# Resolución al 2do Parcial de EBD 2010

Disclaimer:

*Estas diapositivas tienen el propósito de mostrar una posible resolución al parcial tomado en la materia EBD en 2010. No deberían ser utilizada como único soporte de estudio. Tampoco implican que los exámenes consistirán en el futuro de ejercicios similares. Y no están completamente verificadas de su correctitud, por lo que puede ser que contengan algún error que haya sido detectado en clase o que aún persista.*

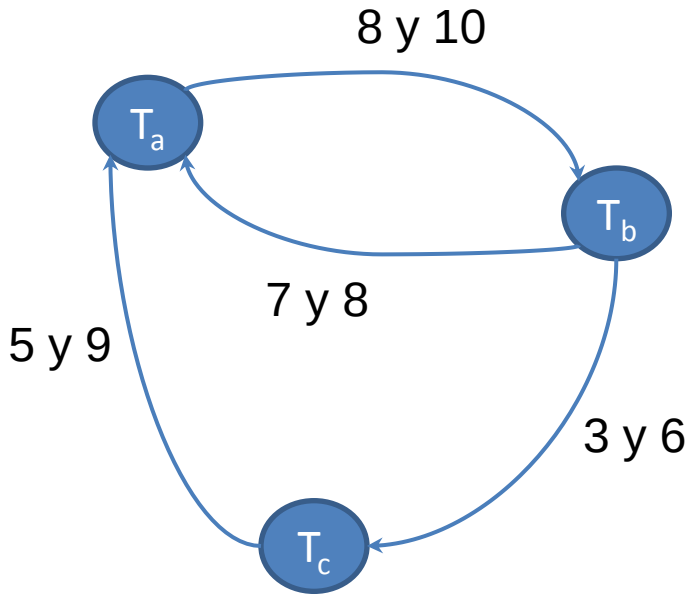
*Resolución 2do Parcial EBD 2010*

***Ejercicio 1:*** Utilizando las siguientes transacciones:

$T_a$: read(X); write(X); read(Z);
$T_b$: read(Y); write(Y); read(X); write(X);
$T_c$: read(Z); write(Z); read(Y); write(Y);

a) Diseñe una planificación concurrente que no sea serializable en conflictos.

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 1 | R(x) | | |
| 2 | | R(y) | |
| 3 | | W(y) | |
| 4 | | | R(z) |
| 5 | | | W(z) |
| 6 | | | R(y) |
| 7 | | R(x) | |
| 8 | W(x) | | |
| 9 | R(z) | | |
| 10 | | W(x) | |
| 11 | | | W(y) |




Como el grafo de precedencia tiene ciclos <Ta,Tb> o <Ta,Tb,Tc> entonces la planificación No es serializable en conflictos

*Resolución 2do Parcial EBD 2010*

***Ejercicio 1:*** Utilizando las siguientes transacciones:

$T_a$: read(X); write(X); read(Z);
$T_b$: read(Y); write(Y); read(X); write(X);
$T_c$: read(Z); write(Z); read(Y); write(Y);

b) Diseñe una planificación concurrente utilizando un protocolo de concurrencia a su elección que resulte en una planificación no recuperable.

i- ¿Cómo podría evitarse dicha situación?

Utilizando un protocolo de bloqueo estricto o riguroso.

ii- ¿La solución propuesta en (b – i) resuelve también el problema de los retrocesos en cascada? Justifique.

Si, ya que ninguna transacción puede depender de un valor escrito por otra transacción activa.

# *Resolución 2do Parcial EBD 2010*

***Ejercicio 1:*** Utilizando las siguientes transacciones:

$T_a$: read(X); write(X); read(Z);
$T_b$: read(Y); write(Y); read(X); write(X);
$T_c$: read(Z); write(Z); read(Y); write(Y);

b) Diseñe una planificación concurrente utilizando un protocolo de concurrencia a su elección que resulte en una planificación no recuperable.

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 1 | R(x) | | |
| 2 | W(x) | | |
| 3 | | R(y) | |
| 4 | | W(y) | |
| 5 | | | R(z) |
| 6 | | | W(z) |
| 7 | | | R(y) |
| 8 | | | W(y) |
| 9 | | | commit |
| 10 | | R(x) | |
| 11 | R(z) | | |
| 12 | | W(x) | |

En 11 Ta falla porque la lectura de Z llega tarde y por lo tanto Ta retrocede y hace retroceder en cascada a Tb por r(x) en 10 y debería retroceder también Tc por 7 pero Tc está en estado de commit por lo cual es No recuperable.

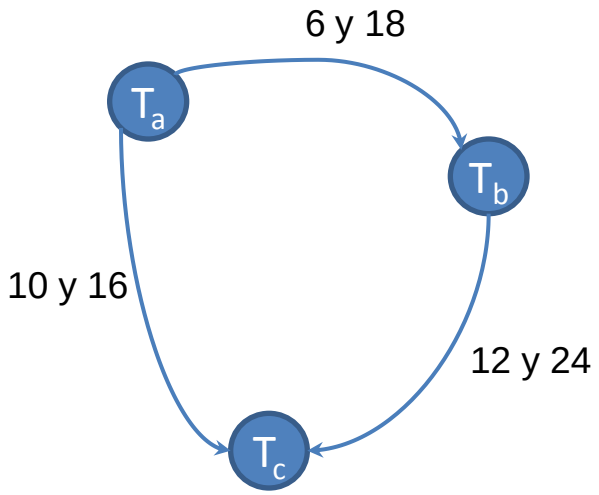
# *Resolución 2do Parcial EBD 2010*

***Ejercicio 1:*** Utilizando las siguientes transacciones:
$T_a$: read(X); write(X); read(Z);
$T_b$: read(Y); write(Y); read(X); write(X);
$T_c$: read(Z); write(Z); read(Y); write(Y);
c) Diseñe una planificación concurrente que resulte de aplicar el *protocolo bloqueo de 2 fases* con posibilidad de *upgrade* y *downgrade (PB2F Refinado)* pero que dicha planificación no sea válida bajo PB2F. ¿La planificación es serializable? Justifique. ¿Cuál es la serie equivalente?

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 1 | L-s(x) | | |
| 2 | R(x) | | |
| 3 | | L-s(y) | |
| 4 | | R(y) | |
| 5 | Up(x) | | |
| 6 | W(x) | | |
| 7 | | | L-s(z) |
| 8 | | | R(z) |
| 9 | L-s(z) | | |
| 10 | R(z) | | |
| 11 | | Up(y) | |
| 12 | | W(y) | |

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 13 | U(x) | | |
| 14 | U(z) | | |
| 15 | | | Up(z) |
| 16 | | | W(z) |
| 17 | | L-s(x) | |
| 18 | | R(x) | |
| 19 | | Up(x) | |
| 20 | | W(x) | |
| 21 | | U(x) | |
| 22 | | U(y) | |
| 23 | | | L-s(y) |
| 24 | | | R(y) |

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 25 | | | Up(y) |
| 26 | | | W(y) |
| 27 | | | U(y) |
| 28 | | | U(z) |

