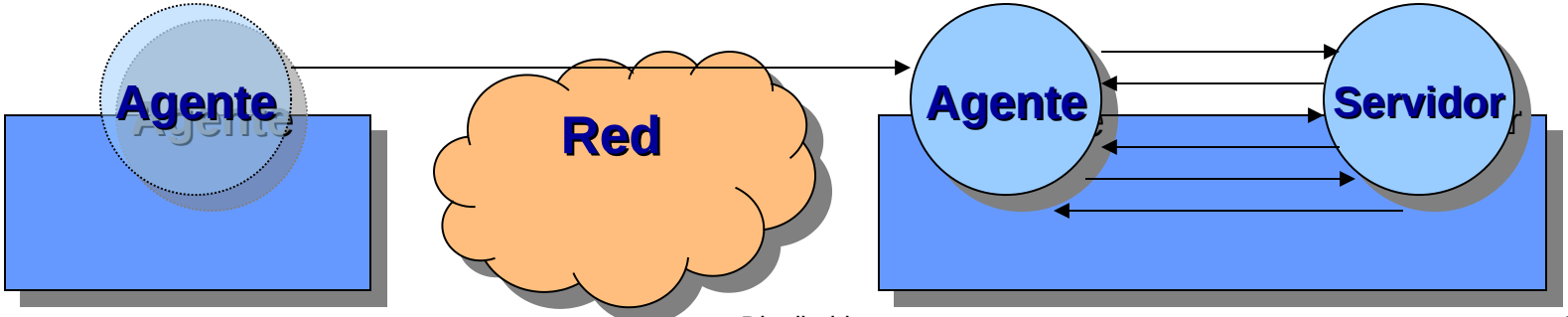
- Agentes **colaborativos**: se construyen como procesos independientes enfocados a ofrecer cierta funcionalidad de forma común (agendas colaborativas).
- Agentes **móviles**: agentes diseñados para migrar de una máquina a otra.
- Agentes **de interfaz**: diseñados para adaptarse al uso que realice el usuario de cierta funcionalidad (brokers).
- Agentes **de información**: dedicados a recopilar información de diferentes medios y operar sobre la información antes de proporcionarla (filtros, cookies).

Agentes móviles

Convencionalmente....



Usando agentes móviles



¿por qué usar agentes?

- **Bajo tráfico de red y latencia.**
  - La comunicación agente-servidor es local.
- **Encapsulación**
  - El código viaja con el agente (¿seguridad?)
- **Navegación autónoma y asíncrona**
  - El agente decide dónde ir y cuando lo hace termina la comunicación con su antigua ubicación.
- **Adaptabilidad**
  - Los agentes pueden dinámicamente agregarse objetos durante sus viajes.
- **Tolerancia a fallas**
  - Un agente puede irse de un nodo si ve que presenta fallas.

En resumen: Ejemplos de Sistemas

Tipos	Sistemas
Paso de mensajes	Socket, PVM, MPI
RPC	Xerox Courier, SunRPC, RMI
Ejecución remota	Servlets, Remote evaluation, Tacoma
Code on Demand	Applets, VB/Jscripts
Migración de Código	Condor, Sprite, Olden
Agentes móviles (Weak Migration)	IBM Aglets, Voyager, Mole
Agentes Móviles (Strong Migration)	Telescript, D'Agent, Ara

En resumen: Movilidad de código

	Datos	Control	Código	Estado de datos	Estado de Ejecución	Autonomía	Dirección de transferencia
Paso de mensajes	SI						E/S
RPC	SI	SI					S
Ejecución remota	SI	SI	SI				S
Code-on-Demand	SI		SI				E
Migración de procesos	SI	SI	SI	SI	SI		E/S
Agentes móviles (weak)	SI	SI	SI	SI		SI	E/S
Agentes móviles (strong)	SI	SI	SI	SI	SI	SI	E/S

En resumen:

Migración de procesos v/s Agentes Móviles

	Migración de procesos	**Agentes Móviles**
Autonomía	Migración decidida por el sistema.	Agentes deciden donde ir
Ejecución de código	Programas compilados y ejecutados en forma nativa.	La mayoría son programas en Java interpretados por la máquina virtual.
Migración Fuerte/Débil	La ejecución es reanudada justo desde el punto de suspensión.	Los agentes Java reinician la ejecución desde el último método llamado.
Estado de Entrada/Salida	E/S permanentes son forwardeadas a la nueva ubicación.	Deben rehacer los enlaces de E/S cada vez que cambian de ubicación.