Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Transacciones

Sistemas distribuidos

Gabriel Vidal Salazar

¿Qué es una transacción?

Operaciones

- Iniciar
- Cerrar
- Abortar

Transacciones: Updates perdidos (A = $100, B = $200, C = $300)

Transaction *T*:		**Transaction *U*:**	
balance = b.getBalance(); *b.setBalance(balance*1.1);* *a.withdraw(balance/10)*		*balance = b.getBalance();* *b.setBalance(balance*1.1);* *c.withdraw(balance/10)*	
balance = b.getBalance();	$200		
		balance = b.getBalance();	$200
		*b.setBalance(balance*1.1);*	$220
*b.setBalance(balance*1.1);*	$220		
a.withdraw(balance/10)	$80		
		c.withdraw(balance/10)	$280

Source: IIC2523 - 2017/2

Transacciones: Updates perdidos (Serialización correcta)

Transaction *T*:		**Transaction *U*:**	
balance = b.getBalance()		*balance = b.getBalance()*	
*b.setBalance(balance*1.1)*		*b.setBalance(balance*1.1)*	
a.withdraw(balance/10)		*c.withdraw(balance/10)*	
balance = b.getBalance()	$200		
*b.setBalance(balance*1.1)*	$220		
		balance = b.getBalance()	$220
		*b.setBalance(balance*1.1)*	$242
a.withdraw(balance/10)	$80		
		c.withdraw(balance/10)	$278

Source: IIC2523 - 2017/2

Transacciones: Lecturas inconsistentes (A = $200, B = $200)

Transaction *V*:		**Transaction *W*:**	
a.withdraw(100) *b.deposit(100)*		*aBranch.branchTotal()*	
a.withdraw(100);	$100		
		total = a.getBalance()	$100
		total = total+b.getBalance()	$300
		total = total+c.getBalance()	
b.deposit(100)	$300	⋮	

Source: IIC2523 - 2017/2

Transacciones: Lecturas inconsistentes (Serialización correcta)

Transaction *V*:		**Transaction *W*:**	
a.withdraw(100); *b.deposit(100)*		*aBranch.branchTotal()*	
a.withdraw(100);	$100		
b.deposit(100)	$300		
		total = a.getBalance()	$100
		total = total+b.getBalance()	$400
		total = total+c.getBalance()	
		...	

Source: IIC2523 - 2017/2

Definiciones

- Equivalencia Serial
- Transacciones Serialmente Equivalentes
- Operaciones conflictivas
  - *read* vs *write*
  - *write* vs *write*

Operaciones conflictivas

Transaction *T*:	Transaction *U*:
x = read(i)	
write(i, 10)	
	y = read(j)
	write(j, 30)
write(j, 20)	
	z = read (i)