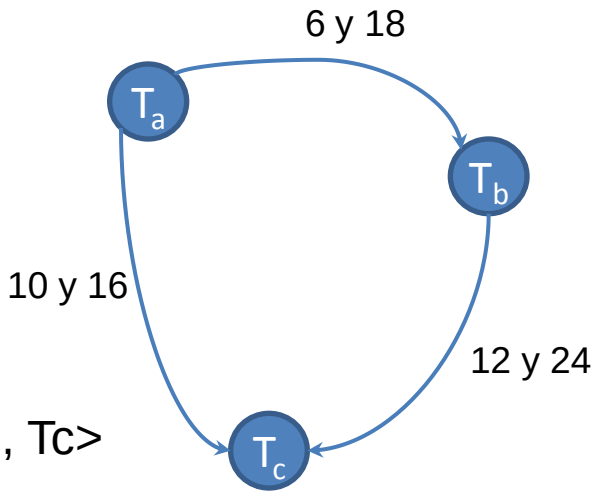


# *Resolución 2do Parcial EBD 2010*

***Ejercicio 1:*** Utilizando las siguientes transacciones:
$T_a$: read(X); write(X); read(Z);
$T_b$: read(Y); write(Y); read(X); write(X);
$T_c$: read(Z); write(Z); read(Y); write(Y);
c) Diseñe una planificación concurrente que resulte de aplicar el *protocolo bloqueo de 2 fases* con posibilidad de *upgrade* y *downgrade (PB2F Refinado)* pero que dicha planificación no sea válida bajo PB2F. ¿La planificación es serializable? Justifique. ¿Cuál es la serie equivalente?

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| … | | | |
| 7 | | | L-s(z) |
| 8 | | | R(z) |
| 9 | L-s(z) | | |
| 10 | R(z) | | |
| … | | | |
| 14 | U(z) | | |
| 15 | | | Up(z) |
| 16 | | | W(z) |

La planificación no sería válida bajo un PB2F porque Tc necesitaría bloquear el dato Z en forma exclusiva por lo cual Ta no podría obtener un bloqueo para lectura




Es serializable en conflicto y la serie equiv. es <Ta, Tb, Tc>

# *Resolución 2do Parcial EBD 2010*

***Ejercicio 1:*** Utilizando las siguientes transacciones:
$T_a$: read(X); write(X); read(Z);
$T_b$: read(Y); write(Y); read(X); write(X);
$T_c$: read(Z); write(Z); read(Y); write(Y);
Diseñe una planificación concurrente siguiendo el protocolo de árbol que utilice el siguiente esquema:
Padre(V,W), Padre(V,Z), Padre(W,X), Padre(W,Y).

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 1 | | | Lock(v) |
| 2 | | | Lock(z) |
| 3 | | | R(z) |
| 4 | | Lock(w) | |
| 5 | | Lock(y) | |
| 6 | | R(y) | |
| 7 | | W(y) | |
| 8 | | U(y) | |
| 9 | | Lock(x) | |
| 10 | | U(w) | |
| 11 | | | W(z) |
| 12 | | | U(z) |

| t | $24T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 13 | | | Lock(w) |
| 14 | | | U(v) |
| 15 | Lock(v) | | |
| 16 | | | Lock(y) |
| 17 | | | U(w) |
| 18 | Lock(w) | | |
| 19 | | R(x) | |
| 20 | | W(x) | |
| 21 | | U(x) | |
| 22 | Lock(x) | | |
| 23 | U(w) | | |
| 24 | R(x) | | |

| t | $T_a$ | $T_b$ | $T_c$ |
|---|---|---|---|
| 25 | W(x) | | |
| 26 | | | R(y) |
| 27 | | | W(y) |
| 28 | U(x) | | |
| 29 | Lock(z) | | |
| 30 | U(v) | | |
| 31 | R(z) | | |
| 32 | U(z) | | |
| 33 | | | U(y) |

# Ejercicio 2

- Considere la siguiente planificación suponiendo que: $0 < ts(T1) < ts(T2) < ts(T3) < ts(T4)$
- Simule la aplicación del *protocolo de Multiversión*, indicando para cada instante de tiempo el valor de la estampilla de lectura (R-ts) y escritura (W-ts) de la versión correspondiente. Suponga que existen:
  - <A0,R-ts=0,W-ts=0>,
  - <B0,R-ts=0,W-ts=0>,
  - <C0,R-ts=0,W-ts=0>.
- Si se produce un retroceso por la violación del protocolo indique en que puntos se produce y que acciones se realizan para recuperarse. Justifique.

| T | $T_1$ | $T_2$ | $T_3$ | $T_4$ | ACCION | Versión | R-Ts | w-Ts |
|---|---|---|---|---|---|---|---|---|
| 1 | | read(B) | | | Lee con éxito | B0 | T2 | 0 |
| 2 | read(A) | | | | Lee con éxito | A0 | T1 | 0 |
| 3 | | | read(A) | | Lee con éxito | A0 | T3 | 0 |
| 4 | | write(B) | | | Crea una versión con éxito | B1 | T2 | T2 |
| 5 | | | write(B) | | Crea una versión con éxito | B2 | T3 | T3 |
| 6 | write(C) | | | | Crea una versión con éxito | C1 | T1 | T1 |
| 7 | | | | write(C) | Crea una versión con éxito | C2 | T4 | T4 |
| 8 | | | | read(B) | Lee con éxito | B2 | T4 | T3 |
| 9 | | write(B) | | | Sobre escribe su version | B1 | T2 | T2 |
| 10 | | | read(C) | | Lee con éxito | C1 | T3 | T1 |
| 11 | write(A) | | | | Falla t1 por 3<br>Retroce en cascada t3<br>Cascada t4 | | | |

- ***Falla T1 y retroceden en cascada T3 y T4***
- ***La única transacción que terminaría es T2***
- ***Desaparece la versión C1***
- ***Desaparece la versión B2***
- ***Desaparece la versión C2***
- ***Se actualiza A0 con tiempo de lectura 0***

- ¿Es una planificación válida bajo el protocolo de estampilla de tiempo con regla de reescritura de Thomas? Justifique.

| T | $T_1$ | $T_2$ | $T_3$ | $T_4$ | ACCION | Dato | R-Ts | W-Ts |
|---|---|---|---|---|---|---|---|---|
| 1 | | read(B) | | | Lee con éxito | B | T2 | -00 |
| 2 | read(A) | | | | Lee con éxito | A | T1 | -00 |
| 3 | | | read(A) | | Lee con éxito | A | T3 | -00 |
| 4 | | write(B) | | | Escribe con éxito | B | T2 | T2 |
| 5 | | | write(B) | | Escribe con éxito | B | T2 | T3 |
| 6 | write(C) | | | | Escribe con éxito | C | -00 | T1 |
| 7 | | | | write(C) | | C | -00 | T4 |
| 8 | | | | read(B) | | B | T4 | T3 |
| 9 | | write(B) | | | No se ignora la escritura falla t2 por 8 | | | |
| 10 | | | read(C) | | Falla t3 por 7 Falla t4 en casdada | | | |
| 11 | write(A) | | | | | | | |

# *Resolución 2do Parcial EBD 2010*

***Ejercicio 3:***

a) Suponga que se utiliza un esquema de **modificación inmediata** y se dispone de la siguiente bitácora:

1. <T1, starts>
2. <T1, X, 1 , 5>
3. <T2, starts>
4. <T2, B, 5, 10>
5. <T1, Y, -1, *#valor1#* >
6. <T2, B, *#valor2#* , *#valor3#* >
7. <T3, starts>
8. <T2, commits>
9. <T3, A, -10, 10>
10. <checkpoint, [ ]>
11. <T1, commits>
12. <T4, starts>
13. <T4, Y, 0, 15>
14. <T3, B, 10, 20>
15. <T1, commits>
16. Fallo del sistema

i - Describa las acciones que debe realizar el sistema al momento de realizar el checkpoint.

