PostgreSQL ed “altra roba”

Ottavio Campana
ottavio at campana dot vi dot it

Storia di PostgreSQL

- 1986, inizio dello sviluppo ad UC Berkeley, *The design of the postgres rules system*, come successore di ingres.
- 1994 aggiunto l'interprete SQL e rilascio di Postgres95
- 1996 nome cambiato in PostgreSQL e rilascio della versione 6.0
- 4 Febbraio 2008 rilasciato PostgreSQL 8.3

Un database acido

Per gestire correttamente la manipolazione dei dati un database deve essere ACID, ovvero deve soddisfare le seguenti quattro proprietà:

Atomicity

Consistency

Isolation

Durability

Atomicità

La proprietà di atomicità assicura che tutte o nessuna delle operazioni di una transazione vengono eseguite.

Per esempio nel caso di trasferimenti di fondi da un conto ad un altro la transazione deve coinvolgerli entrambi o nessuno ma non modificarne uno solo.

Consistenza

La consistenza assicura che ad ogni operazione ci si muova da uno stato valido ad un altro stato valido.

E' quindi necessario assicurare che tutti i vincoli sui dati siano rispettati.

Consistenza

A differenza di altri database, PostgreSQL ha sempre implementato tutta la struttura necessaria per assicurare l'integrità referenziale.

Inoltre è presente nel database tutta la gestione dei trigger e delle eccezioni per le violazioni di condizioni, quali per esempio di unicità.

Isolamento

Ogni operazione all'interno di un database deve poter lavorare sui dati indipendentemente dalle operazioni che avvengono in concorrenza.

Isolamento

Un metodo per garantire che operazioni concorrenti non interferiscano molti database implementano meccanismi di locking per gestire l'accesso ai dati.

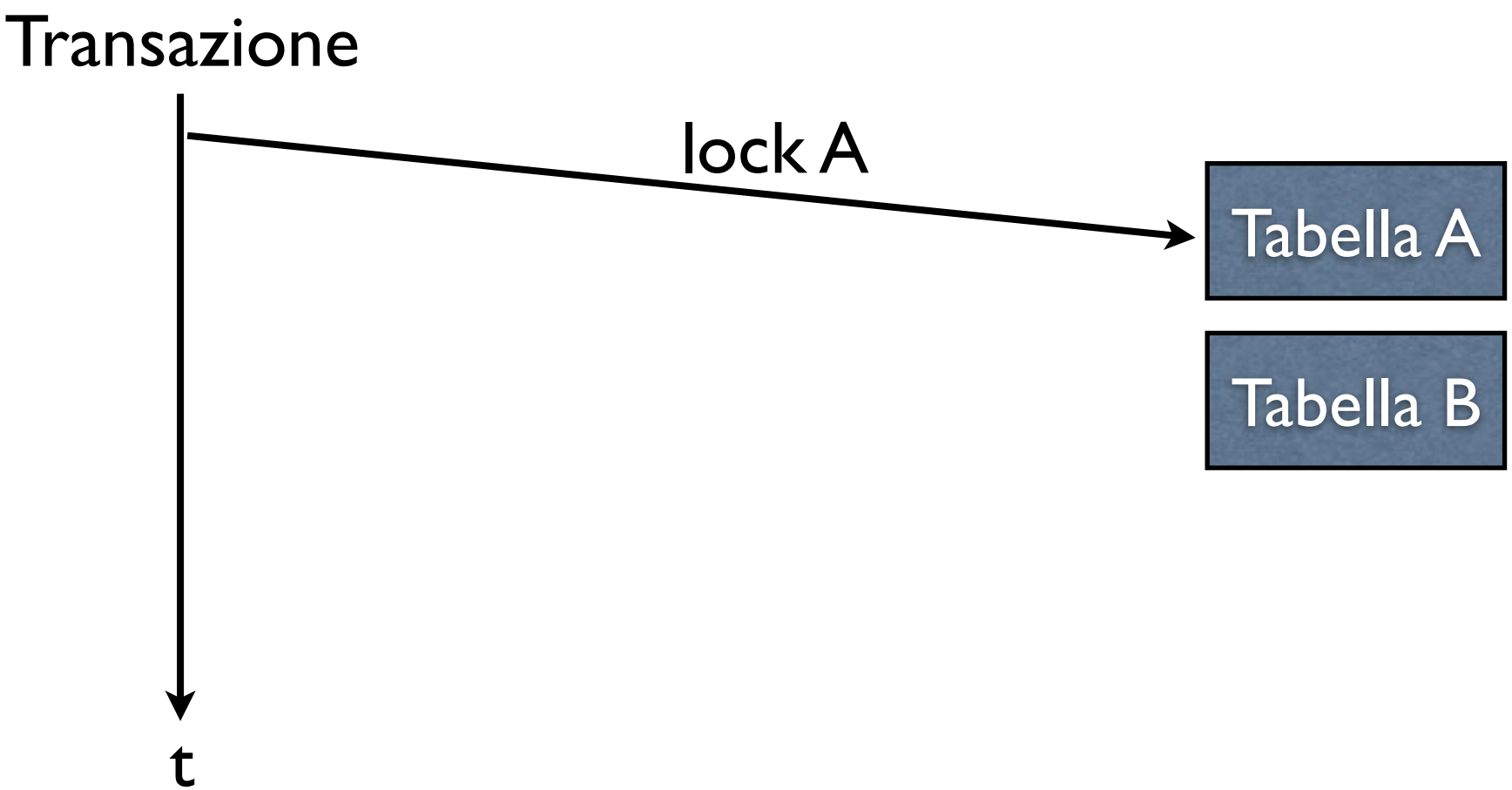
I lock in generale hanno due svantaggi principali: prestazioni ridotte e rischi di stallo (*deadlock*) che bloccano il database.

Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:

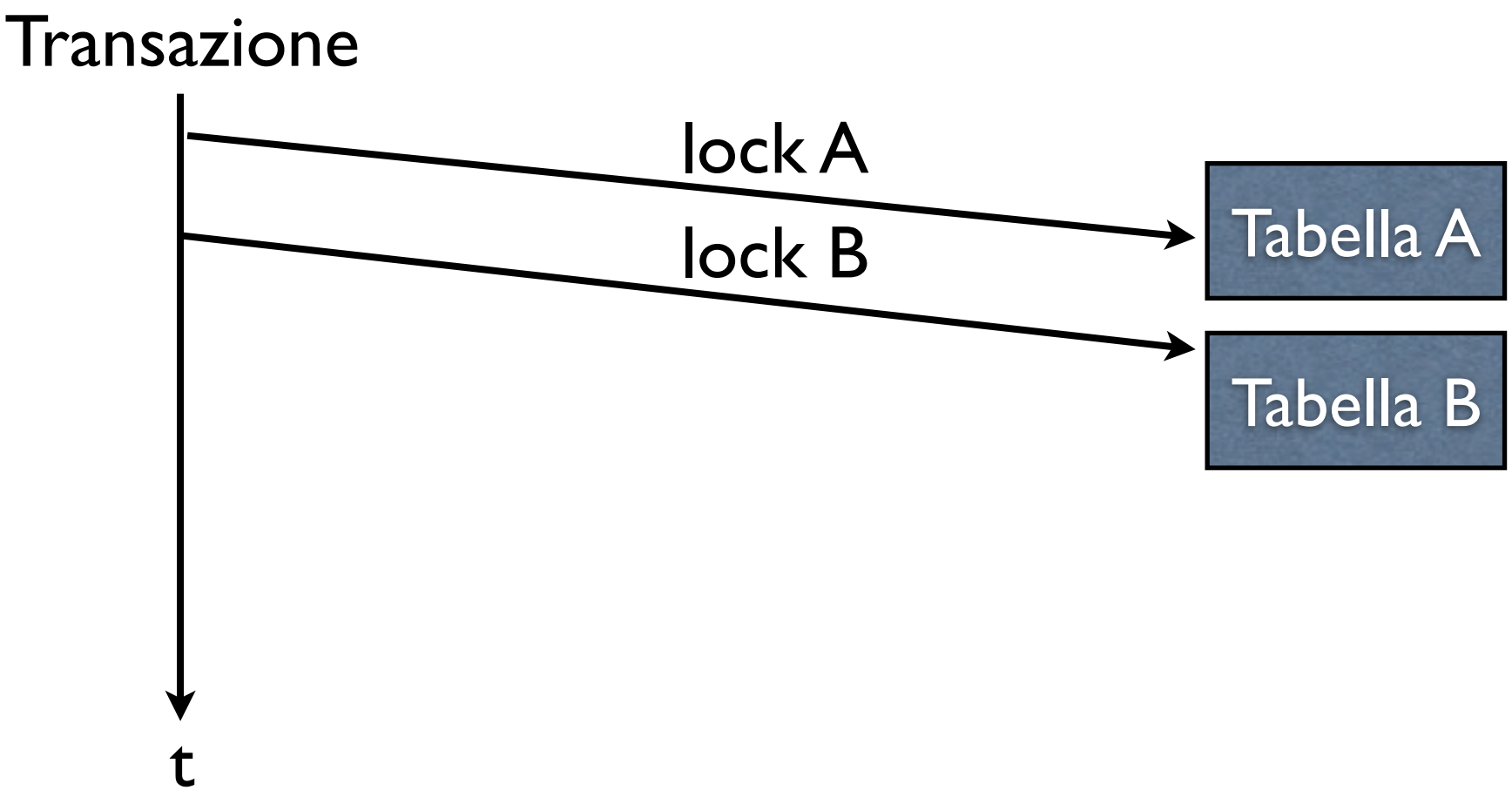
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