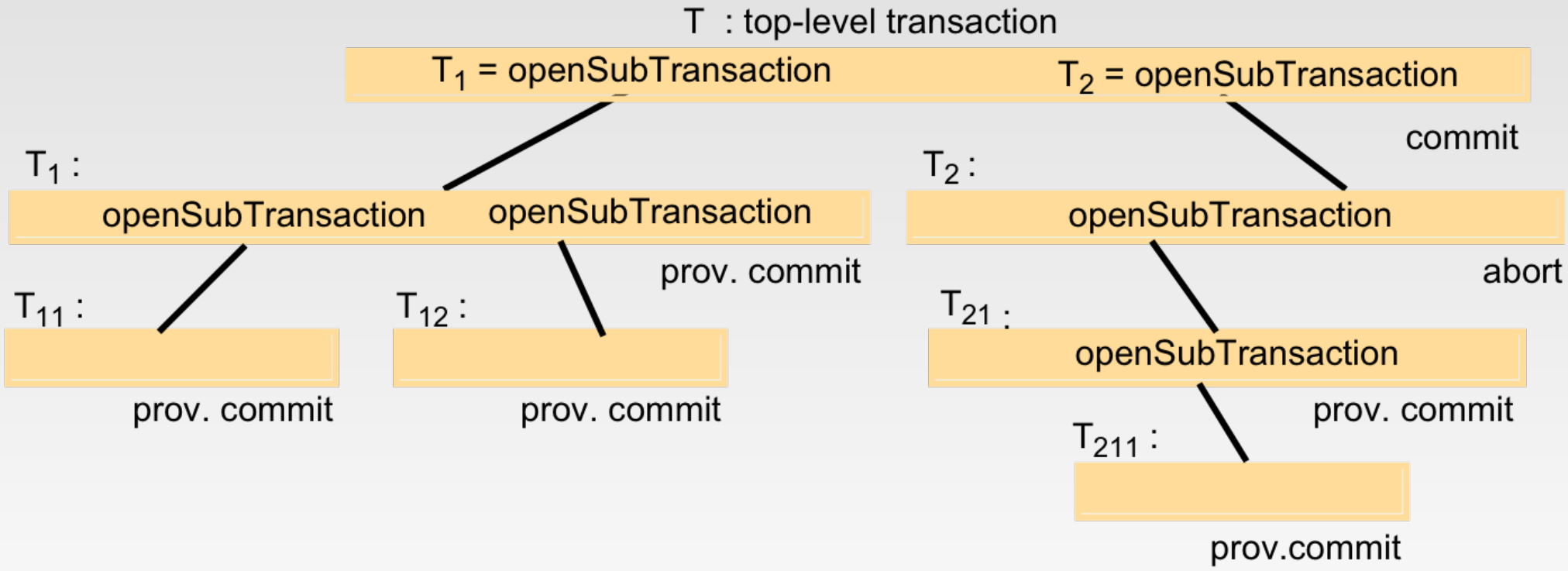
Source: IIC2523 - 2017/2

Manejo de aborts

Transaction *T*:	Transaction *U*:
a.getBalance() *a.setBalance(balance + 10)*	*a.getBalance()* *a.setBalance(balance + 20)*
balance = a.getBalance() $100	
a.setBalance(balance + 10) $110	
	balance = a.getBalance() $110
	a.setBalance(balance + 20) $130
	commit transaction
abort transaction	

Source: IIC2523 - 2017/2

Transacciones anidadas



Source: IIC2523 - 2017/2

Controlar concurrencia

- *Locks*
- Control de concurrencia optimista
- Ordenamiento de *timestamps*

Locks

Transaction *T*:		Transaction *U*:	
balance = b.getBalance() *b.setBalance(bal*1.1)* *a.withdraw(bal/10)*		*balance = b.getBalance()* *b.setBalance(bal*1.1)* *c.withdraw(bal/10)*	
Operations	Locks	Operations	Locks
openTransaction			
bal = b.getBalance()	lock *B*		
*b.setBalance(bal*1.1)*		*openTransaction*	
a.withdraw(bal/10)	lock *A*	*bal = b.getBalance()*	waits for *T*'s lock on *B*
closeTransaction	unlock *A*, *B*	• • •	
			lock *B*
		*b.setBalance(bal*1.1)*	
		c.withdraw(bal/10)	lock *C*
		closeTransaction	unlock *B*, *C*

Source: IIC2523 - 2017/2

Locks

Transaction *T*		Transaction *U*	
Operations	**Locks**	**Operations**	**Locks**
a.deposit(100);	write lock *A*		
		b.deposit(200)	write lock *B*
b.withdraw(100)			
•••	waits for *U*'s lock on *B*	*a.withdraw(200);*	waits for *T*'s lock on *A*
		•••	
•••		•••	
•••		•••	

Locks

Transaction T		Transaction U	
Operations	**Locks**	**Operations**	**Locks**
a.deposit(100);	write lock *A*		
		b.deposit(200)	write lock *B*
b.withdraw(100)			
• • •	waits for *U*'s lock on *B*	*a.withdraw(200);*	waits for T's lock on *A*
	(timeout elapses)	• • •	
T's lock on *A* becomes vulnerable, unlock *A*, abort T		• • •	
		a.withdraw(200);	write locks *A* unlock *A*, *B*

