Tolerancia a Fallas en SD

9

Sistemas Operativos y Distribuidos

Mg. Javier Echaiz

D.C.I.C. – U.N.S.

http://cs.uns.edu.ar/~jechaiz

je@cs.uns.edu.ar

Conceptos Básicos

La “dependibilidad” incluye:

- Disponibilidad
- Confiabilidad
- Seguridad
- Mantenimiento

Conceptos Básicos

Disponibilidad: se define como la propiedad de que un sistema está disponible para ser usado inmediatamente.

Confiabilidad: se refiere a la propiedad de que un sistema corra continuamente (24/7) sin fallas.

Seguridad: se refiere a la situación en la que un sistema falla temporalmente y nada catastrófico ocurre.

Mantenimiento: se refiere a cuan fácil puede ser reparado un sistema fallado.

Conceptos Básicos

Un sistema se dice que falla cuando no puede cumplir con su propósito.

Un error es parte del estado de un sistema que lleva a una falla.

La causa de un error es un falta/falla.

Tolerancia a las fallas implica que el sistema puede proveer sus servicios aún en presencia de fallas.

Las fallas, en general pueden clasificarse en: transitorias, intermitentes y permanentes.

Modelos de Fallas

Tipos de fallas

Tipo de Falla	Descripción
Crash	Un servidor se detiene, pero estaba funcionando normalmente hasta la detención
Falla por Omisión Omisión de *Receive* Omisión de *Send*	Un servidor falla a responder a los requerimientos de entrada Falla al recibir un mensaje de entrada Falla al enviar mensajes
Fallas de Timing	La respuesta de un servidor cae fuera de un intervalo específico de tiempo
Falla en la Respuesta *Falla en Valor* *Falla en Transición*	La respuesta del servidor es incorrecta El valor de la respuesta es errónea El servidor se desvía del flujo de control correcto
Falla Arbitraria / Bizantina	Un servidor puede producir respuestas arbitrarias en tiempos arbitrarios

Modelos de Fallas

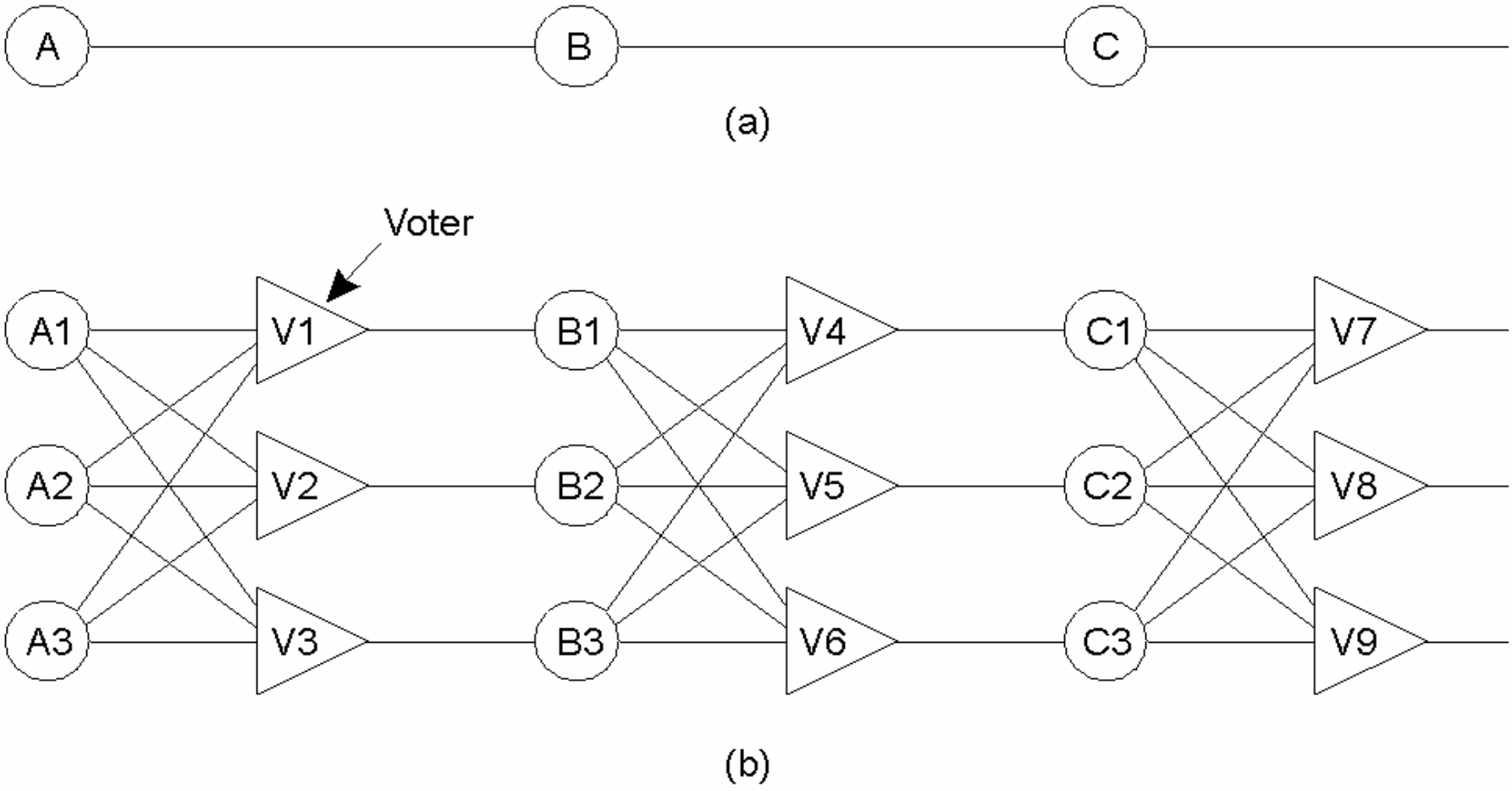
Las fallas arbitrarias son también conocidas como fallas bizantinas (*).

Puede ocurrir que un servidor produzca una salida que nunca debería haberse producido pero que no puede ser calificada como incorrecta.

Peor si hay servidores que actúan maliciosamente (seguridad).