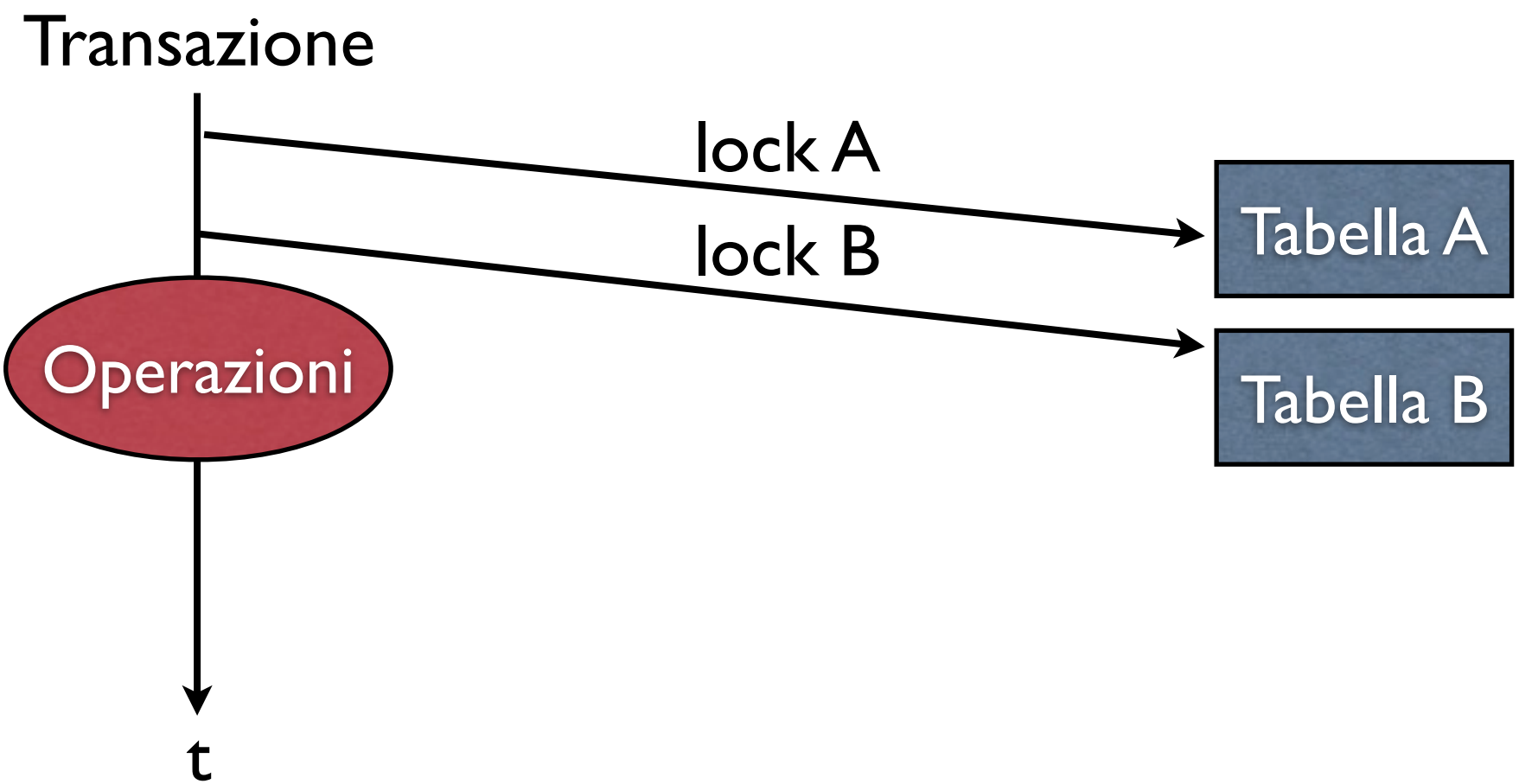
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



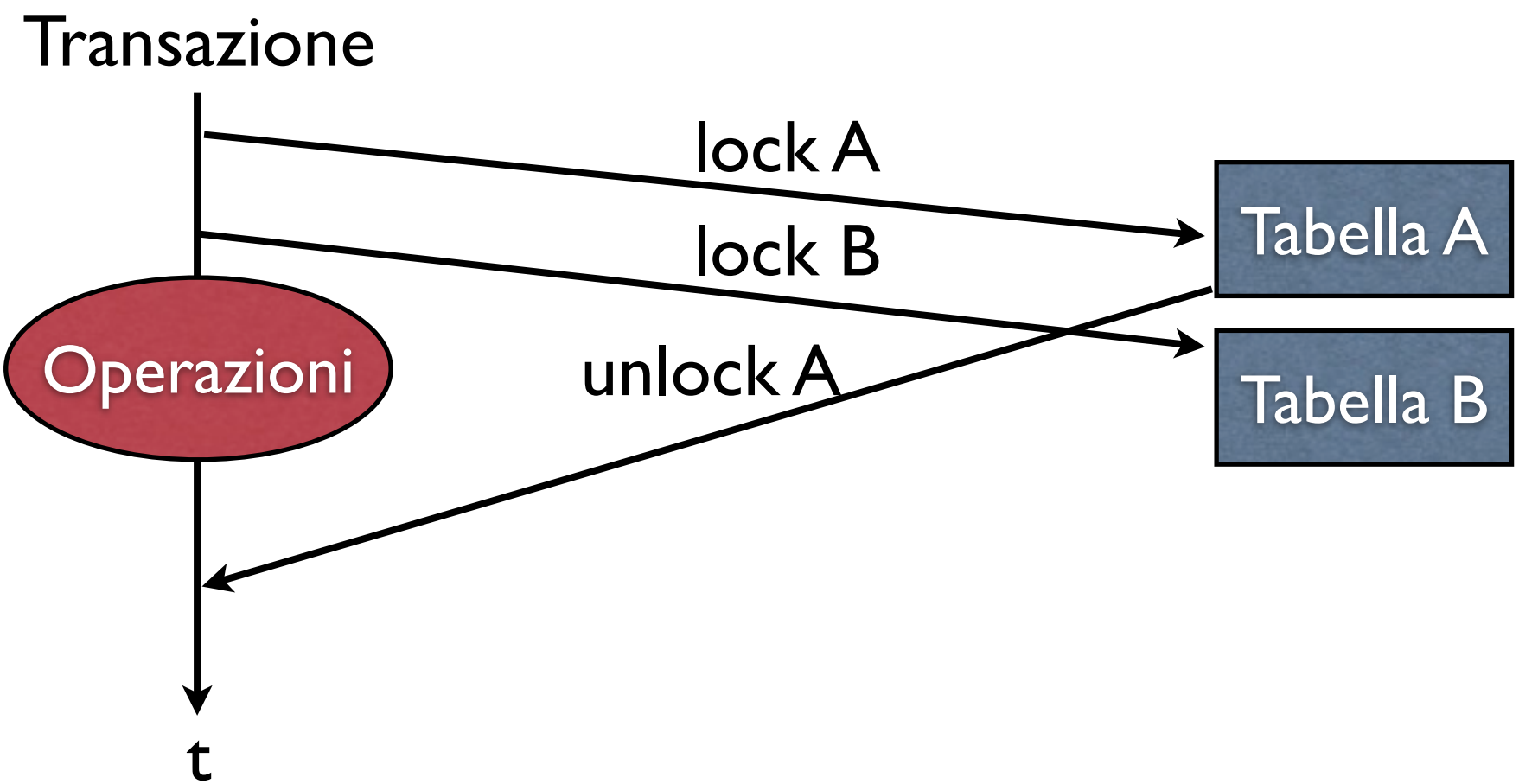
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



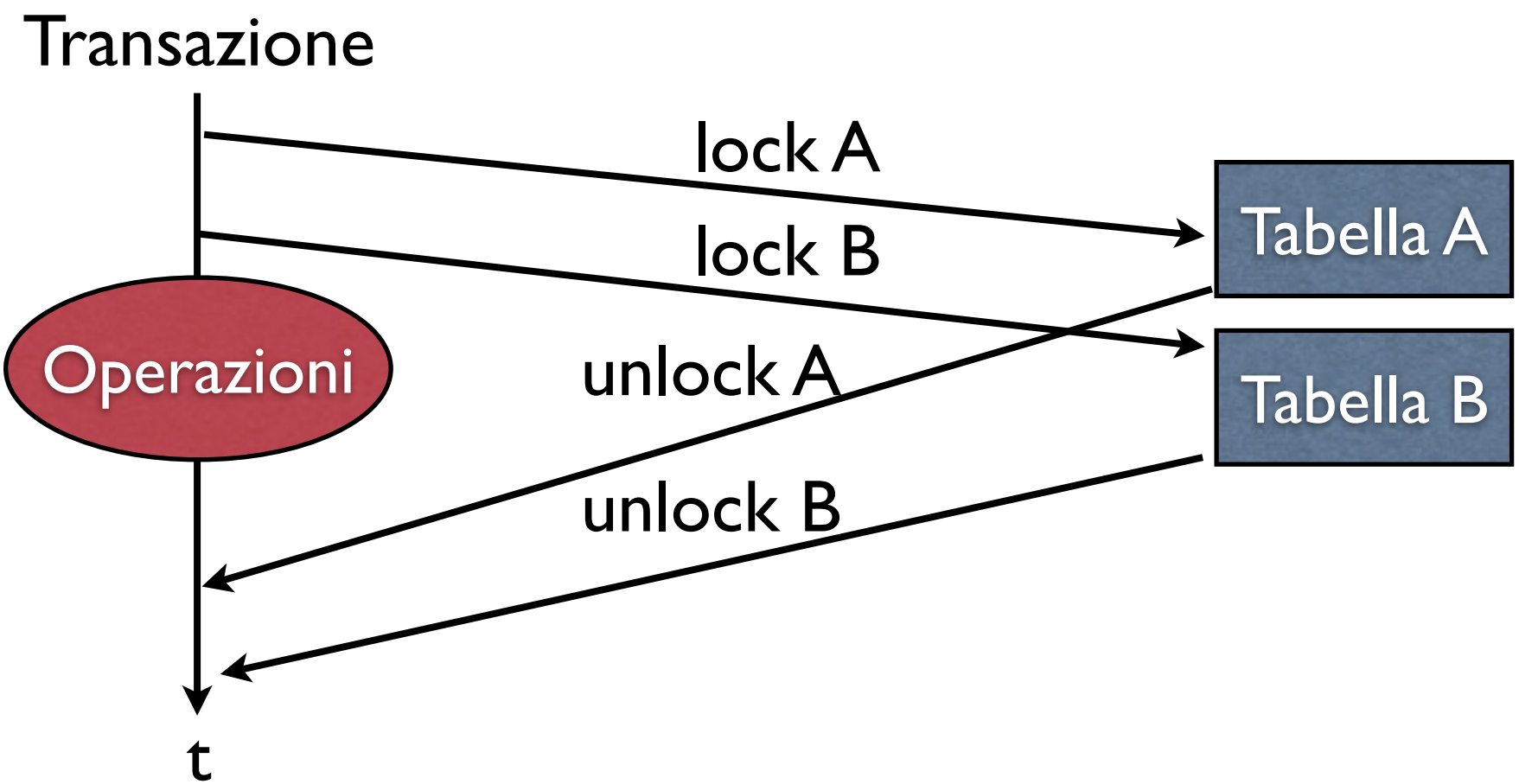
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