## Puntos de Verificación (Checkpoints)

**Solución**

- Se agrega al sistema de recuperación un procedimiento de **checkpointing** o puntos de verificación, que sigue los pasos:
  1. Grabar en disco (output) todos los registros de bitácora que están en memoria principal.
  2. Grabar en disco (output) los bloques modificados de los registros intermedios (buffer de MP).
  3. Grabar un registro de bitácora <checkpoint> en memoria estable.
- Durante el proceso de checkpointing se *suspenden todas las otras tareas* para realizar sólo *tareas de sistema.*

EBD2010_21 - Lic. Mercedes Vitturini

# *Resolución 2do Parcial EBD 2010*

***Ejercicio 3:***

a) Suponga que se utiliza un esquema de **modificación inmediata** y se dispone de la siguiente bitácora:

1. <T1, starts>
2. <T1, X, 1 , 5>
3. <T2, starts>
4. <T2, B, 5, 10>
5. <T1, Y, -1, *#valor1#* >
6. <T2, B, *#valor2#* , *#valor3#* >
7. <T3, starts>
8. <T2, commits>
9. <T3, A, -10, 10>
10. <checkpoint, [ ]>
11. <T1, commits>
12. <T4, starts>
13. <T4, Y, 0, 15>
14. <T3, B, 10, 20>
15. <T3, commits>
16. Fallo del sistema

ii - Complete el registro de checkpoint (registro 10 de la bitacora).

10. <checkpoint, [ T1, T3 ]>

iii - Indique que valores tienen los campos *#valor1#, #valor2# y #valor3#*

*#valor1# = 0 (ver instantes 5 y 13)*
*#valor2# = 10 (ver instantes 4 y 6)*
*#valor3# = 10 (ver instantes 6 y 14)*

iv – Especifique que acciones se realizan durante la recuperación del fallo de sistema indicando: Que listas se construyen y el contenido de cada una.

Redo-List( [T1,T3] )
Undo-List( [T4] )

| Dato | Valor | Reg. Bitacora | Acción |
|---|---|---|---|
| Y | 0 | 13. <T4, Y, 0, 15> | Undo(T4) |
| B | 20 | 14. <T3, B, 10, 20> | Redo(T3) |

# Checkpoints – Modelo Concurrente

**Caída de Sistema (*Crash*)**

- El proceso de recuperación involucra a todas las transacciones ejecutadas y en ejecución (posiblemente más de una) a partir del último punto de verificación (*checkpoint*).
- El sistema de recuperación recorre la bitácora para construir dos listas:
  - **undo-list**
  - **redo-list,**



# Undo-List y Redo-List

- El recorrido de la bitácora es hacia atrás hasta el primer <*checkpoint*, L>:
  - Por cada registro **<$T_i$, *commit*>**, se agrega $T_i$ a *redo-list*.
  - Por cada registro **<$T_i$, *start*>**, si $T_i$ no está en *redo-list* entonces se agrega a la lista *undo-list*.
- Al llegar al checkpoint se controla la lista ***L*** de transacciones activas (<*checkpoint*, L>)
  - Por cada transacción $T_i$ de ***L*** si $T_i$ no esta incluida en *redo-list* se agrega a *undo-list*.



# Recuperación ante Fallo de Sistema

1. Se vuelve a recorrer la bitácora hacia atrás, realizando **undo** de cada registro que corresponde a una transacción de la lista *undo-list*. El proceso se detiene cuando se encuentra en la bitácora el item <$T_i$, start> de cada transacción $T_i$ en la lista *undo-list*.
2. Se avanza ubicando el más reciente <checkpoint> en la bitácora.
3. Se recorre la bitácora hacia adelante desde el más reciente <checkpoint> y se realiza un **redo** de instrucción que corresponda a una transacción $T_i$ en la lista *redo-list*.



# Recuperación ante Fallos

- Es importante respetar el orden de ejecución de las operaciones para la recuperación.
- Las operaciones UNDO deben realizarse siempre antes que las operaciones REDO.
- Las operaciones de UNDO se realizan recorriendo la bitácora desde abajo hacia arriba, esto es, en orden inverso al que se ejecutaron.
- Las operaciones de REDO se realizan recorriendo la bitácora desde arriba hacia abajo, esto es, en el mismo orden en que se actualizó.

# *Resolución 2do Parcial EBD 2010*

***Ejercicio 3:***

b) En general, una vez realizado un checkpoint, ¿pueden eliminarse (borrar de memoria estable) todos los registros de bitácora anteriores al checkpoint sin comprometer la recuperación de futuras fallas?

No. Al momento de realizar la recuperación podría suceder que se requiera retroceder más allá del punto de checkpoint debido a que es necesario deshacer una transacción activa al Momento de la falla y que habia comenzado su ejecución antes del checkpoint.

c) Mencione al menos dos ventajas del uso de checkpoint.