Source: IIC2523 - 2017/2

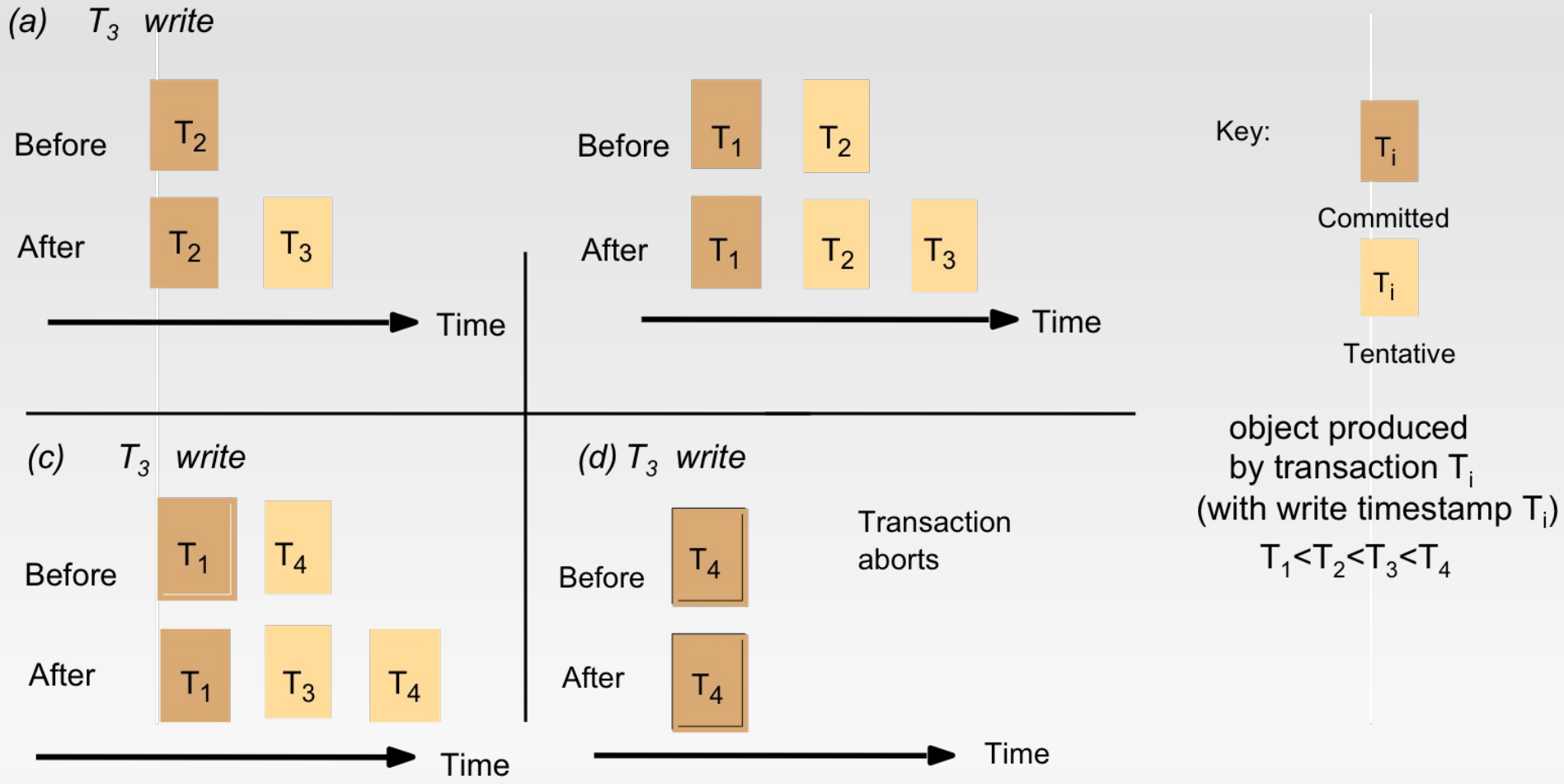
Concurrencia optimista

- Todo va a salir bien
  - No hay checkeos de conflictos en la ejecución
  - Sólo hay verificación cuando se hace commit
    - Si hay conflicto se aborta la transacción

