Replicación de Datos en Sistemas Distribuidos

Joaquín Seoane
Alejandro Alonso

Dpto. Ing. de Sistemas Telemáticos

Tabla de contenidos

1. Introducción a la replicación

- Varias copias de la información o servidores
- ¿Para qué?:
  - Mejora de rendimiento
  - Mejora de la disponibilidad
  - Tolerancia a fallos
- Requisitos a considerar:
  - **Transparencia**: clientes no conscientes de la replicación
  - **Coherencia**: diferencias en la respuesta entre las réplicas
- Realización:
  - Modelo arquitectónico.
  - Comunicación de grupos.
  - Ejemplos.

Caches y sugerencias

- Caches:
  - Son réplicas parciales bajo demanda.
  - Suponen probabilidad de acceso futuro.
  - Ej: caches de sistemas de ficheros, caches de páginas en memoria virtual distribuida, navegadores y proxis de WWW.
- Sugerencias (*hints*):
  - Variante no fiable de cache.
  - Permite detectar fallos y encontrar la información verdadera.
  - Ej: volumen de un fichero en AFS, propietario probable en memoria virtual distribuida, ...
- Replicación completa explícita:
  - Ej: Ficheros replicados, noticias electrónicas, etc.

Replicación y mejora de rendimiento

- ☑ Acceso a réplica cercana
- ☑ Reparto de carga

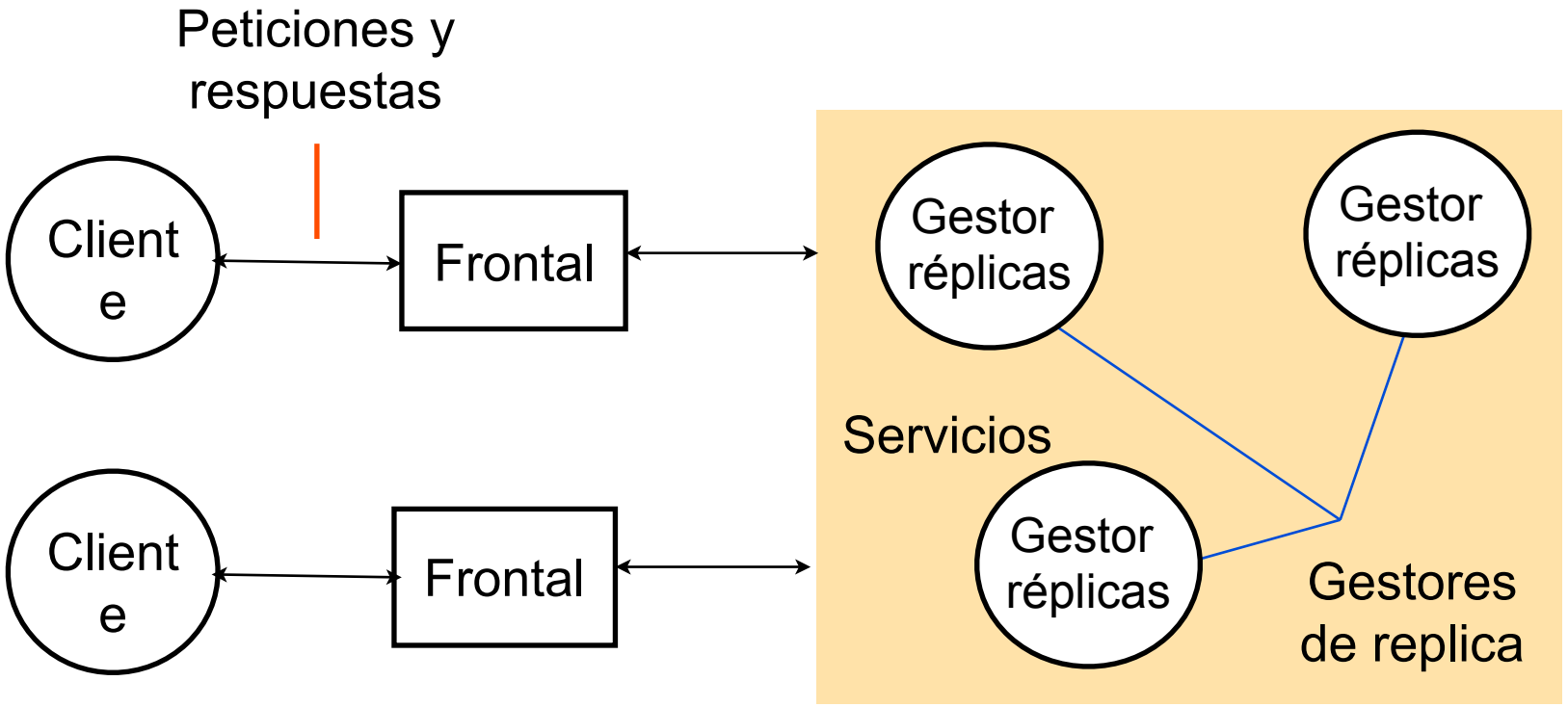
Replicación y mejora de disponibilidad

- Si n copias accesibles fallan, aún puedo acceder a la copia n + 1 (si la hay)
- La probabilidad de fallo (en un intervalo de tiempo t) se reduce con la copias
  - **pf = $\prod$ pf$_i$** (si independientes)
  - **d = 1 − pf** (aumenta la disponibilidad)
- Si los sistemas fueran interdependientes, disminuiría la disponibilidad.
- Un enlace es parte de un sistema y su fallo causa el de las máquinas conectadas → enlaces redundantes.
- Las caches no aumentan la disponibilidad (en

Replicación y tolerancia a fallos

- Fallo: Comportamiento no coherente con la especificación
- Fallos de programas:
  - Varias implementaciones del mismo servicio.
- Respuesta en tiempo real:
  - Fallo es no poder responder a tiempo.