1. Reducir el tamaño de la bitácora que será utilizada para la recuperación.
2. Agilizar el proceso de recuperación.

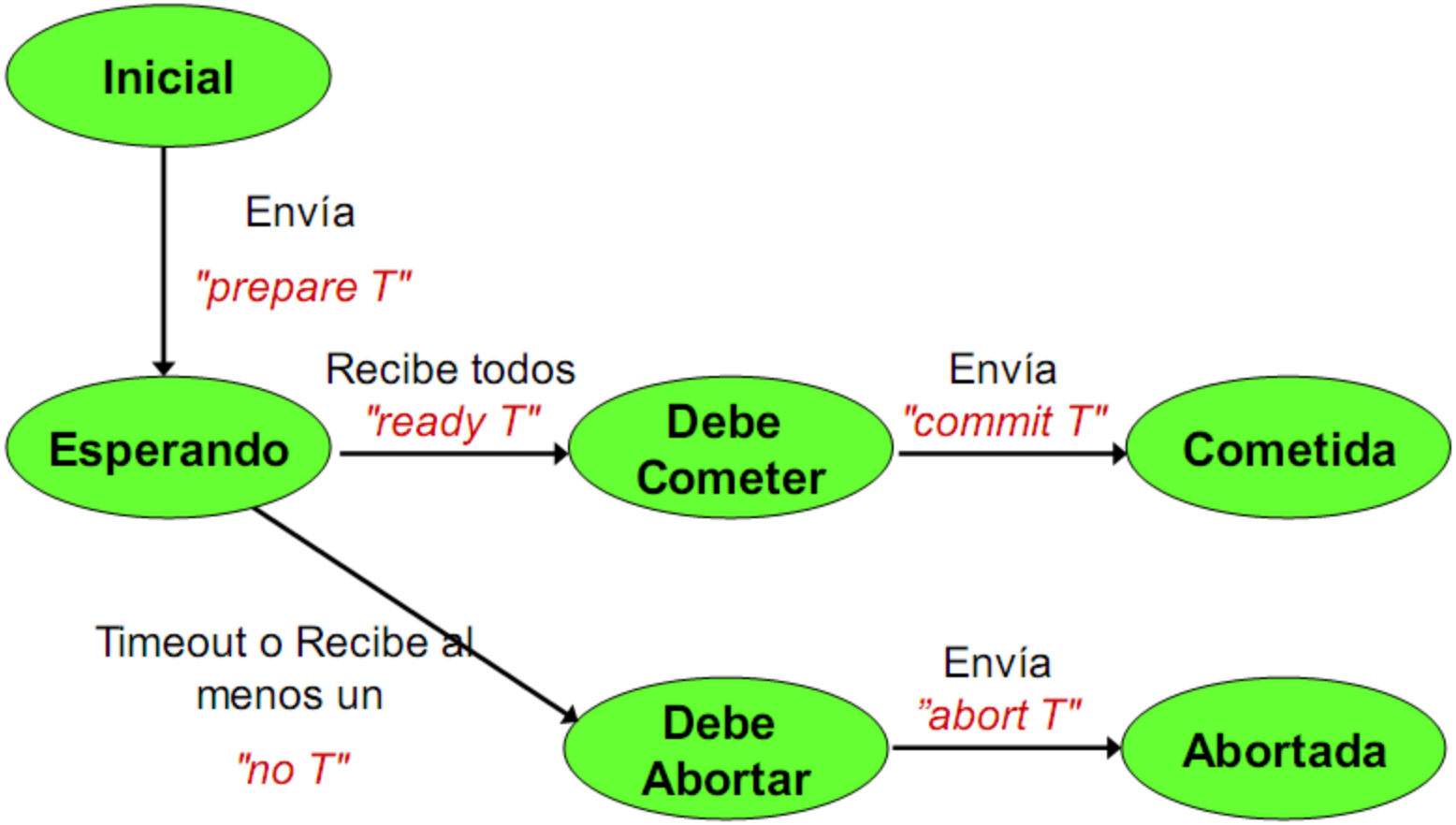
d) Suponga que en un esquema de modificación diferida, una transacción $T_i$ falla antes de cometer y ninguna otra transacción leyó datos generados por $T_i$. ¿Qué acciones se deben realizar para realizar una recuperación exitosa ante la falla de dicha transacción?

No es necesario realizar ninguna acción. Simplemente se ignora la transacción ya que sus modificaciones no han sido efectivas todavia en la BD. Opcionalmente, podría agregarse un registro en la bitácora <Ti, abort>.