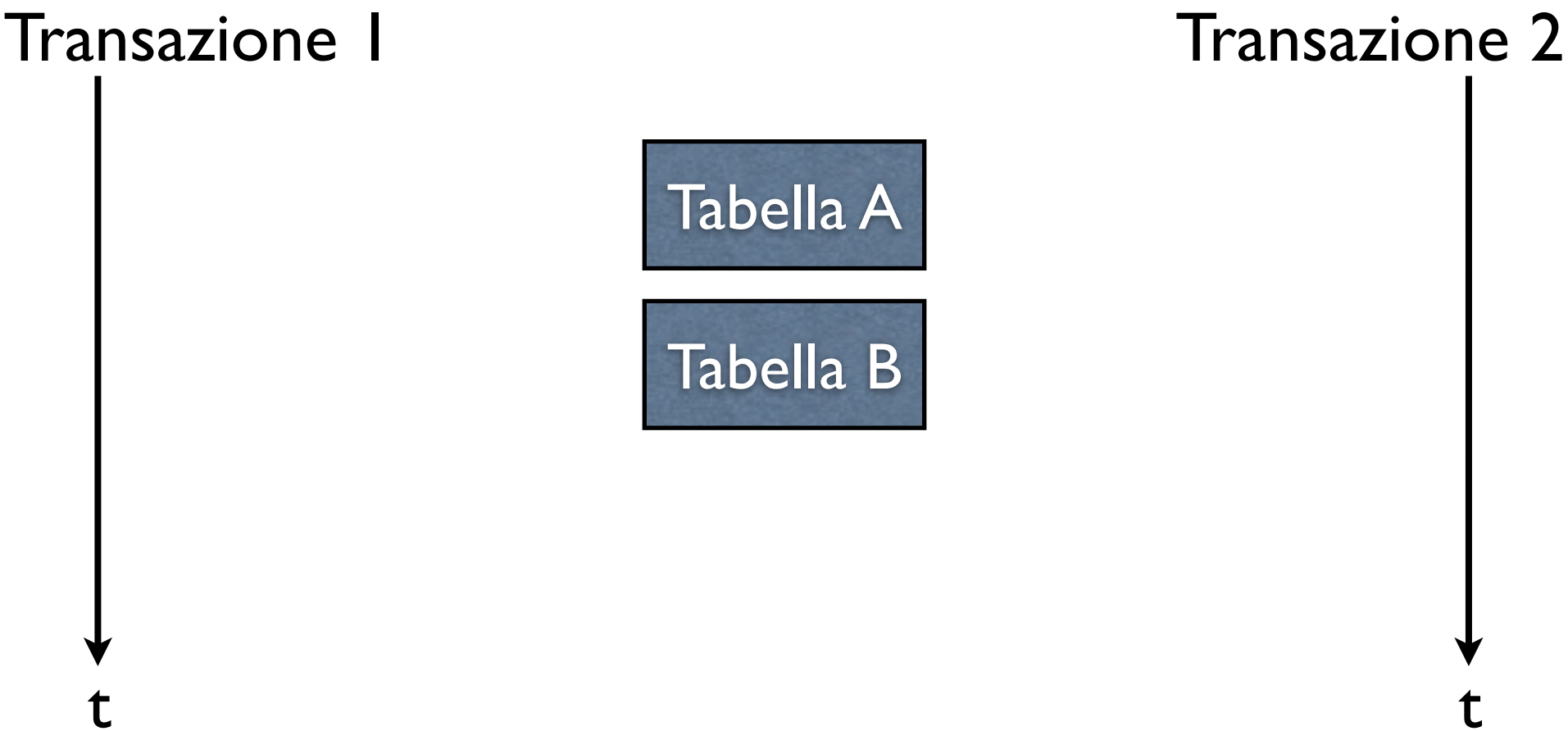
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



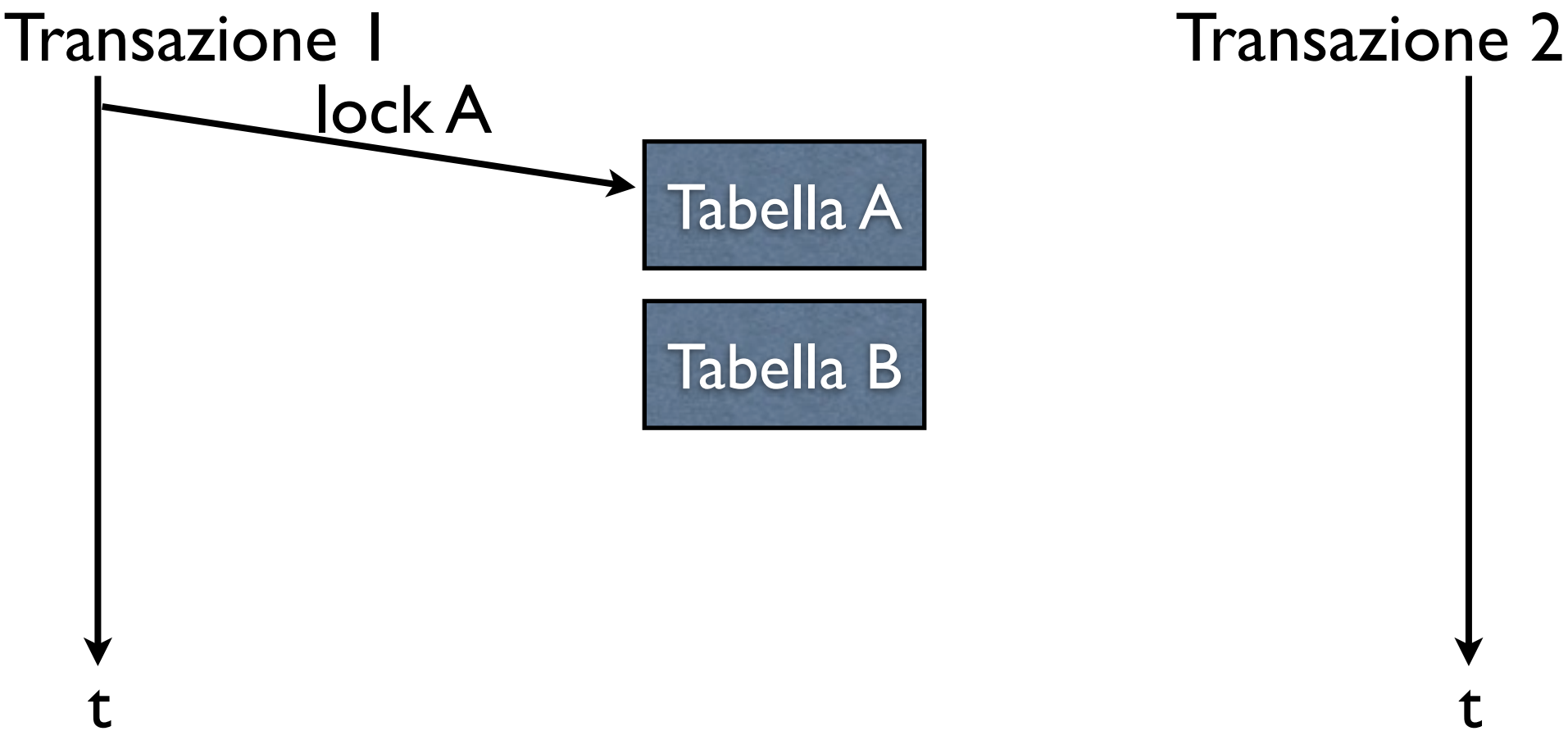
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