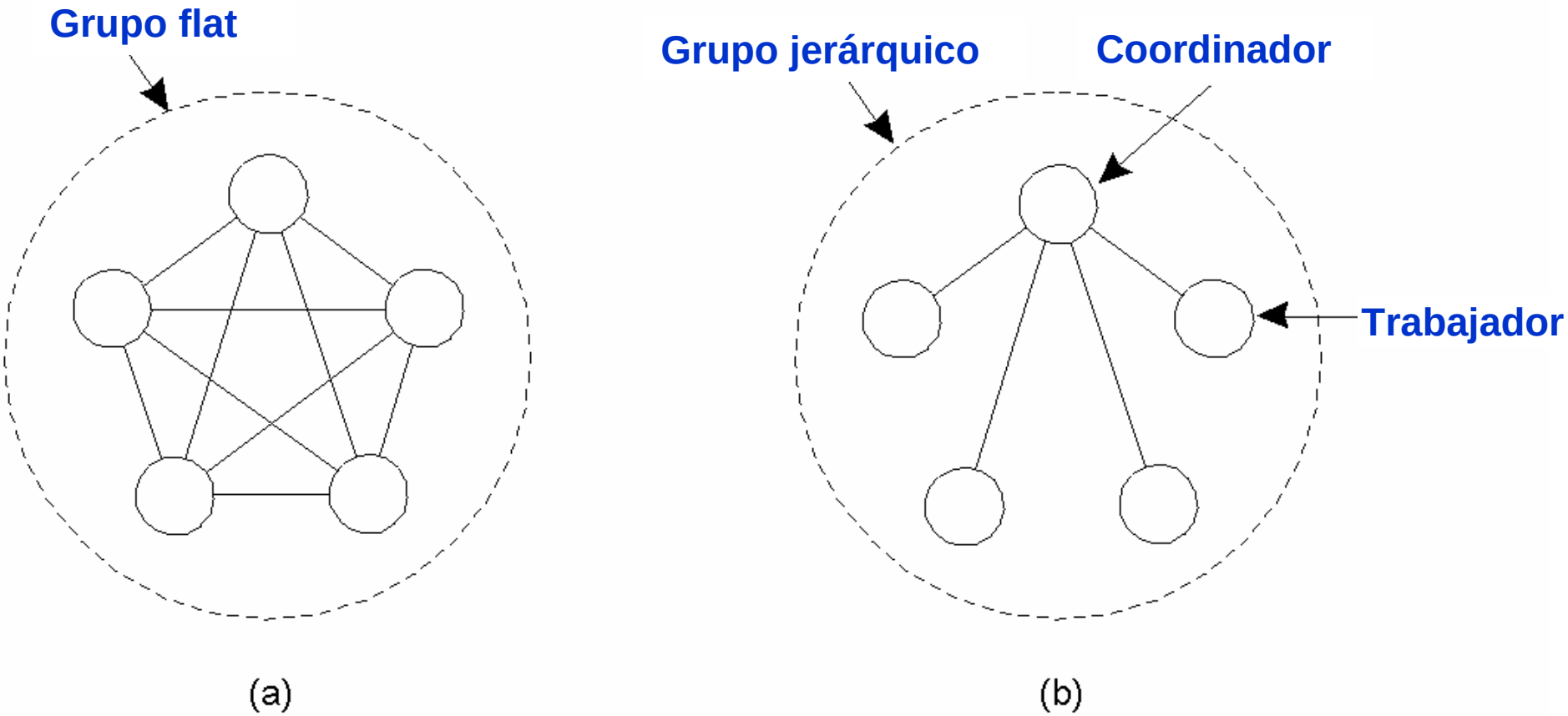
** bizantinas por el imperio Bizantino (330-1453) donde las conspiraciones, intrigas, traiciones y mentiras eran moneda corriente.*

Enmascaramiento de Fallas por Redundancia



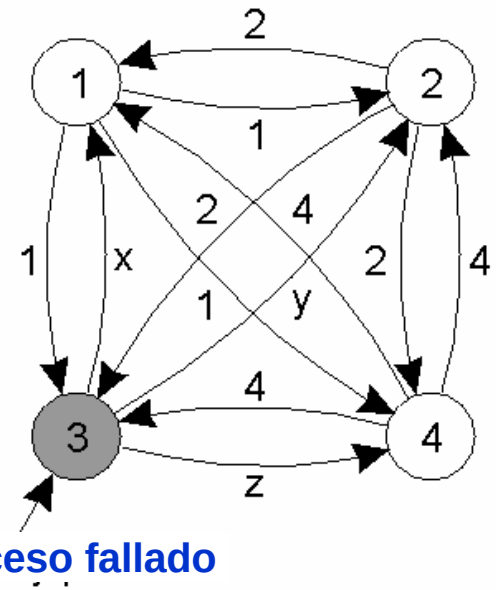
Triple redundancia modular

Grupos “Flat” versus Grupos Jerárquicos



a) Comunicación en un grupo “flat”.
b) Comunicación en un único grupo jerárquico

Acuerdo en Sistemas con Fallas (1)



(a)

1 **Obt**(1, 2, x, 4)
2 **Obt**(1, 2, y, 4)
3 **Obt**(1, 2, 3, 4)
4 **Obt**(1, 2, z, 4)

(b)

1 **Obt**	2 **Obt**	4 **Obt**
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

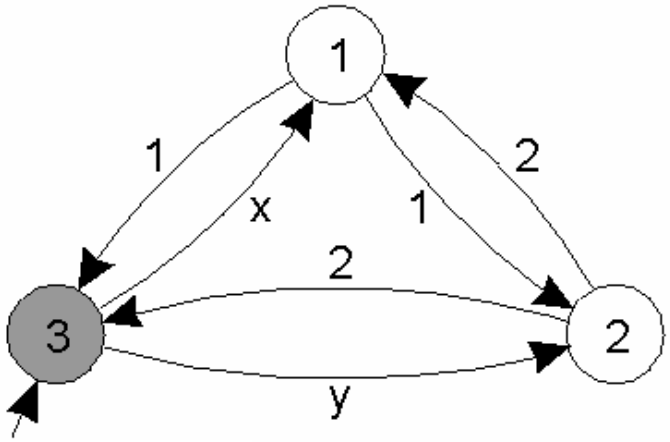
El problema de los generales Bizantinos para tres generales leales y un traidor.

a) Los generales anuncian la fortaleza de su tropa (en unidades de “kilosoldados”).

b) Los vectores que cada general arma basados en (a)

c) Los vectores que cada general recibe en el paso 3.

Acuerdo en Sistemas con Fallas (2)



Proceso fallado

(a)

1 Obt (1, 2, x)
2 Obt (1, 2, y)
3 Obt (1, 2, 3)

(b)

1 Obt
(1, 2, y)
(a, b, c)

2 Obt
(1, 2, x)
(d, e, f)

(c)

Idem slide previa, ahora con dos generales leales y un traidor.

Comunicación Confiable Cliente-Servidor

Una comunicación confiable punto a punto es lograda haciendo uso de un protocolo confiable como TCP.

Este enmascara las fallas en los mensajes pero no hay manera de enmascarar la ruptura del canal de comunicaciones.

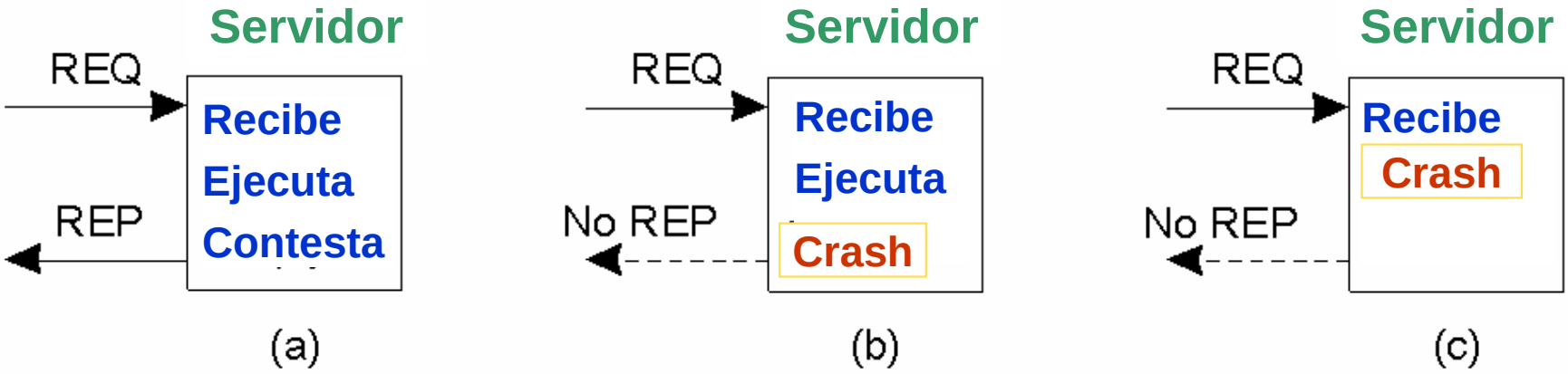
Se puede tomar como ejemplo de comunicación cliente-servidor el uso de la *facility* de comunicación de alto nivel como RPC (o RMI).