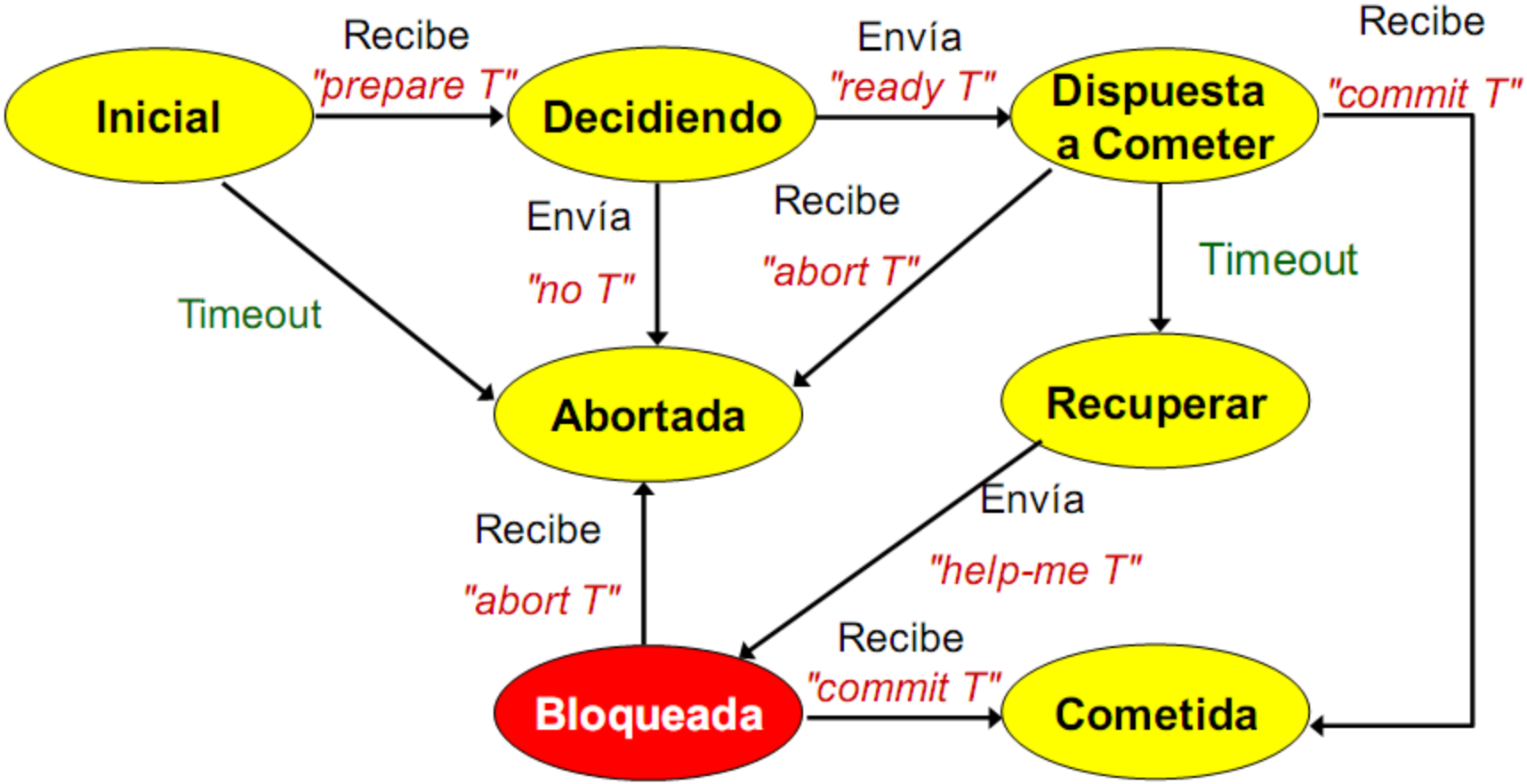
# Segundo Parcial

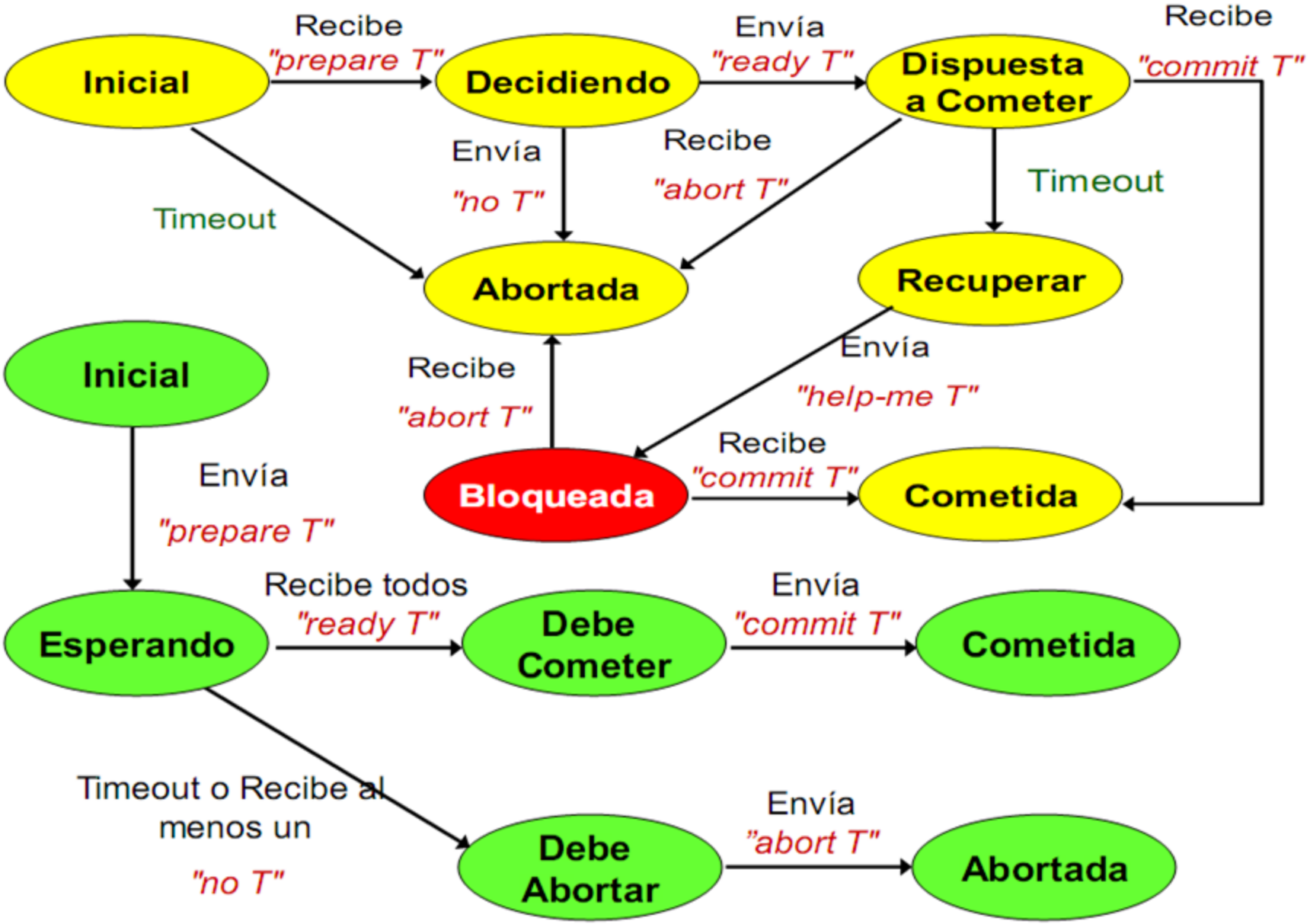
Explicación Ejercicio 4

# 2PC – Diagrama de Estado Coordinador



# 2PC – Diagrama de Estado Participante



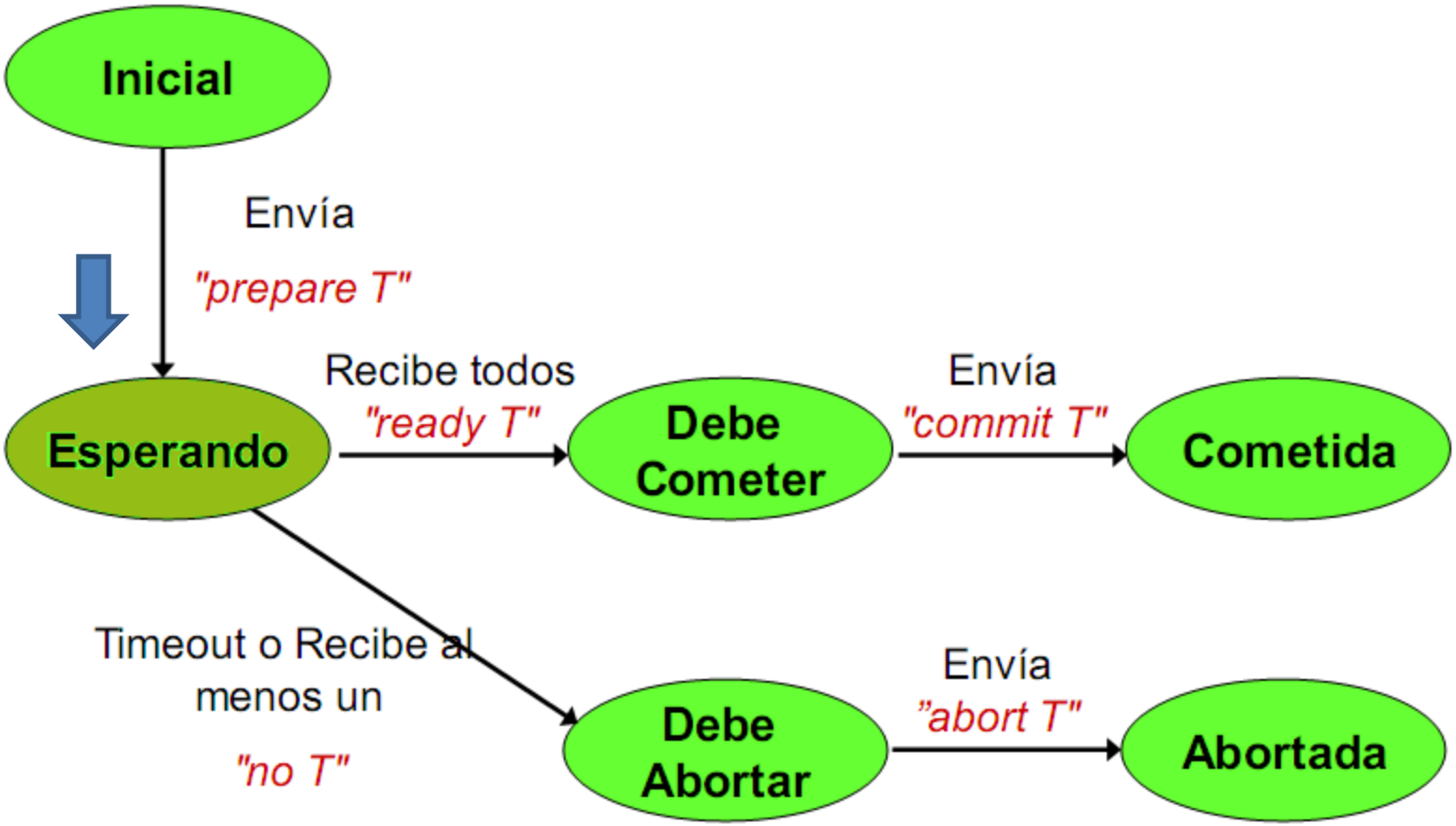

Inicial
Recibe
"prepare T"
Decidiendo
Envía
"ready T"
Dispuesta a Cometer
Recibe
"commit T"
Timeout
Envía
"no T"
Recibe
"abort T"
Timeout
Abortada
Recuperar
Inicial
Recibe
"abort T"
Envía
"help-me T"
Envía
"prepare T"
Bloqueada
Recibe
"commit T"
Cometida
Esperando
Recibe todos
"ready T"
Debe Cometer
Envía
"commit T"
Cometida
Timeout o Recibe al menos un
"no T"
Debe Abortar
Envía
"abort T"
Abortada

