Replicazione e Consistenza

Replicazione

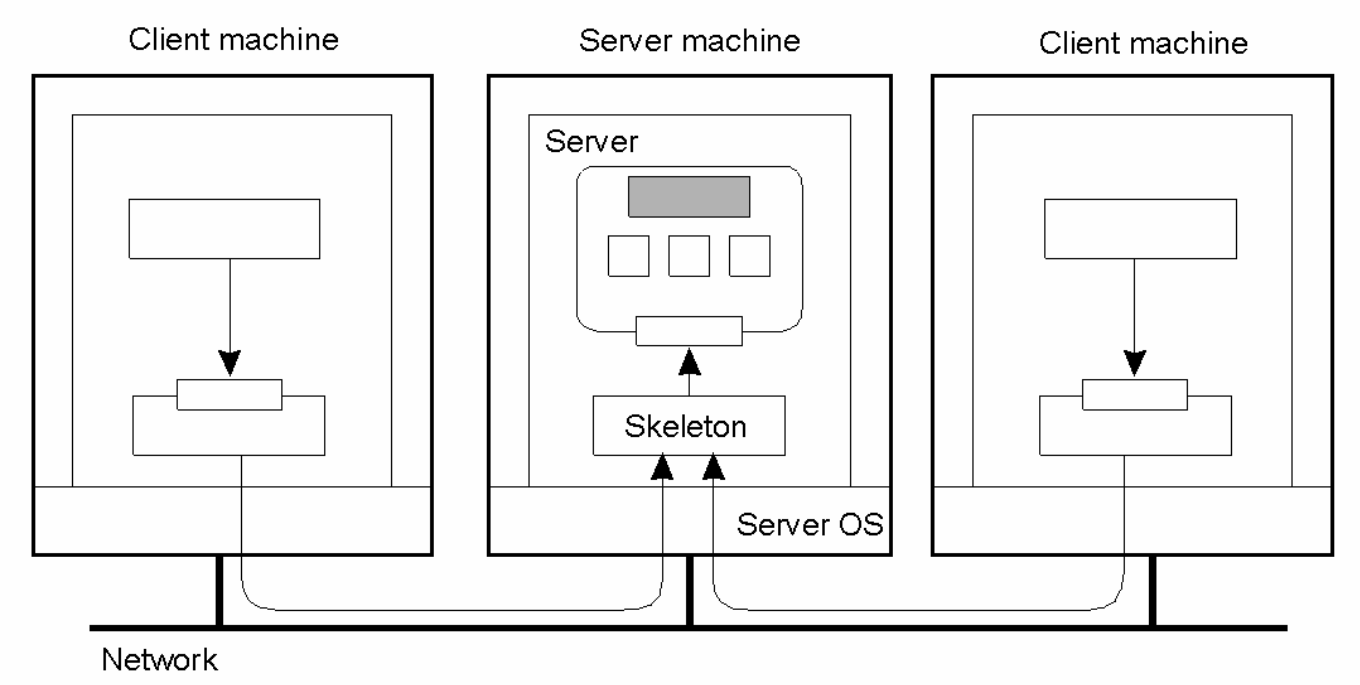
La Replicazione di dati è usata per migliorare:

- le performance
- l'affidabilità.

Tuttavia, le **repliche** devono essere mantenute **in uno stato consistente.**

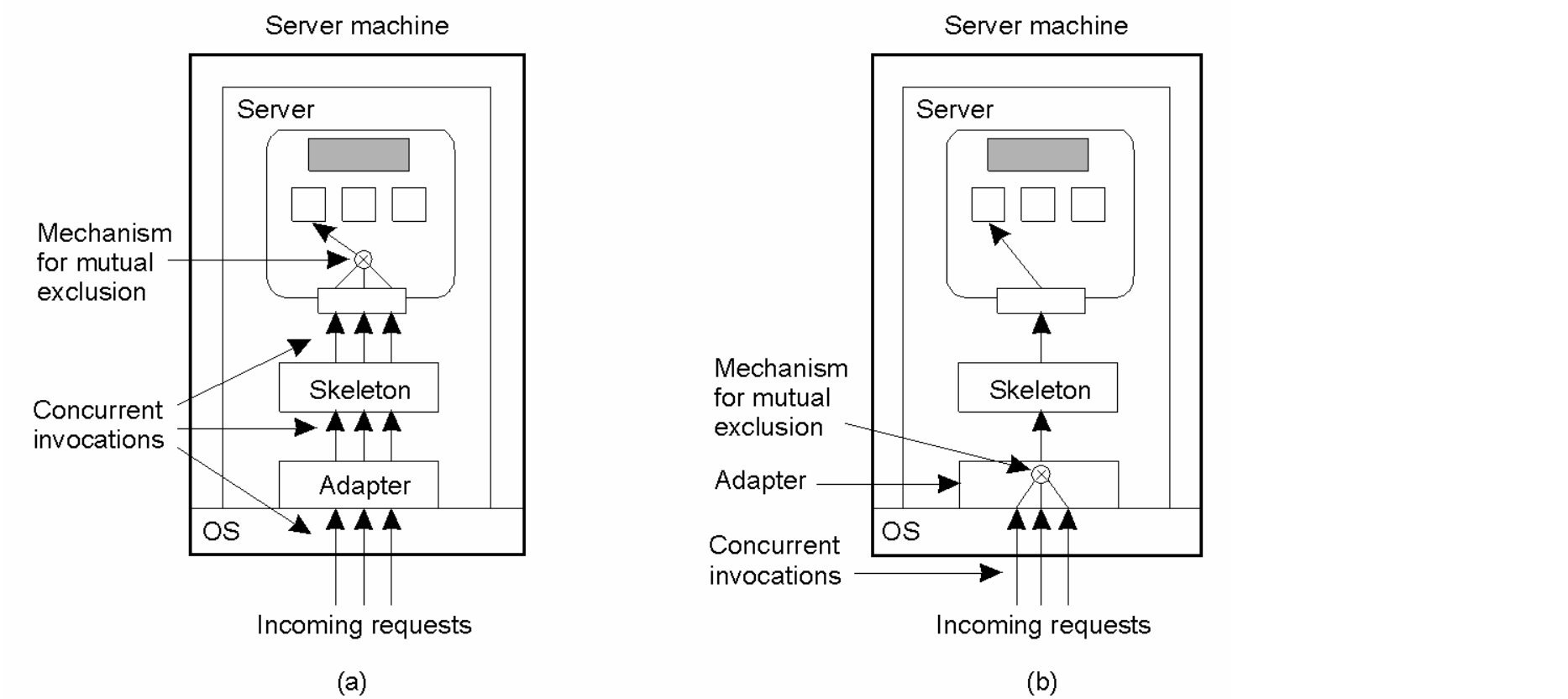
Modelli Efficienti di consistenza sono complessi da implementare.

Replicazione di Oggetti (1)



- Organizzazione di un distributed remote object condiviso da due clienti su nodi differenti.
- Come accedere all'oggetto condiviso?