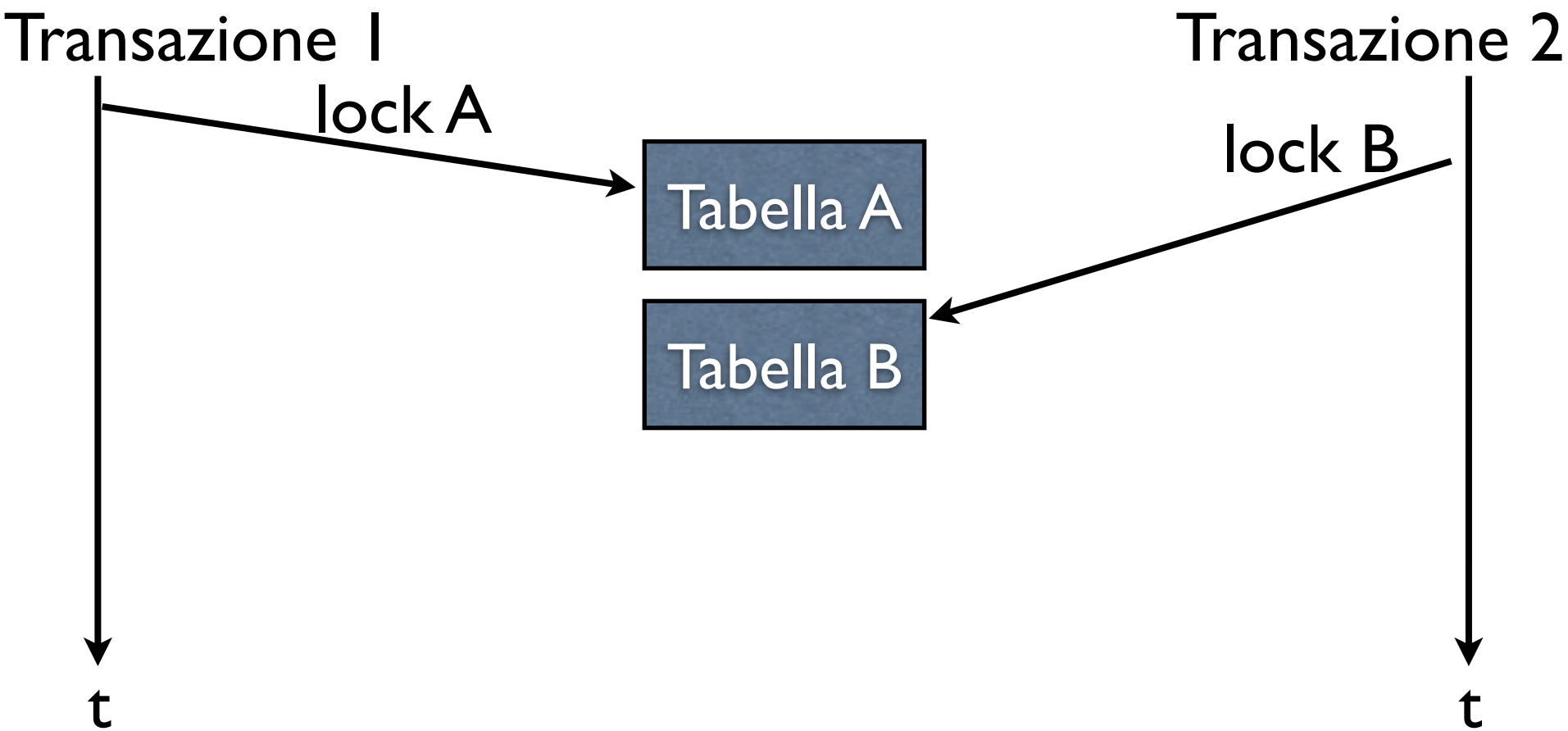
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



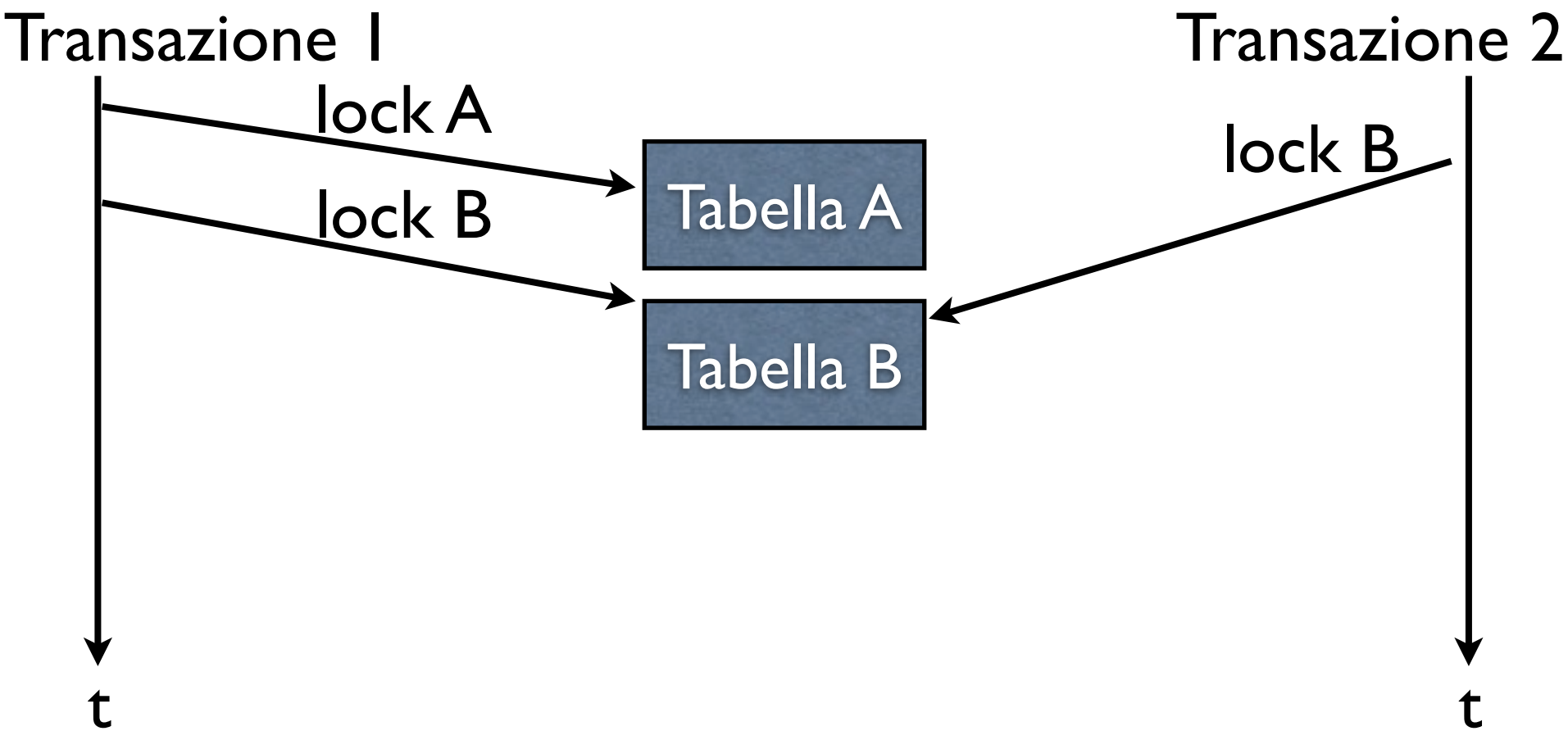
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



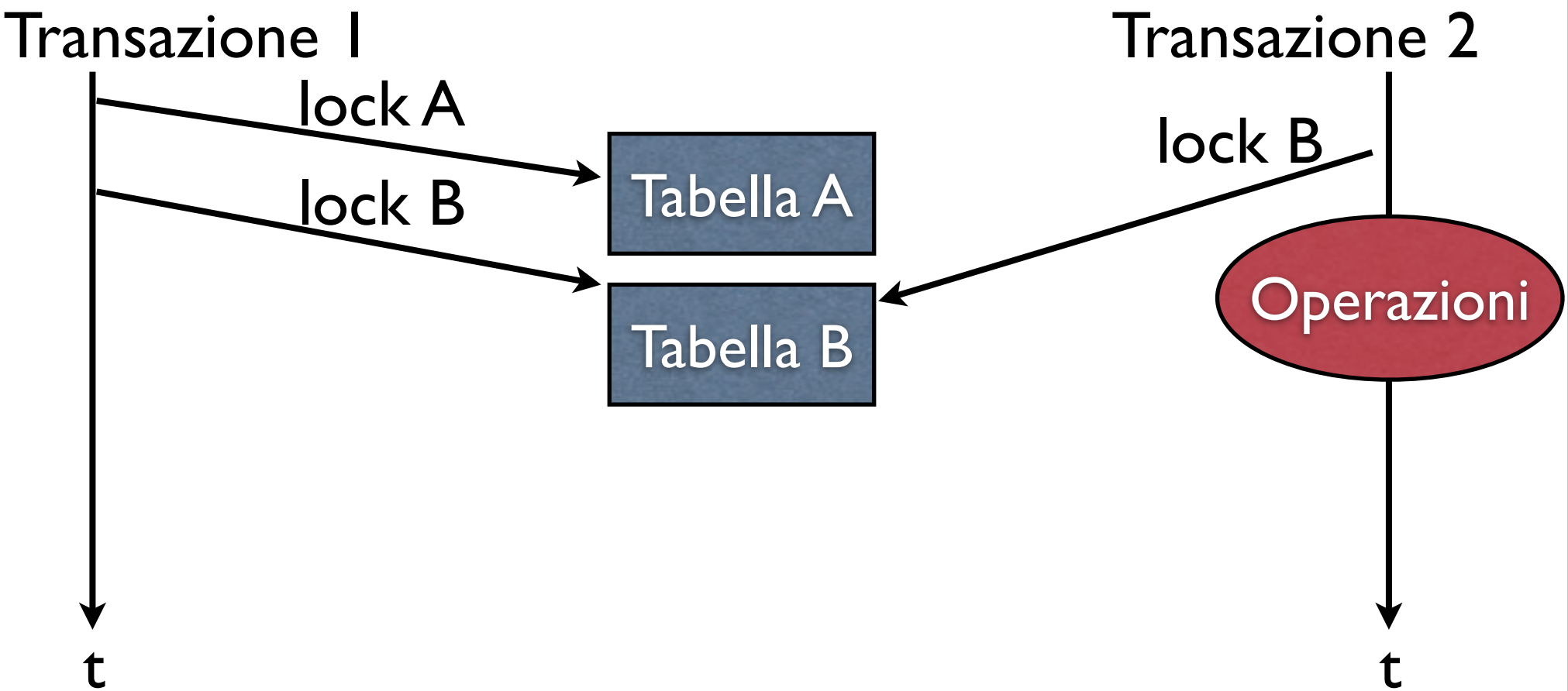
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