# Inciso a) i)

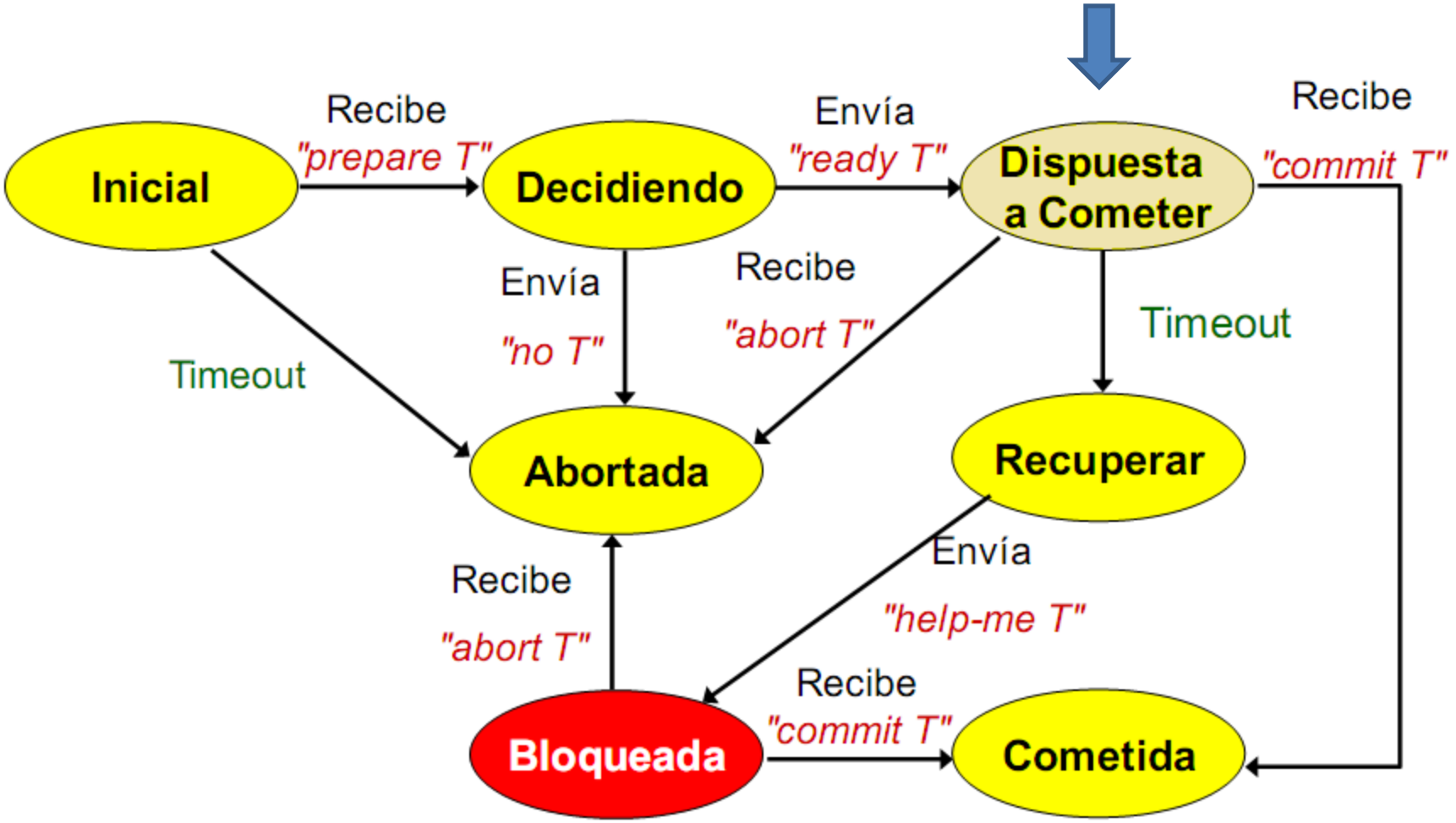
a) Suponga la situación para una transacción distribuida. Existen cinco sitios S1, S2, S3, S4 y S5 y en cuatro de ellos se ejecuta una transacción sincronizada bajo el protocolo de compromiso de dos fases. Se produce una falla del coordinador, explicar que acciones se llevarán adelante. Considere que el coordinador podría recuperarse inmediatamente o no recuperarse por un tiempo prolongado. Analizar cada una de estas dos situaciones.

   i) Cuando 3 de los 4 participantes tienen como último registro en su bitácora <ready T> y el participante restante no tiene dicho registro.

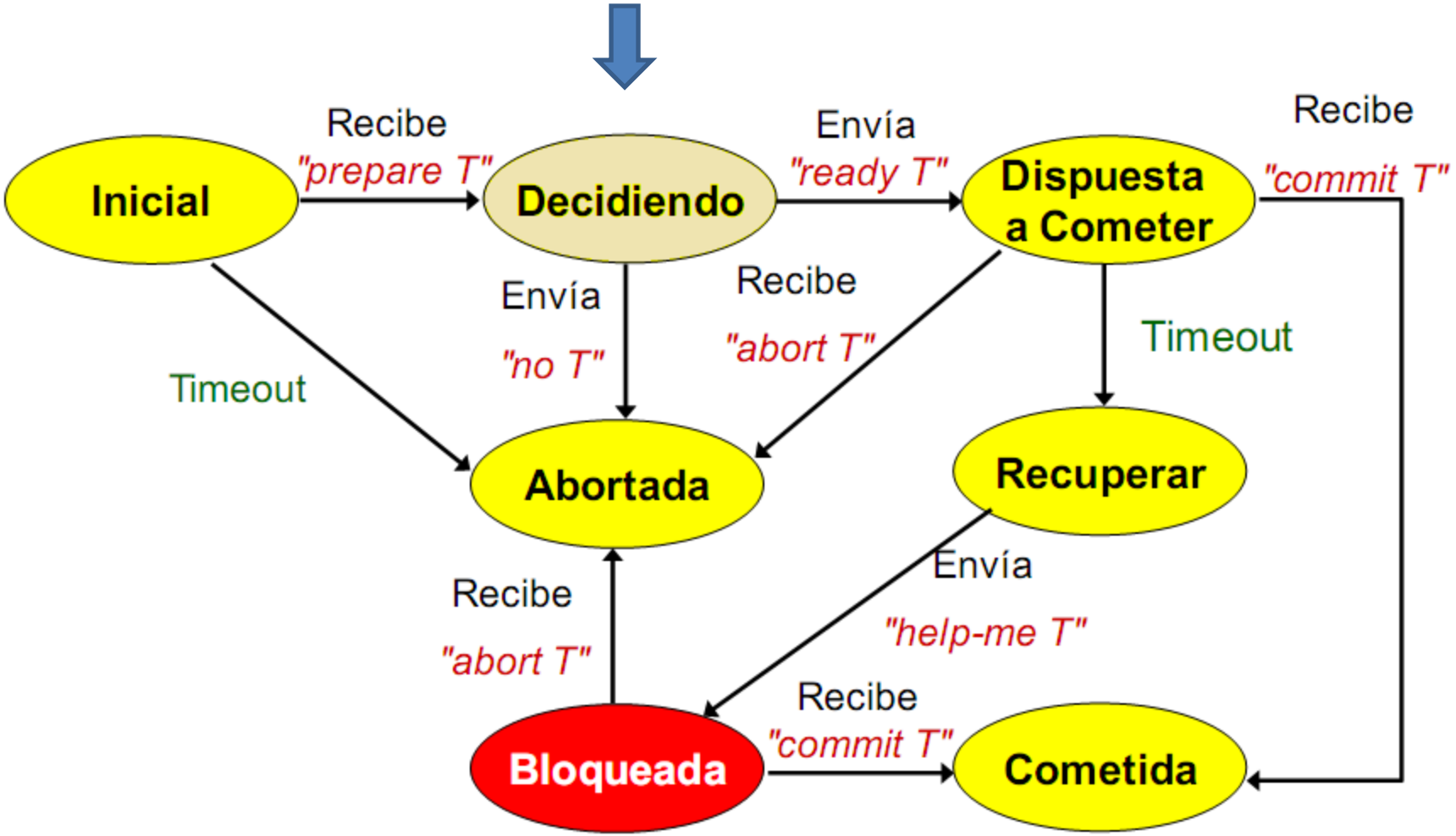
El coordinador recibió 3 “ready T”  y esta en el estado “Esperando”`cuando falla