2. Modelo de sistema



Cliente: Ven un servicio que les permite acceder a objetos virtuales
Frontal: Gestionan las peticiones de los clientes y proporcionan transparencia
Gestor de réplica: Proporcionan los servicios y se pueden comunicar entre sí.

Transparencia de replicación

- La replicación puede ser manejada por el cliente (no transparente).
- La transparencia se logra con un frontal interpuesto.
- Las operaciones retornan un sólo valor o fallan.
- Normalmente sólo se ve una copia
  - Salvo si hay que resolver conflictos (como en CODA).

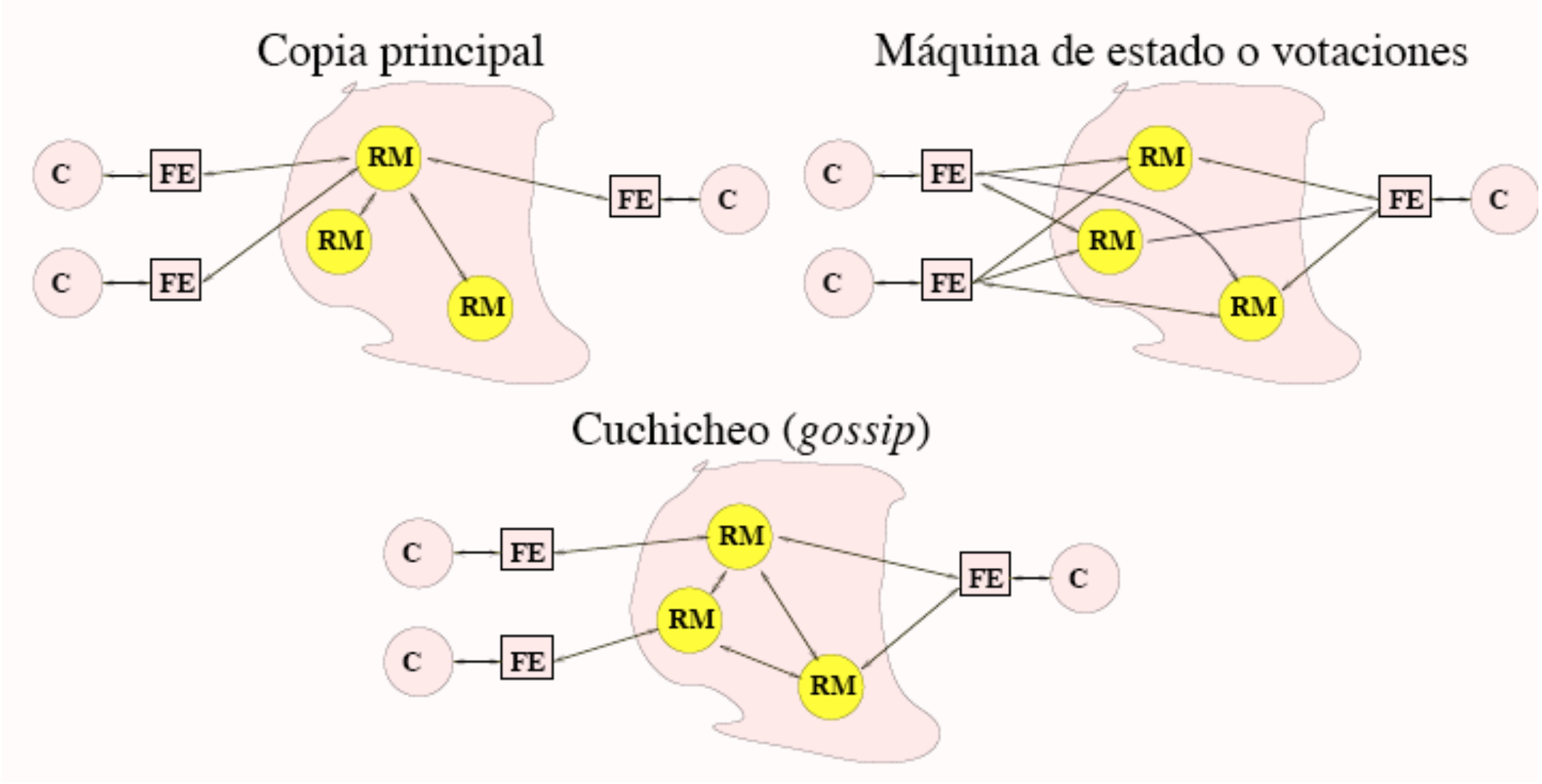
Fases en la realización de una petición

1. Petición del frontal a:
   - Un gestor de réplicas, que se comunica con los demás.
   - Todos los gestores de réplicas (multienvío).
2. Coordinación:
   - Si se realiza o no
   - Ordenación: FIFO, causal, total.
3. Ejecución (quizá tentativa).
4. Acuerdo (quizá abortando).
5. Respuesta al frontal (uno o varios)