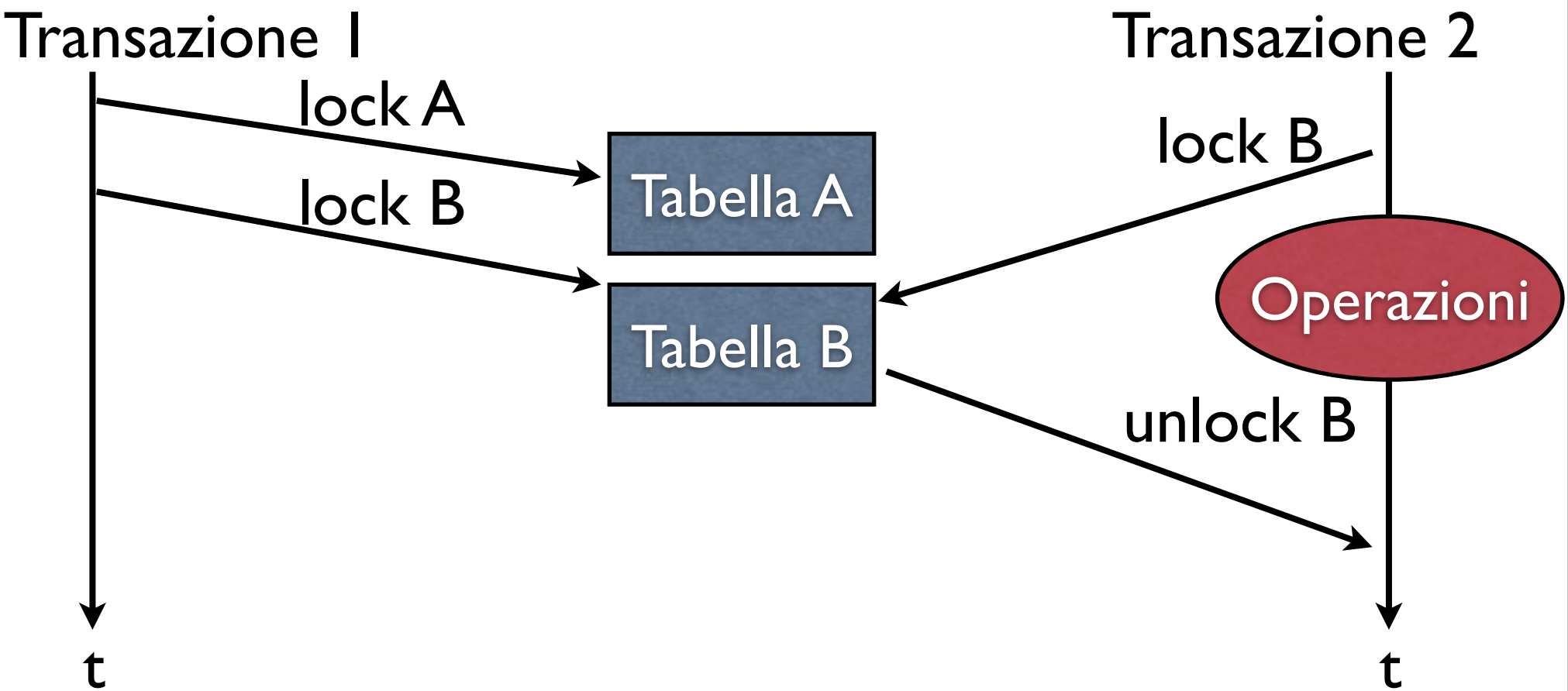
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



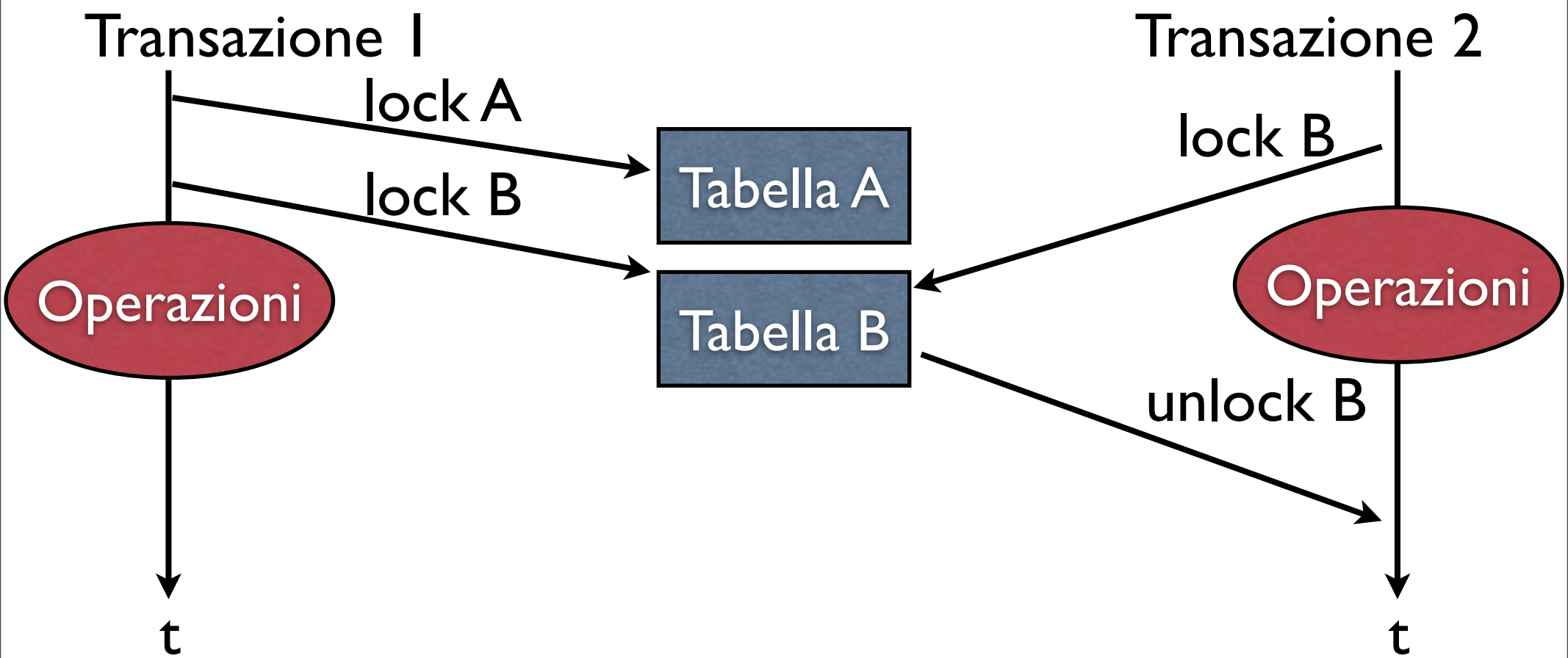
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



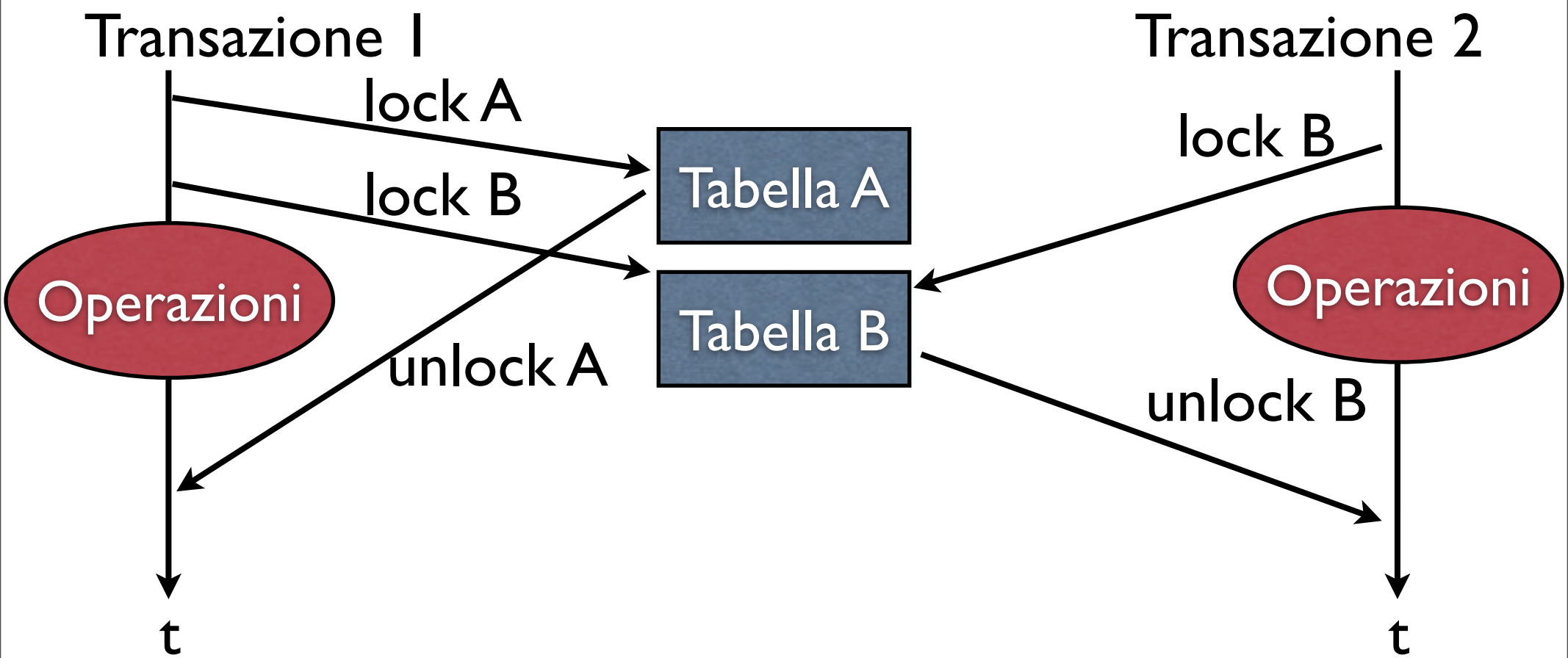
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