Comunicación Confiable Cliente-Servidor

Pueden ocurrir cinco clases de errores en un sistema RPC:

1) El cliente no puede ubicar al servidor.
2) Se pierde el requerimiento del cliente al servidor.
3) El servidor se cae después de recibir un requerimiento.
4) Se pierde la respuesta del servidor al cliente.
5) El cliente cae luego de enviar un requerimiento.

Pérdida de Mensajes de Requerimiento: Crash del Servidor (1)



Un servidor en la comunicación cliente-servidor.

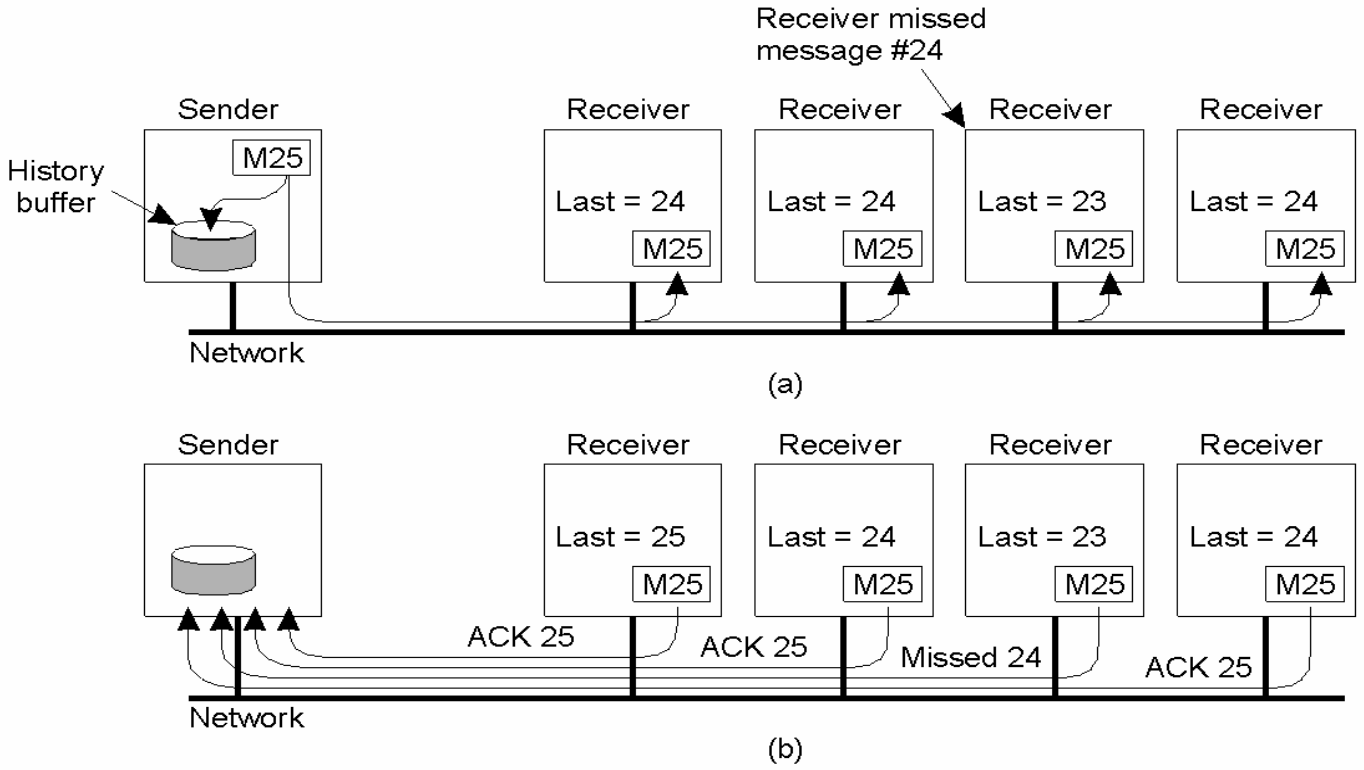
a) Caso normal.
b) Crash luego de ejecución.
c) Crash antes de ejecución.

Crash del Servidor (2)

Diferentes combinaciones de estrategias de cliente y servidor en la presencia de caídas del servidor.

Cliente	Servidor					
	Estrategia M -> P			Estrategia P -> M		
Estrategia de reenvío	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Siempre	DUP	OK	OK	DUP	DUP	OK
Nunca	OK	CERO	CERO	OK	OK	CERO
Solo cuando ACK	DUP	OK	CERO	DUP	OK	CERO
Solo cuando no ACK	OK	CERO	OK	OK	DUP	OK

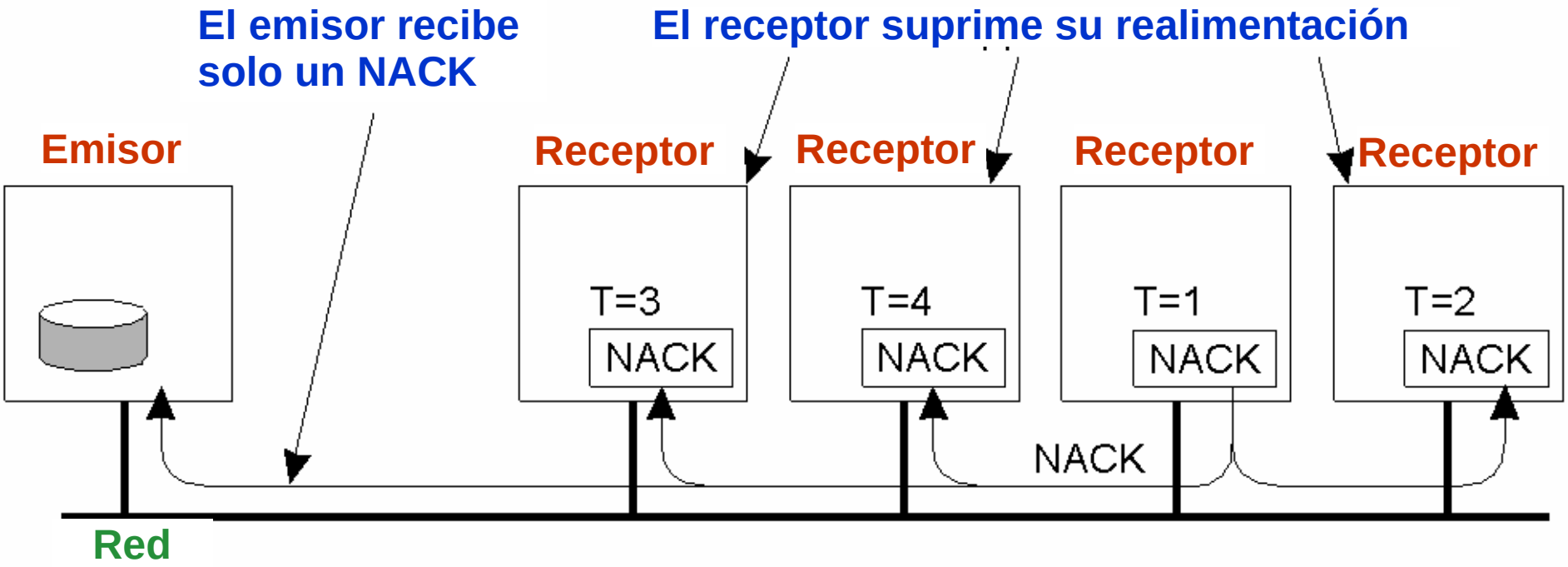
Esquemas de Multicasting Confiable



Una solución simple para el multicasting confiable cuando todos los receptores son conocidos y se suponen sin fallas.