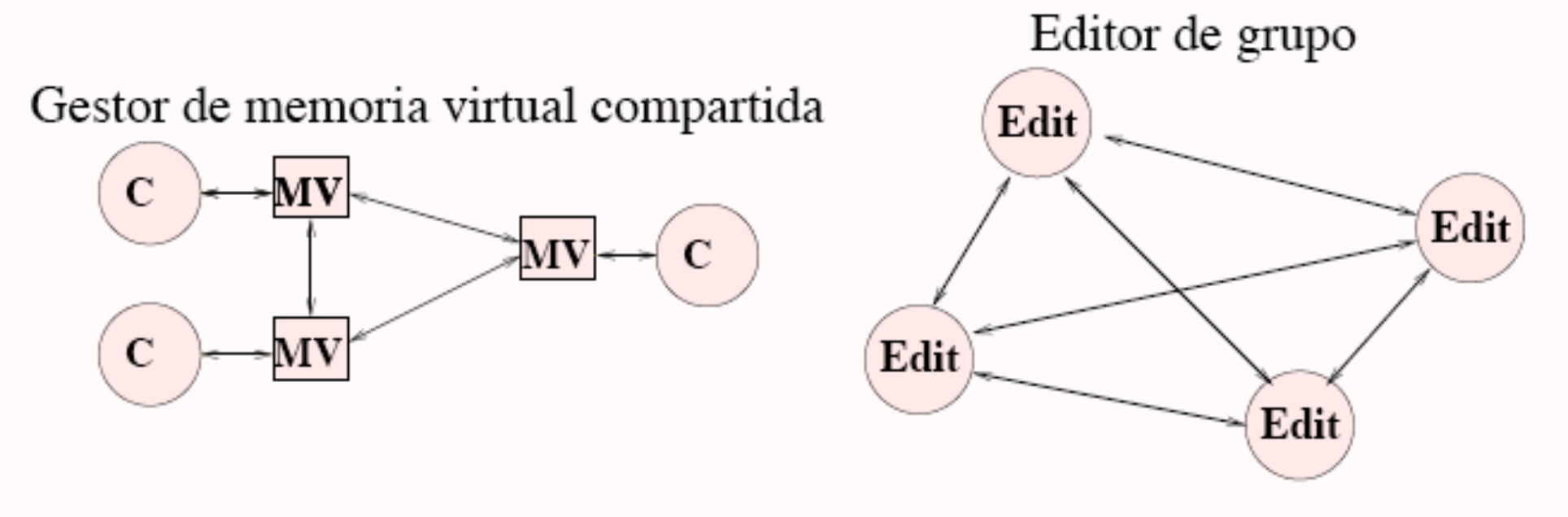
Tipos de orden

- FIFO
  - Si un FE envía una petición r y luego r', entonces todas la réplicas correctas procesan r antes que r'
- Causal
  - Si r ocurre-antes que r',
    entonces todas la réplicas correctas procesan r antes que r'
- Total:
  - Si una réplica correcta procesa r antes que r',
    entonces todas las réplicas correctas procesan r antes que r'

Ejemplos de Arquitecturas



Arquitecturas combinadas



Coherencia (consistencia)

- Las propiedades del sistema se cumplen siempre:
  - Pesimismo: impide que no se cumplan.
  - Optimismo: permite avanzar siempre, si se pueden detectar las incoherencias más adelante.
- Requisitos de las aplicaciones:
  - Orden total.
  - Orden causal.
  - Orden total y causal.
  - Desorden total.
  - La igualdad y coherencia de réplicas son operadores dependientes de la aplicación.
- A mayor coherencia, menor rendimiento.

Ejemplo: BBS incoherente

os.interesting

Smith

23	Hanlon	Mach
24	Joseph	Microkernels
25	Hanlon	Re: Microkernels
26	Hereux	RPC performance
27	Walker	Re: Mach

Jones

20	Joseph	Microkernels
21	Hanlon	Mach
22	**Sabiner**	**Re: RPC performance**
23	Walker	Re: Mach
24	**Hereux**	**RPC performance**
25	Hanlon	Re: Microkernels

Ejemplo: BBS causal

os.interesting

Smith

23	Hanlon	Mach
24	Joseph	Microkernels
25	Hanlon	Re: Microkernels
26	Hereux	RPC performance
27	Walker	Re: Mach

Jones

20	Joseph	Microkernels
21	Hanlon	Mach
22	Walker	Re: Mach
23	**Hereux**	**RPC performance**
24	**Sahiner**	**Re: RPC performance**
25	Hanlon	Re: Microkernels