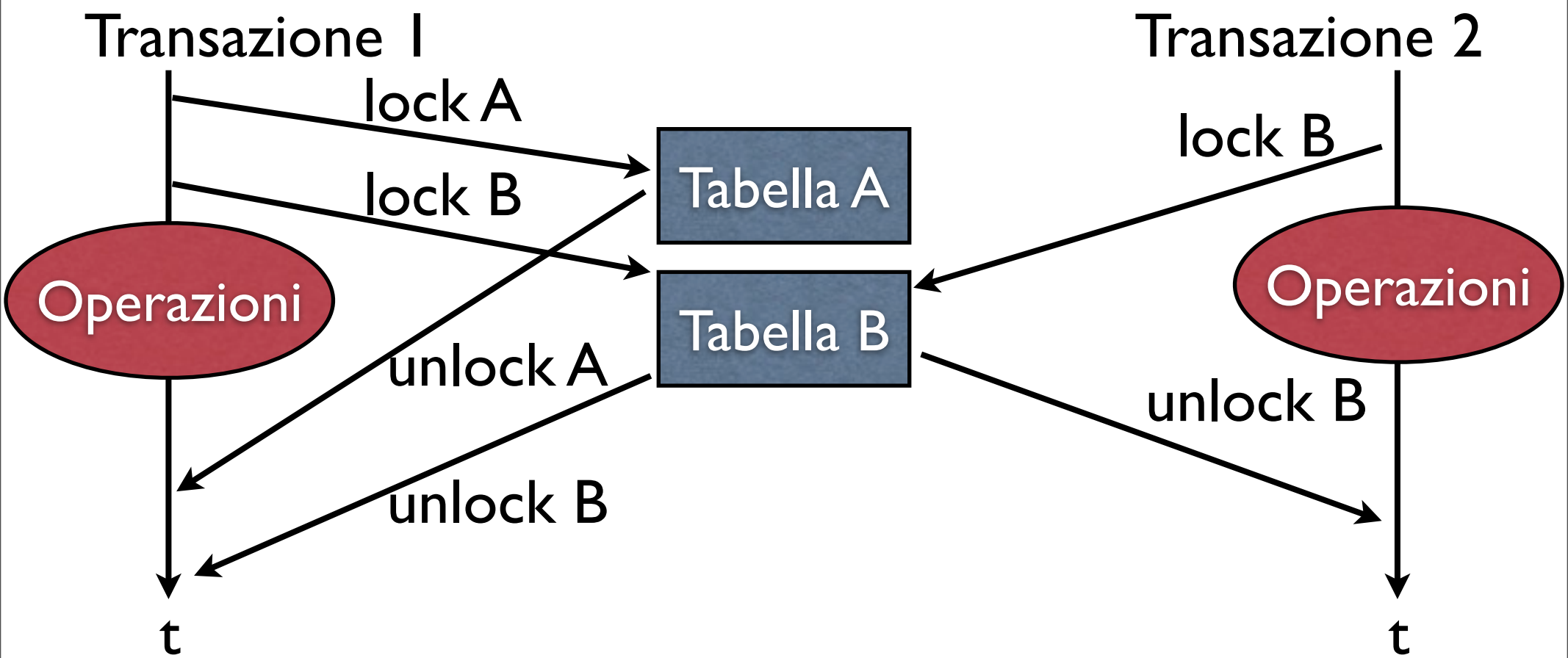
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



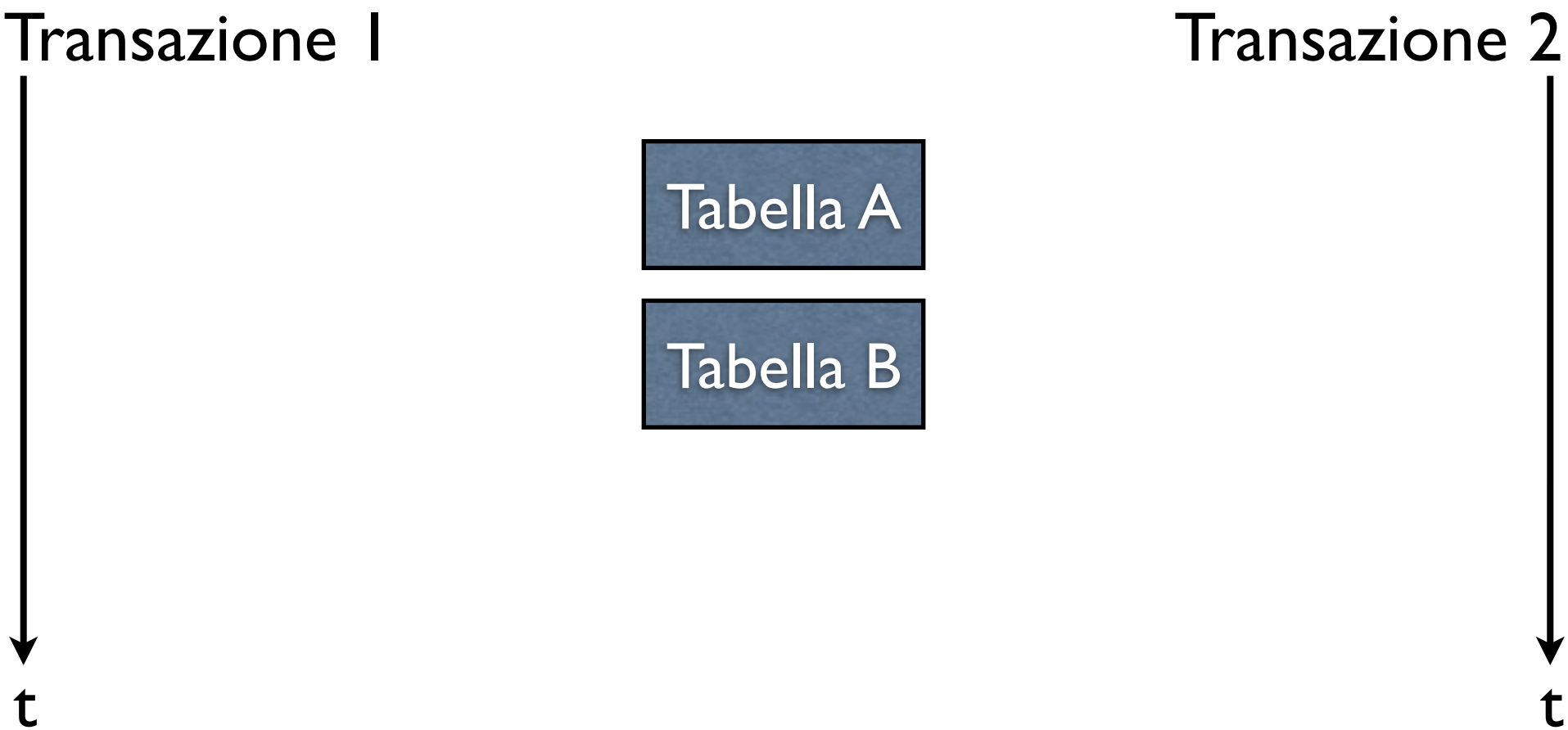
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



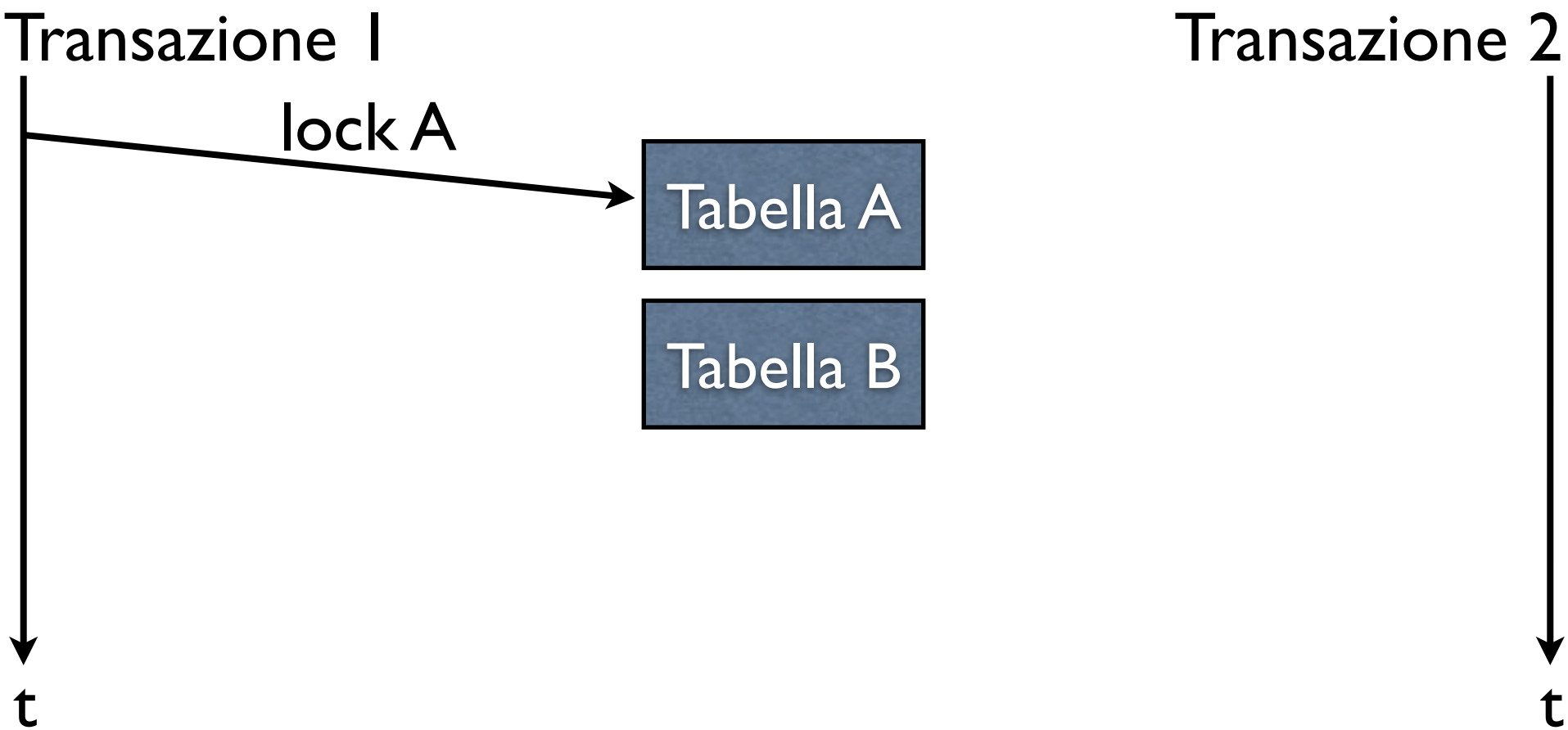
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



