



# **Bases de datos distribuidas**



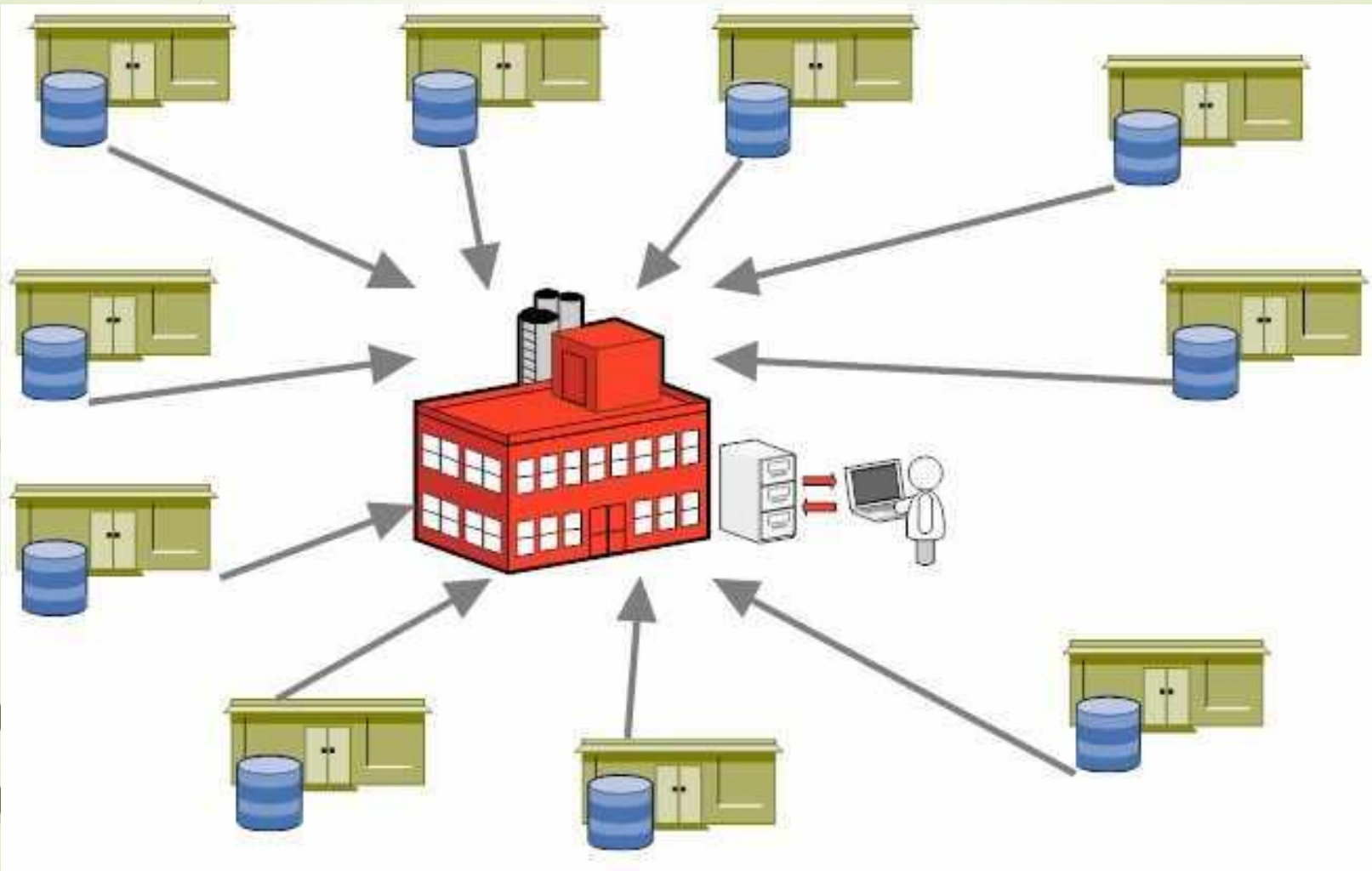
# Bases de datos distribuidas

- Una base de datos distribuida es una colección de múltiples bases de datos lógicas interrelacionadas y distribuidas a través de una red informática.
- SGBD distribuido es el sistema de software que permite la gestión de las BD distribuidas y hace que la distribución sea transparente para los usuarios.

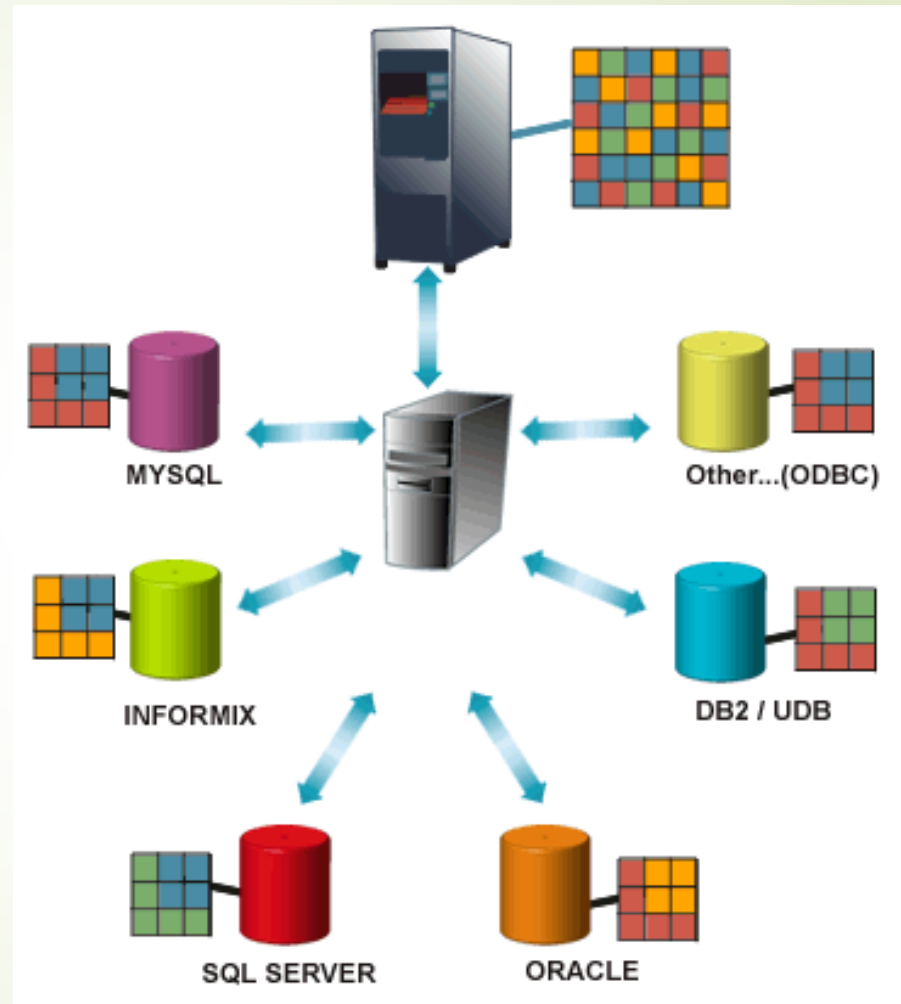
# Base de datos centralizada



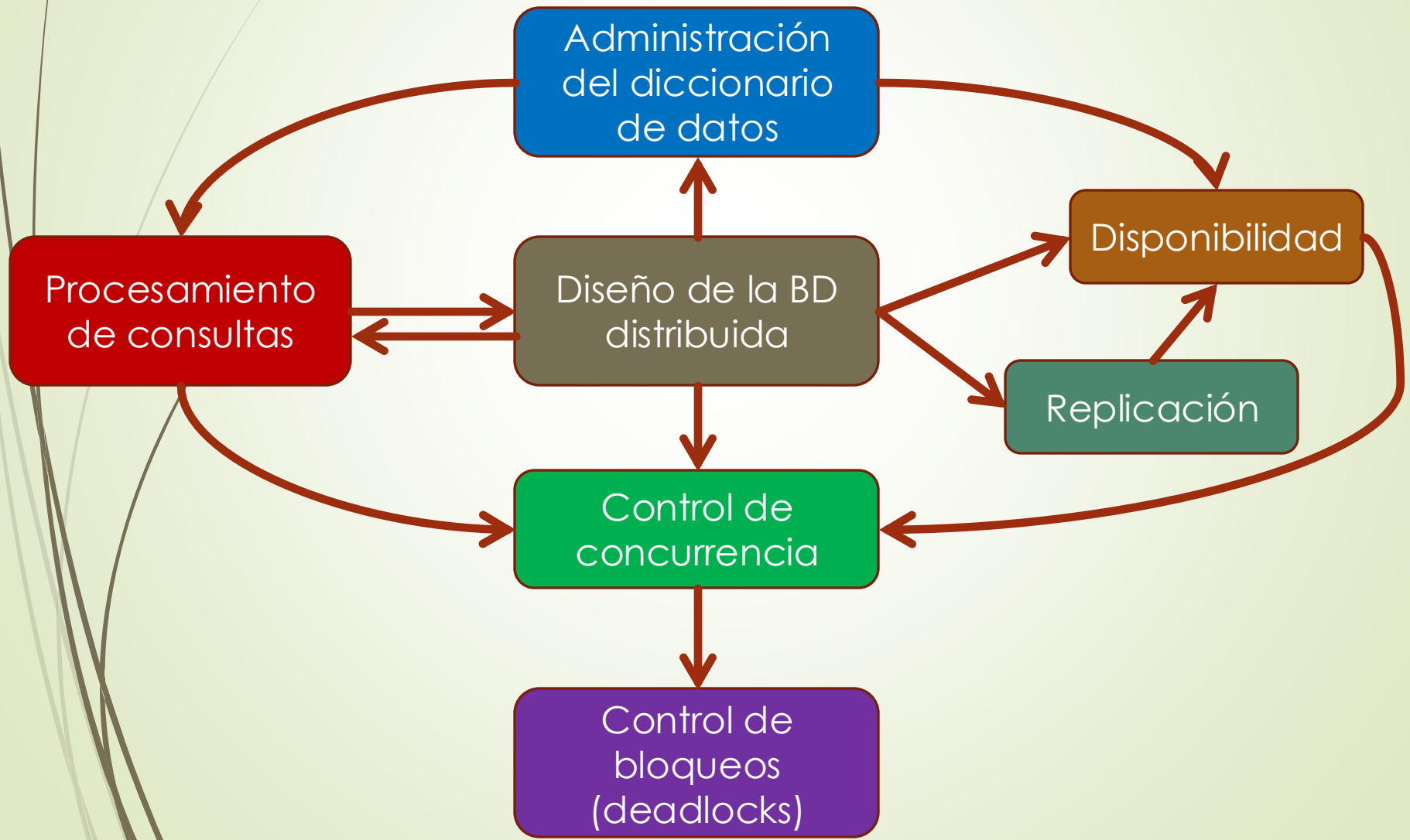
# Base de datos distribuida



Lo que no  
es una base  
de datos  
distribuida



# Problemática de la distribución



# Características de un SGBDD (DDBMS)



Administración transparente de datos replicados y distribuidos.



Independencia de Datos

La estructura de los datos puede modificarse sin alterar la operación de las aplicaciones.



Transparencia de Red

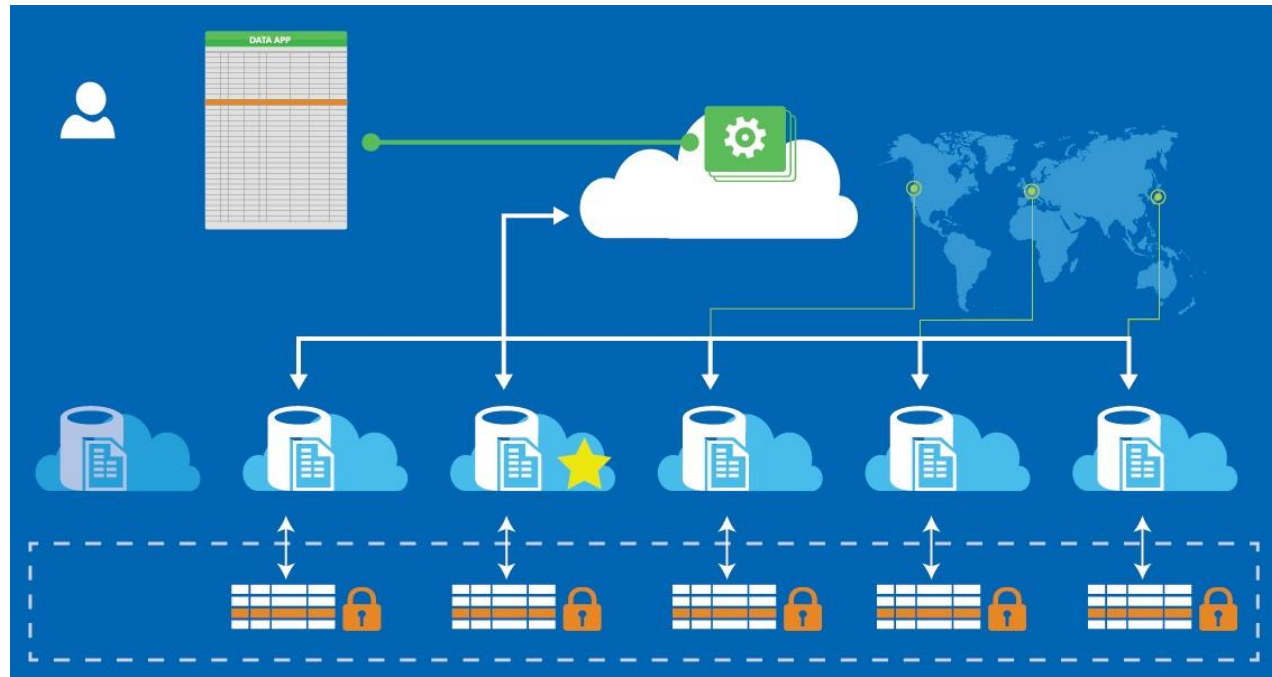
Location & Naming



# Transparencia de red

- Capacidad del sistema para abstraer a los usuarios de los detalles de donde y como están almacenados los datos en el sistema distribuido.
- Aspectos a considerar:
  - ✓ Formas de distribuir los datos:
    - Réplica
    - Fragmentación
  - ✓ Denominación de los datos (Naming)
  - ✓ Localización de fragmentos y réplicas