Ejemplo: BBS ordenado totalmente

os.interesting

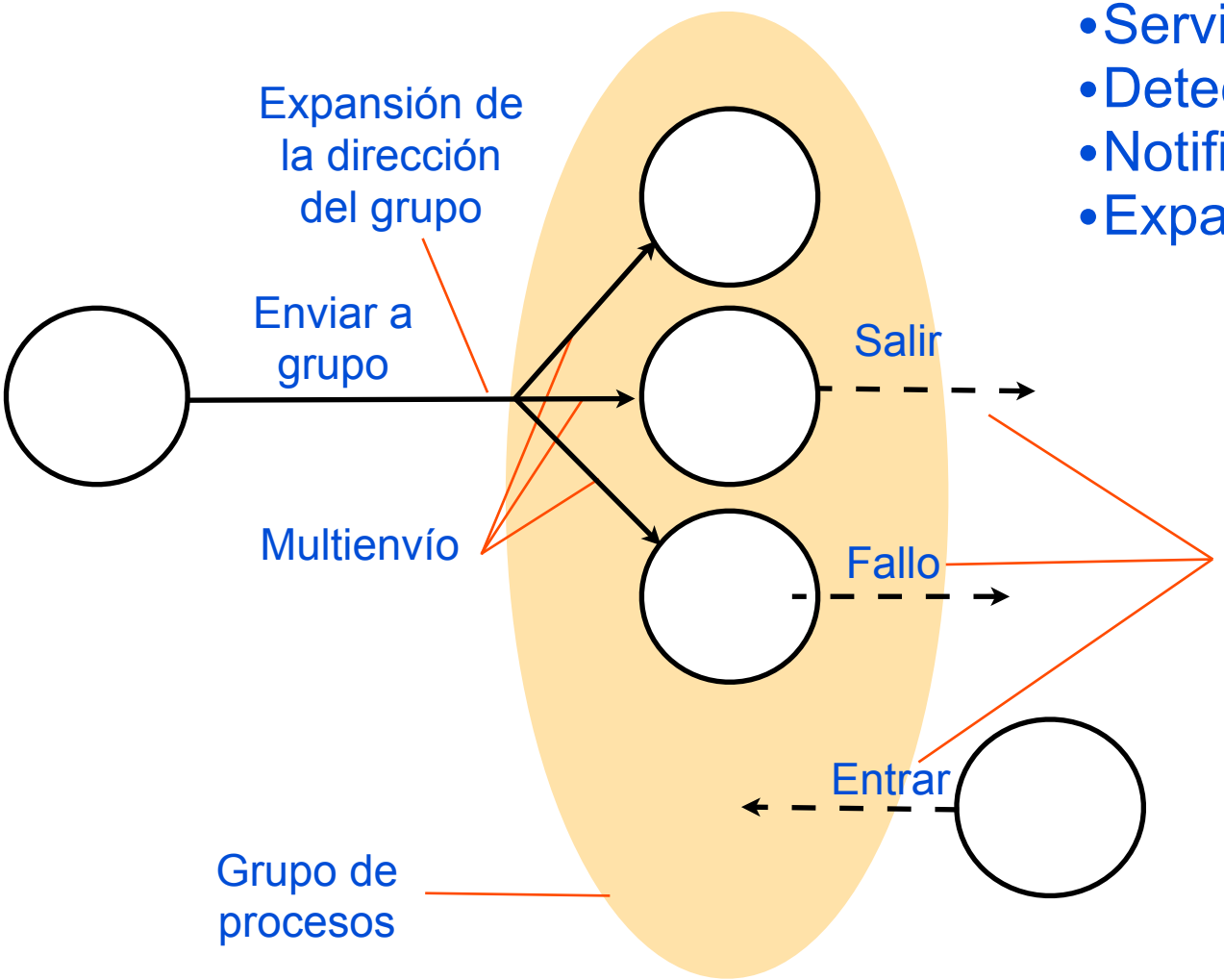
Smith

23	Hanlon	Mach
24	Joseph	Microkernels
25	Hanlon	Re: Microkernels
26	Hereux	RPC performance
27	Walker	Re: Mach
28	Sahiner	Re: RPC performance

Jones

23	Hanlon	Mach
24	Joseph	Microkernels
25	Hanlon	Re: Microkernels
26	Hereux	RPC performance
27	Walker	Re: Mach
28	Sahiner	Re: RPC performance

3. Servicio de gestión de grupos



- Servicio de pertenencia
- Detector de fallos
- Notificación de miembros
- Expansión de direcciones

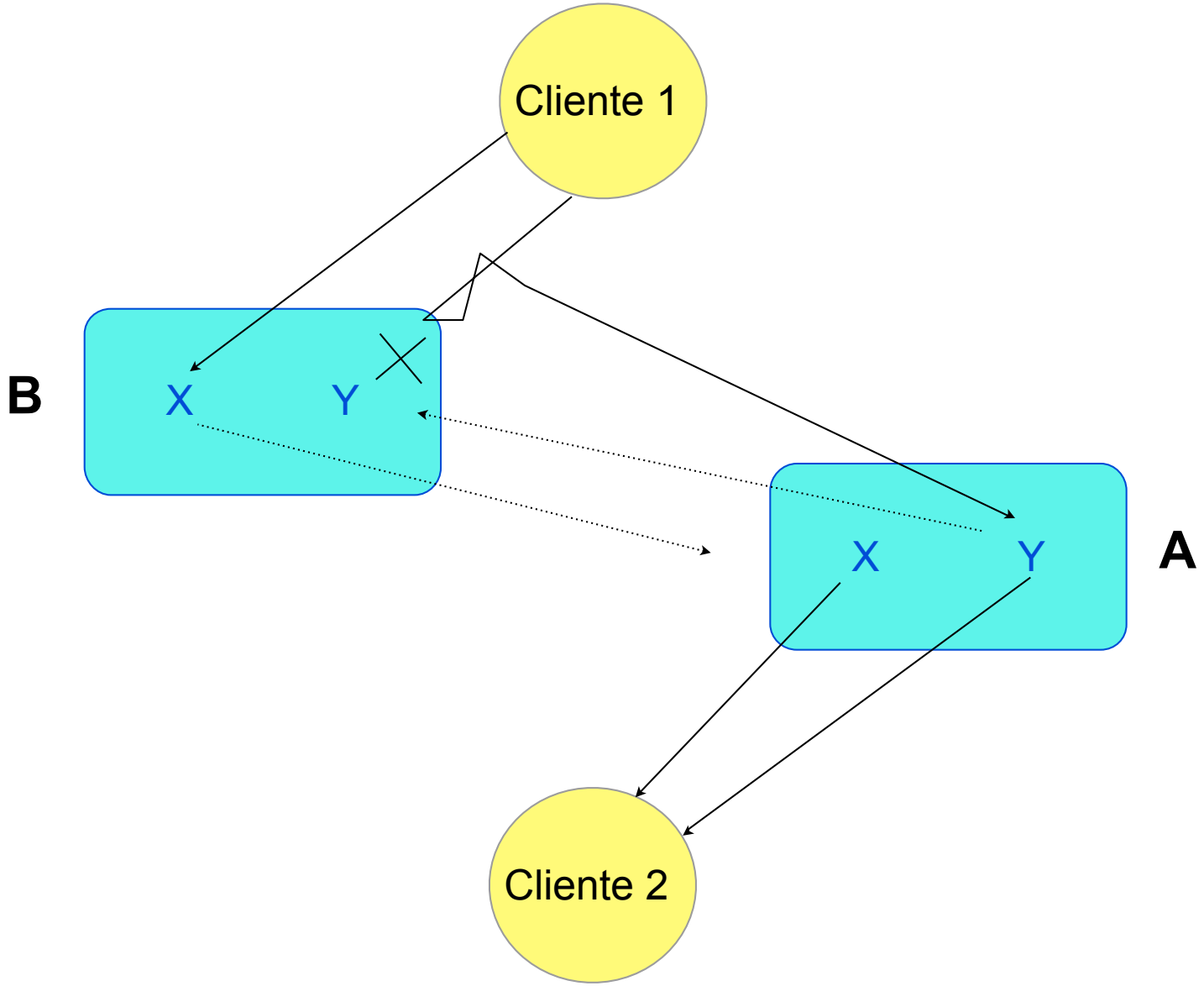
4. Servicios tolerantes a fallos

- En presencia de fallos cumplen la especificación (respuesta correcta)
- Obtener respuesta suele ser parte de la especificación (disponibilidad)
- Por su repetición, los fallos pueden ser:
  - Transitorios, intermitentes, permanentes
- Tipos de fallos
  - Fallo silencioso/caída: funcionan bien o no funcionan
    - No es inmediato saber que han fallado
  - Fallo parada: funcionan bien o no funcionan
    - Facilitan que otros nodos detecten este evento
  - Fallos bizantinos/arbitrarios: Respuesta errónea y maliciosa
    - Requieren votación