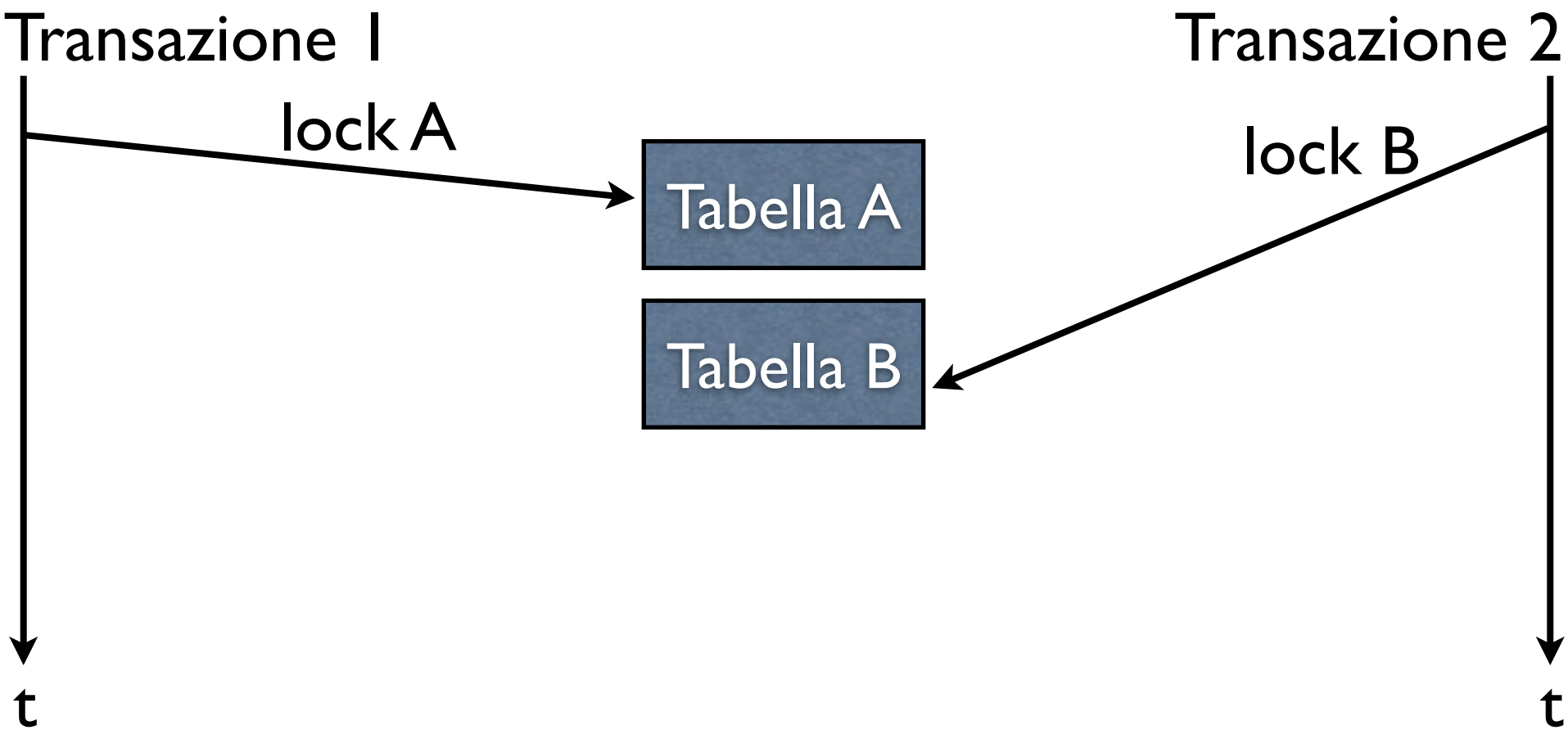
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



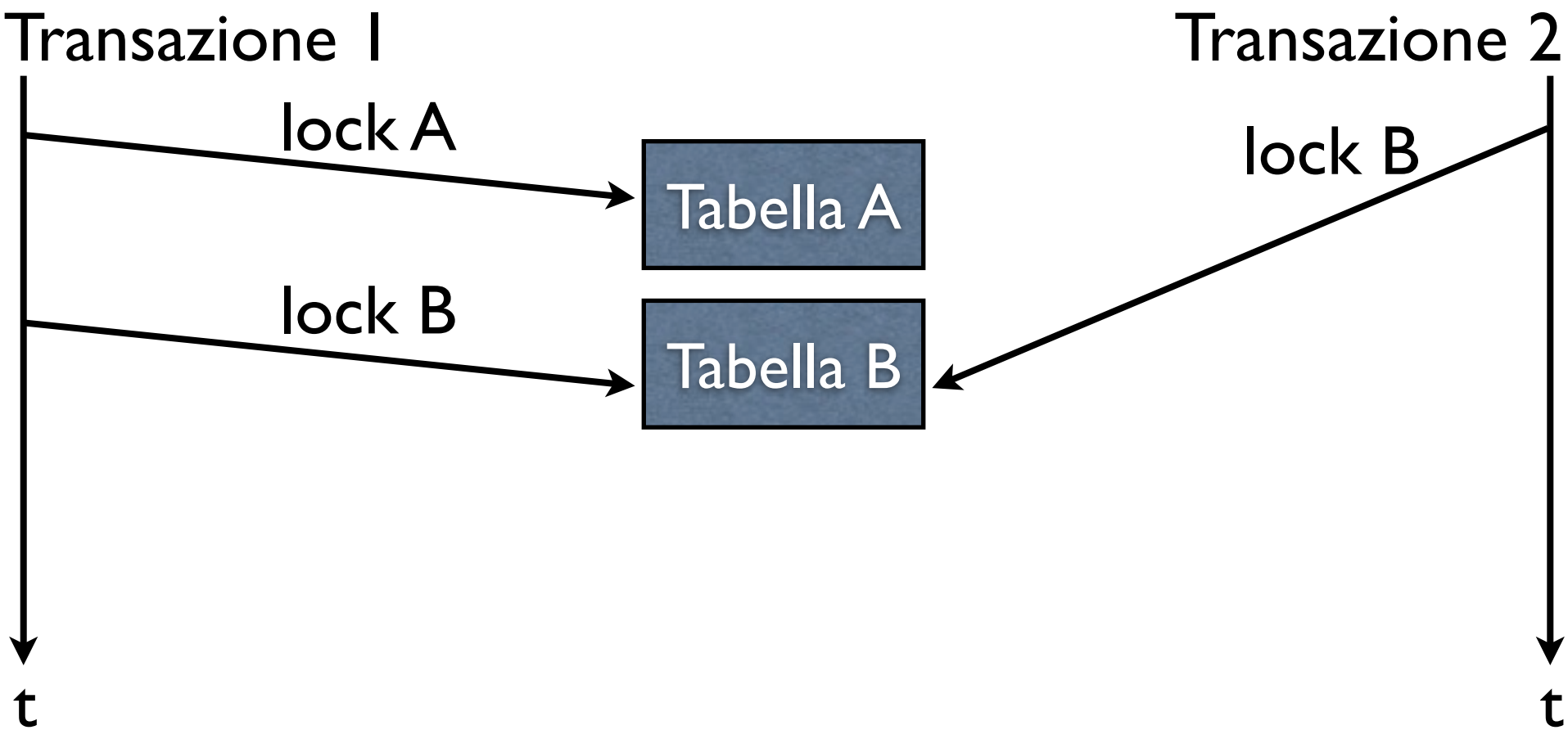
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