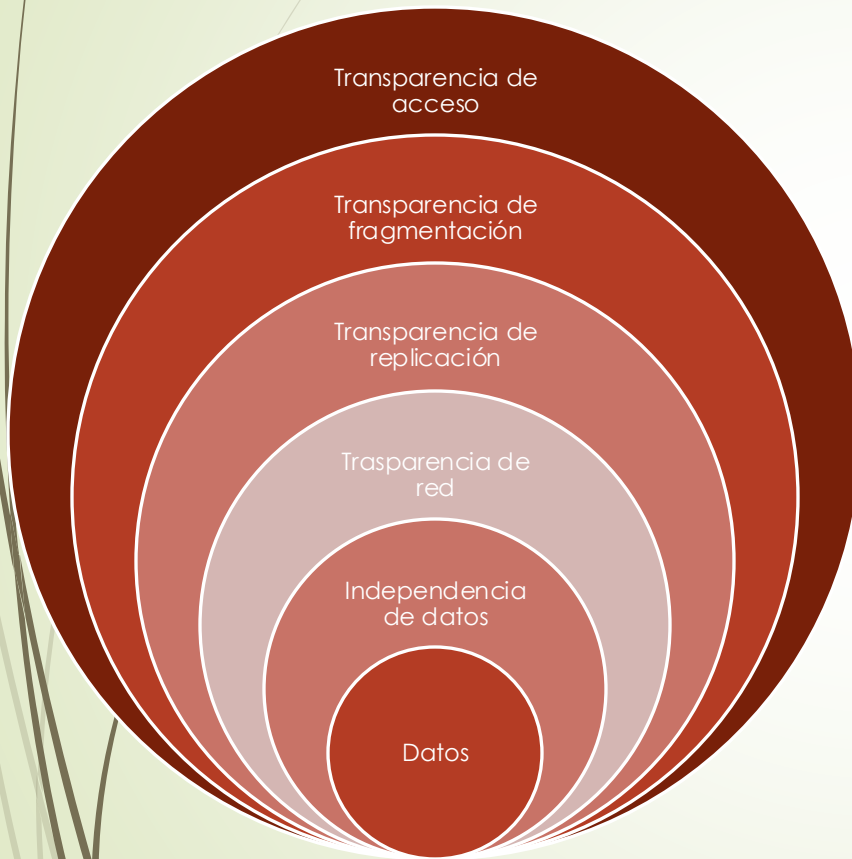


## Transparencia de red

**Location Transparency:** se pueden reubicar los datos sin afectar las aplicaciones.

**Naming Transparency:** el nombre de un objeto es independiente de su ubicación.

# Características de un SGBDD (DDBMS)



## Transparencia de replicación

- El usuario no necesita saber de la existencia de una réplica (o que está usando una réplica).
- Convergencia de datos.

## Transparencia de fragmentación

- El usuario accede a un único objeto lógico completo.
- por Rendimiento, Disponibilidad
- Horizontal y Vertical

# Características de un SGBDD (DDBMS)



Mayor  
disponibilidad

Replicación, Parcialidad.  
Transacciones distribuidas.  
Stand-by systems / Off-reporting.



Mayor  
rendimiento

Paralelismo horizontal  
Distributed Query Processing



Facilidad de Expansión



# Almacenamiento de datos distribuido

## Replicación

- Se mantienen múltiples copias de los datos almacenadas en diferentes nodos.

## Fragmentación

- La relación es particionada en fragmentos almacenados en diversos nodos, pero sólo una copia en total.

## Distribución

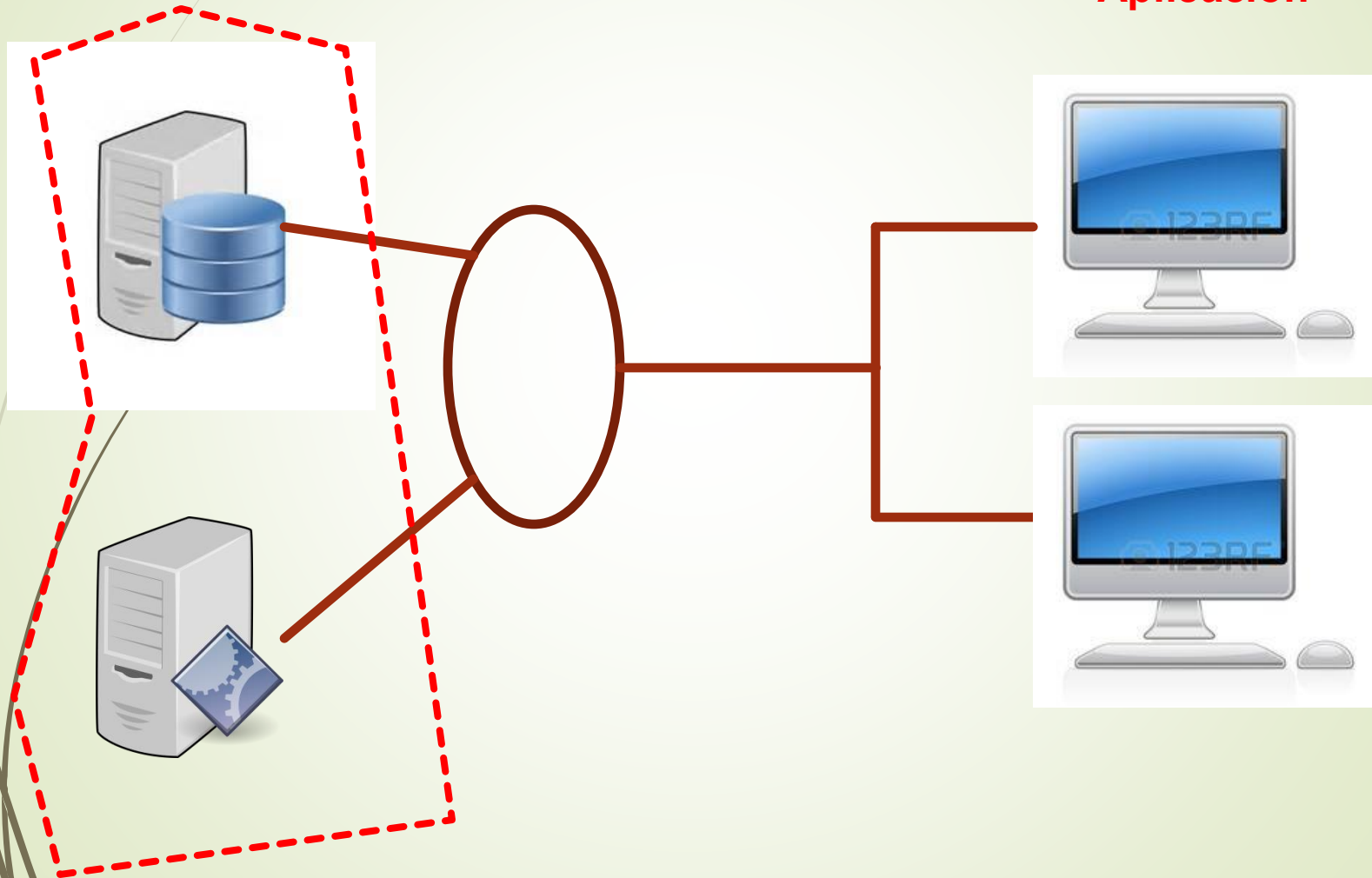
- Fragmentación Zero: la tabla completa existe una única vez en la BD global.

## Híbrido

- Replicación y fragmentación combinadas.