Ordenamiento de timestamps

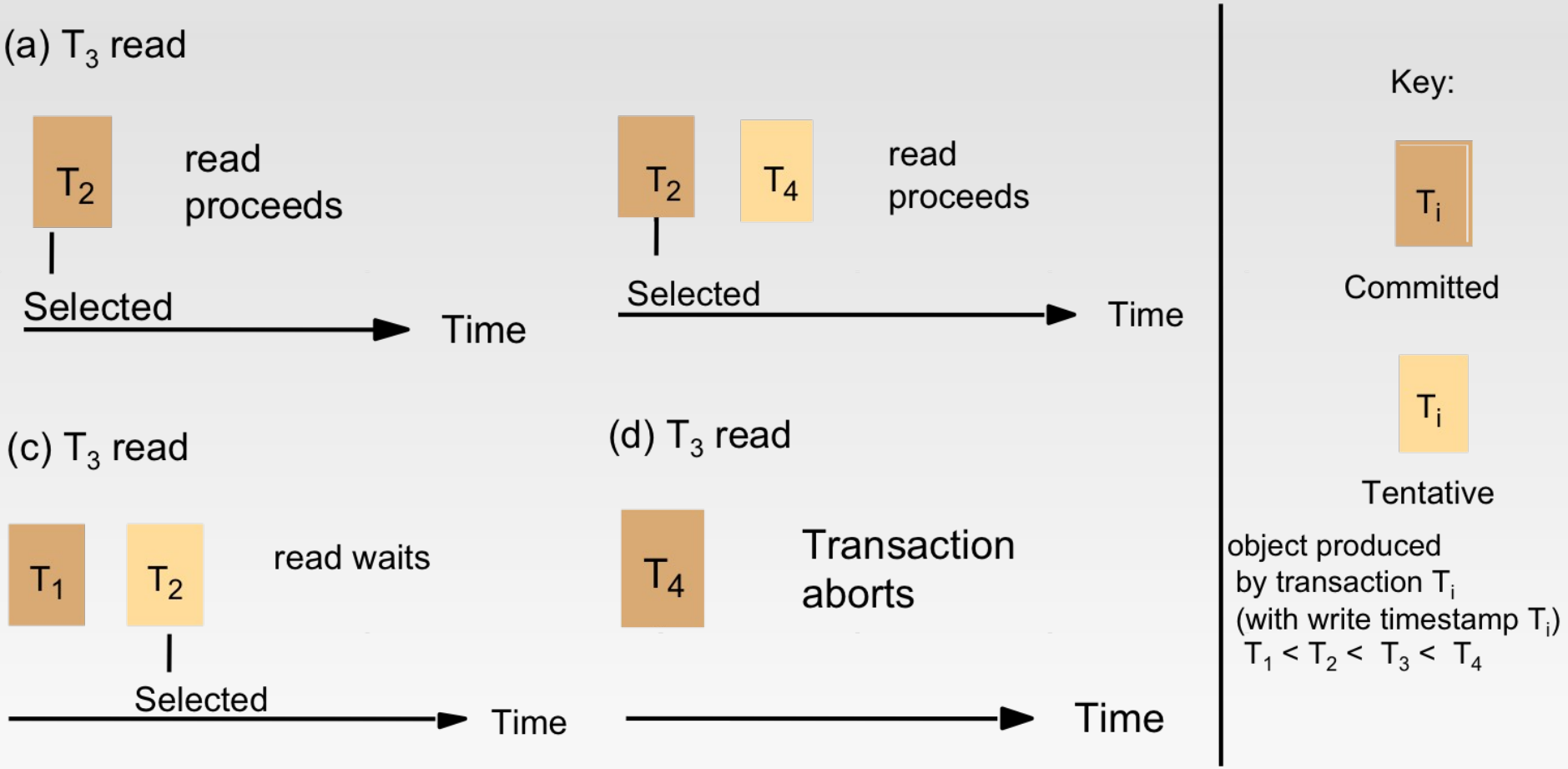
- Servidor guarda *read* y *write* más recientes para la operación sobre un objeto
- De acuerdo a su *timestamp* la nueva operación puede ser:
  - Inmediata
  - Demorada
  - Rechazada
- Operaciones conflictivas
  - Write / read, write / write, read / write