Los participantes que enviaron “ready T” estan en el estado “Dispuesta a Cometer”



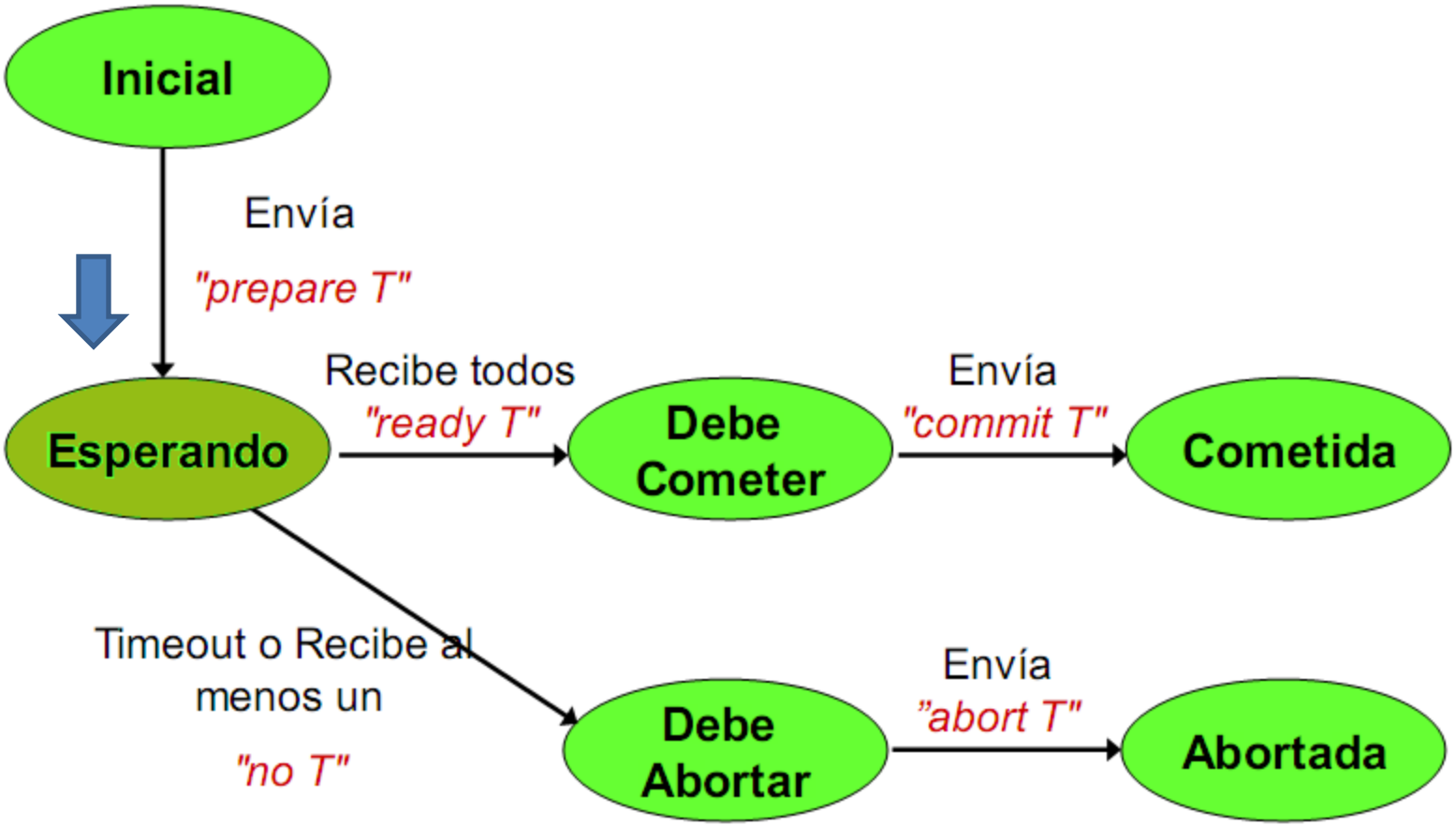
El participante que no envio “ready T”, ni “no T” esta en el estado “Decidiendo”