# BD centralizada

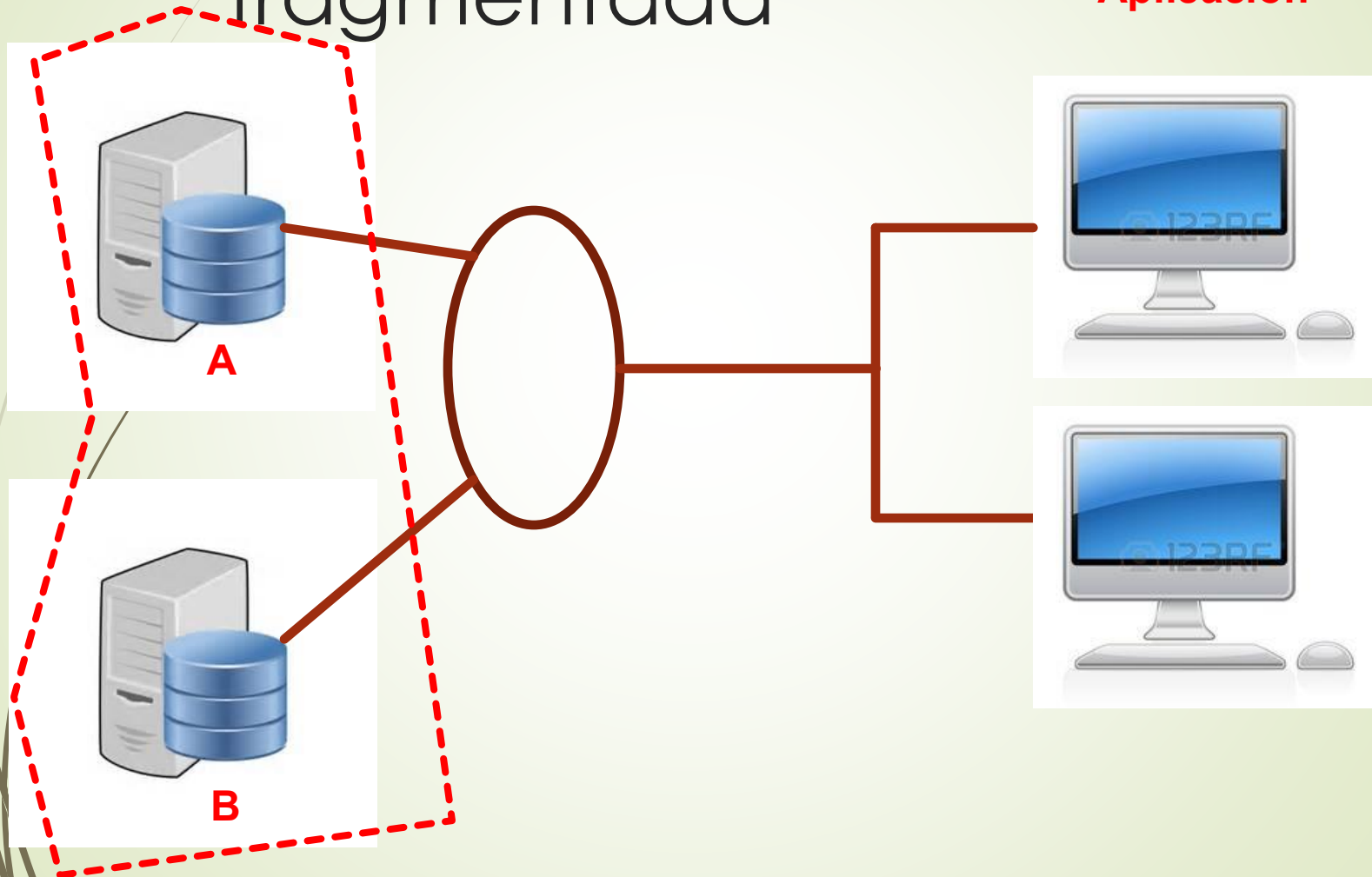
**Aplicación**



**Base de datos**

# BD distribuida / fragmentada

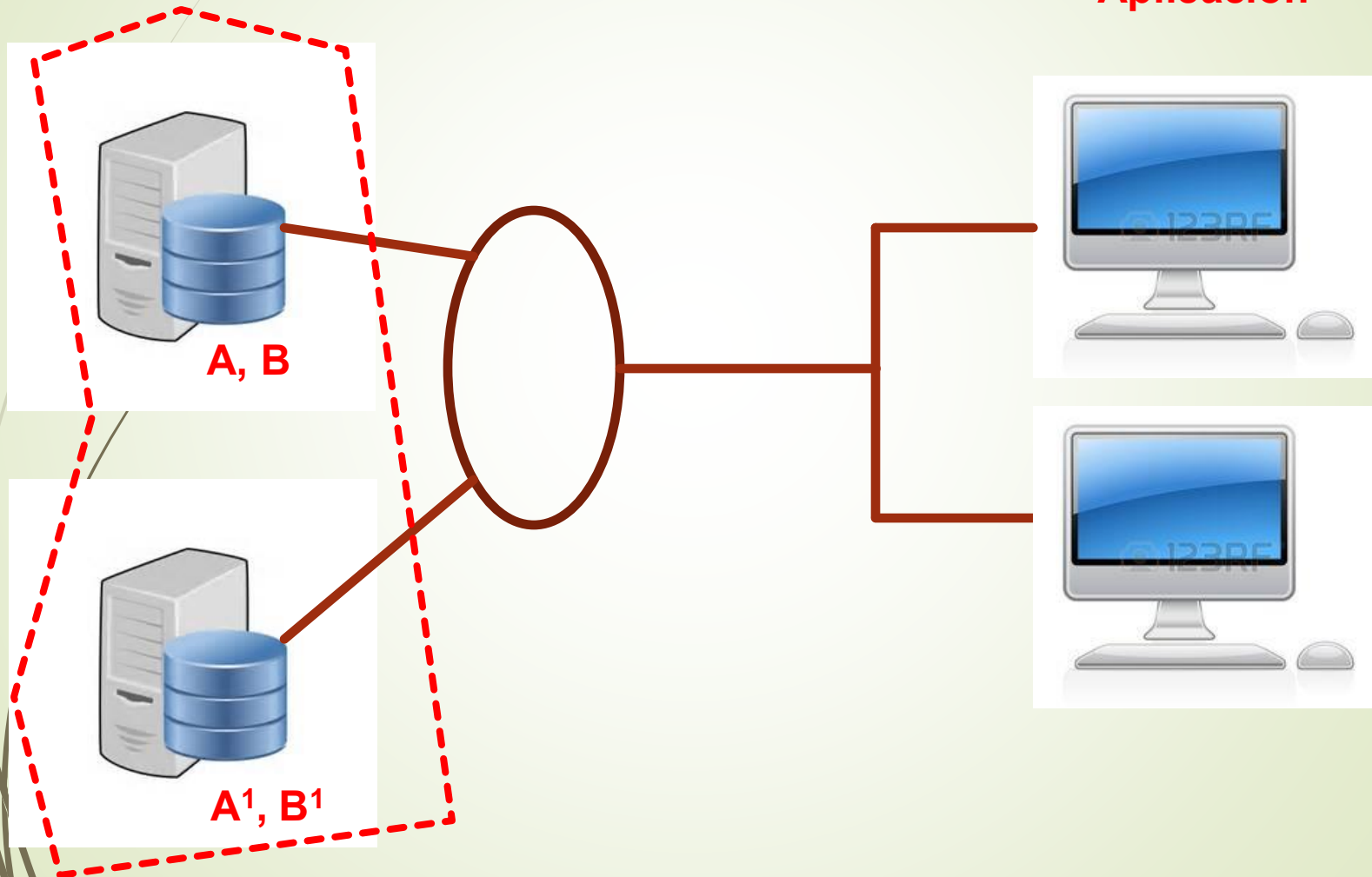
**Aplicación**



**Base de datos = A + B**

# BD replicada

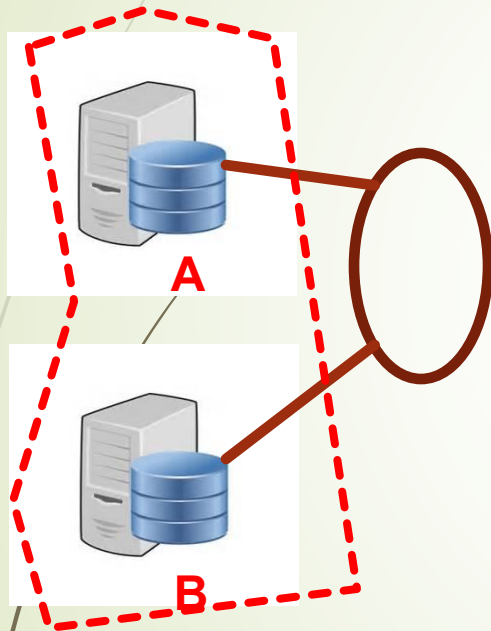
Aplicación



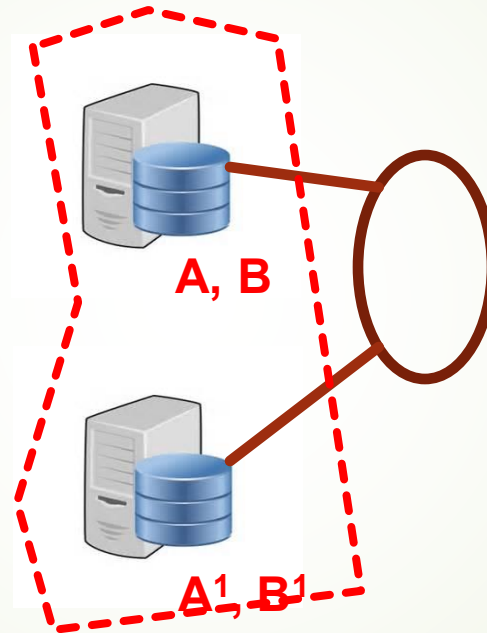
Base de datos = A + B



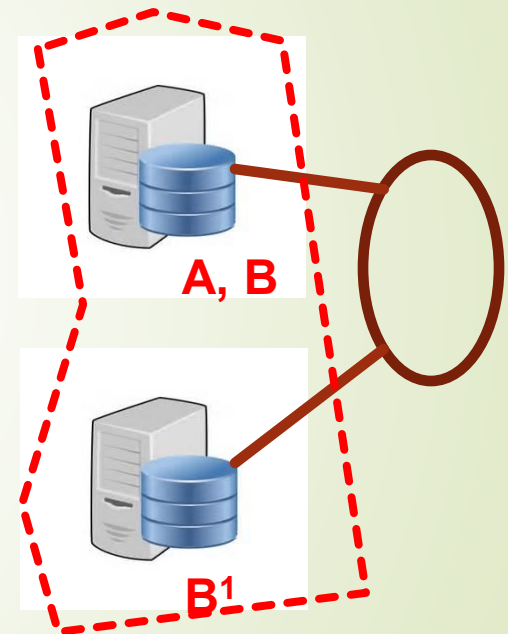
# Resumen



**Distribuido /  
fragmentado**



**Replicado**



**Híbrido**

**Base de datos = A + B**

# Ventajas de la replicación



## Alta Disponibilidad

Si un sitio con una relación falla, hay copias de la relación en otros sitios para continuar trabajando.



## Paralelismo

Consultas sobre la relación pueden ser ejecutadas en diversos nodos a la vez.



## Transferencia de datos

La relación está disponible en cada nodo que tenga una réplica de la relación.



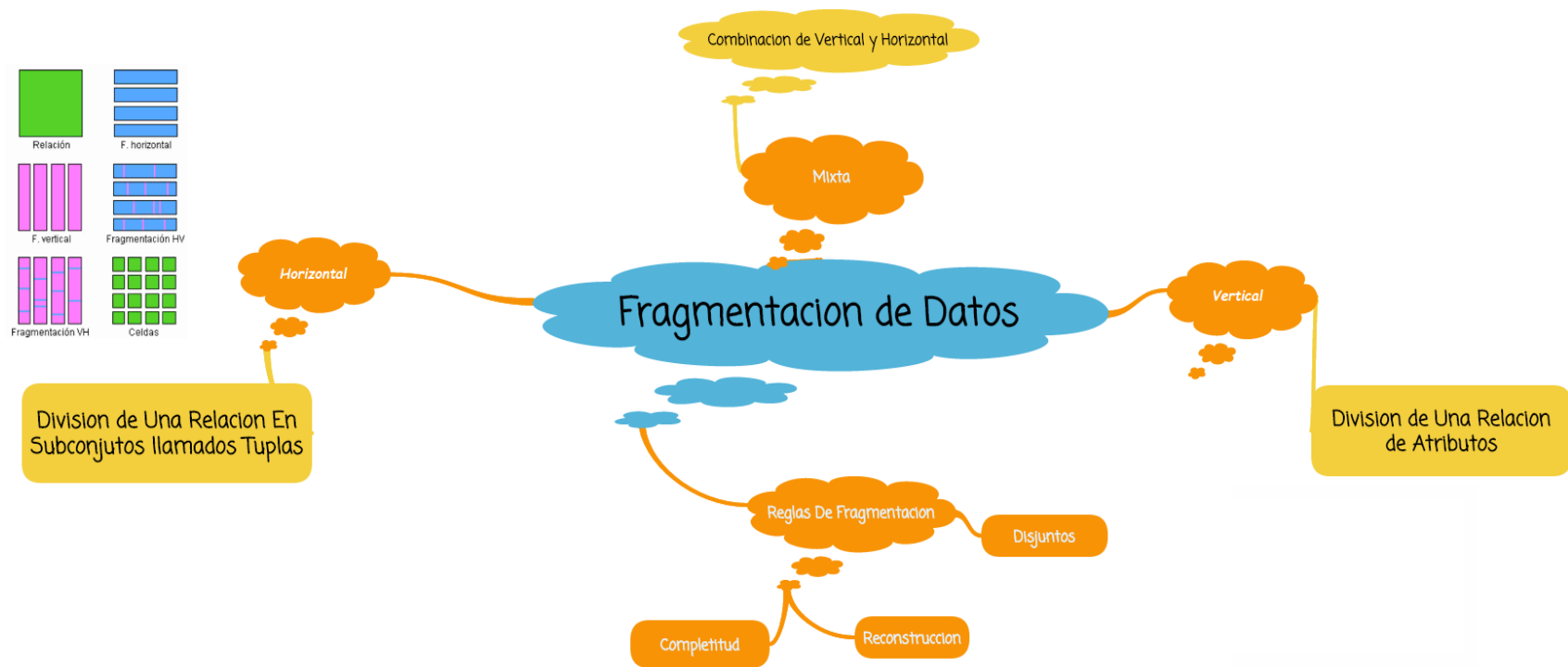
# Desventajas de la replicación

## Complejidad y costo de la actualización

- Cada replica debe ser actualizada
- off-line o on-line?

## Complejidad en el Control de la Concurrency

- Actualizaciones concurrentes llevan a la base de datos global a un estado inconsistente.
- No siempre es posible converger a un estado consistente.



1. Fragmentación Horizontal [IMAG.01].

## Fragmentación

- Los datos se distribuyen al almacenar tablas individuales en nodos diferentes, una única vez.
- También puede lograrse al “fragmentar” una tabla y almacenar estas piezas en diferentes nodos.
- Horizontal o Vertical.



# Ventajas de la fragmentación

## Usabilidad

- En general las aplicaciones usan un subconjunto de los datos.
- Ej: los empleados en Costa Rica vs USA

## Eficiencia

- La relación se almacena cerca de donde es más frecuentemente utilizada.

## Paralelismo

- Una transacción puede ser dividida en varias subconsultas para aumentar la concurrencia.
- Distributed Query Processing.

## Seguridad

- Los datos están sólo donde son necesarios.

# Desventajas de la fragmentación

## Rendimiento

- Consultas pueden ser más lentas debido al “data shipping”.

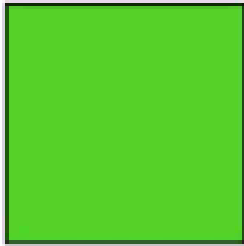
## Disponibilidad

- Si un nodo falla, la base de datos entera falla.

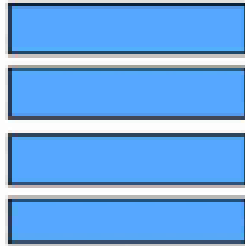
## Control de la Integridad



## Fragmentación horizontal



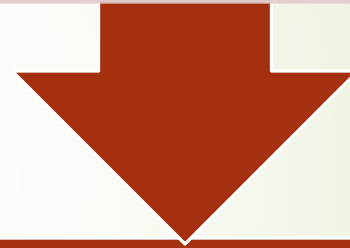
Relación



F. horizontal

Cada fragmento,  $T_i$ , de una tabla  $T$  contiene un subconjunto de las filas de  $T$

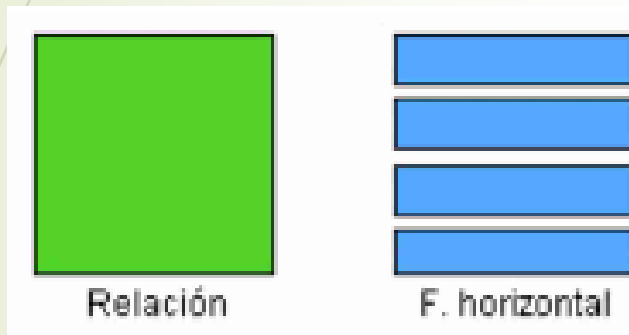
$$T = \bigcup_{i=1 \dots n} (T_i)$$



Cada tupla de  $T$  es asignada a uno o más fragmentos.



## Fragmentación horizontal



Se almacenan los datos en la sucursal en la que se utilizan.

Hay tantas tuplas como tuplas tengan todas las sucursales:

Difícil obtener una única vista.

Eficiente operar sobre una sucursal específica.

# Fragmentación horizontal

01	pedro	jupiter	plp
02	luis	venus	lls
03	maria	jupiter	mrn
04	ana	venus	ana
05	pedro	sol	ppp
06	elena	marte	cmr
07	carlos	jupiter	crc
08	sofia	venus	sofi
09	elena	jupiter	elsb
10	elena	venus	els
11	elisa	pluton	eli
12	elena	neptuno	elsc

Sitio A

01	pedro	jupiter	plp
02	luis	venus	lls
03	maria	jupiter	mrn
04	ana	venus	ana
05	pedro	sol	ppp

Sitio B

06	elena	marte	cmr
07	carlos	jupiter	crc
08	sofia	venus	sofi
09	elena	jupiter	elsb
10	elena	venus	els
11	elisa	pluton	eli
12	elena	neptuno	elsc

# Fragmentación vertical

- Cada fragmento,  $T_i$  de  $T$ , contiene un subconjunto de las columnas, cada columna está en al menos un fragmento y cada fragmento incluye la llave:
- $T_i = \pi \text{ attr\_list}_i (T)$
- $T = T1 \mid X \mid T2 \dots \mid X \mid Tn$
- Un atributo especial, llamado el tuple-id, podría agregarse para servir como una llave candidata artificial.



# Fragmentación vertical

Sitio A

01	pedro	jupiter	01
02	luis	venus	02
03	maria	jupiter	03
04	ana	venus	04
05	pedro	sol	05
06	elena	marte	06
07	carlos	jupiter	07
08	sofia	venus	08
09	elena	jupiter	09
10	elena	venus	10
11	elisa	pluton	11
12	elena	neptuno	12

Sitio B

01	plp	01
02	lls	02
03	mrm	03
04	ana	04
05	ppp	05
06	cmr	06
07	crc	07
08	sofi	08
09	elsb	09
10	els	10
11	eli	11
12	elsc	12

# Objetivos de las bases de datos distribuidas

**Autonomía local.** Los sitios en un sistema distribuido deben ser **autónomos**.

**No dependencia de un sitio central.** La autonomía local implica que todos los sitios deben ser tratados como iguales.

**Operación continua.** Una ventaja de los sistemas distribuidos es que deben proporcionar mayor *confiabilidad* y mayor *disponibilidad*.