Soluciones tolerantes a fallos

- Redundancia:
  - De información (datos replicados, códigos redundantes, CRC).
  - Temporal (retransmisiones, repetición de transacciones abortadas).
  - Física (componentes, máquinas, conexiones replicados).
  - De información y física combinadas (discos espejo, RAID).
- La redundancia puede introducir a su vez fallos (respuestas incorrectas).

Ejemplo: replicación perezosa de cuentas bancarias



Inconsistencia replicación perezosa

Cliente 1:	Cliente 2:
$Escribe_B$ (x, 1)	
$Escribe_A$ (y, 2)	
	$Lee_A(y) \rightarrow 2$
	$Lee_A(x) \rightarrow 0$

Viola el sentido común, ya que y = 2
se hizo después que x = 1

Algunos criterios de consistencia

- ☑ Linealizabilidad.
- ☑ Consistencia secuencial.
- ☑ Consistencias débiles.

Linealizabilidad

- Un objeto replicado es linealizable ≡

  ∀ ejecución ∃ un entrelazado de las operaciones de los clientes que verifica:

  - Coincide con un posible entrelazado con el objeto sin replicar.
  - El orden de las operaciones en el entrelazado es consistente con el tiempo real en que ocurrieron en los clientes.

Consistencia secuencial

- Un objeto replicado es secuencialmente consistente ≡ ∀ ejecución ∃ un entrelazado de las operaciones de los clientes que verifica:
  - Coincide con un posible entrelazado con el objeto sin replicar.
  - El orden de las operaciones en el entrelazado es consistente con el orden del programa en que las ejecutaron los clientes.

Ejemplo

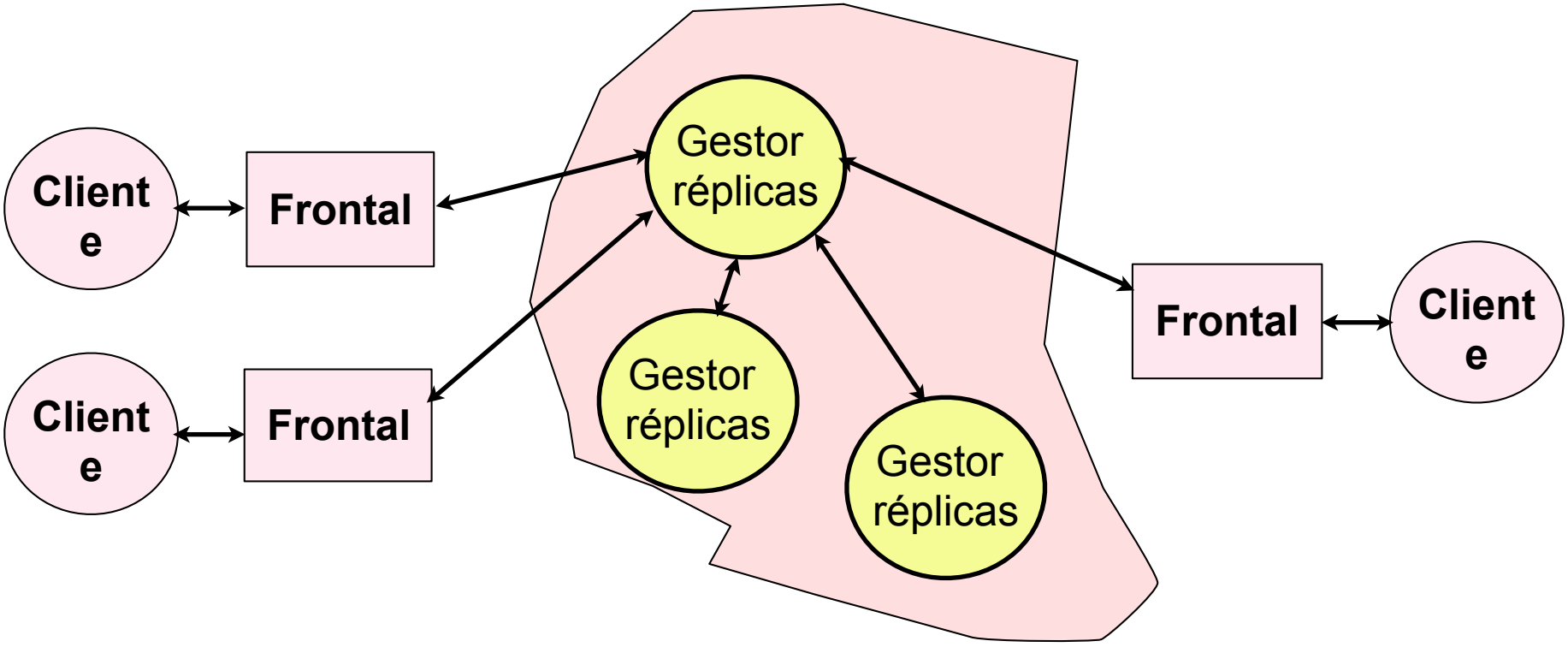
- Entrelazado secuencialmente consistente, no linealizable

Cliente 1:	Cliente 2:
$Escribe_B$ (x, 1)	
	$Lee_A(y) \rightarrow 0$
	$Lee_A(x) \rightarrow 0$
$Escribe_A$ (y, 2)	

Este entrelazado cumple la consistencia secuencial:
$Lee_A(y) \rightarrow 0$, $Lee_A(x) \rightarrow 0$, $Escribe_B(x, 1)$, $Escribe_A(y, 2)$

4.1 Replicación pasiva

- También conocida como principal/respaldo



Replicación pasiva

- Todas las peticiones al principal
  (si van a respaldo, se redirigen)
- Servidores de respaldo mantienen el estado por si tienen que sustituir al primario
- Más barato y usado que
  las máquinas de estado replicadas
- Muchas veces primario + respaldo +
  otros respaldos apagados

Fases de la replicación pasiva

1. Petición: Con identificador único al principal.
2. Coordinación: El principal elimina duplicados.
3. Ejecución: El principal ejecuta y obtiene la respuesta.
4. Acuerdo: Si es escritura actualiza los respaldos (canales FIFO y asentimiento).
5. Respuesta: El primario responde.