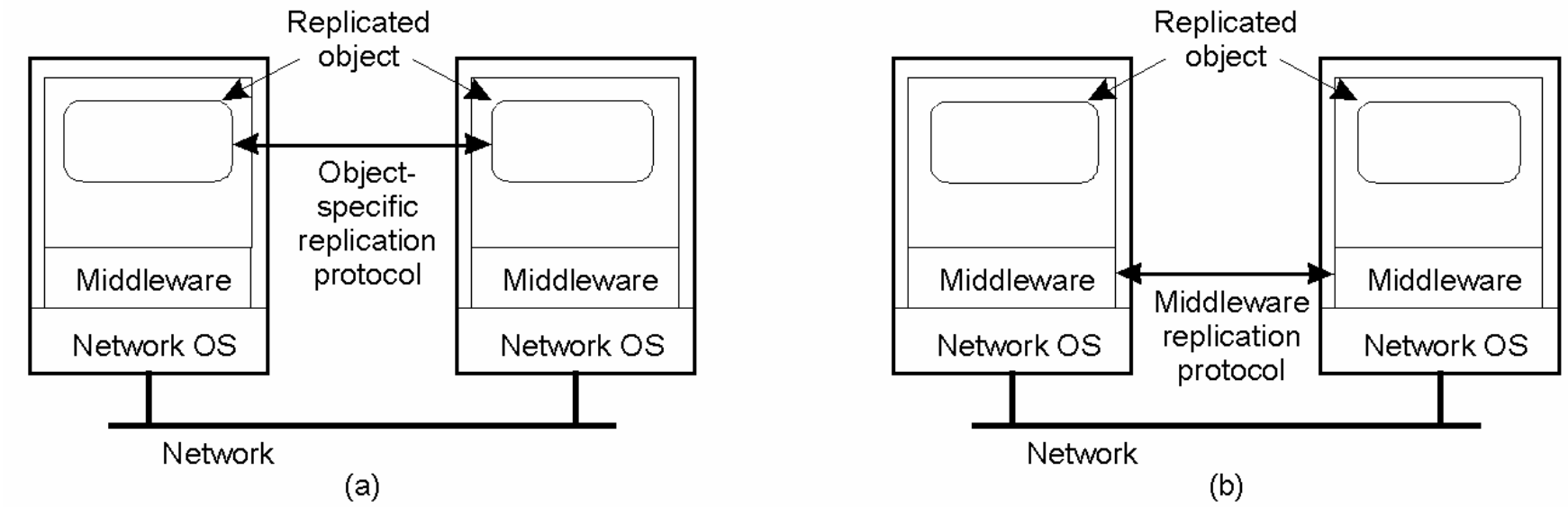
Replicazione di Oggetti (2)



(a) Un oggetto remoto capace di gestire invocazioni concorrenti (in proprio).

(b) Un oggetto remoto che usa un object adapter per gestire invocazioni concorrenti

Replicazione di Oggetti (3)



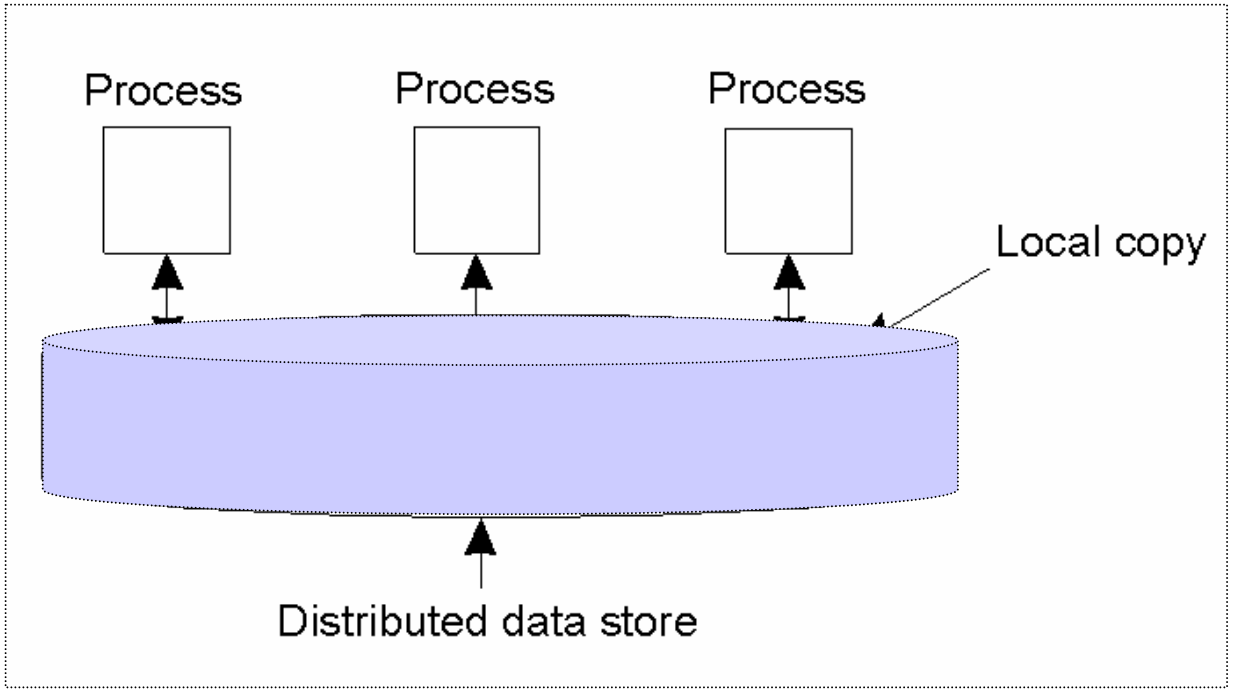
Come gestire la sincronizzazione delle repliche?

(a) Un sistema distribuito per oggetti distribuiti **replication-aware** (*Globe*, *SOS*).

(b) Un sistema distribuito responsabile per la gestione delle repliche (*Piranha*).

Modelli di Consistenza Data Centric

L'organizzazione generale di un **logical data store** fisicamente distribuito e replicato tra più macchine.

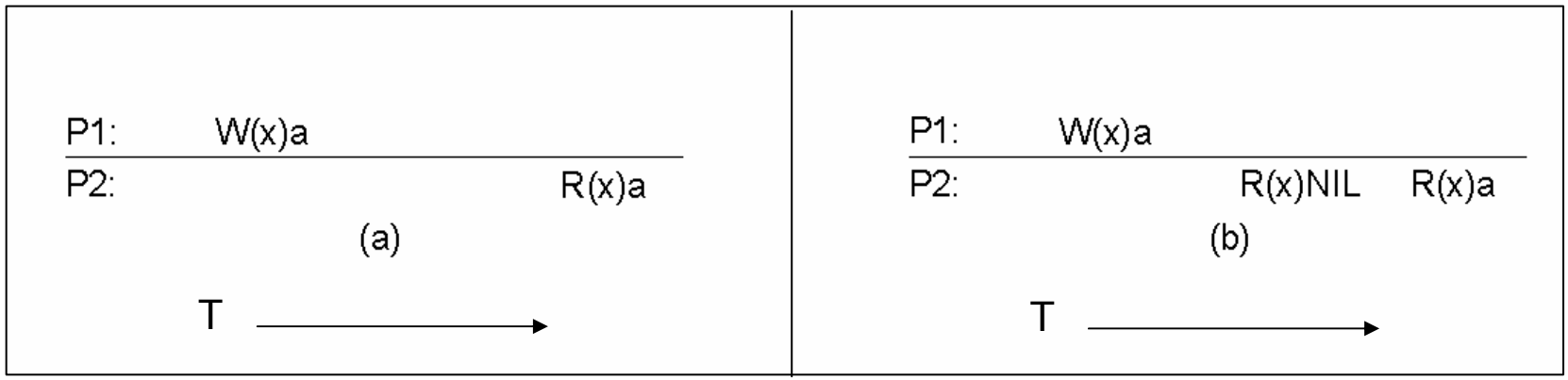


Un **modello di consistenza** è un contratto tra processi e il data store: **se i processi rispettano alcune regole il data store contiene valori corretti**.