**Independencia de ubicación.** Conocida también como *transparencia de ubicación*.

## Objetivos de las bases de datos distribuidas (cont.)




**Independencia de fragmentación.** Un sistema soporta la *fragmentación de datos* cuando puede ser dividida en o partes o fragmentos, para efectos de almacenamiento físico.



**Independencia de replicación.** El sistema soporta *replicación de datos* cuando un *fragmento* puede ser representado por muchas copias distintas, o réplicas, guardadas en muchos sitios distintos.



**Procesamiento de consultas distribuidas.** La optimización es más importante en un sistema distribuido que en uno centralizado.



# Objetivos de las bases de datos distribuidas (cont.)

---

**Administración de transacciones distribuidas.** Existen dos aspectos principales en la administración de transacciones: *control de recuperación y control de la concurrencia.*

---

**Independencia de hardware.** Soporte para un gran número de máquinas diferentes. Poder integrar todos los datos de todos estos sistemas y presentar al usuario una “imagen del sistema único”.

---

**Independencia de sistema operativo.** Obviamente es necesario no sólo tener la posibilidad de ejecutar el mismo DBMS en diferentes plataformas de hardware, sino también ejecutarlo en diferentes plataformas de sistema operativo.





# Objetivos de las bases de datos distribuidas (cont.)

---

**Independencia de red.** Si el sistema tiene la posibilidad de soportar muchos sitios distintos es obviamente necesario tener la posibilidad de soportar también una variedad de redes de comunicación distintas.

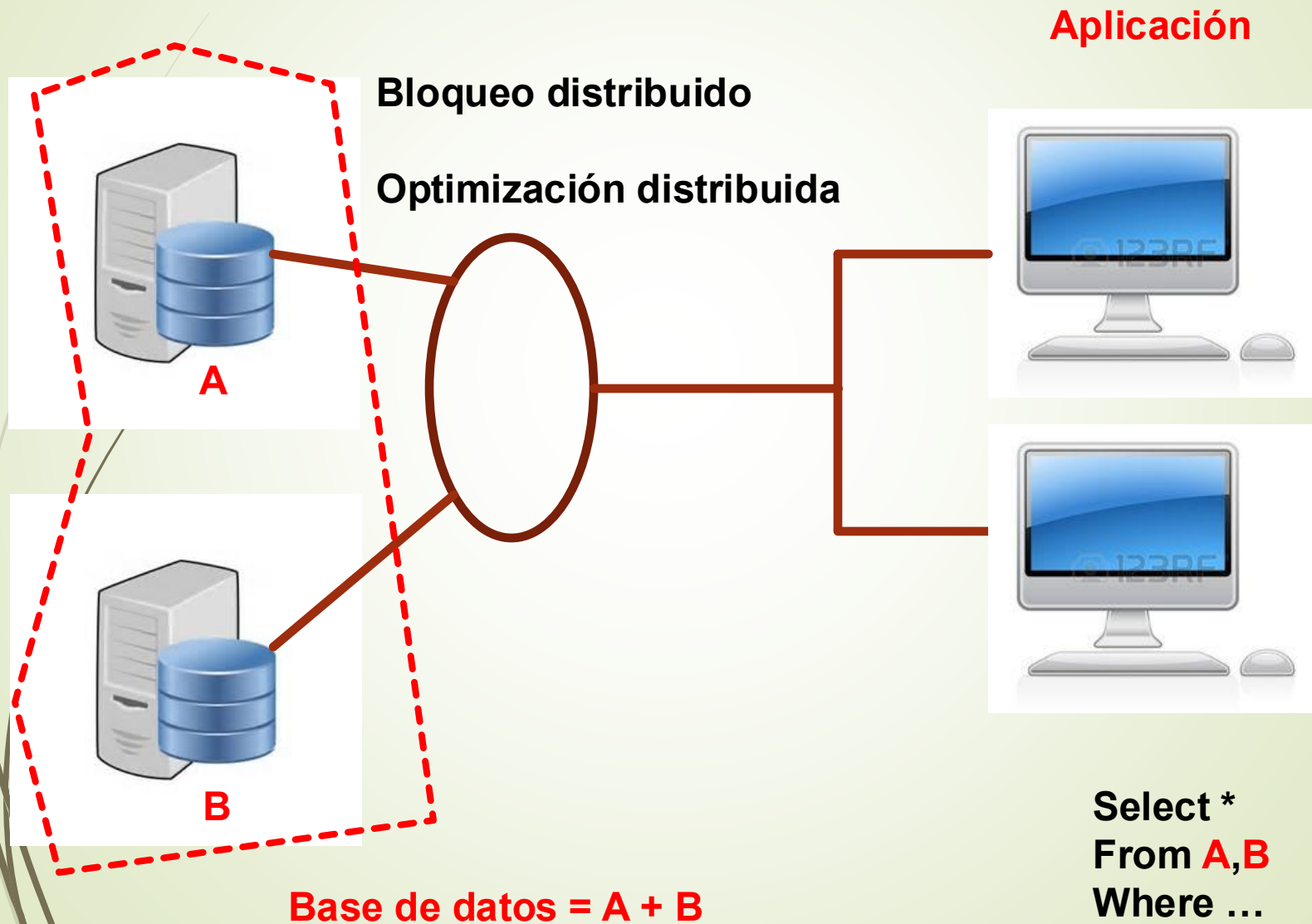
---

**Independencia de DBMS.** Lo que se necesita es que *todos* los ejemplares de DBMS en sitios diferentes *soporten la misma interfaz*.

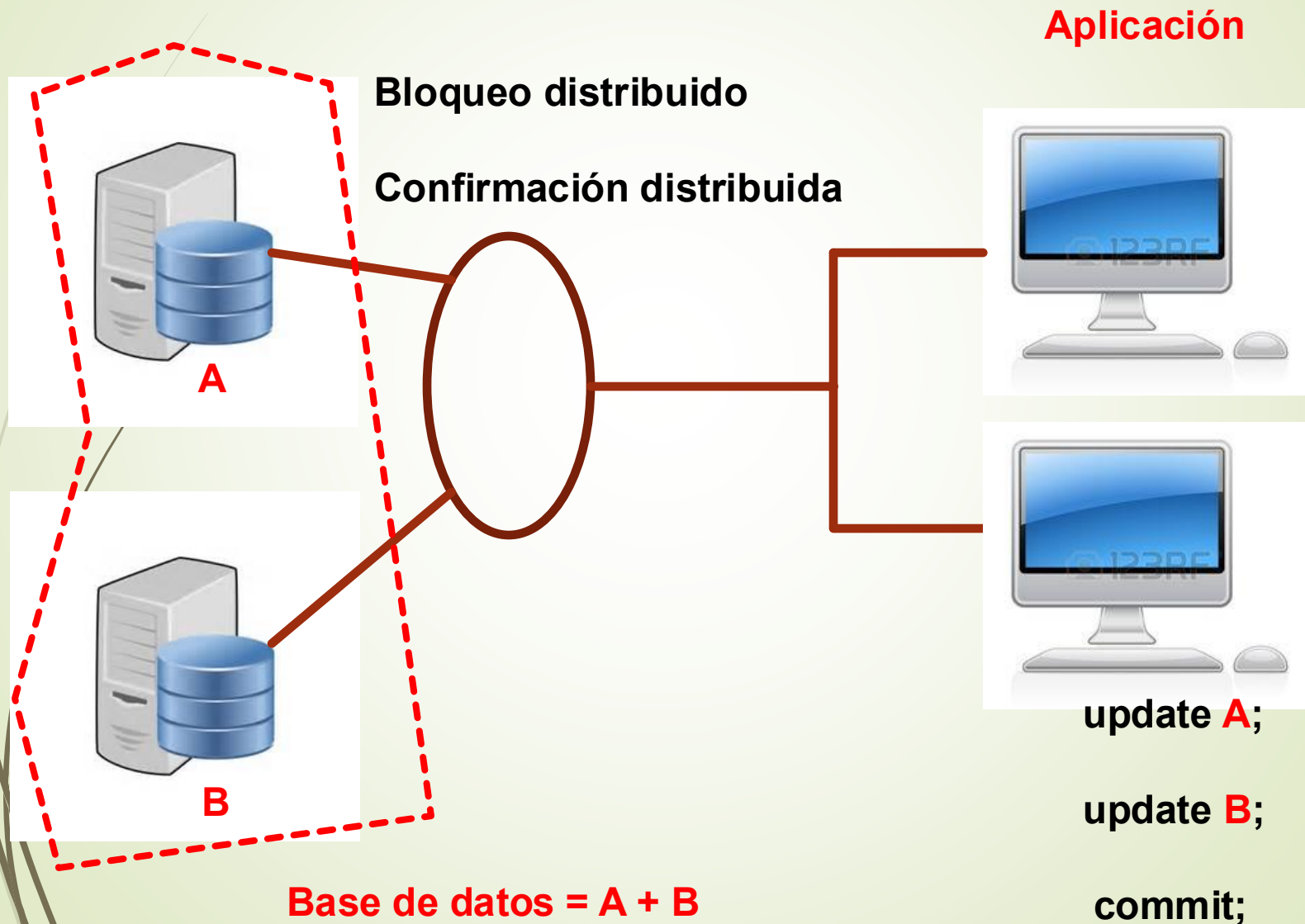
# Replicación y distribución

	Replicación	Replicación parcial	Distribución
Procesamiento de consultas	simple	difícil	difícil
Manejo del diccionario de datos	muy simple	difícil	difícil
Control de concurrencia	medio	complejo	simple
Confiableidad	muy alta	alta	baja
Uso	posible	típica	posible

# BD distribuida



# BD distribuida



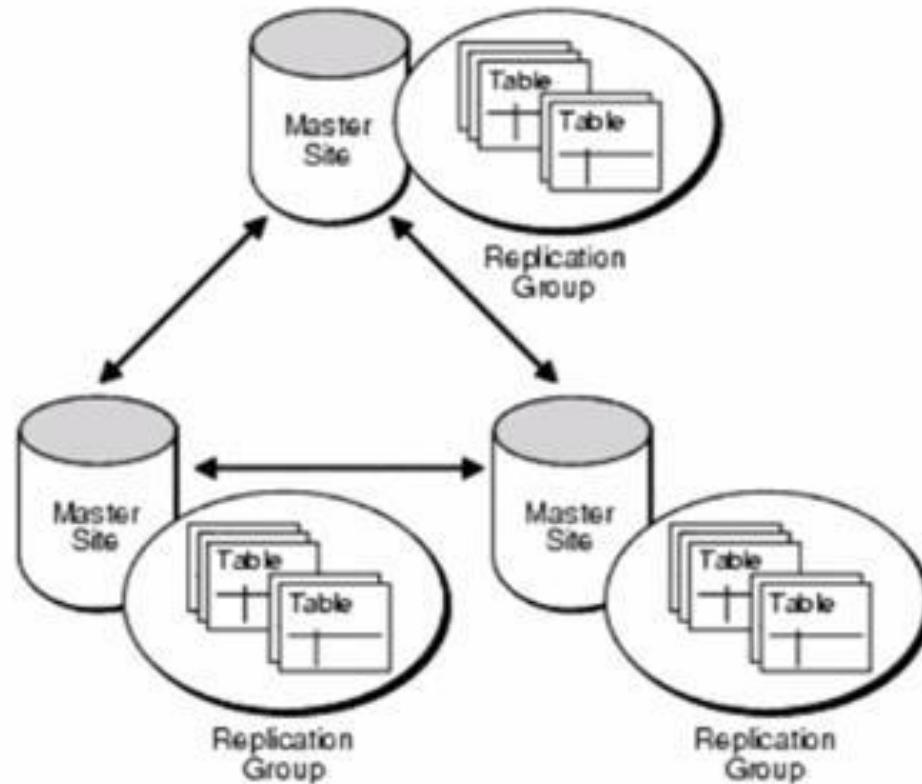
# Consultas distribuidas y transacciones distribuidas

Consulta distribuida

Un sólo SELECT que accede información de diferentes sitios remotos/locales.

Transacción distribuida

Una transacción que modifica información en diferentes nodos remotos/locales.



# Transacciones distribuidas

La confirmación (commit) debe ser atómica: todos o nadie.

# Entrega de datos

- La información en un DDBMS debe ser enviada desde el nodo que tiene los datos hacia el nodo que hace la consulta
  - Pull (de los clientes al servidor)
    - Se envían mensajes (consulta y actualización)
    - Tiempo de respuesta al cliente depende del tiempo de búsqueda y actualización
  - Push (del servidor a los clientes)
    - El servidor tiene la lista de réplicas de clientes y cachés
    - Se envían actualizaciones
    - Tiempo de respuesta al cliente depende del tiempo de búsqueda y actualización, pero puede considerarse inmediato
  - Hybrid
  - Periodic, Conditional, Ad-Hoc



# Problemas

## Falla un nodo

- Se pierden mensajes
- Falla el enlace de comunicación
- “Network partition”:
- Se dice que la red se ha particionado cuando se ha dividido en dos o mas sub-redes que no tienen conexión entre ellas.

No es posible distinguir un falla en la red de una falla en el nodo!

# Protocolos de confirmación (commit)

Asegura la atomicidad de las transacciones a través de todos los nodos:

- Confirmación en todos los nodos o aborto en todos los nodos.
- No es aceptable tener una transacción que sea confirmada en un nodo y abortada en otro.

Two-phase commit (2PC)

Three-phase commit (3PC)

- Más complicado y costoso, resuelve algunos problemas del 2PC.
- No se usa en la práctica

# Confirmación distribuida

Cada sitio involucrado en una transacción tiene:

- Local Transaction Manager:
  - Mantiene el log de transacciones (undo or redo)
  - Coordina la ejecución concurrente en este nodo.