Ordenamiento de *timestamps*: writes



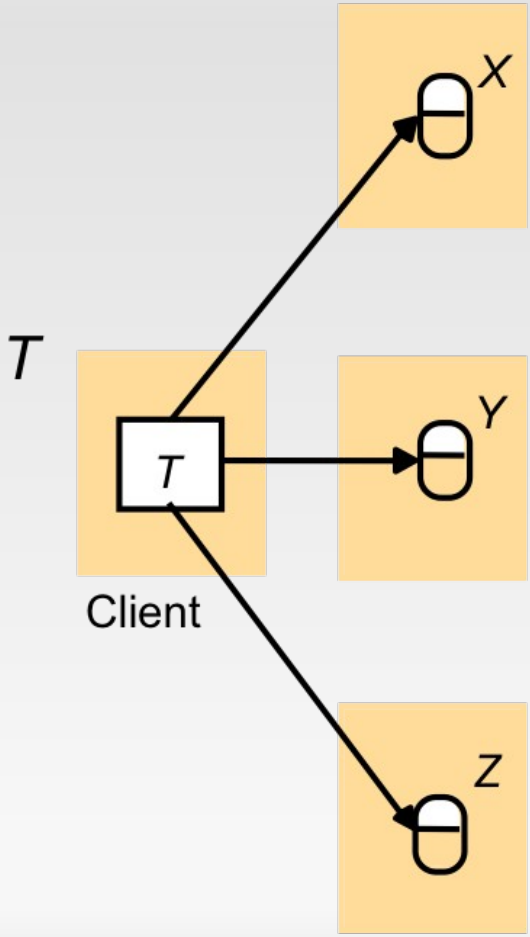
Source: IIC2523 - 2017/2

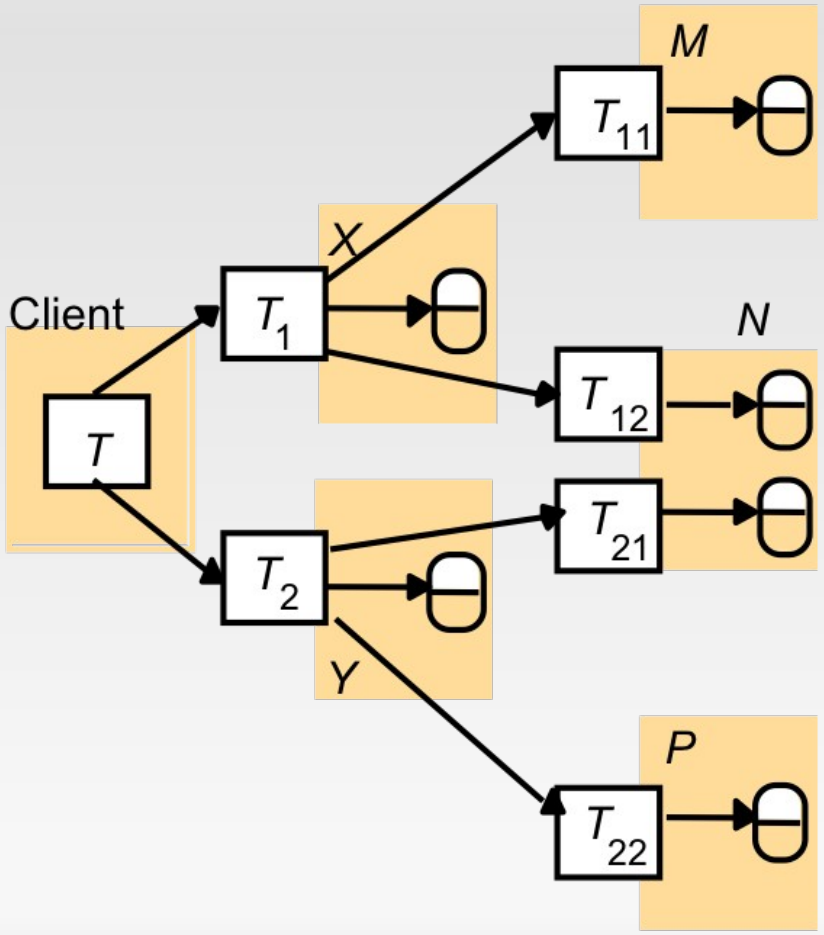
Ordenamiento de *timestamps*: reads