Strict Consistency

Qualsiasi lettura su un dato ***x*** *ritorna un valore corrispondente al* ***risultato della più recente write su x.***

Un tempo globale è necessario.

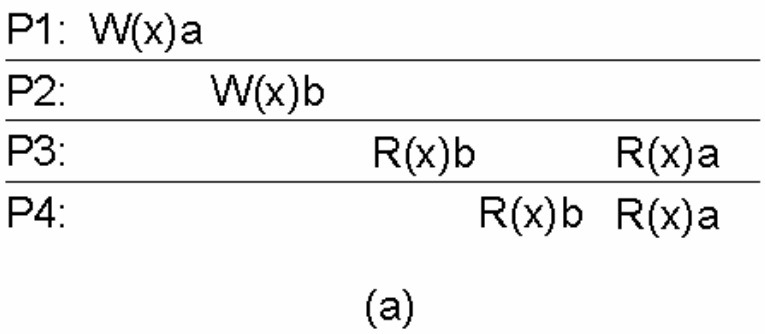


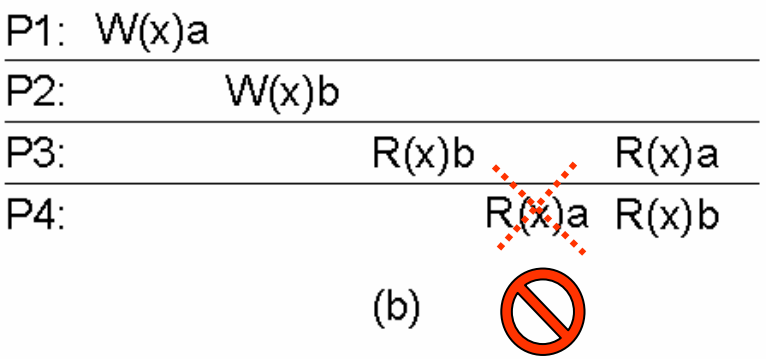
Comportamento di due processi che operano sullo stesso dato x
(a) Con memoria strictly consistent.
(b) Con memoria non strictly consistent.

Nella consistenza stretta (rigorosa) le write sono viste da tutti i processi **istantaneamente**.

Consistenza Sequenziale

Il risultato di una qualsiasi esecuzione è uguale a quello ottenuto se le operazioni di tutti i processi sul data store fossero eseguiti ***in qualche ordine sequenziale*** *e le operazioni di ogni singolo processo nella sequenza sono comunque* ***fatte nell'ordine indicato dal suo programma***





(a) Un data store sequenzialmente consistente.

(b) Un data store non sequenzialmente consistente.

Linearizzabilità (1)

Il risultato di una qualsiasi esecuzione è uguale a quello ottenuto se le operazioni di tutti i processi sul data store fossero eseguiti ***in qualche ordine sequenziale*** *e le operazioni di ogni singolo processo nella sequenza sono comunque* ***fatte nell'ordine indicato dal suo programma***
Inoltre, se $ts_{OP1}(x) < ts_{OP2}(y)$*, allora l'operazione OP1(x) deve precedere l'operazione OP2(y) nella sequenza.*

- Si usa un timestamp da assegnare ad ogni operazione.
- Un data store linearizzabile è anche sequenzialmente consistente.

Linearizzabilità (2)

Tre processi in esecuzione concorrentemente

Process P1	**Process P2**	**Process P3**
x = 1; print (y, z);	y = 1; print (x, z);	z = 1; print (x, y);

x = 1;
print (y, z);
y = 1;
print (x, z);
z = 1;
print (x, y);
 	
Prints: 001011
 	
Signature: 001011
	x = 1;
	y = 1;
	print (x,z);
	print(y, z);
	z = 1;
	print (x, y);
	
	Prints: 101011
	
	Signature: 101011
		y = 1;
		z = 1;
		print (x, y);
		print (x, z);
		x = 1;
		print (y, z);
		
		Prints: 010111
		
		Signature: 010111
			y = 1;
			x = 1;
			z = 1;
			print (x, z);
			print (y, z);
			print (x, y);
			
			Prints: 111111
			
			Signature: 111111

Quattro sequenze di esecuzione valide per i processi.

Consistenza Causale (1)

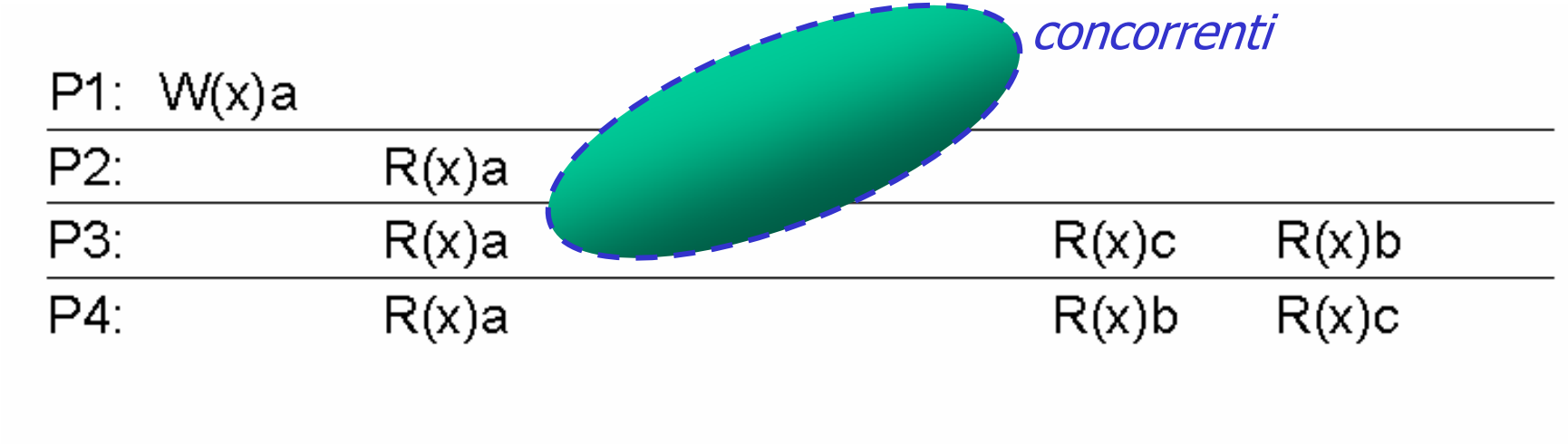
*Le operazioni di write che **potenzialmente** sono **causalmente correlati** devono essere viste da tutti processi nello stesso ordine. Write concorrenti possono essere viste in ordine differente su macchine differenti.*

Se due processi simultaneamente scrivono su due variabili, le due ***write*** non sono potentialmente causalmente correlati (***write*** concorrenti).