Transaction Coordinator:

- Inicia la ejecución de transacciones distribuidas que se originan en este sitio
- Distribuye las sub-transacciones a los nodos apropiados para su ejecución
- Coordina la terminación de las transacciones que se originan en ese sitio

# Two Phase Commit (2PC)

- Asume un modelo de **fail-stop**
  - Los nodos que fallan simplemente dejan de funcionar sin causar ningún otro daño, como enviar mensajes erróneos a los demás nodos
- La ejecución del protocolo es iniciada por el **coordinador** una vez que todas las operaciones de la transacción han terminado y se prepara para hacer commit.
- El protocolo involucra a todos los nodos que participan en la transacción.
- Si  $T$  es iniciada en el nodo  $N_i$ ,  $C_i$  es el coordinador de transacciones en  $N_i$ .

# Fase 1

El **coordinador** pide a todos los **Nodos participantes** prepararse para hacer commit de la transacción T.

- El coordinador agrega un <prepare T> en el log y lo graba permanentemente.
- Envía un mensaje de preparación a todos los nodos donde T ejecutó operaciones.

En el nodo, el Transaction Manager determina si puede hacer commit de T localmente:

- **Si no puede hacer commit**, agregar un registro <no T> en el log y envía un mensaje abortando T al Coordinador (voto).
- **Si puede** hacer commit entonces:
  - Agrega <ready T> al log
  - Graba todos los registros de T permanentemente
  - Envía un <ready T> al Coordinador (voto)

# Fase 2



T fue aplicada (committed) si el Coordinador recibe un mensaje “ready” T de todos los **participantes**: en caso contrario T debe abortarse.



El Coordinador agrega un <commit T> o <abort T> al log y lo graba permanentemente.

Una vez grabado es irrevocable (aun si luego hay fallas).

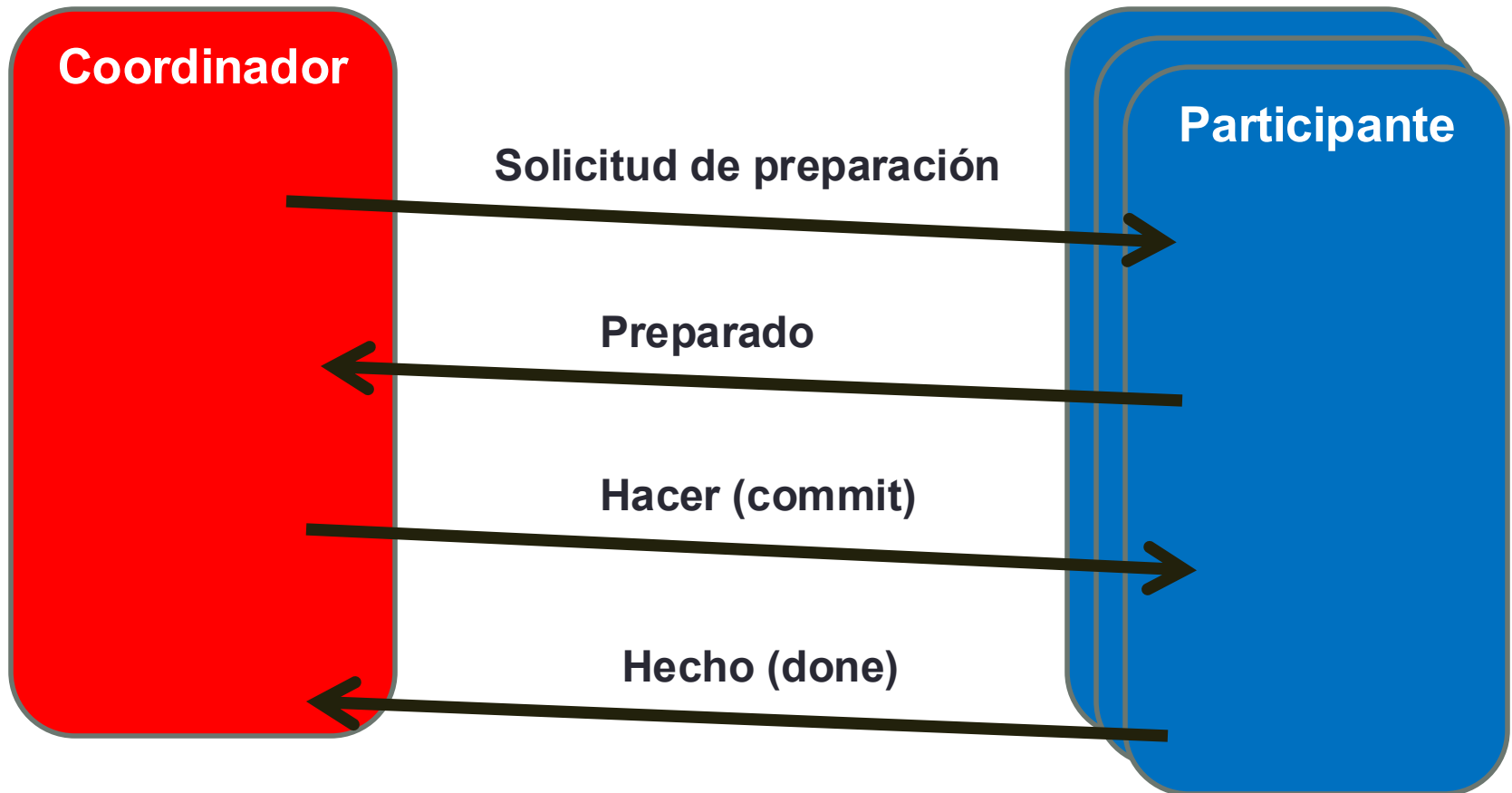


El Coordinador envía un mensaje a cada participante indicando la decisión (commit o abort).

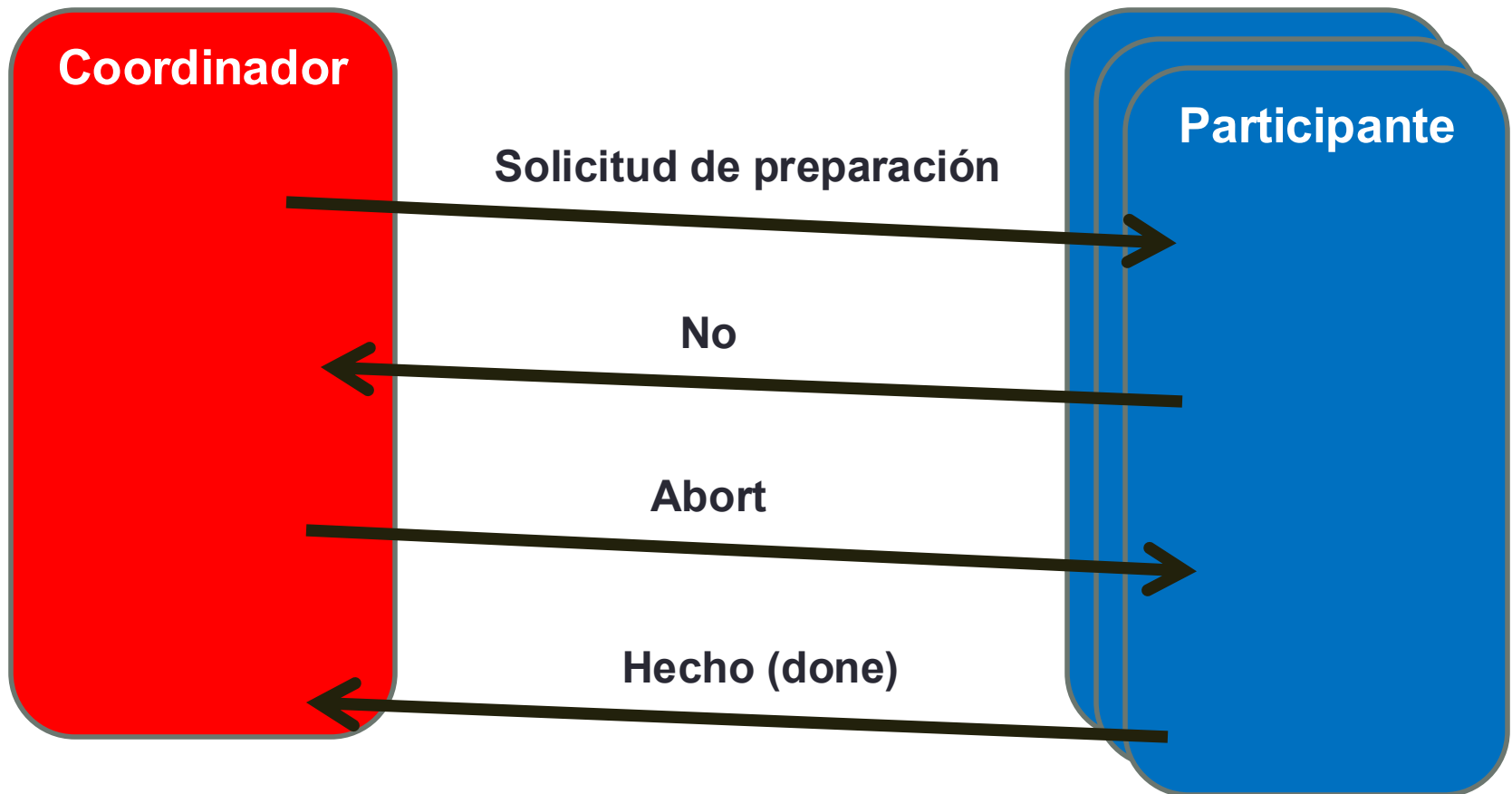


El Participante toma la acción apropiada localmente.

# Caso 1: Commit



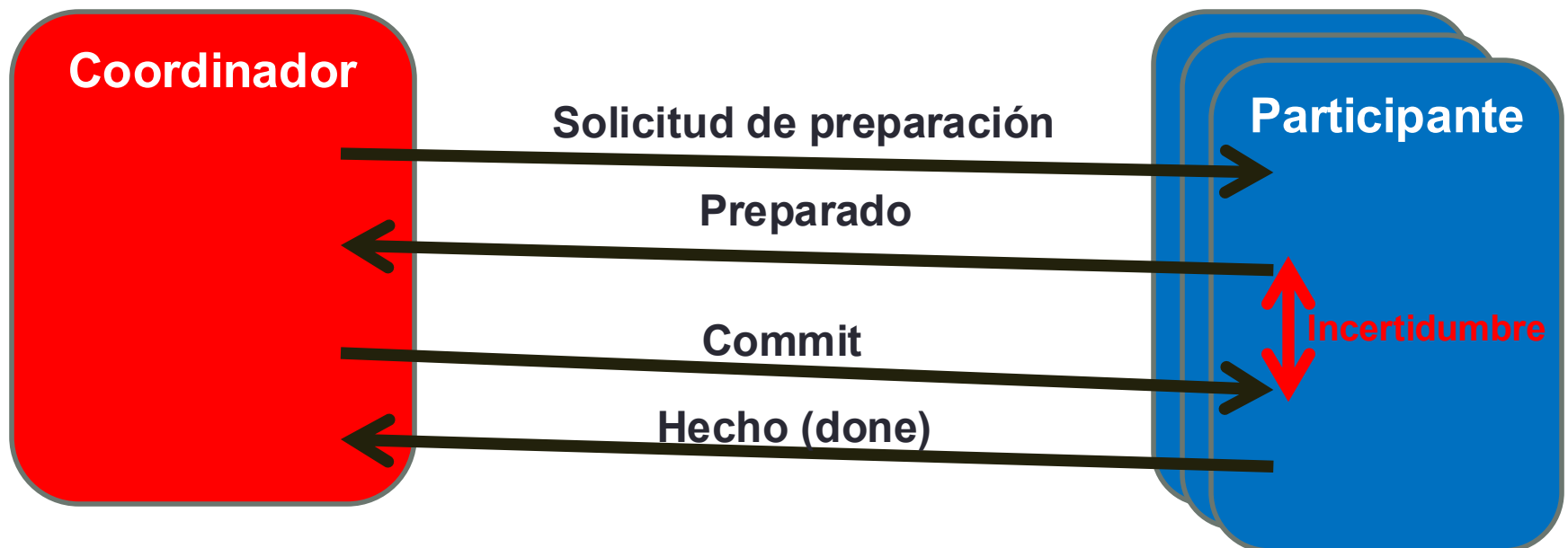
## Caso 2: Abort





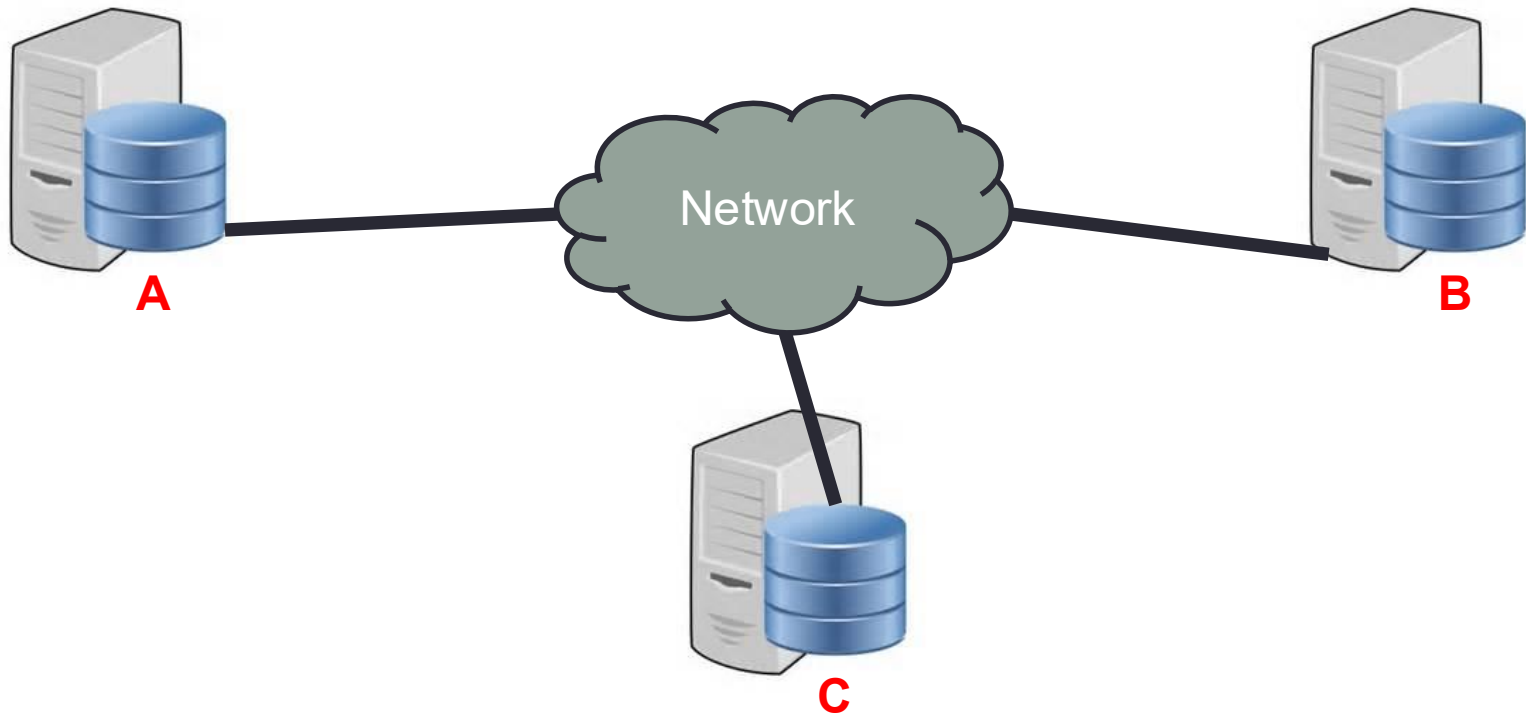
# Incertidumbre (“In-doubt”)