Source: IIC2523 - 2017/2

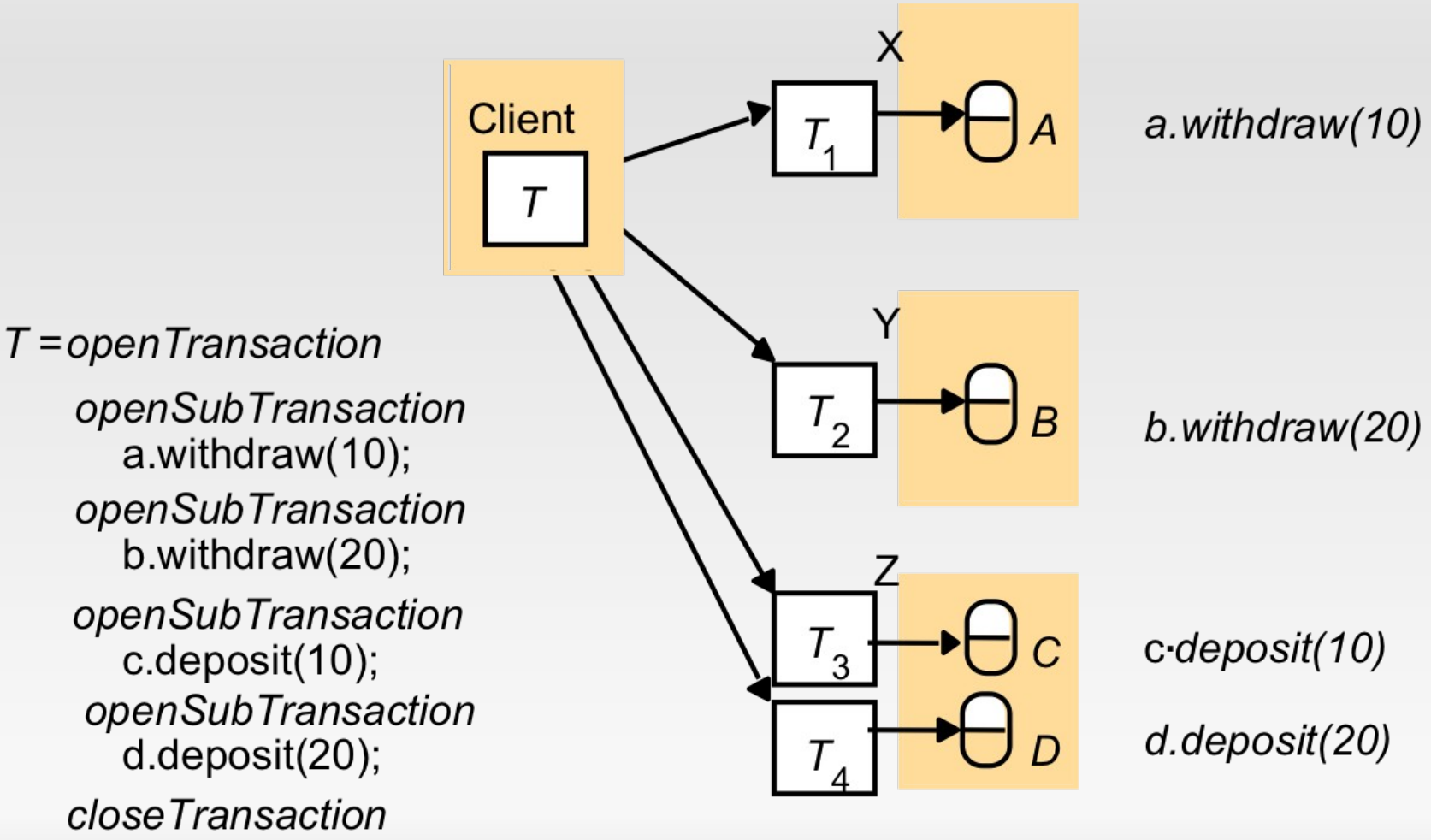
Transacciones distribuidas





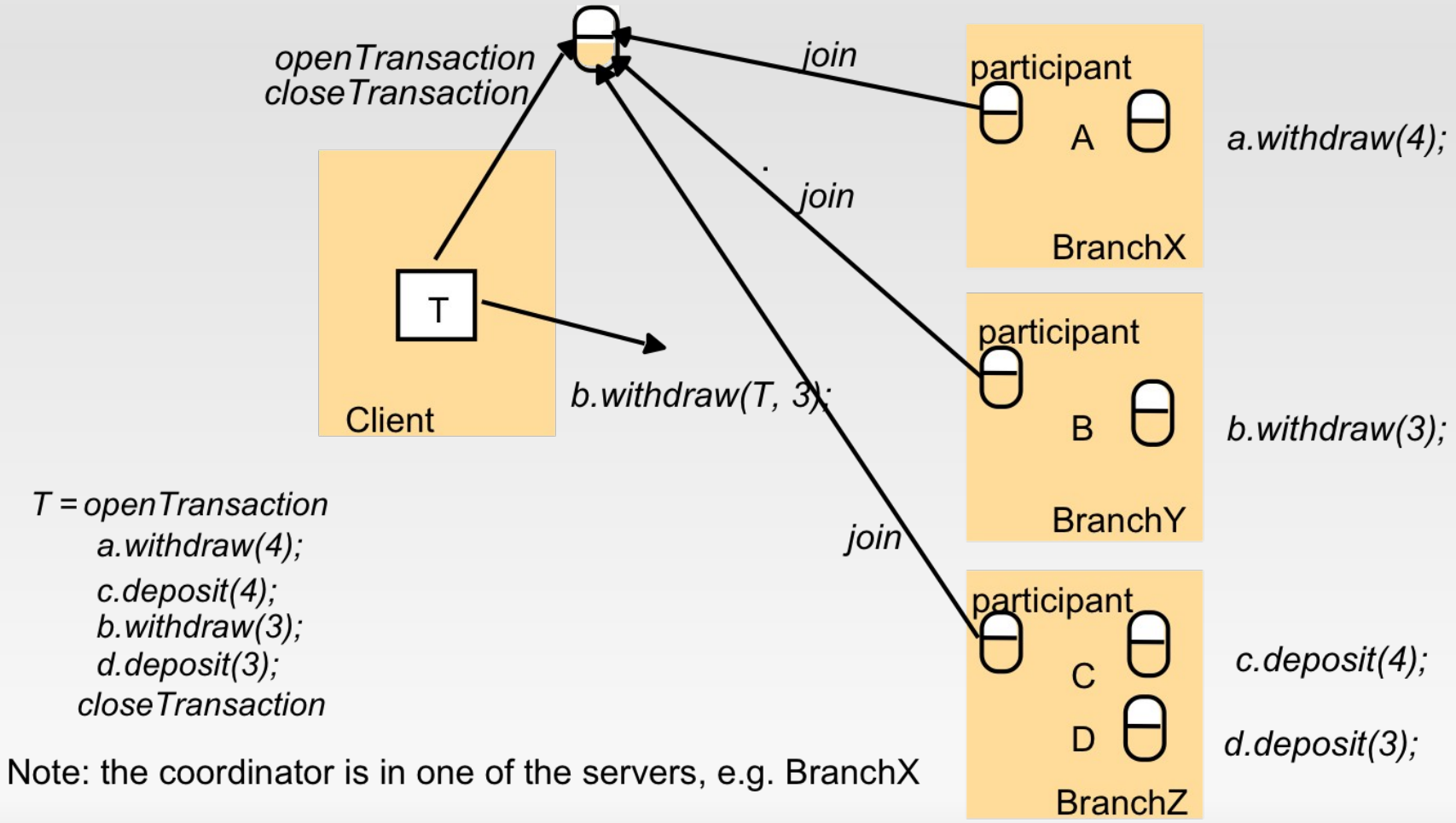
Source: IIC2523 - 2017/2

Transacciones distribuidas y anidadas



Source: IIC2523 - 2017/2

Transacciones distribuidas y anidadas



Source: IIC2523 - 2017/2

Commit protocols

- *One-phase commit*
- *Two-phase commit*