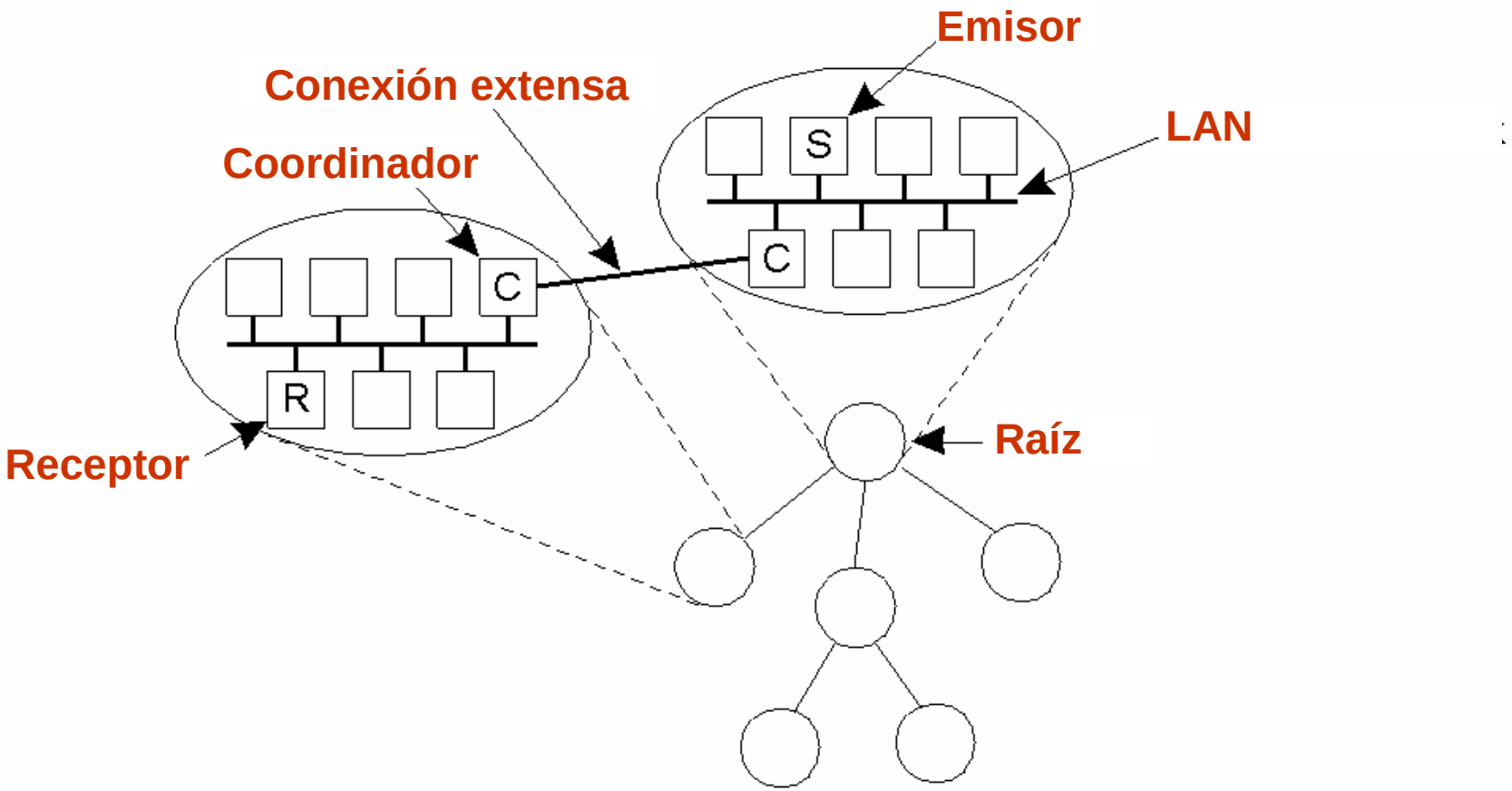
a) Transmisión de mensajes.
b) Reportando (ACK) recepción.

Control Realimentado No Jerárquico



Varios receptores han planificado su requerimiento para retransmisión, pero el primer requerimiento de retransmisión lleva a suprimir el de los otros.

Control Realimentado Jerárquico



Esencia de un multicasting jerárquico confiable.

a) Cada coordinador local reenvía el mensaje a sus hijos.

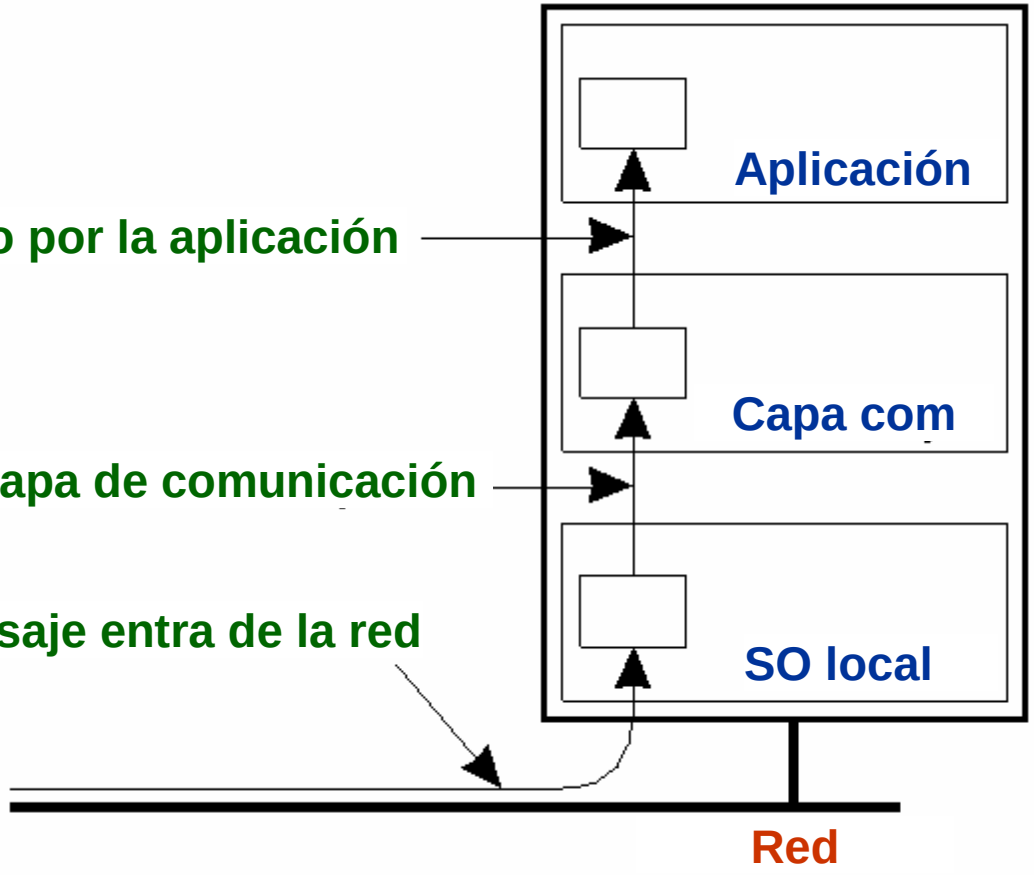
b) Un coordinador local gestiona los requerimientos de retransmisión.