Replicación pasiva: Linealizabilidad

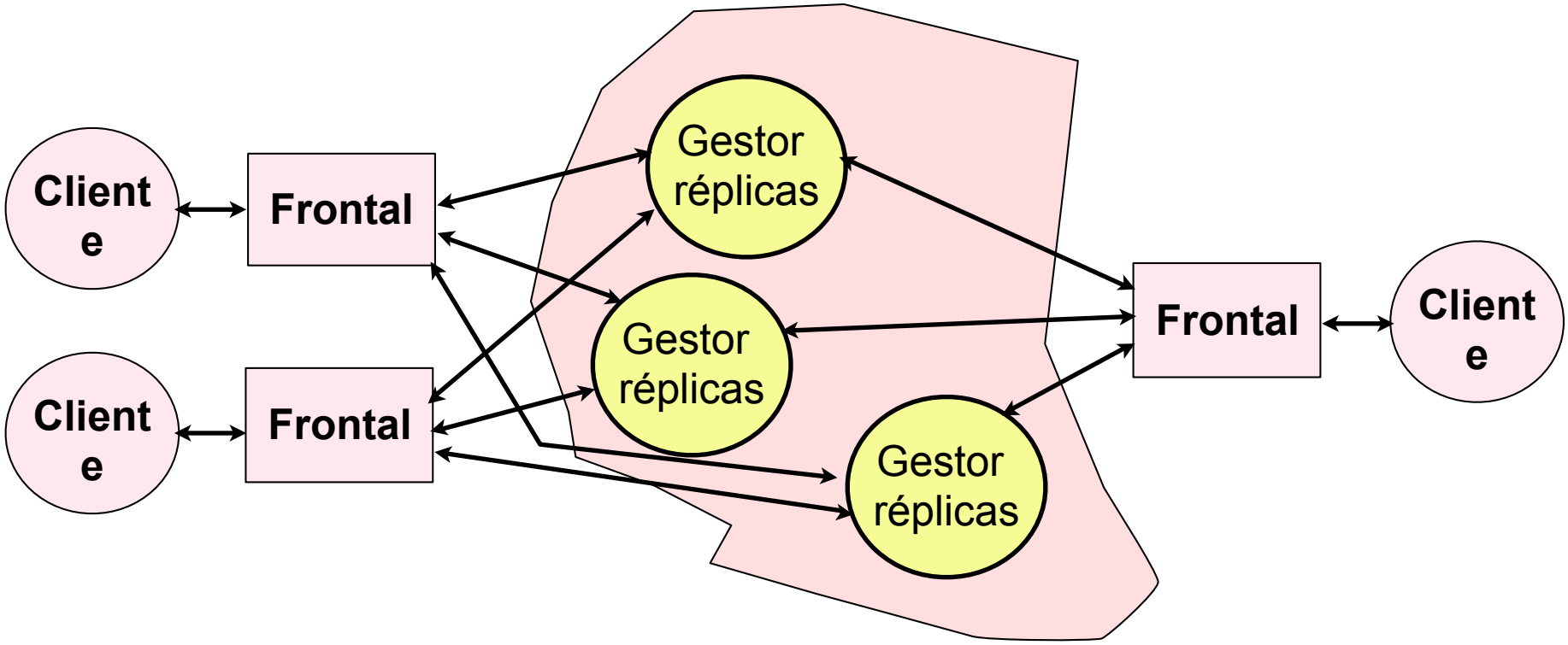
- Obvio si el primario es correcto
  - Se supone que las operaciones se bloquean hasta que se haya entregado la respuesta.
- Si el primario cae:
  - Es reemplazado por un único respaldo
  - Las réplicas acuerdan qué operaciones se realizaron antes del cambio
- Se puede realizar como grupo con comunicación síncrona de vistas

Replicación pasiva: observaciones

- Sobrevive a *f* fallos con *f+1* réplicas
- No tolera fallos bizantinos (el primario es vital)
- No tolera particiones
- Cuello de botella en el primario
- Mucho retardo en la actualización
- Si se hacen lecturas de respaldos, pueden ser obsoletas e incoherentes:
  - Más eficiencia
  - Servicios de nombres (sugerencias)

4.2 Replicación activa

- También llamada de máquinas de estado



Replicación activa

- Autómata finito determinista:
  - Lecturas que extraen información del estado.
  - Escrituras que modifican el estado.
  - El estado no es modificado por el tiempo, sensores, etc.
- Replicabilidad con la misma semántica → Orden total.
- Tolerancia a t fallos
  - Fallo/parada: *f* + *1* réplicas (basta una respuesta en lecturas).
  - Bizantino: *2f* + *1* réplicas (hay que votar).
  - Particiones: no utilizable.

Fases de la replicación activa

1. Petición: Multienvío atómico con identificador único.
2. Coordinación: Entrega fiable y con orden total (atómica).
3. Ejecución: Toda réplica ejecuta la petición.
4. Acuerdo: Ya lo hizo el multienvío atómico.
5. Respuesta:
   - ✓ Fallo/parada: Primera respuesta.
   - ✓ Bizantino: Mayoría de respuestas iguales.