Una ***read*** seguita da una ***write*** possono essere potentialmente causalmente correlati:

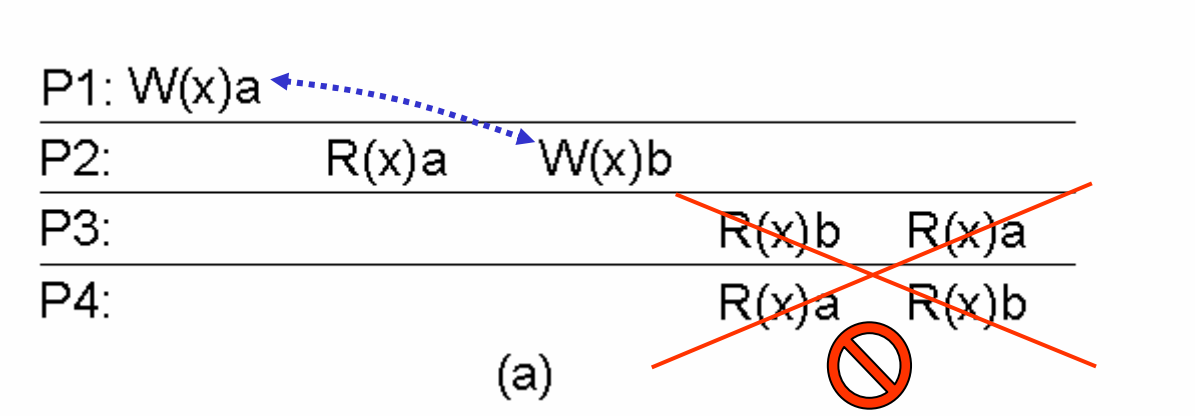
Se P1 scrive x e P2 legge x e usa il suo valore per scrivere y, la lettura di x e la scrittura di y sono potenzialmente causalmente correlati.

Consistenza Causale (2)

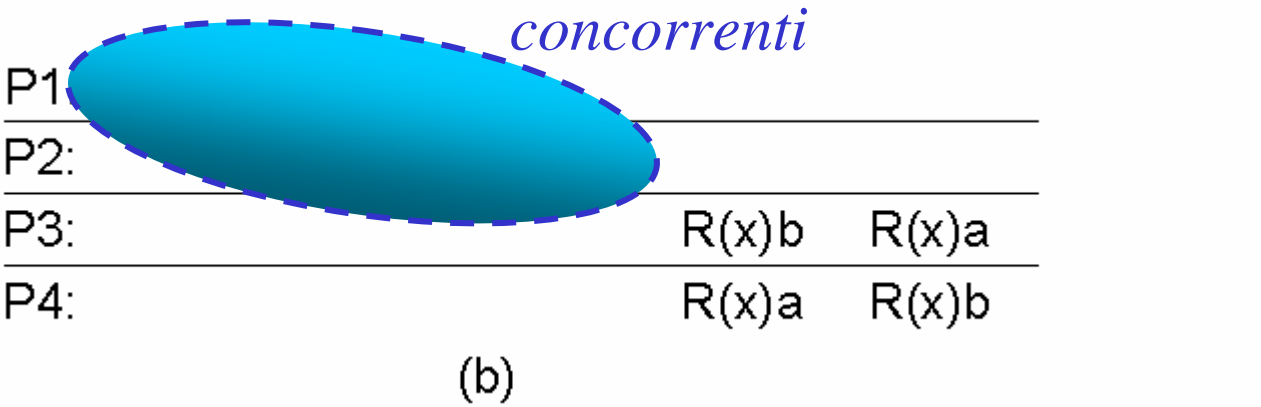


Questa sequenza è permessa in un data store causally-consistent, ma non in una memoria sequentially o strictly consistent: ***write concorrenti possono essere viste in ordine differente***.

Consistenza Causale (3)



(a) Una violazione della memoria causalmente consistente.



(b) Una sequenza corretta di eventi in una memoria consistente causalmente.

Consistenza FIFO (1)

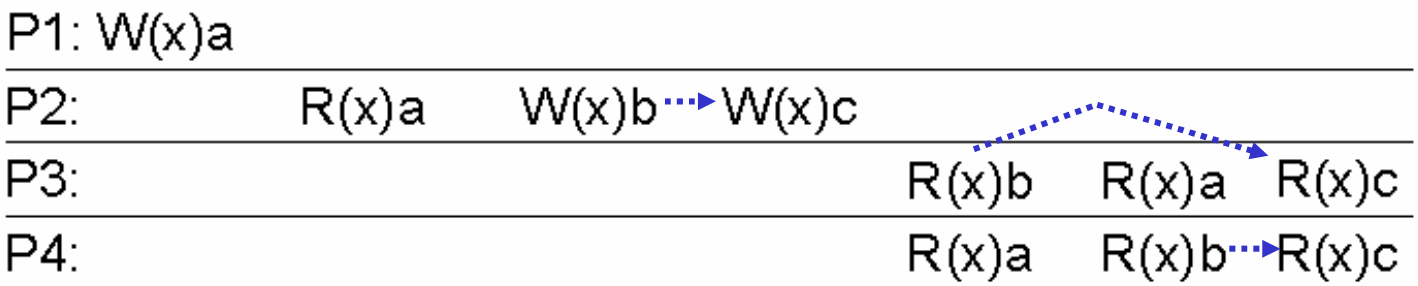
Le write fatte da un singolo processo sono viste da tutti gli altri processi nell'ordine in cui queste vengono effettuate, ma le write effettuate da processi differenti possono essere viste in ordine differente da processi differenti.

Esempi: In un sistema di 5 processi:

- 3 modifiche effettuate da write eseguite dallo stesso processo ($P4_{W(x)}$, $_{W(y)}$, $_{W(z)}$) devono essere viste nello stesso ordine dai processi P1, P2, P3 e P5 (e P4).

- 3 modifiche effettuate da write eseguite da tre differenti processi ($P4_{W(x)}$, $P1_{W(y)}$, $P5_{W(z)}$) possono essere viste in ordine diverso dai 5 processi.

Consistenza FIFO (2)



Una sequenza valida di eventi secondo la consistenza FIFO

La consistenza FIFO è semplice da implementare

Consistenza FIFO (3)

Processo P1	Processo P2	Processo P3
x = 1; print (y, z);	y = 1; print (x, z);	z = 1; print (x, y);

x = 1;
print (y, z);
y = 1;
print(x, z);
z = 1;
print (x, y);
 	
Prints: **00**
	x = 1;
	y = 1;
	print(x, z);
	print (y, z);
	z = 1;
	print (x, y);
	
	Prints: **10**
		y = 1;
		print (x, z);
		z = 1;
		print (x, y);
		x = 1;
		print (y, z);
		
		Prints: **01**

I valori di x,y e z sono inizializzati a 0.

Esecuzione degli statement vista dai tre processi.

Gli statement in grassetto sono quelli che generano l'output mostrato.

Consistenza FIFO (4)

I valori di x e y sono inizializzati a 0.

Processo P1	**Processo P2**
x = 1; if (y == 0) kill (P2);	y = 1; if (x == 0) kill (P1);

Due processi concorrenti che possono essere terminati con consistenza FIFO (perchè ognuno può "vedere" le operazioni dell'altro in un'ordine diverso).

Modelli di Consistenza

Consistenza	Descrizione
Strict	Ordinamento temporale assoluto di tutti gli accessi.
Sequenziale	Tutti i processi vedono tutti gli accessi condivisi nello stesso ordine. Gli accessi non sono ordinati temporalmente
Linearizzabilità	Tutti i processi vedono tutti gli accessi condivisi nello stesso ordine. Alcuni accessi sono ordinati temporalmente secondo un timestamp globale
Causale	Tutti i processi vedono tutti gli accessi causalmente correlati nello stesso ordine.
FIFO	Tutti i processi vedono tutte le write di un altro processo nello stesso ordine in cui sono eseguite. Write di processi differenti possono non essere viste nello stesso ordine.