- Antes de “votar” un participante puede abortar unilateralmente.
- Luego de indicar que está **preparado** y antes de recibir la decisión del coordinador, la transacción es incierta: no puede abortar de manera unilateral hasta no conocer el resultado.

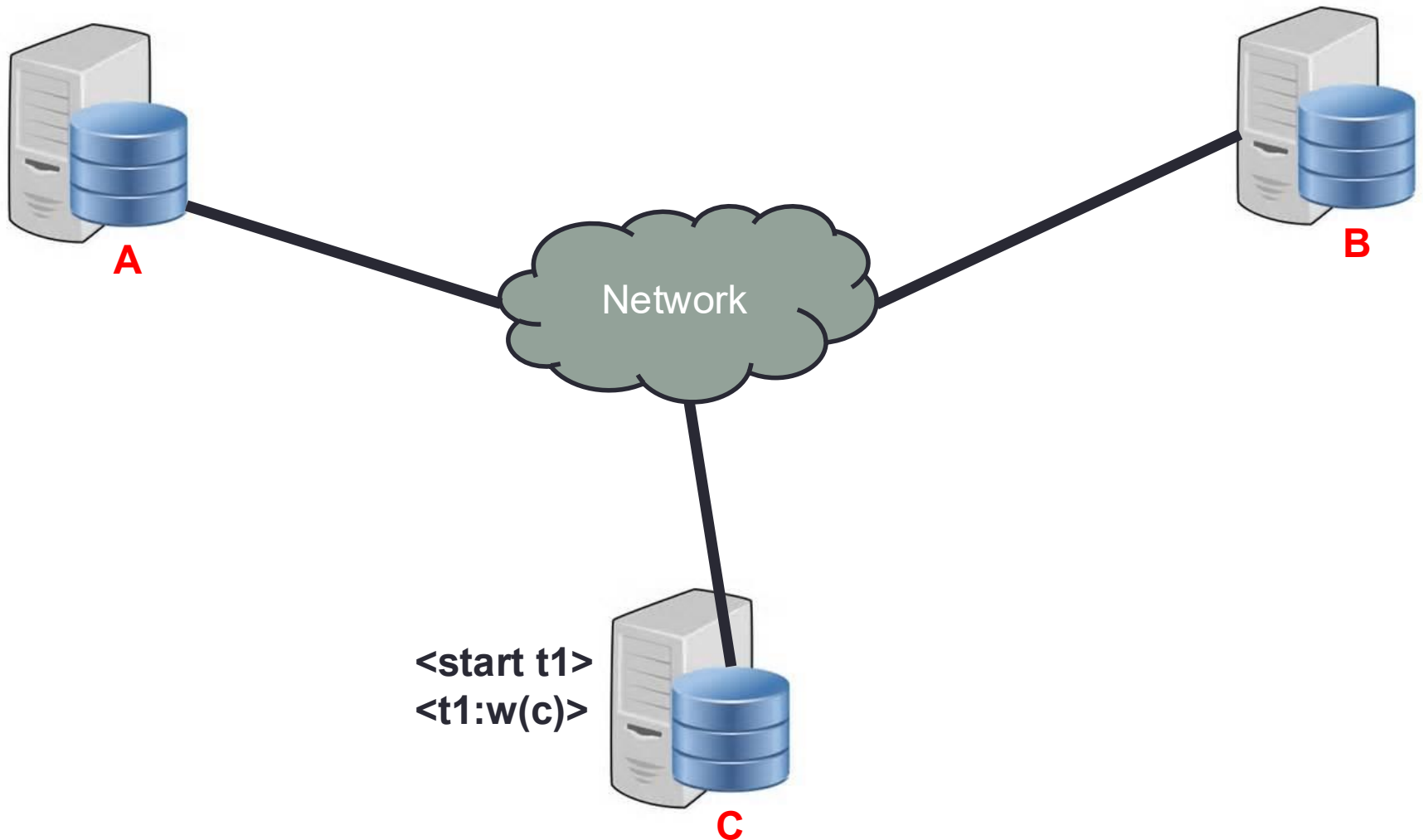


# Incertidumbre (“In-doubt”)

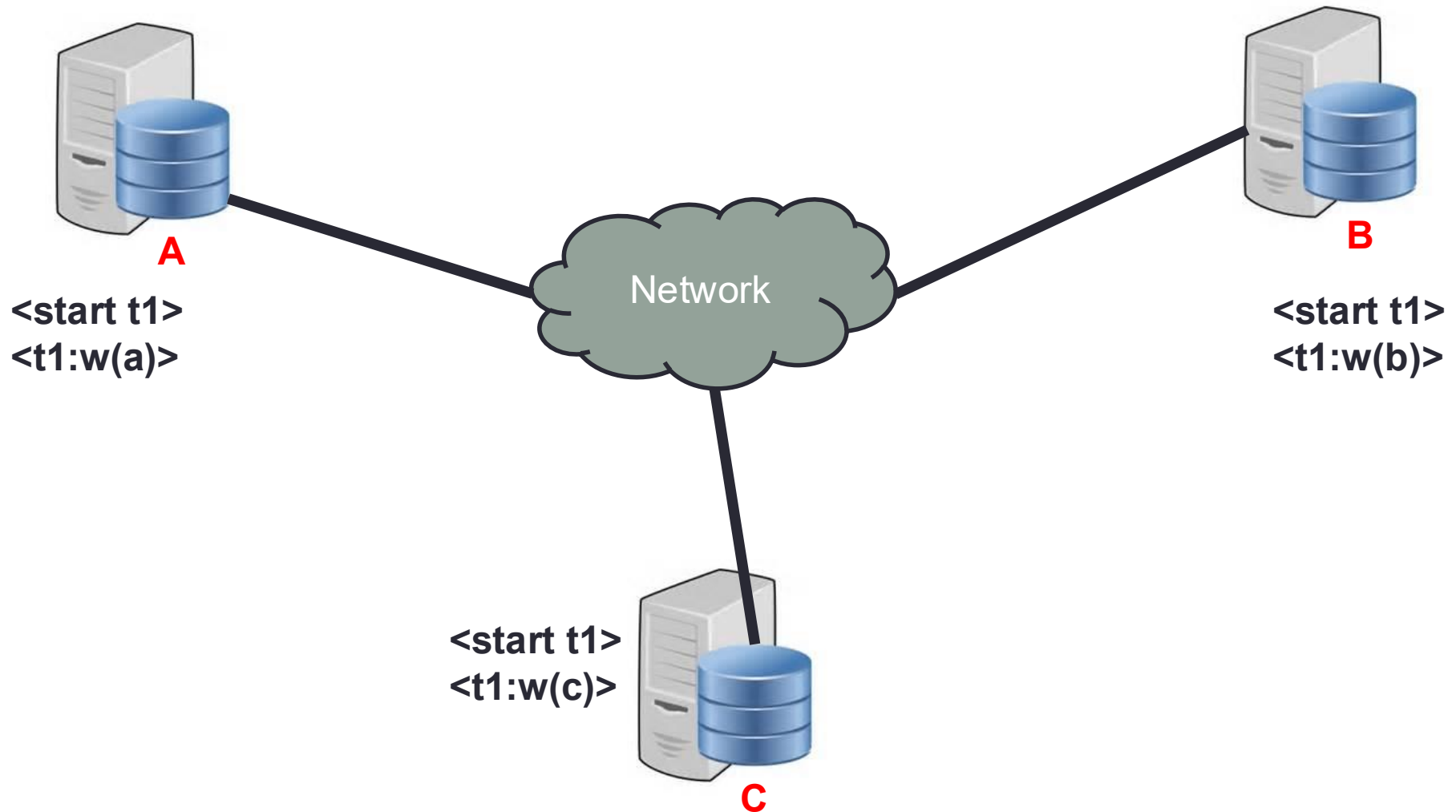
- El Coordinador nunca esta incierto.
- Si un Participante falla o se desconecta, cuando se recupere debe buscar cómo quedó la transacción.



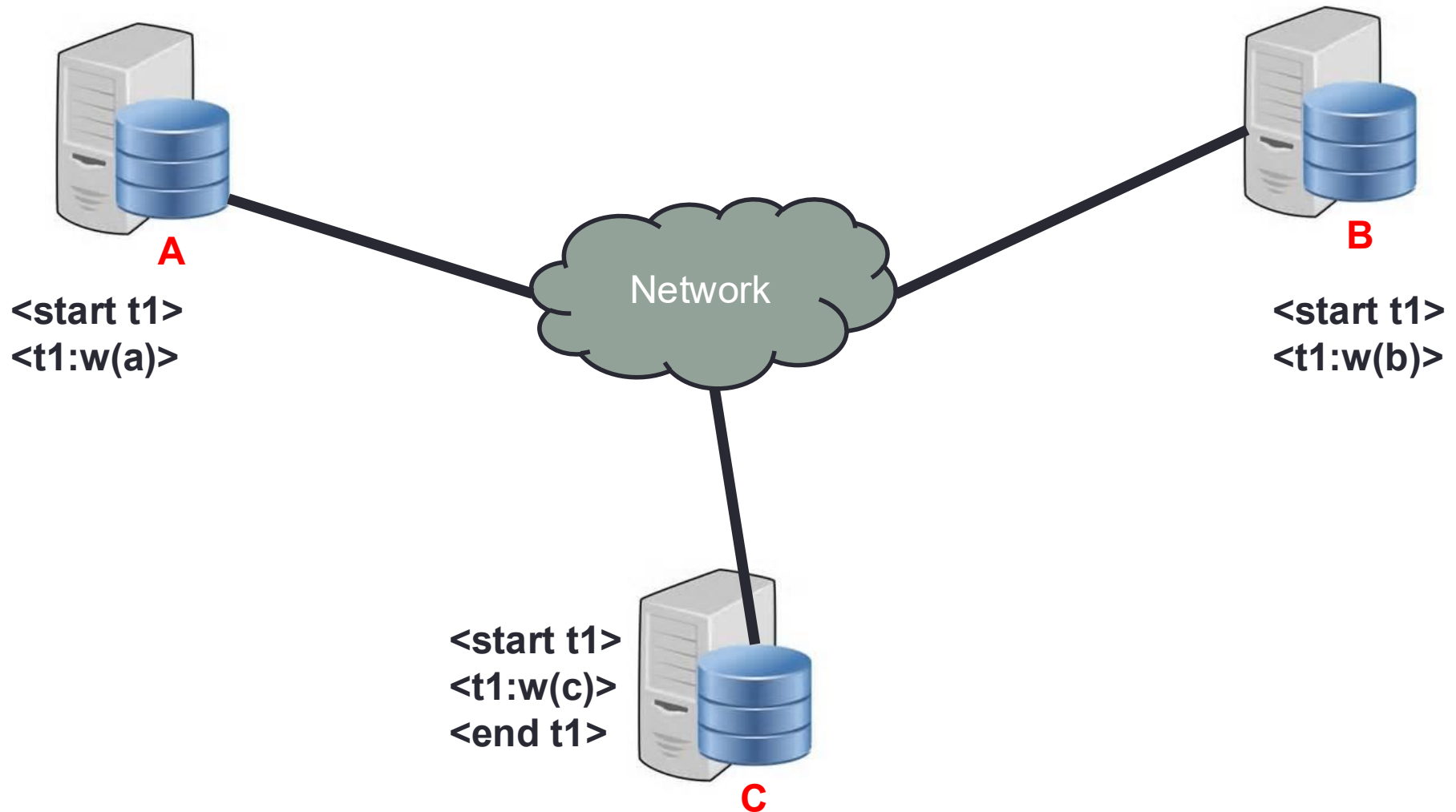
# 2PC (commit)



# 2PC (commit)



# 2PC (commit)



# 2PC (commit)

Nodo1 (n1)



**A**

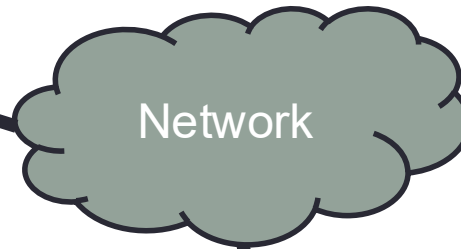
<start t1>  
<t1:w(a)>

Nodo2 (n2)