Observaciones de la replicación activa

- No linealizable
- Secuencialmente consistente (orden FIFO)
- Sólo implementable en sistemas síncronos
- Costoso
- Lecturas sólo a una réplica
- Relajación de orden total: operaciones que conmutan:
  - Lecturas.
  - Escrituras disjuntas.

5. Alta disponibilidad

- Tiempo de respuesta rápido.
- Relajación de consistencia o pesimismo.
- Ejemplos:
  - Arquitectura de cuchicheo de Ladin, Liskov,..
  - CODA.

Arquitectura de cuchicheo

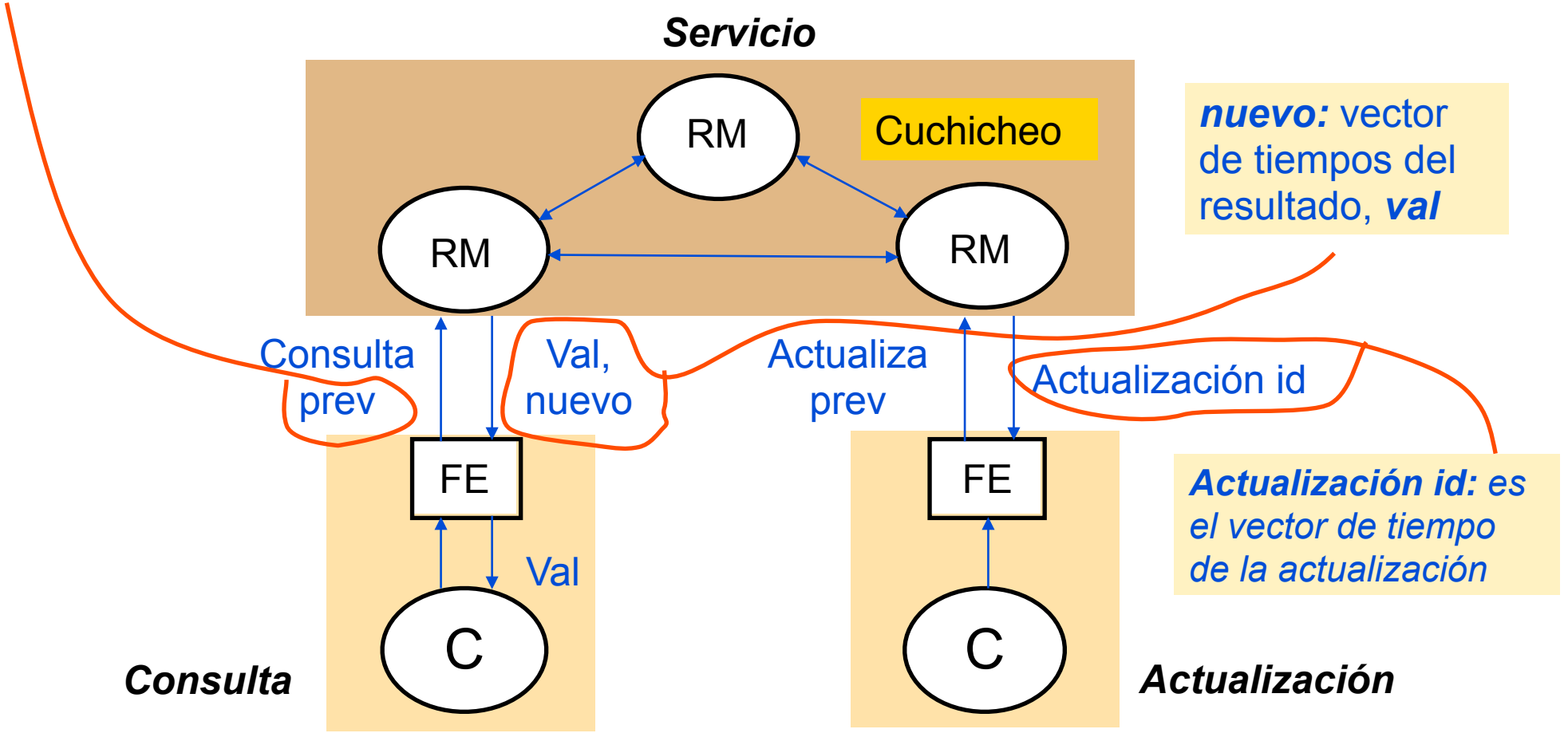
- Propagación perezosa de actualizaciones (por lotes).
- Comunicación con un gestor de réplicas cercano (cualquiera, cambiable).
- Consistencia relajada:
  - No es para obtener consistencia secuencial.
  - Toda consulta obtiene siempre información ya observada.

Operaciones

- Operaciones:
  - Actualización (escritura sin lectura):
    - Causal.
    - Forzada (causal y total).
    - Inmediata (sincronización).
  - Consulta (causal).
- Retorno:
  - Consulta: cuando llegue el dato.
  - Actualización: inmediata o *f* + *1* réplicas