Sincronismo Virtual (1)

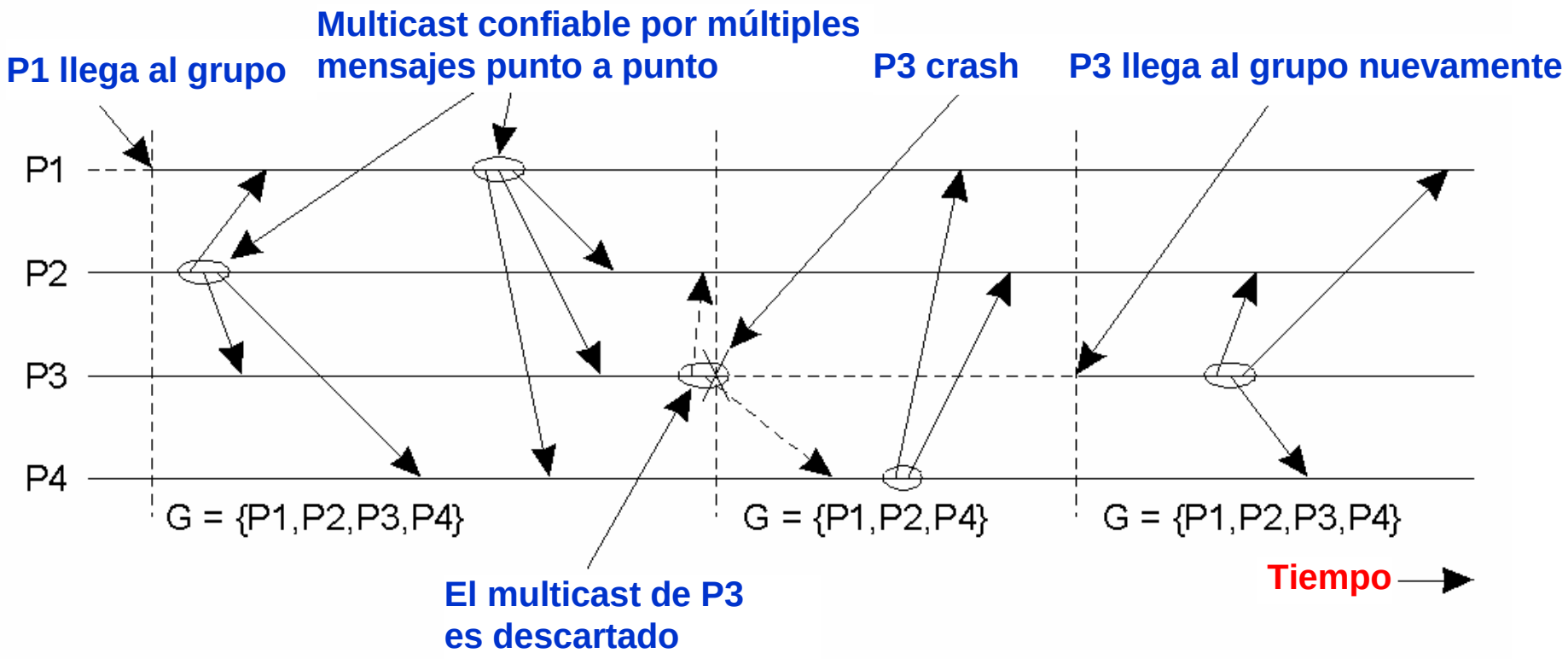
El mensaje es recibido por la aplicación

El mensaje es recibido por la capa de comunicación

El mensaje entra de la red



Sincronismo Virtual (2)



El principio del multicast de sincronismo virtual.

Ordenamiento de Mensajes

Se distinguen cuatro diferentes ordenamientos:

1) Multicast sin orden.
2) Multicast ordenados *First-Input First-Output*.
3) Multicast causalmente ordenados.
4) Multicast totalmente ordenados.

Ordenamiento de Mensajes (1)

Multicast sin orden

Proceso P1	Proceso P2	Proceso P3
send m1	receive m1	receive m2
send m2	receive m2	receive m1

Tres procesos comunicándose en el mismo grupo. El orden de los eventos por proceso es mostrado a lo largo del eje vertical.

Ordenamiento de Mensajes (2)

Multicast ordenados First-Input First-Output

Proceso P1	Proceso P2	Proceso P3	Proceso P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

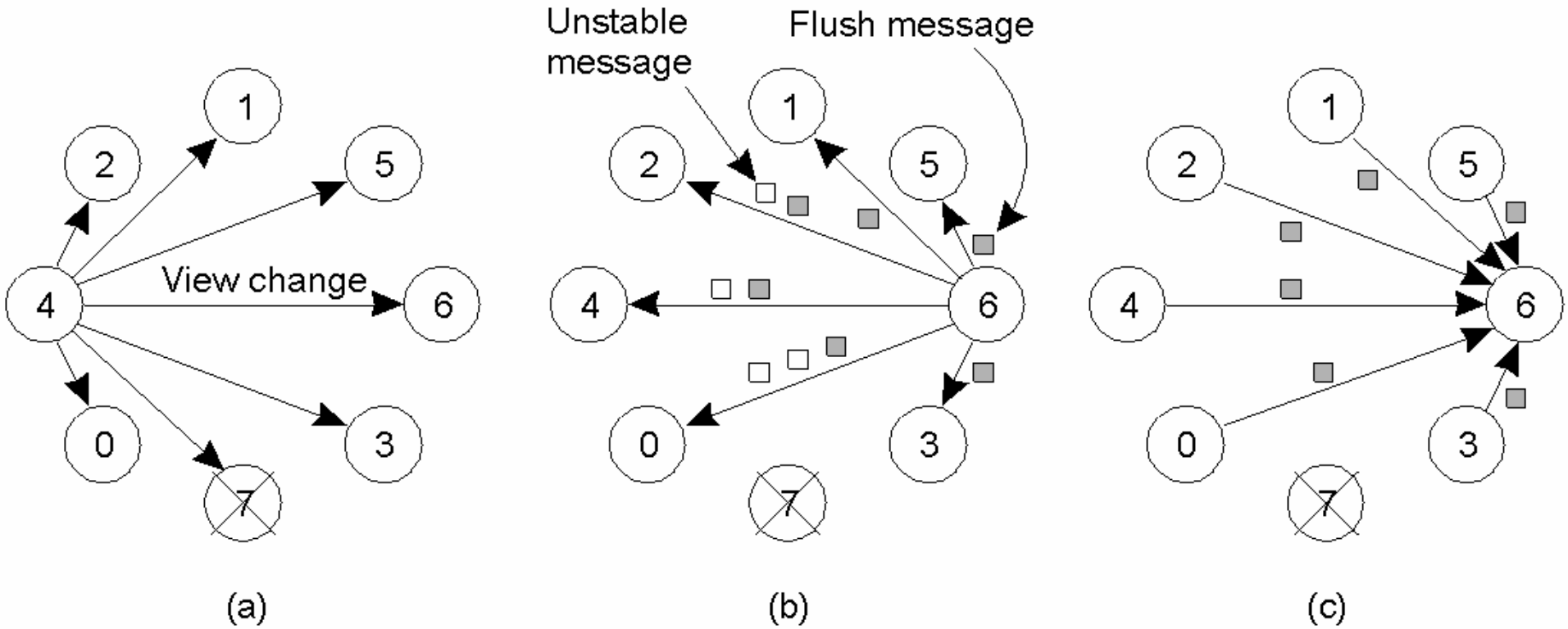
Cuatro procesos en el mismo grupo con dos diferentes emisores y un posible orden de recepción de mensajes bajo un multicast con ordenamiento.