**B**

<start t1>  
<t1:w(b)>



**C**

**Coordinador**

<start t1>  
<t1:w(c)>  
<end t1>

# 2PC (commit)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>

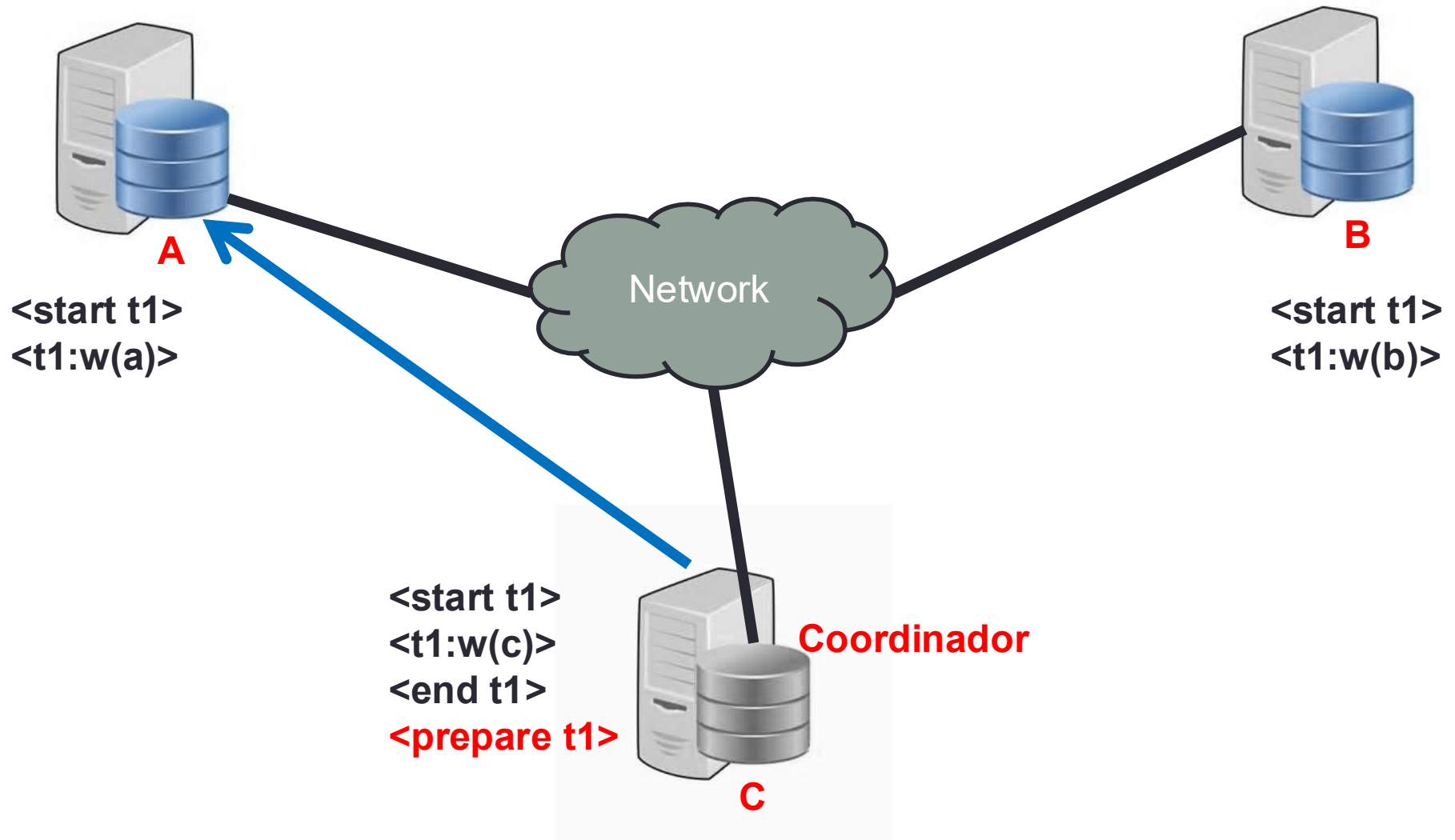
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>

Coordinador



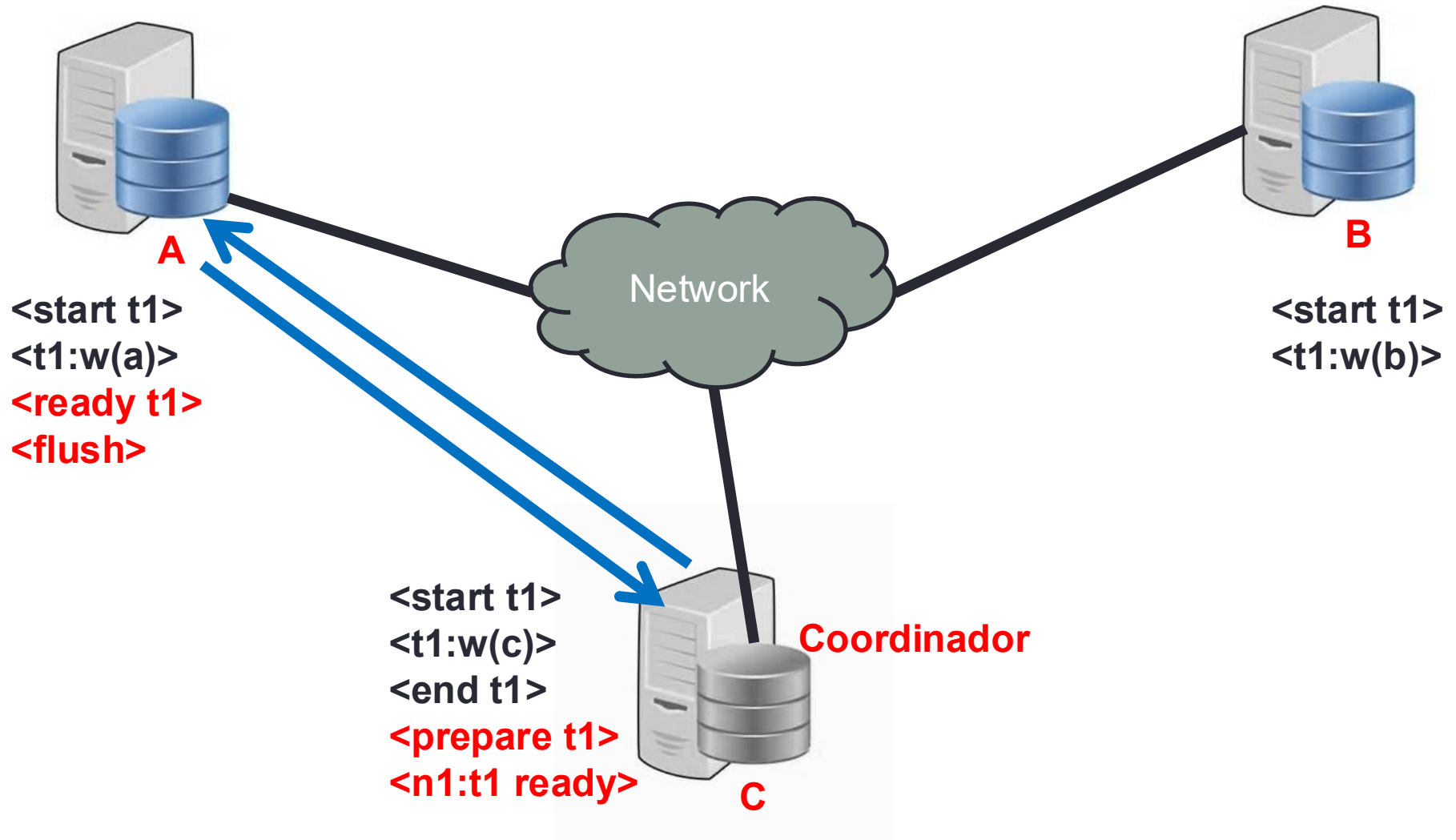
C



# 2PC (commit)

Nodo1 (n1)

Nodo2 (n2)





# 2PC (commit)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>

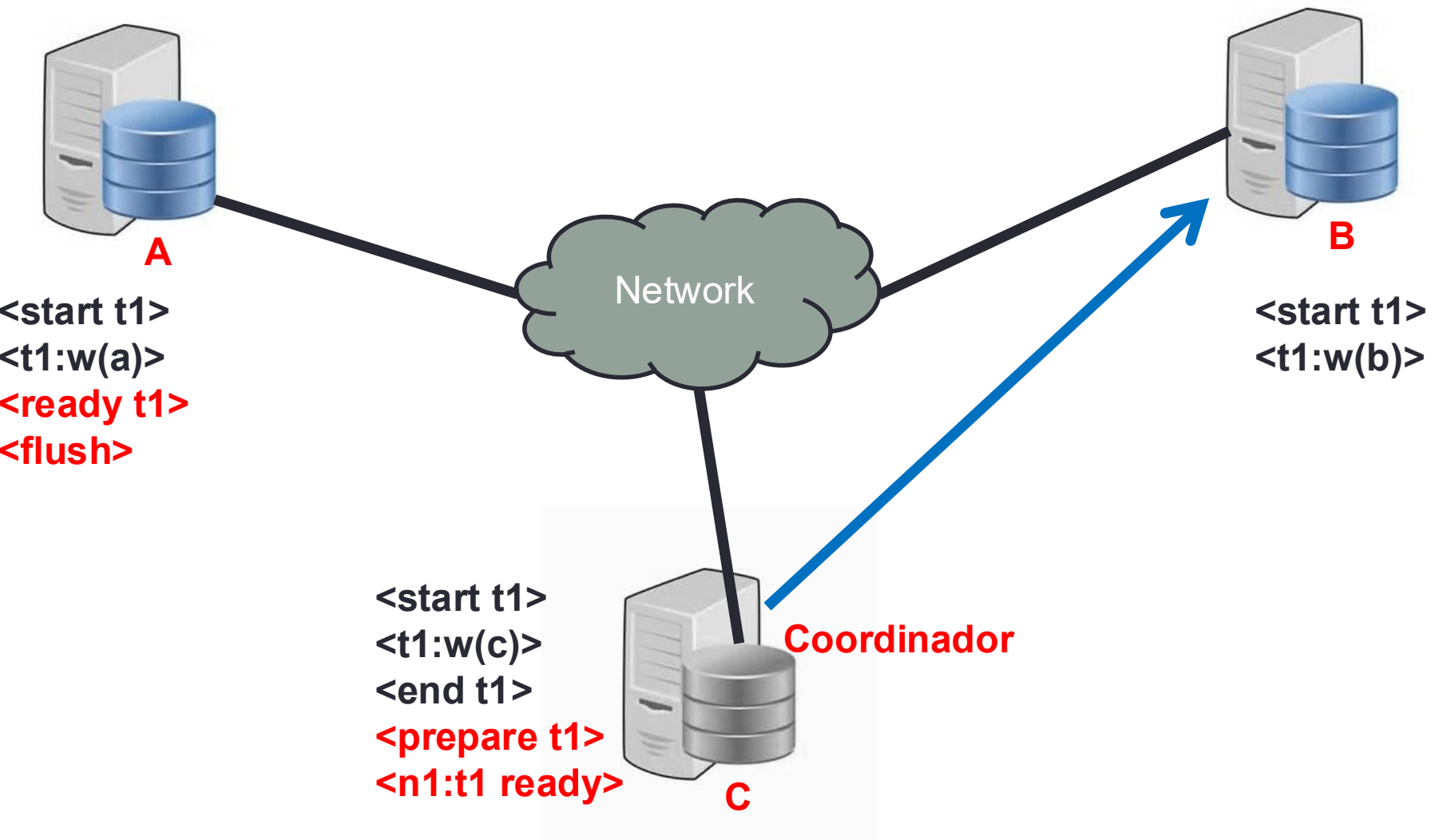
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>



C

Coordinador



# 2PC (commit)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>  
<ready t1>  
<flush>

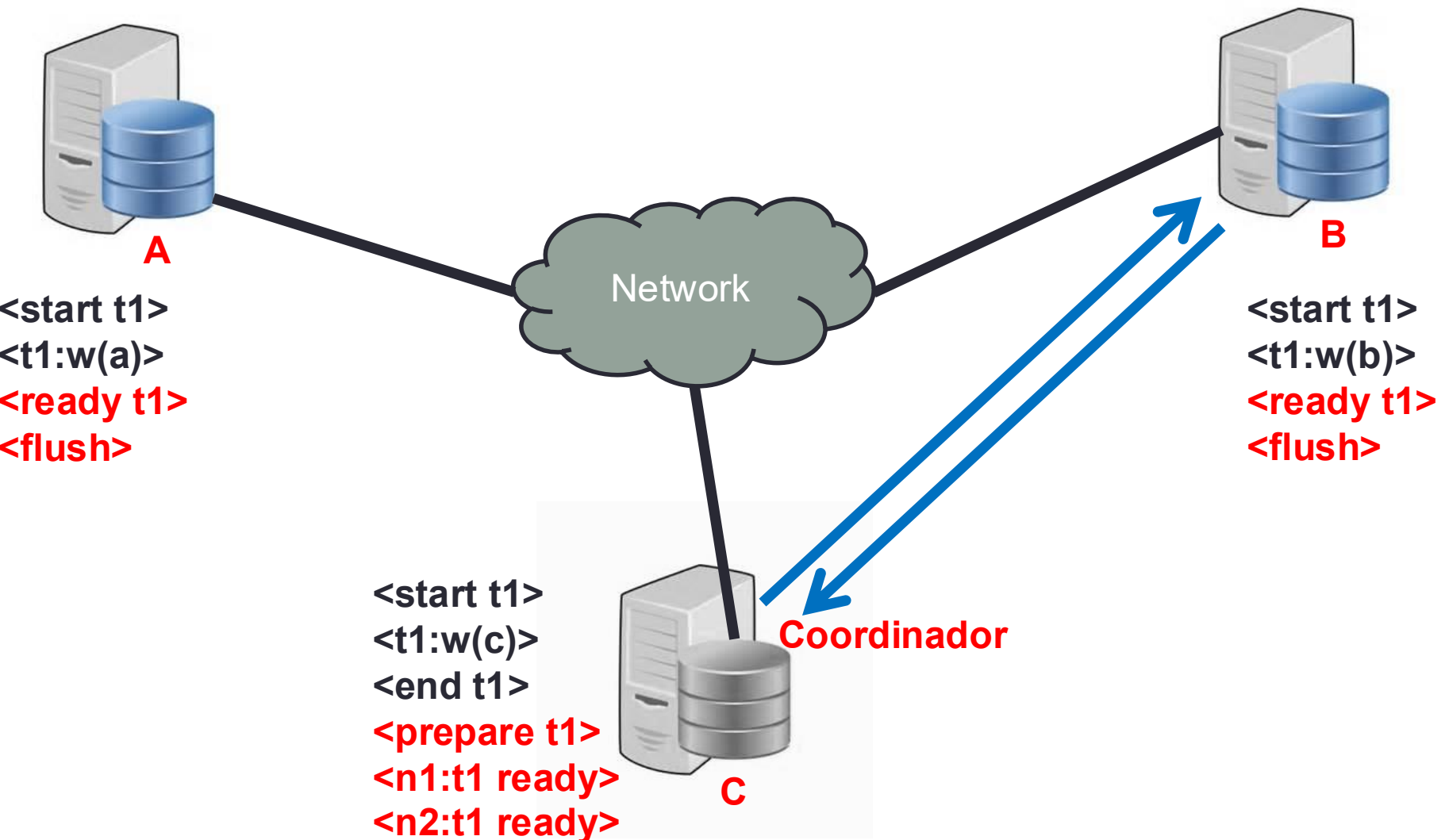
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>  
<n2:t1 ready>