Weak Consistency (1)

I processi fanno uso di **variabili sincronizzate** che permettono di sincronizzare tutte le copie locali del data store tramite l'operazione

synchronize(S)

Proprietà:

- *Gli accessi a variabili sincronizzate associate ad un data store sono sequenzialmente consistenti.*
- *Nessuna operazione su una variabile sincronizzata può essere eseguita finchè tutte le precedenti write non siano state completate su tutte le copie.*
- *Nessuna operazione di read o write su un dato è permessa finchè tutte le precedenti operazioni sulle variabili sincronizzate non siano state eseguite.*

Weak Consistency (2)

La weak consistency rafforza la consistenza di un **gruppo di operazioni** non di singole read o write.

Un frammento di programma in cui alcune variabili possono essere mantenute in registri. Quando **f** viene eseguita, **a** e **b** devono essere tenute in memoria.

```
int a, b, c, d, e, x, y;                /* variables */
int *p, *q;                             /* pointers */
int f( int *p, int *q);                 /* function prototype */

a = x * x;                              /* a stored in register */
b = y * y;                              /* b as well */
c = a*a*a + b*b + a * b;                /* used later */
d = a * a * c;                          /* used later */
p = &a;                                 /* p gets address of a */
q = &b                                  /* q gets address of b */
e = f(p, q)                             /* function call */
```

Weak Consistency (3)

P1: W(x)a	W(x)b	S			
P2:			R(x)a	R(x)b	S
P3:			R(x)b	R(x)a	S

(a)

Una sequenza di eventi valida secondo la weak consistency.

P1: W(x)a	W(x)b	S		
P2:			S	R(x)a

(b)

Una sequenza di eventi non valida secondo la weak consistency.

Release Consistency (1)

Sono definite due operazioni :

- **acquire :** per segnalare l'ingresso in una regione critica, e aggiornare tutte copie dei dati replicati
- **release :** per segnalare l'uscita da una regione critica.

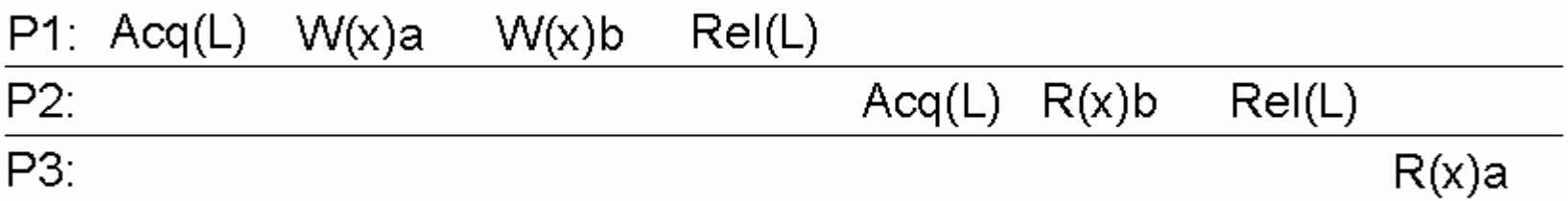
Queste operazioni sostituiscono e specializzano l'operazione synchronize della weak consistency differenziando tra sincronizzazione prima di leggere i dati o dopo averli scritti.

Release Consistency (2)

Regole da rispettare per garantire la release consistency di un data store distribuito:

- *Prima che una operazione di read e write su dati condivisi sia eseguita, tutte le precedenti* ***acquires*** *eseguite dal processo devono essere eseguite con successo.*
- *Prima che una* ***release*** *sia eseguita, tutte le precedenti read e write devono essere completati da un processo.*
- *Accessi a variabili di sincronizzazione sono FIFO consistenti (la consistenza sequenziale non è richiesta).*

Release Consistency (2)



Una sequenza di eventi valida per la release consistency.

Nota bene: **P3** non esegue una acquire prima di leggere i dati, per cui la lettura del valore **a** è permessa.

Entry Consistency (1)

- **Ogni dato deve essere associato a qualche variabile di sincronizzazione.**
- Liste di dati condivisi sono associati ad una variabile di sincronizzazione.
- acquire e release sono più efficienti.
- Accessi ad insiemi disgiunti di dati condivisi possono essere concorrenti.
- Il processo che ha acquisito una variabile di sincronizzazione diviene il proprietario di quella variabile.
- Un processo che vuole acquisire una variabile di sincronizzazione deve richiederla al proprietario corrente insieme al valore aggiornato dei dati associati.

Entry Consistency (2)

Regole da rispettare per garantire la entry consistency di un data store distribuito :

1. *Un accesso tramite acquire ad una variabile di sincronizzazione da parte di un processo non può essere eseguito finchè tutte le update sui dati condivisi controllati (guarded) non siano state eseguite per quel processo.*

   Questo significa che quando una acquire è eseguita, tutte le modifiche remote sui dati controllati (guarded) dalla variabile associata alla acquire devono essere visibili.

Entry Consistency (3)

2 *Prima che un accesso in modo esclusivo ad una variabile di sincronizzazione da parte di un processo è permesso, nessun altro processo può possedere la variabile di sincronizzazione neanche in modo non esclusivo.*

Cioè, prima di aggiornare un dato condiviso, un processo deve entrare in una regione critica in modo esclusivo per evitare che un altro processo possa aggiornare il dato.

Entry Consistency (4)

3 *Dopo che sia stato eseguito un accesso in modo esclusivo ad una variabile di sincronizzazione, qualsiasi altro accesso di un processo in modo non esclusivo alla variabile di sincronizzazione non può essere eseguito finchè il proprietario di quella variabile non abbia registrato le modifiche.*

Cioè: se un processo vuole entrare in una regione critica in modo non esclusivo, deve controllare con il proprietario della variabile di sincronizzazione di accedere a copie aggiornate dei dati.

Entry Consistency (5)

P1:	Acq(Lx)	W(x)a	Acq(Ly)	W(y)b	Rel(Lx)	Rel(Ly)					
P2:							Acq(Lx)	R(x)a			R(y)NIL
P3:									Acq(Ly)	R(y)b	

Una sequenza di eventi valida per la entry consistency.

Sintesi dei Modelli di Consistenza

Consistenza	Descrizione
Strict	Ordinamento temporale assoluto di tutti gli accessi.
Sequenziale	Tutti i processi vedono tutti gli accessi condivisi nello stesso ordine. Gli accessi non sono ordinati temporalmente
Linearizzabilità	Tutti i processi vedono tutti gli accessi condivisi nello stesso ordine. Alcuni accessi sono ordinati temporalmente secondo un timestamp globale (nonunico)
Causale	Tutti i processi vedono tutti gli accessi causalmente correlati nello stesso ordine.
FIFO	Tutti i processi vedono tutte le write di un altro processo nello stesso ordine in cui sono eseguite. Write di processi differenti possono non essere visti nello stesso ordine.

(a)

Consistenza	Descrizione
Weak	I dati condivisi possono essere considerati consistenti solo dopo una sincronizzazione
Release	I dati condivisi possono essere considerati consistenti all'uscita da una regione critica
Entry	I dati condivisi riguardanti una regione critica possono essere considerati consistenti all'ingresso di una regione critica.

(b)

(a) Modelli di consistenza che non fanno uso di operazioni di sincronizzazione.

(b) Modelli che fanno uso di operazioni di sincronizzazione.