Implementación de Sincronismo Virtual (1)

Seis diferentes versiones de un multicasting confiable virtualmente sincrónico.

Multicast	Orden de Mensajes Básico	¿Recepción totalmente ordenada?
Multicast confiable	Ninguno	No
Multicast FIFO	Recepción FIFO-ordenada	No
Multicast Causal	Recepción causalmente ordenada	No
Multicast Atómico	Ninguna	Si
Multicast Atómico FIFO	Recepción FIFO-ordenada	Si
Multicast Atómico Causal	Recepción causalmente ordenada	Si

Implementación de Sincronismo Virtual (2)

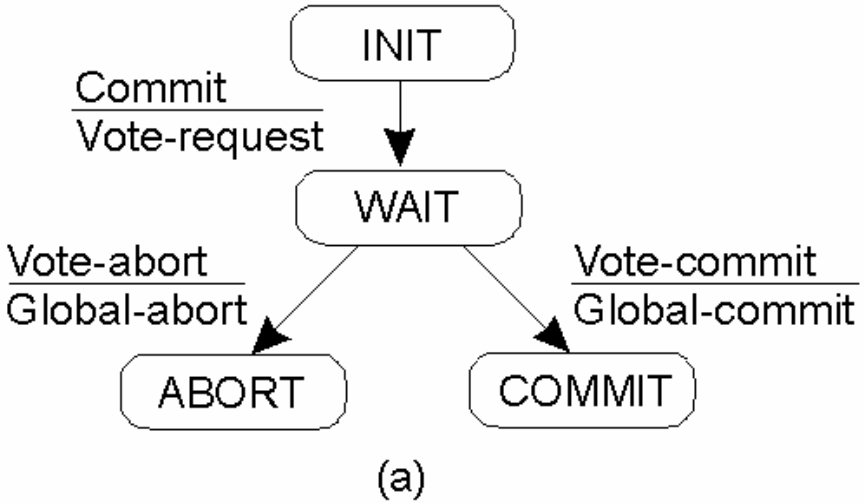


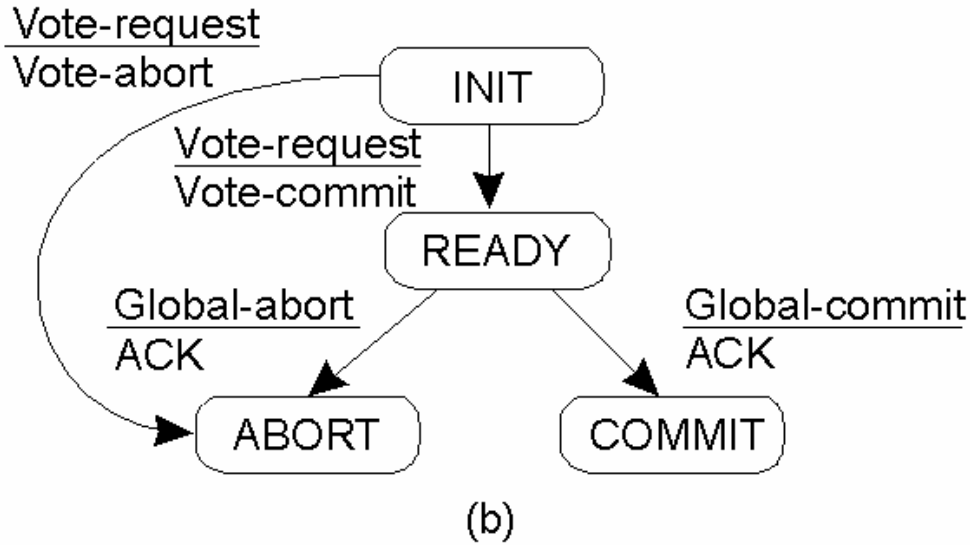
a) El proceso 4 nota que el proceso 7 ha caído, envía un cambio de vista.

b) El proceso 6 emite todos sus mensajes inestables, seguidos de un mensaje flush.

c) El proceso 6 instala la nueva vista cuando ha recibido un mensaje flush de todos los demás.

Commit de Dos Fases (2PC)





a) Máquina de estados finitos para el coordinador en 2PC.

b) Máquina de estados finitos para el participante.

Commit de Dos Fases (2)

Estado de Q	Acción P
COMMIT	Hace la transición a COMMIT
ABORT	Hace la transición a ABORT
INIT	Hace la transición a ABORT
READY	Contacte a otro participante

Acciones tomadas por el participante *P* cuando está en estado *READY* y tiene contactado a otro participante *Q*.

Commit de Dos Fases (3)

actions by coordinator:

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        while GLOBAL_ABORT to local log;
        multicast  GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
    write GLOBAL_COMMIT to local log;
    multicast GLOBAL_COMMIT to all participants;
} else {
    write GLOBAL_ABORT  to local log;
    multicast GLOBAL_ABORT to all participants;
}
```

Pasos seguidos por el coordinador en el protocolo de commit de dos fases.

Commit de Dos Fases (4)

Pasos seguidos por el participante en 2PC.

actions by participant:

```
write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
    write VOTE_ABORT to local log;
    exit;
}
if participant votes COMMIT {
    write VOTE_COMMIT to local log;
    send VOTE_COMMIT to coordinator;
    wait for DECISION from coordinator;
    if timeout {
        multicast DECISION_REQUEST to other participants;
        wait until DECISION is received; /* remain blocked */
        write DECISION to local log;
    }
    if DECISION == GLOBAL_COMMIT
        write GLOBAL_COMMIT to local log;
    else if DECISION == GLOBAL_ABORT
        write GLOBAL_ABORT to local log;
} else {
    write VOTE_ABORT to local log;
    send  VOTE ABORT to coordinator;
}
```

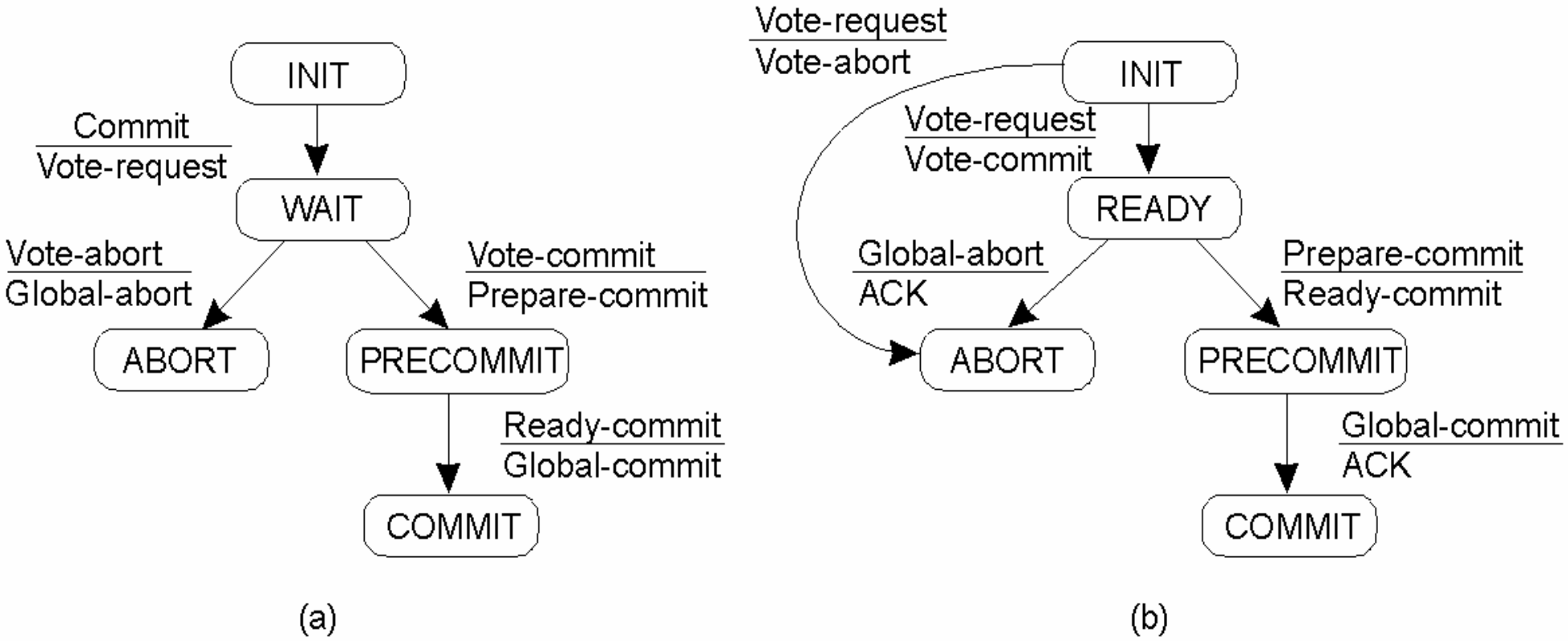
Commit de Dos Fases (5)

actions for handling decision requests: /* executed by separate thread */