C

Coordinador



# 2PC (commit)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>  
<ready t1>  
<flush>

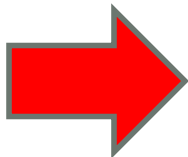
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>  
<n2:t1 ready>

Coordinador



C



# 2PC (commit)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>  
<commit t1>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>  
<ready t1>  
<flush>  
<commit t1>

Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>  
<n2:t1 ready>  
<commit t1>



C

Coordinador

# 2PC (abort)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>

Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>



C

Coordinador

# 2PC (abort)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>

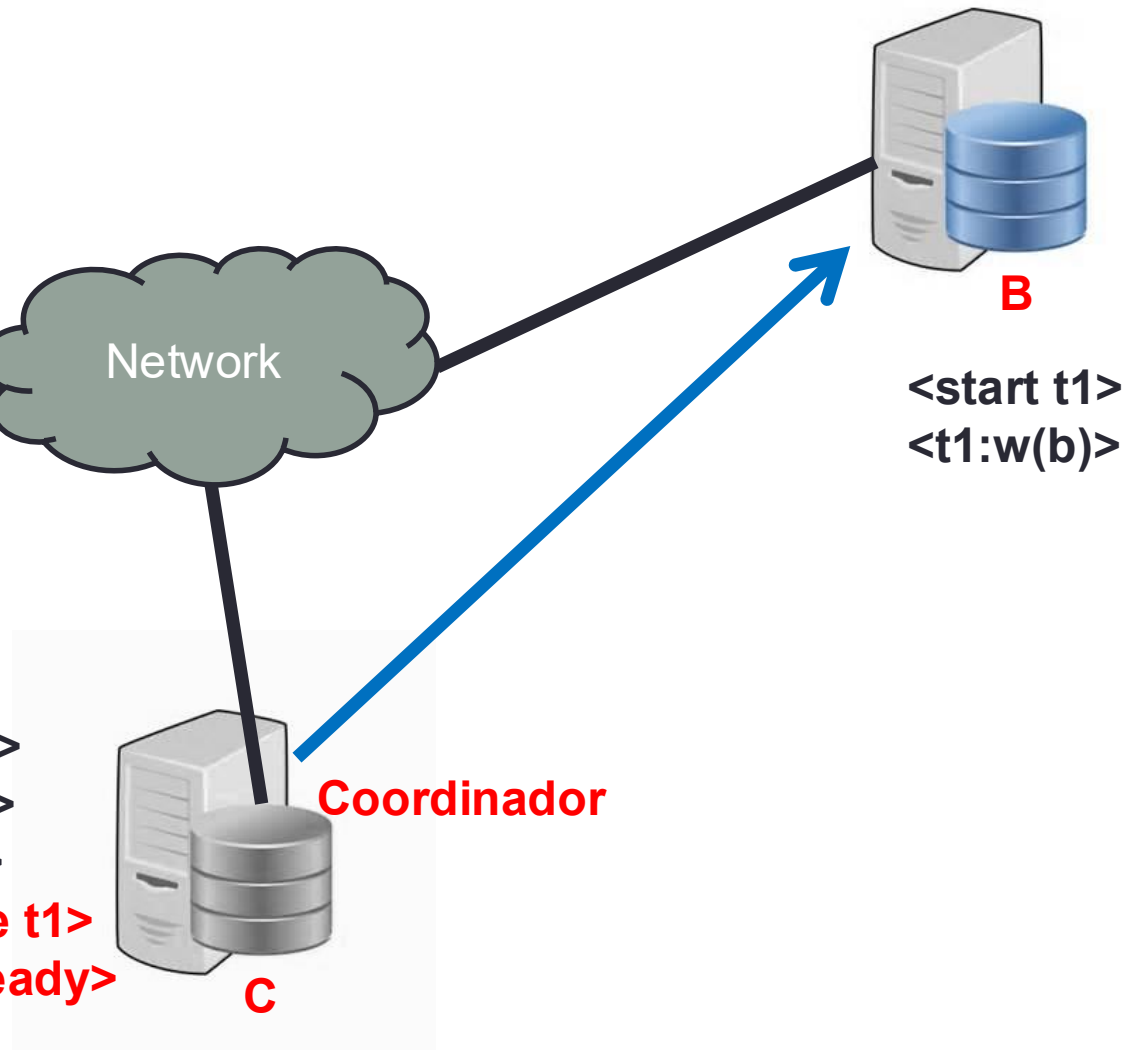
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>



C

Coordinador



# 2PC (abort)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>  
<no t1>  
<flush>

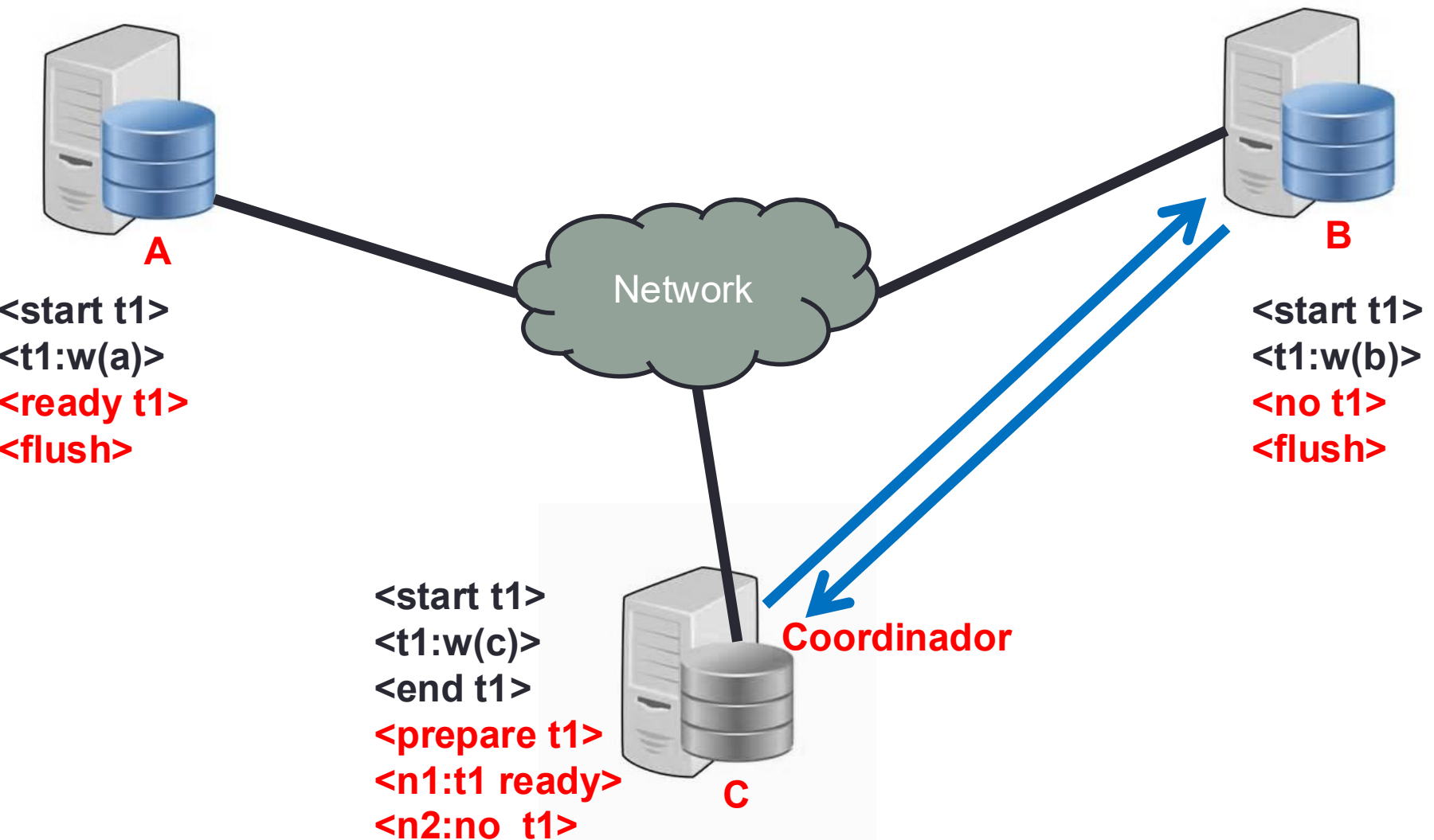
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>  
<n2:no t1>



C

Coordinador



# 2PC (abort)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>  
<no t1>  
<flush>

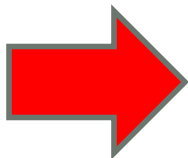
Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>  
<n2:no t1>

Coordinador



C





# 2PC (abort)

Nodo1 (n1)



A

<start t1>  
<t1:w(a)>  
<ready t1>  
<flush>  
<abort t1>

Nodo2 (n2)



B

<start t1>  
<t1:w(b)>  
<no t1>  
<flush>  
<abort t1>

Network

<start t1>  
<t1:w(c)>  
<end t1>  
<prepare t1>  
<n1:t1 ready>  
<n2:no t1>  
<abort t1>



C

Coordinador

# Rendimiento



Sin fallas, el 2PC requiere 3 rondas de mensajes antes de tomar una decisión:

Request-to-prepare  
Votos (ready, no)  
Decision (commit, abort)



Mensajes de “Done” son simplemente de confirmación.

No afectan el tiempo de respuesta.  
Pueden operar en batch.

# Falla de un nodo

Cuando el nodo  $N_i$  se recupera, examina el log para determinar la suerte de las transacciones que estaban activas al momento de la falla.

Si el log contiene:

- <commit T>: el nodo ejecuta un redo(T).
- <abort T>: el nodo ejecuta un undo(T).
- <ready T>: estamos en "in-doubt": el nodo debe consultar al nodo coordinador para saber qué pasó con T:
  - Si T hizo commit, entonces hace un redo(T).
  - Sino, hace un undo(T).
- No tiene registro de la respuesta al nodo coordinador, entonces debe asumir que T se abortó.
  - Se ejecuta un undo (T)

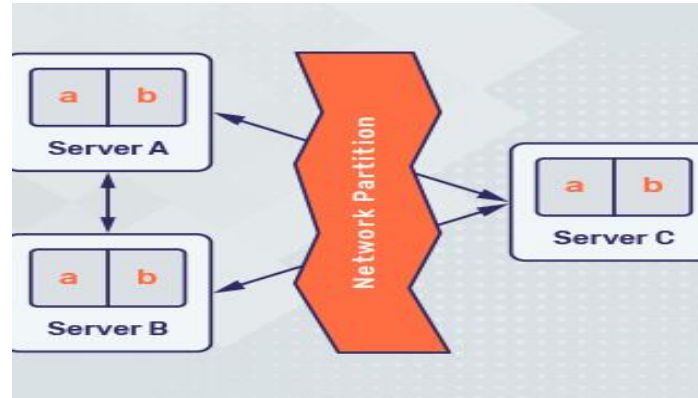
# Falla del coordinador

- Entonces los nodos participantes deben decidir qué hacer con T:
  - Si un nodo participante tiene en el log un `<commit T>` entonces T debe ser aplicada.
  - Si un nodo participante tiene en el log un `<abort T>` entonces T debe ser abortada.
  - Si alguno de los participantes no tiene un `<ready T>` en el log, quiere decir que el nodo coordinador no llegó a decidirlo, por tanto, T debe ser abortada.
  - Si todos los nodos tienen un `<ready T>` pero no un `<abort T>` o `<commit T>` entonces deben esperar por el nodo coordinador para recuperarse, para determinar qué hacer con T.
    - Blocking problem: los nodos activos deben esperar que el nodo coordinador se recupere.

# “Network Partition”



**Si el Coordinador y todos los nodos participantes están en la misma partición, no se afecta el protocolo de commit.**



**Si el Coordinador y los participantes quedan en redes desconectadas entonces:**

Los nodos que no están con el nodo coordinador piensan que el coordinador falló y manejan el protocolo indicado en ese caso:

- No hay daño, funciona, aunque deben esperar.

El Coordinador asume que los nodos en la otra partición fallaron, y sigue el protocolo usual

- De nuevo, no hay daño