Modelli Client-Centric: Eventual Consistency

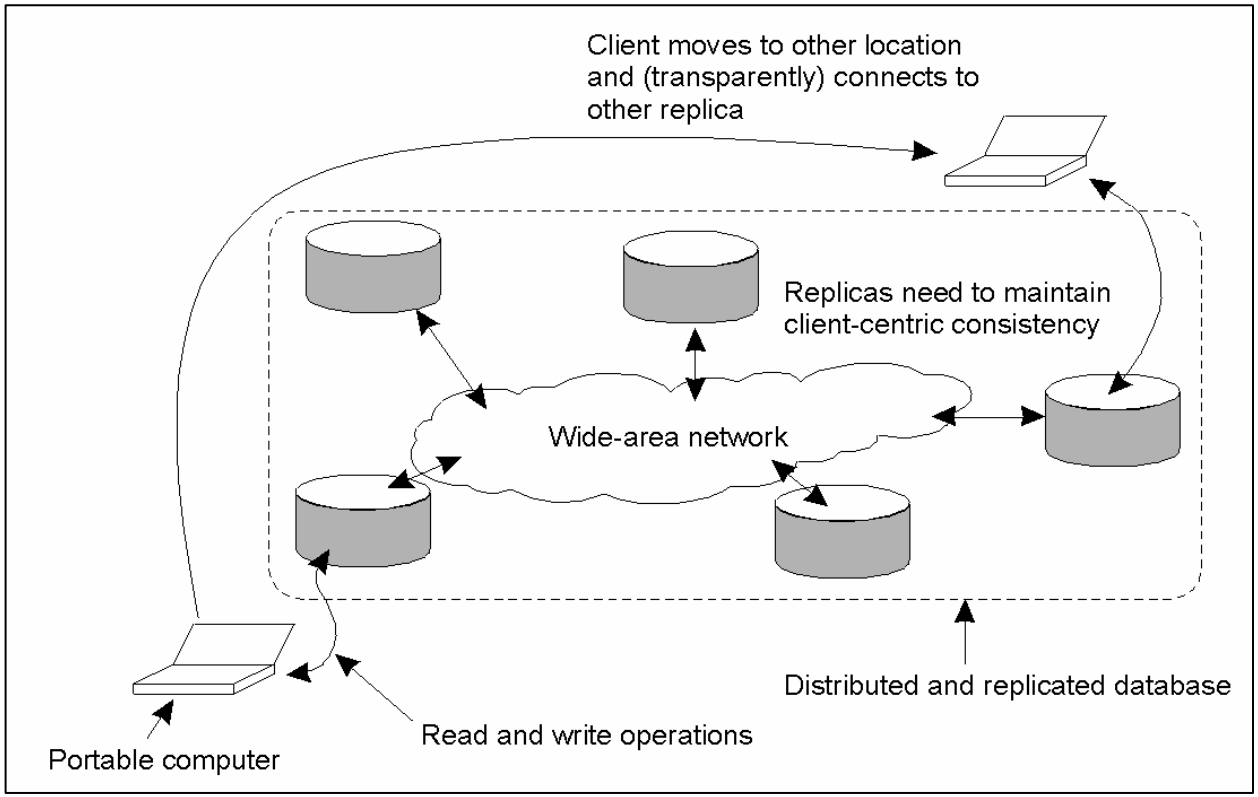
- Se gli aggiornamenti non vengono eseguiti per lungo tempo, le repliche diventono inconsistenti e diventeranno consistenti lentamente quando verranno eseguiti gli aggiornamenti
- Alcuni sistemi possono tollerare questa situazione (siti Web, DNS).

- Questa forma di consistenza è detta **eventual consistency**.

- Può funzionare se i clienti accedono una replica. Possono sorgere problemi se vengono accedute più repliche.

Eventual Consistency

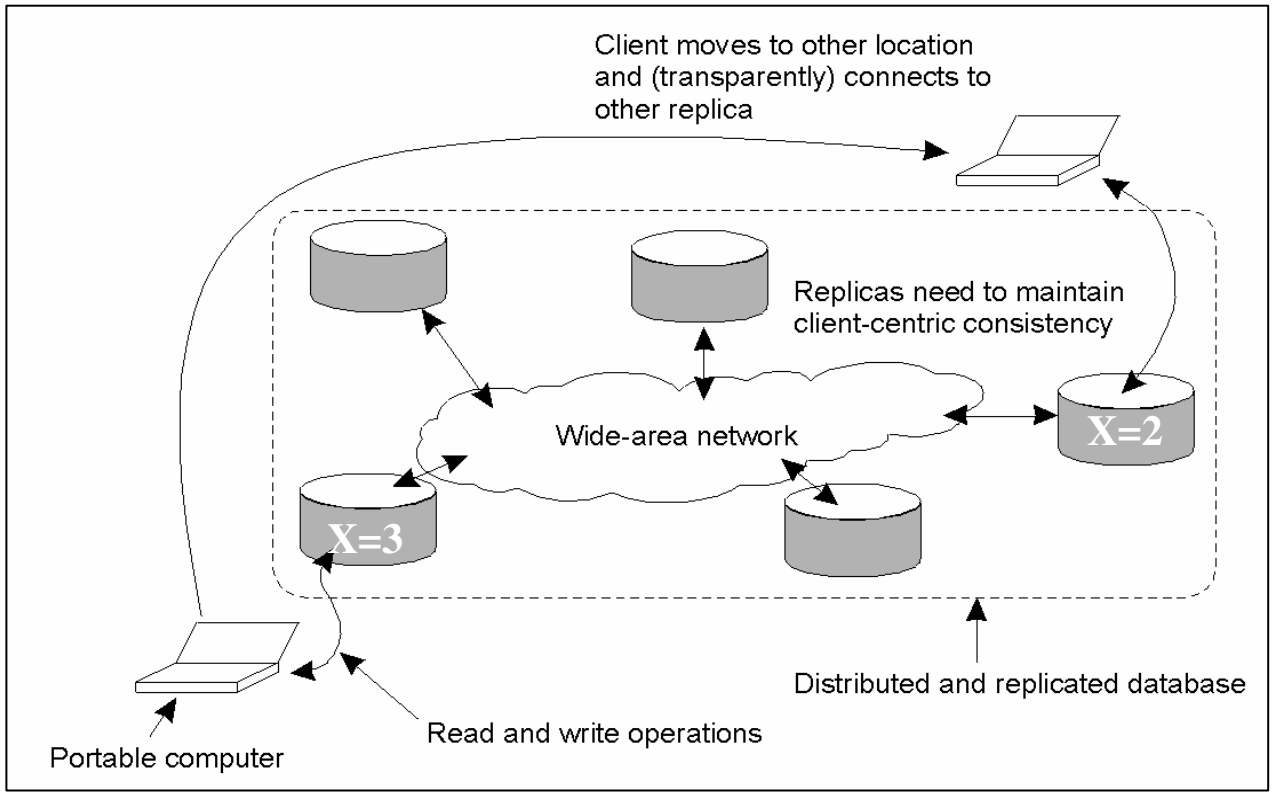
In un ambiente mobile, accessi a repliche differenti possono portare ad inconsistenze.



Modello di un utente mobile che accede differenti repliche di un database distribuito.

Client-Centric Consistency

Se un utente si sposta può accedere a dati inconsistenti.