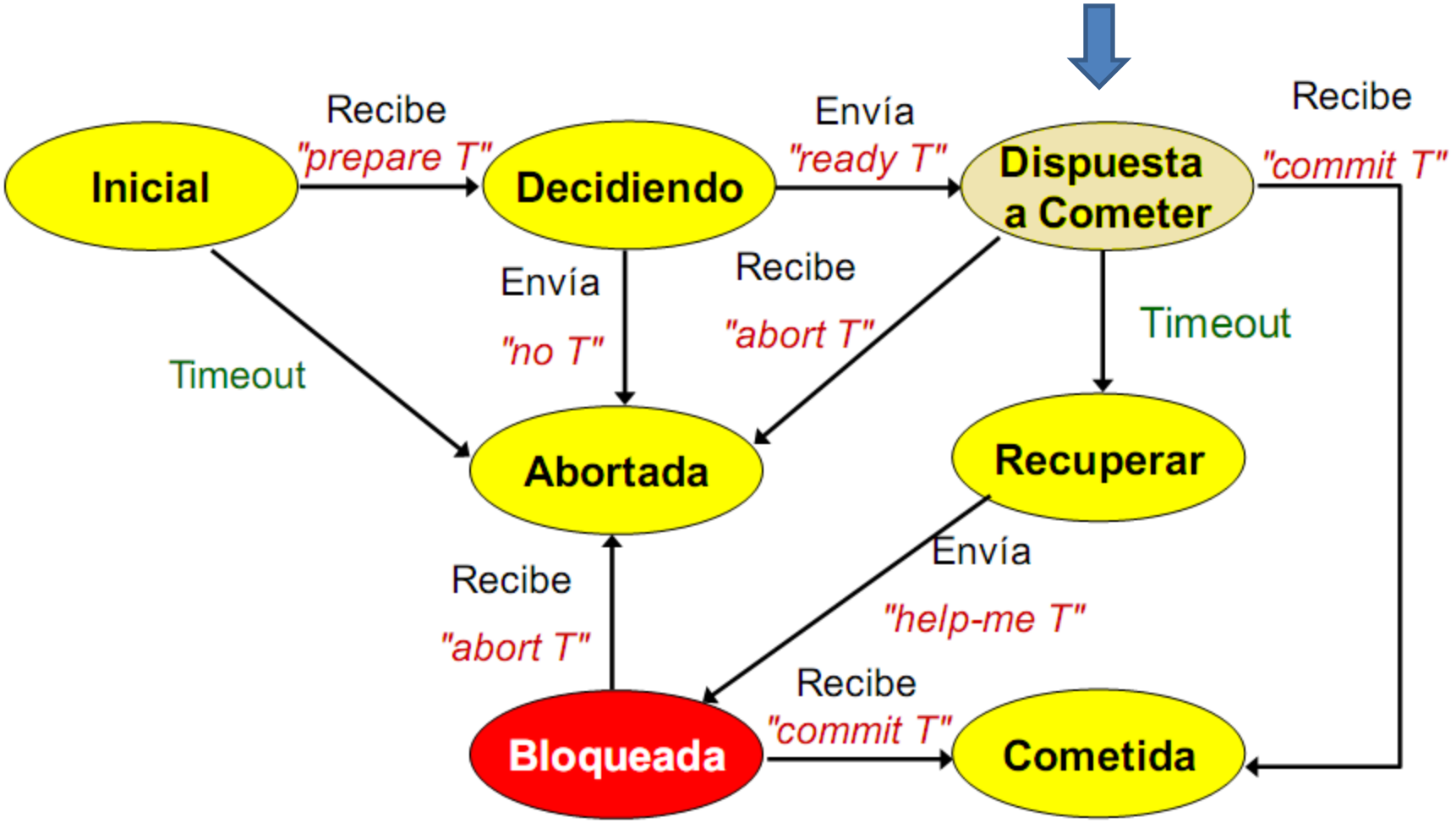
# Inciso a) ii)

ii) Cuando todos los participantes tienen como último registro en su bitácora <ready T>. ¿Se modifican las acciones a realizar si el coordinador es uno de los participantes? Justifique.

El coordinador recibió 4 “ready T”  y esta en el estado “Esperando” cuando falla



Los participantes enviaron “ready T” y estan en el estado “Dispuesta a Cometer”

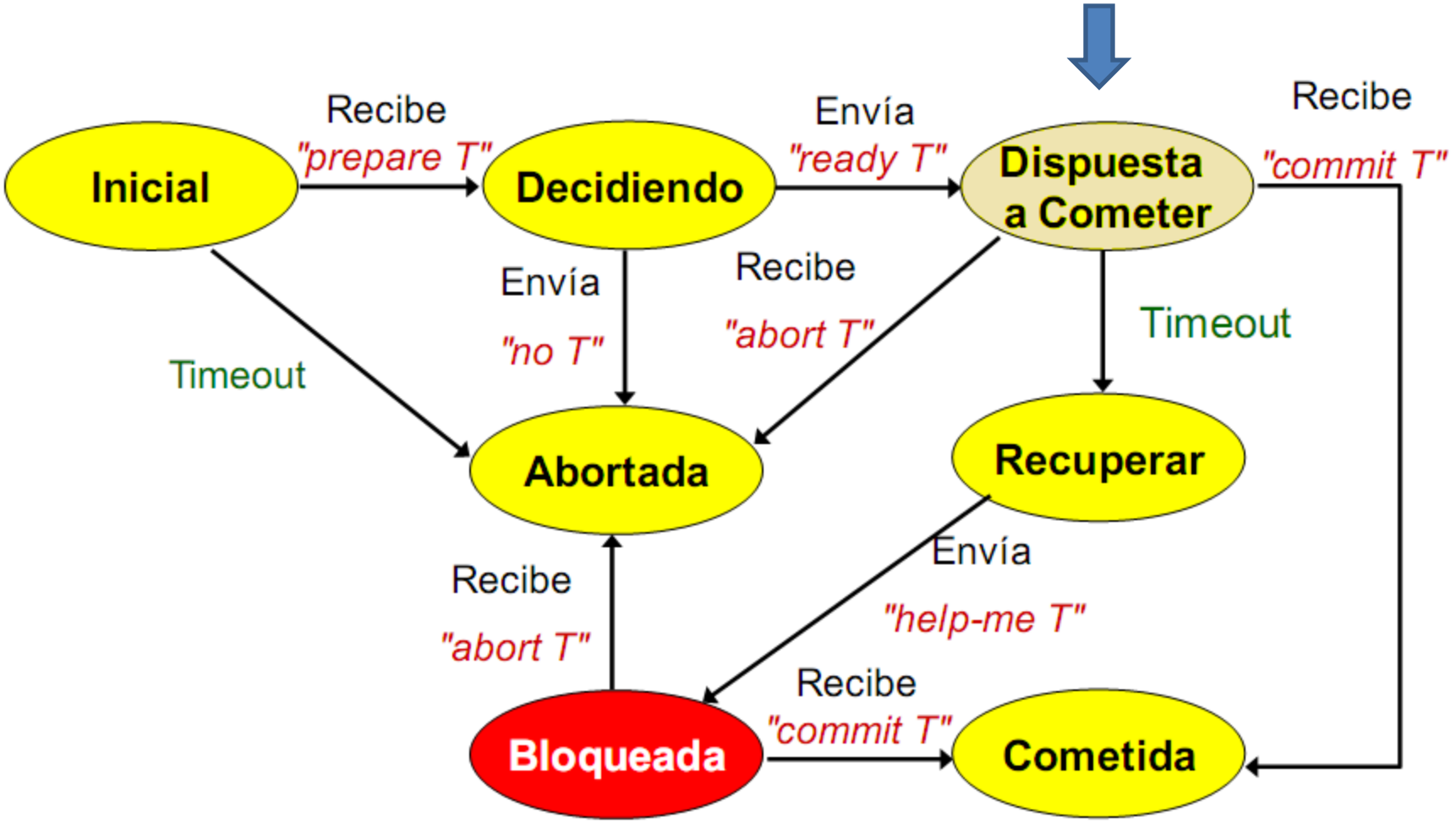


Si el coordinador es también un participante …

- Si suponemos que se recupera de inmediato, es igual a lo anterior.

- Si no se recupera de inmediato, el resto de los participantes no pueden decidir nada entre ellos y quedan bloqueados esperando al coordinador/participante.

Los participantes que enviaron “ready T” estan en el estado “Dispuesta a Cometer”



# Inciso b)

b) Por qué razón los participantes bajo el protocolo de compromiso de dos fases están obligados a escribir en su bitácora <ready T> antes de enviar al coordinador el mensaje “ready T”?. Para justificar su respuesta utilice un ejemplo que ilustre un escenario de falla mostrando que sucede si un participante no sigue el orden previsto (primero escribir en bitácora y luego enviar mensaje) y que sucede en caso contrario.

El participante envió “ready T” pero no guardo el registro <ready T> en bitácora.