```
while true {
    wait until any incoming DECISION_REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip;  /* participant remains blocked */
```

Pasos seguidos para gestionar requerimientos de decisión entrantes.

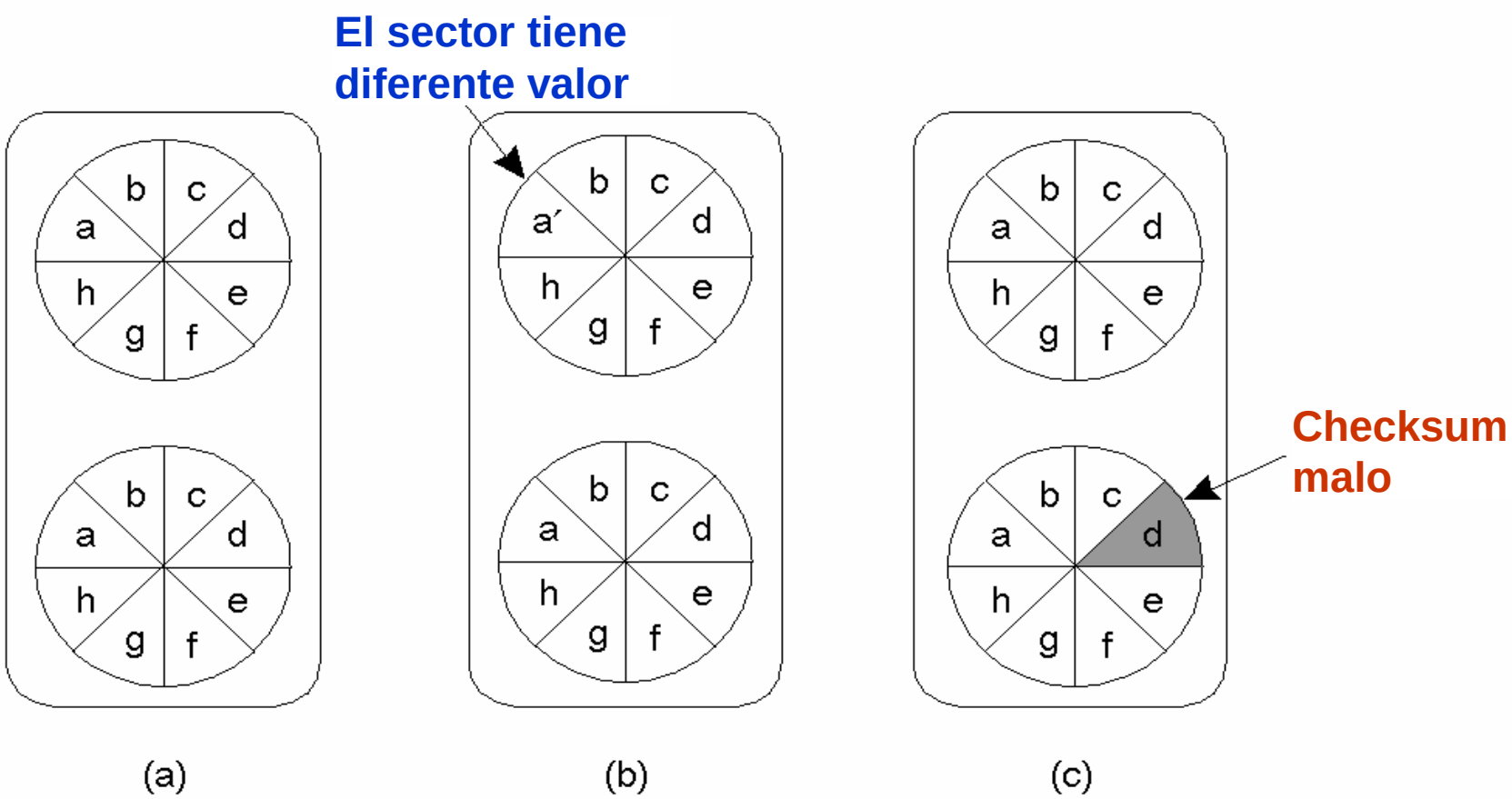
Commit de Tres Fases (3PC)



a) Máquina de estados finitos para el coordinador en 3PC.

b) Máquina de estados finitos para el participante.

Recuperación con Almacenamiento Estable



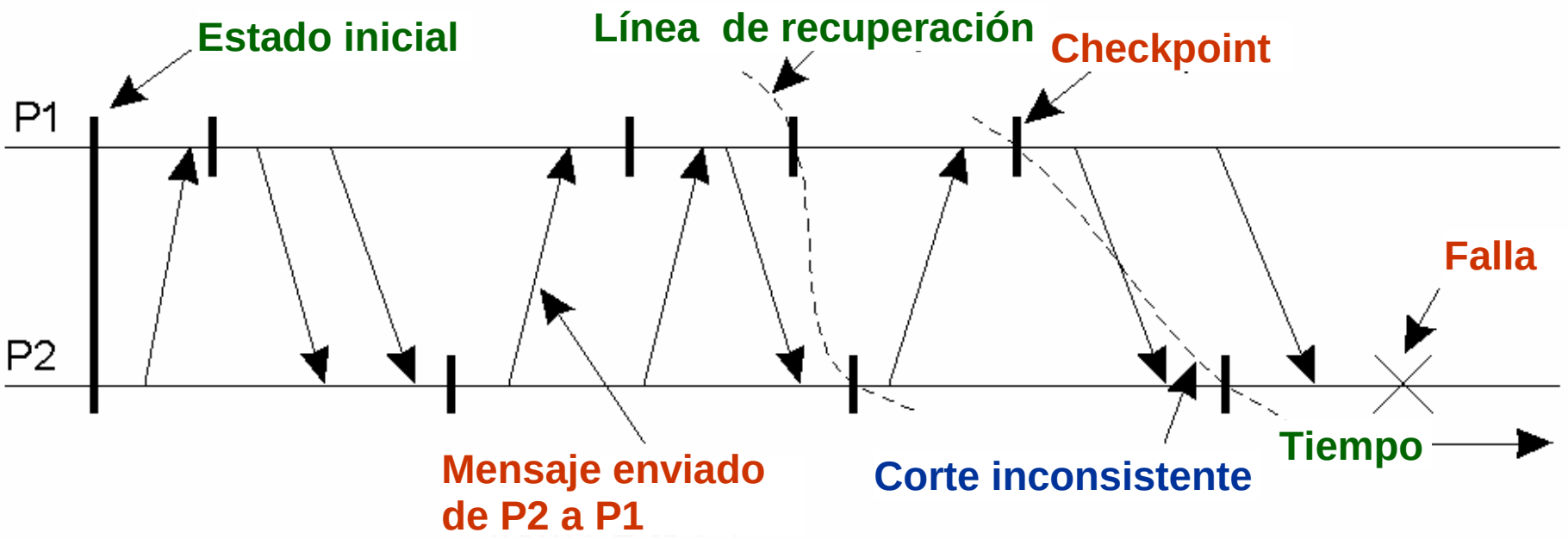
a) Almacenamiento estable.

b) Crash después que el drive 1 es actualizado.

c) Falla el *checksum.*

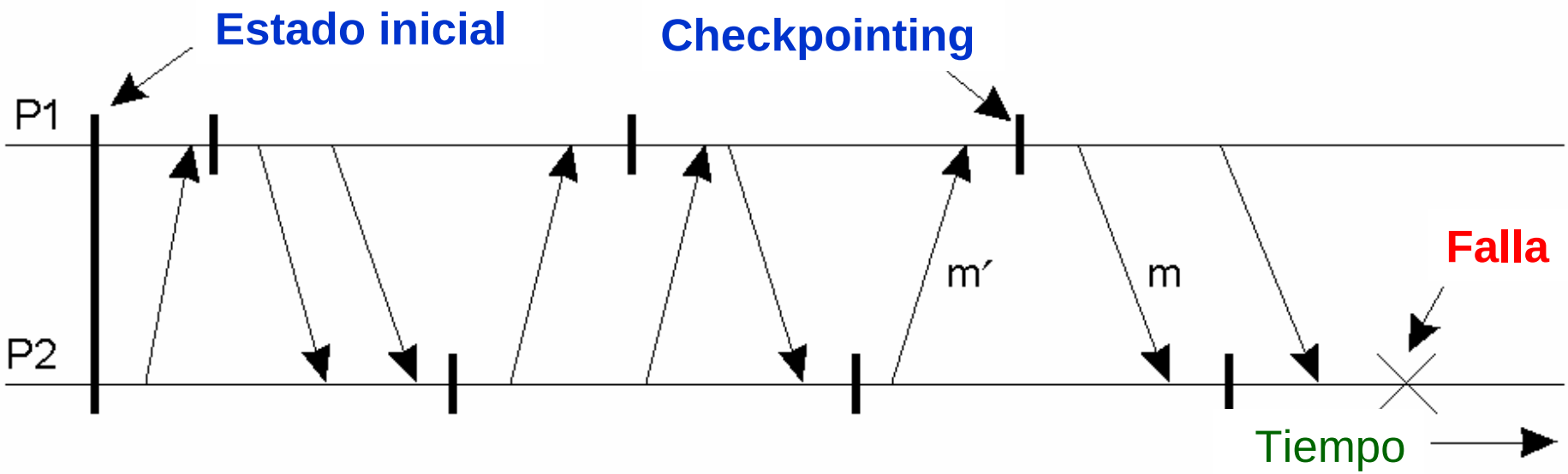
Checkpointing

Una línea de recuperación.



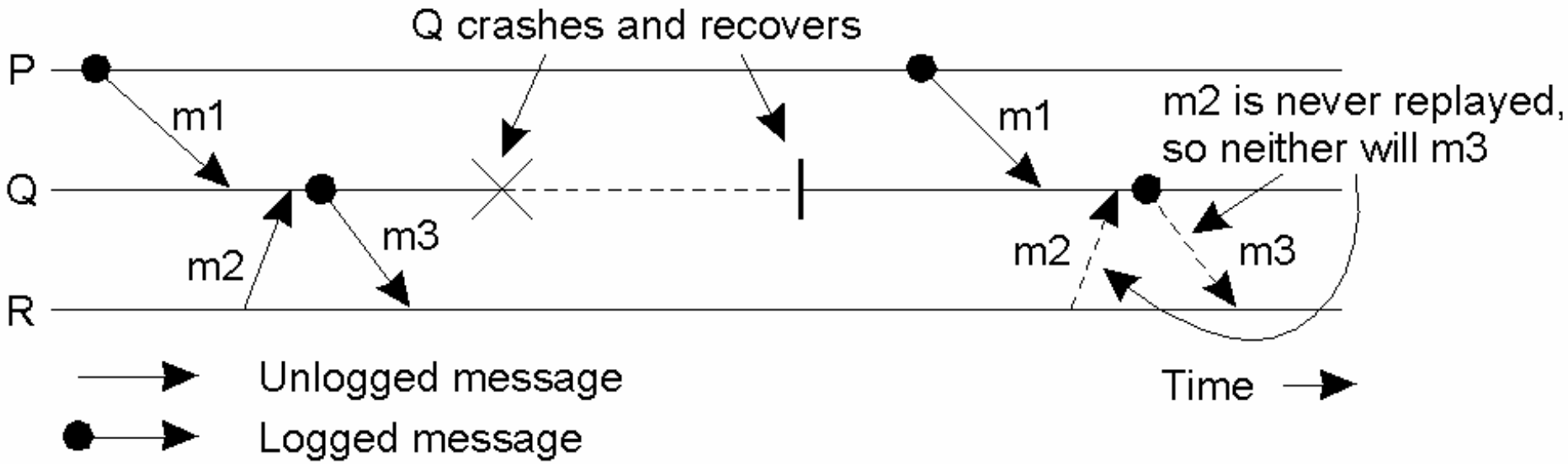
Checkpointing Independiente

El efecto dominó.



Logging de Mensajes

Respuestas incorrectas después de la recuperación, generando un proceso huérfano.



Coming Next