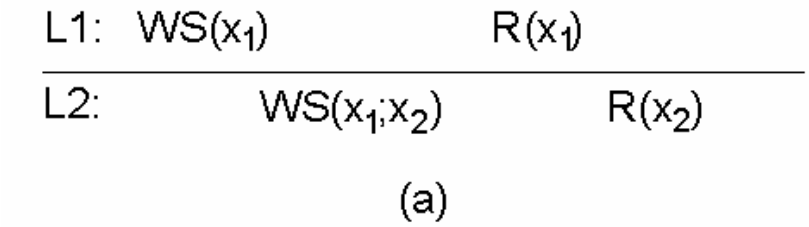


In questi casi possono essere usati modelli di consistenza **Client-centric** che si preoccupano di garantire la consistenza degli accessi di un singolo cliente

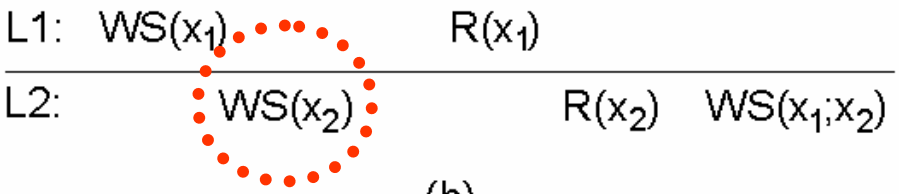
Monotonic Reads

Letture (Read) successive da parte di un processo di un dato x ritornano lo stesso valore o un valore più recente.

Le operazioni read eseguite da un singolo processo P in due differenti copie locali dello stesso data store.



(a)



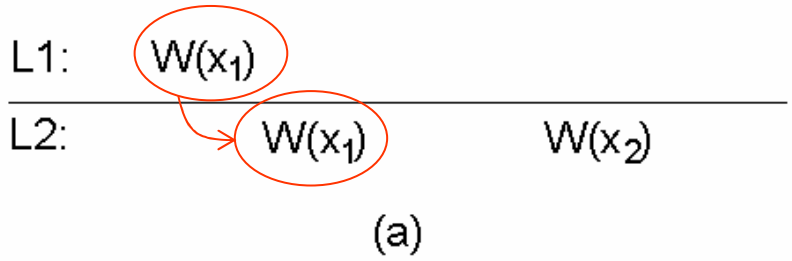
(b)

(a) Un data store monotonic-read consistent

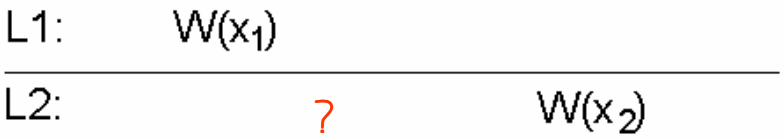
(b) Un data store non monotonic-read consistent.

Monotonic Writes

Una operazione di write da parte di un processo su un dato x è completata prima di qualsiasi successiva write su x da parte dello stesso processo.



(a)



(b)

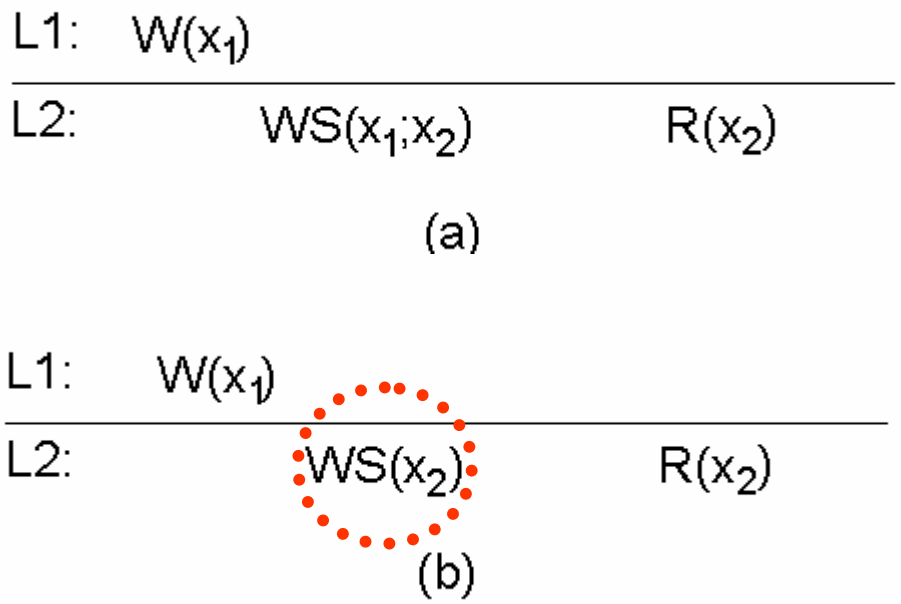
Le operazioni di write eseguite da un processo *P* su due differenti copie locali dello stesso data store

a) Un data store monotonic-write consistent.

b) Un data store non monotonic-write consistent.

Read Your Writes

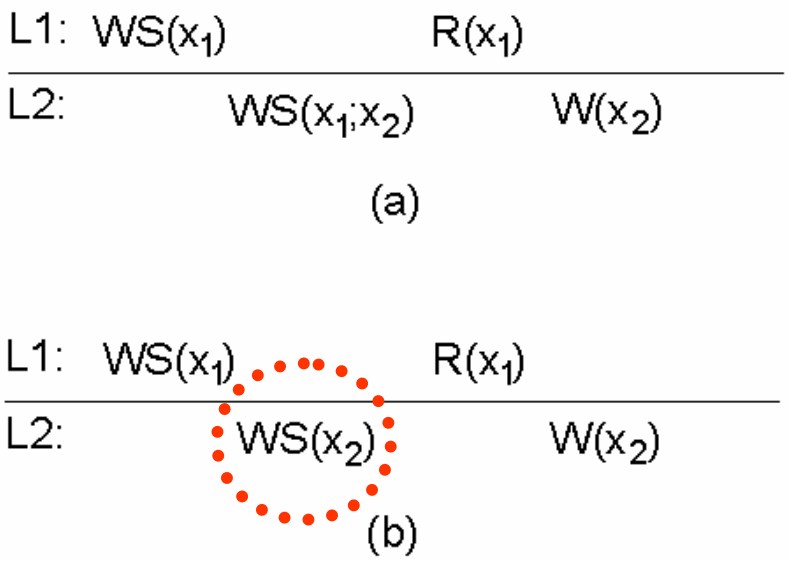
L'effetto di una write di un processo sul dato x sarà sempre visibile da una successiva read su x da parte dello stesso processo.



(a) Un data store che garantisce la consistenza read-your-writes.
(b) Un data store che non garantisce la consistenza read-your-writes.

Writes Follow Reads

Una write di un processo sul dato x dopo una precedente read su x viene effettuata sullo stesso valore letto o su un valore più recente di x.