Interacción cliente/servidores

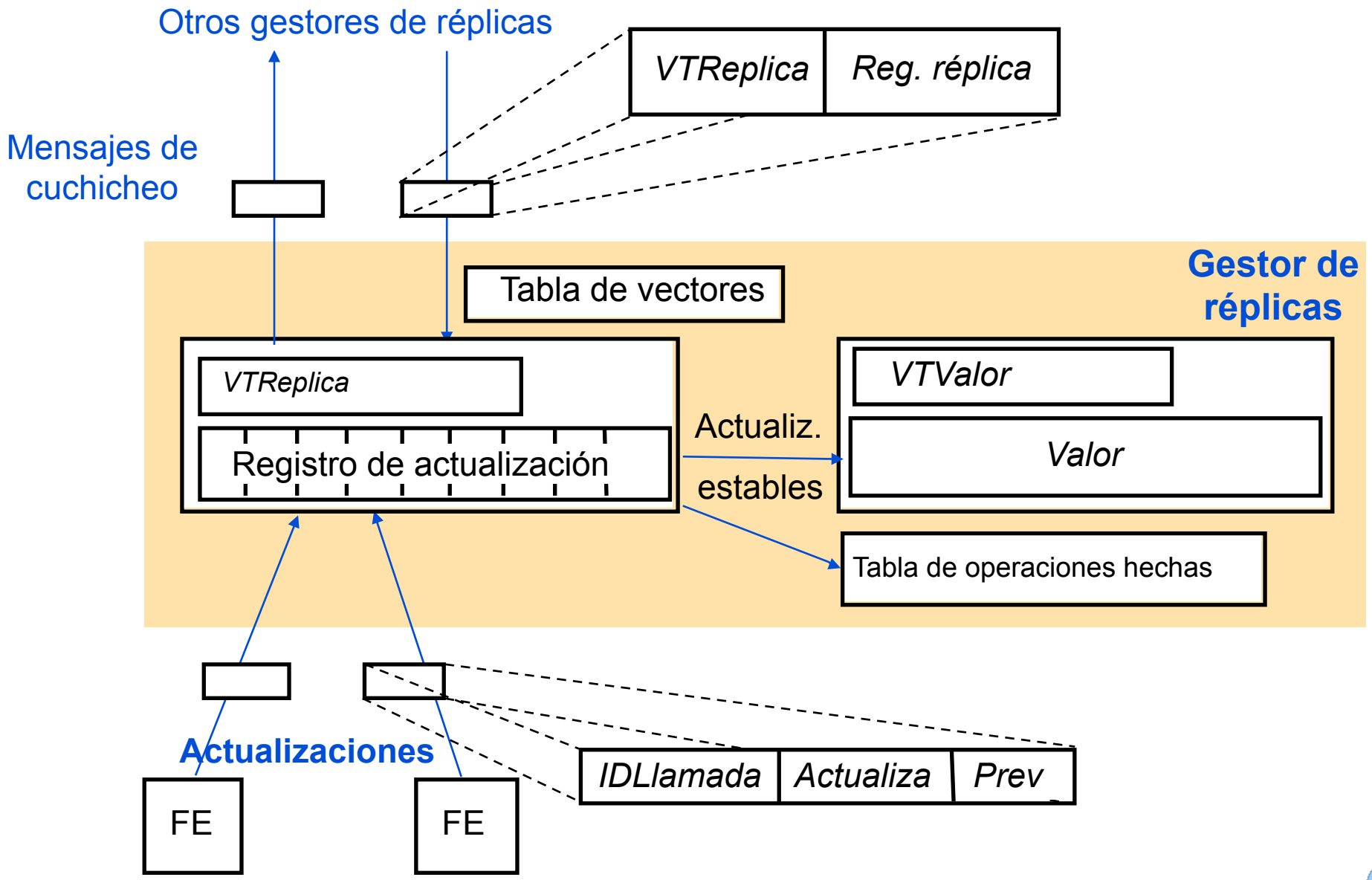
prev: vector con marcas de tiempo con la última versión vista por el frontal



Estado de un gestor de réplicas

- *Valor*
- Vector de tiempo para *Valor(VTValor)*.
  Representa actualizaciones aplicadas;
  - $VT_i[i]$ = número de actualizaciones hechas en réplica i
  - $VT_i[j]$ = número de actualizaciones conocidas en réplica *j* ($j \neq i$)
- Registro de actualizaciones:
  - No aplicables a *Valor* por inestables
  - No confirmada su aplicación en las otras réplicas
- Vector de tiempo de la réplica (*VTReplica*).
  Actualizaciones aceptadas, no necesariamente aplicadas.
- Identificadores de actualizaciones aplicadas (evita repeticiones).
- Tabla de vectores de tiempo de las demás réplicas.

Estado de un gestor de réplicas



Proceso de consultas

- Sólo se puede responder si el *FE* no sabe más que el *RM*:

  $q.prev \leq VTValor$

- Si no, el RM debe:
  - Esperar que cuchicheos actualicen *VTValor*
  - Contactar con los RM relevantes: aquéllos en que $q.prev[i] > VTValor[i]$.

- Una vez que el frontal obtiene respuesta:

  $VTFE := merge(VTFE, new)$

Proceso de actualizaciones causales

- El frontal proporciona (a una o varias réplicas)
  < u.op, u.prev, u.id >
- La réplica comprueba si vista (por *u.id*),
  en cuyo caso la descarta
- Incrementa *VTReplica[i]*
- Se apunta en el registro de actualizaciones
  pendientes: *<i, vt, u.op, u.prev, u.id >*
- Donde *vt* es un vector único:
  vt[i] = VTReplica[i] y *vt[j] = u.prev[j] (j ≠ i)*
- El FE une *vt* con su vector

Proceso de actualizaciones causales

- Condición de estabilidad: $u.prev \leq VTValor$
- Si se cumple, se aplica la operación:
  Valor := aplicar(Valor, u.op)
  VTValor := merge(VTValor, u.vt)
  Ejecutados := Ejecutados ∪ *{u.cid}*
- Si no se cumple, se espera a mensajes de cuchicheo, reevaluando.

Proceso de cuchicheos

- Se envían a un ritmo no especificado
- Contienen el registro y *VTReplica*
- Cuando se recibe:
  - Se mezclan los registros:
    - Añadiendo los que no estén ni tengan vectores menores que *VTValor*
  - Se aplican actualizaciones estables no hechas ya en orden causal
  - Se mezclan los *VTReplica*
  - Se limpia registro de actualizaciones y de operaciones hechas, que se hayan hecho en todas las réplicas

Discusión

- Tratadas sólo las actualizaciones causales
- Ritmo de intercambio de mensajes se ajusta para la aplicación
- No se usan multienvíos
- Se puede contactar con varios *RM* para mejorar latencia, a costa de ancho de banda