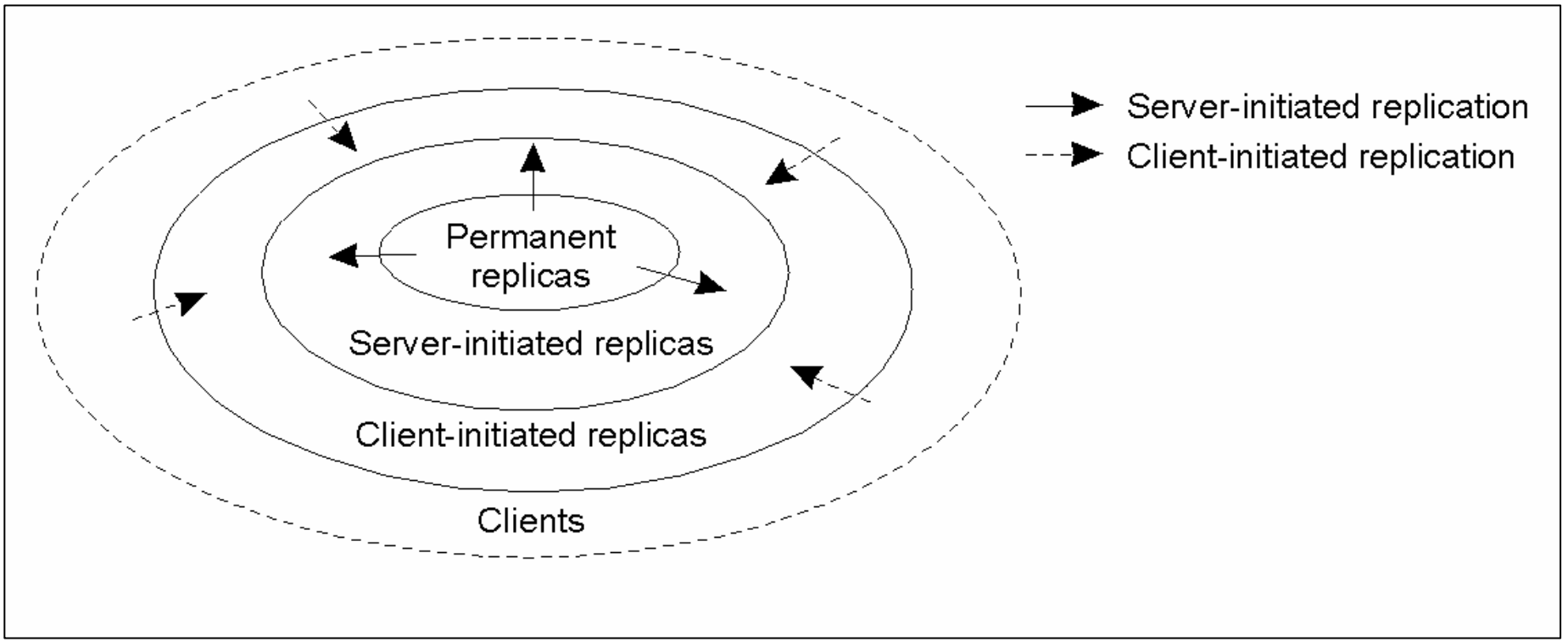
(a) Un data store che rispetta la consistenza writes-follow-reads

(b) Un data store che non rispetta la consistenza writes-follow-reads

PROTOCOLLI DI DISTRIBUZIONE

- Come distribuire gli aggiornamenti delle repliche?
- Sono necessari protocolli di consistenza per la distribuzione delle repliche
- Differenti modi di implementazione independenti dal modello di consistenza supportato.
  1. Replica placement
  2. Update propagation
  3. Epidemic protocols

PROTOCOLLI DI DISTRIBUZIONE delle REPLICHE



L'organizzazione logica di differenti tipi di copie di un data store.

Repliche Permanenti

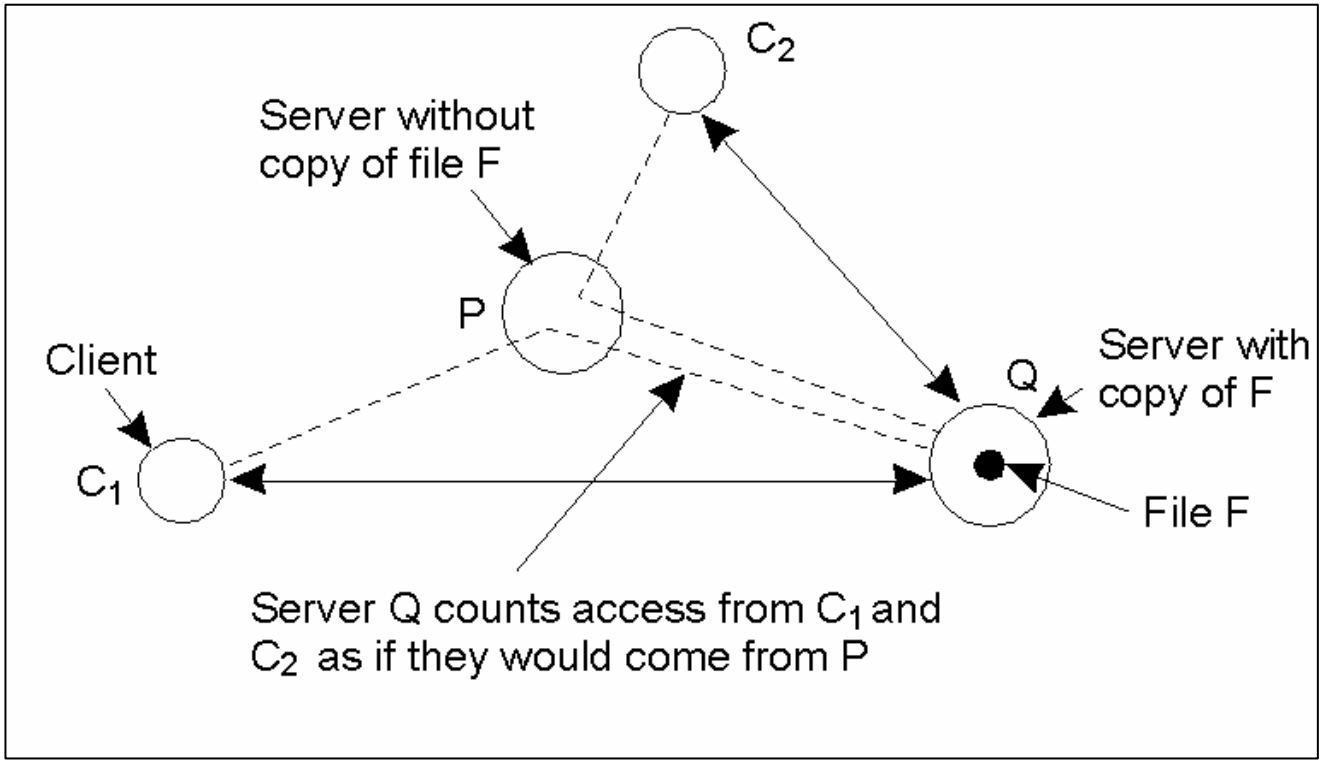
Le repliche principali di un data store distribuito in poche copie.

Esempi:

- Replicazione di un sito Web su una LAN
- Web mirroring
- Replicazione di un Database su un COW (Cluster of Workstations)

Repliche Server-Initiated

La creazione delle repliche è decisa dal server sulla base di una analisi del carico e delle prestazioni del sistema e della rete.



Conteggio di richieste di accessi da clienti differenti.

Repliche Client-Initiated

- Il cliente usa una sua cache per memorizzare una copia di un dato temporaneamente.
- Le cache possono essere locali o “vicine” al nodo client.
- Il server non si cura della consistenza dei dati.
- Più clienti possono condividere una cache.

Update Propagation

- Generalmente la propagazione degli aggiornamenti sono iniziate da un client e inviate ad una delle copie.
- Come propagare i valori aggiornati ?
  - Stato vs Operazioni (inviare i dati o le operazioni ?)
  - Protocolli Pull vs protocolli Push (client-based o server-based ?)
  - Unicasting vs Multicasting (update singoli o di gruppo?)

Protocolli: Pull vs Push

	Push-based	**Pull-based**
Stato del server	Lista delle repliche e delle cache dei client	Nulla
Messaggi inviati	Update (e possibilmente fetch update dopo)	Poll e update
Tempo di risposta nel client	Immediato (o il tempo di fetch-update)	Tempo di fetch-update

Un confronto tra protocolli push-based e pull-based nel caso di più clienti e un singolo server.

Protocolli di Consistenza

- Un **protocollo di consistenza** definisce una implementazione di uno specifico modello di consistenza.
- Protocolli Primary-based
- Protocolli Replicated-write
- Protocolli di Cache-coherence.