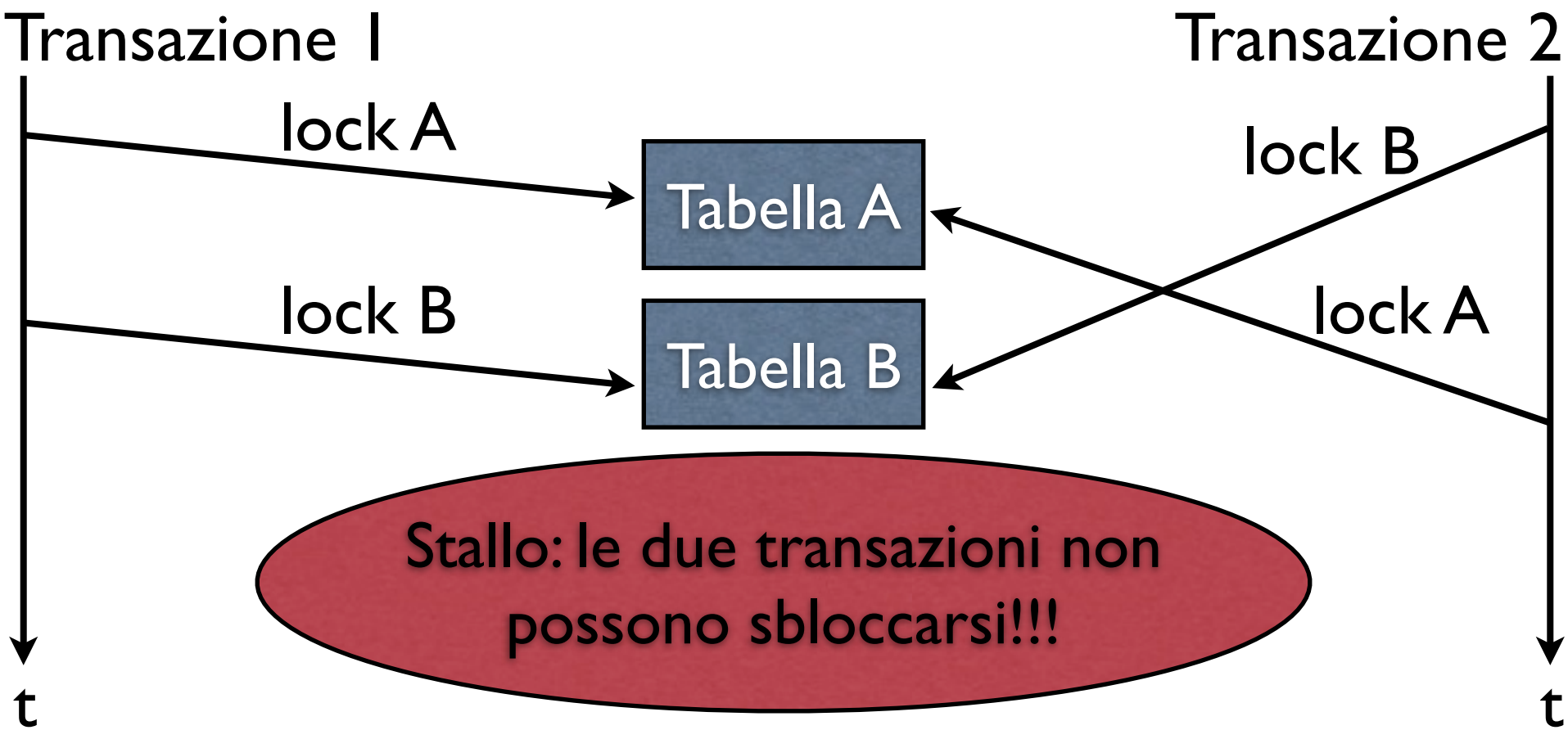
Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



Isolamento

Un deadlock si verifica quando due transazioni concorrenti accedono alle stesse risorse:



Isolamento

Le operazioni di locking possono portare a gravi situazioni di stallo che non solo impediscono la scrittura dei dati nel database, ma che bloccano i programmi che le stanno eseguendo.

Per ovviare a questo problema PostgreSQL offre due meccanismi:

- un meccanismo per risolvere i deadlock;
- il *multiversion concurrency control*.

Multiversion Concurrency Control

In PostgreSQL ogni riga ha ID:

- creation transaction ID ;
- expiration transaction ID .

Ad ogni operazione PostgreSQL crea delle nuove righe aggiornate e marca come obsolete quelle vecchie.

Con i due ID è possibile isolare senza lock.

Multiversion Concurrency Control

Sebbene MVCC sia più complicato da implementare che le semantiche di locking, esso offre molti vantaggi:

- row-level locking;
- facilità di implementazione dei rollback, visto che le righe obsolete possono essere usate per tornare indietro nel tempo;
- hot backup, consistenti senza fermare il db.

Durabilità

La durabilità assicura che una volta che una transazione è stata eseguita i dati rimangano stabili nel database.

Un ruolo particolarmente importante nella durabilità è quello della resistenza alle rotture dei server per salvaguardare i dati.

Durabilità

Sebbene il MVCC permetta di effettuare dei backup a caldo senza fermare il server, l'operazione di restore non è in generale fattibile in tempo breve o reale, soprattutto quando la mole di dati necessari è grande.

Durabilità

Diverse soluzioni sono disponibili per PostgreSQL per garantire affidabilità:

- shared disk failover
- warm stand-by using Point-In-Time Recovery
- master-slave replication
- statement-based replication middleware
- synchronous/asynchronous multimaster replication

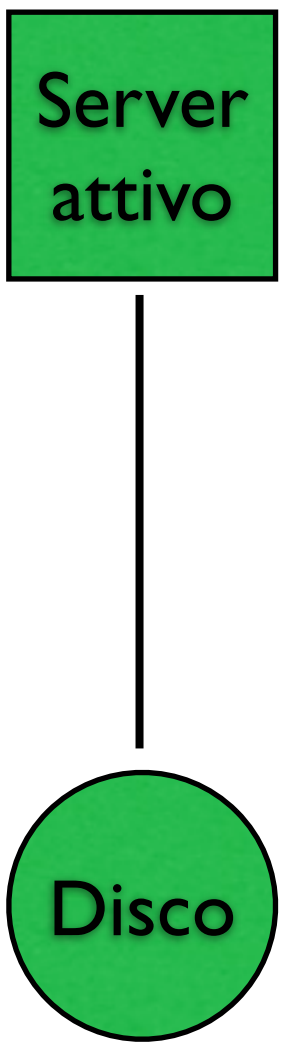
Shared Disk Failover

Le soluzioni basate sui dischi condivisi permettono di eliminare il sovraccarico della replica utilizzando una sola copia del database.

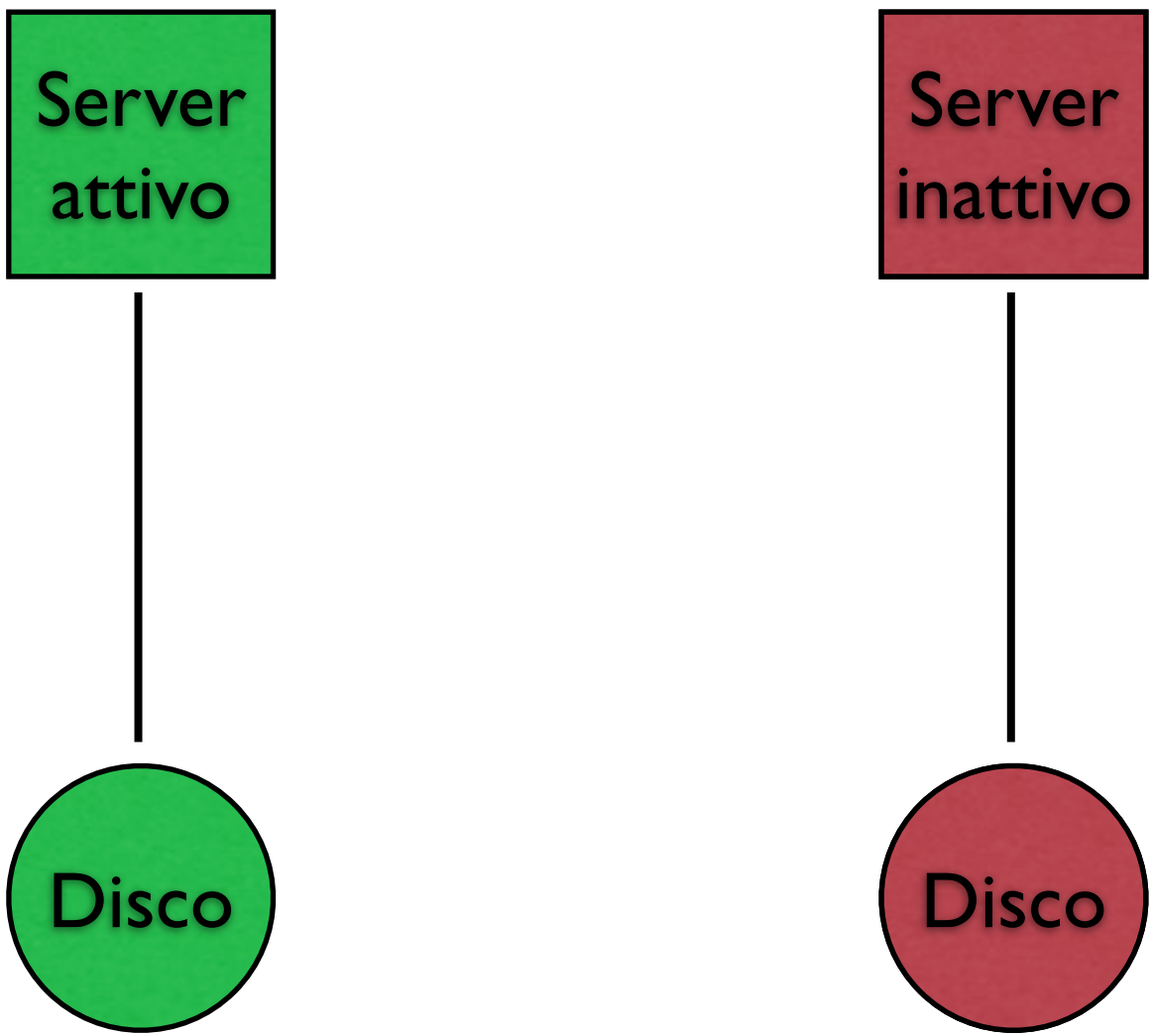
In caso di rottura di un server un secondo server può essere avviato per fornire il servizio.

Per evitare di spostare i dischi è possibile usare DRBD, che replica i dispositivi a blocchi sotto GNU/Linux.

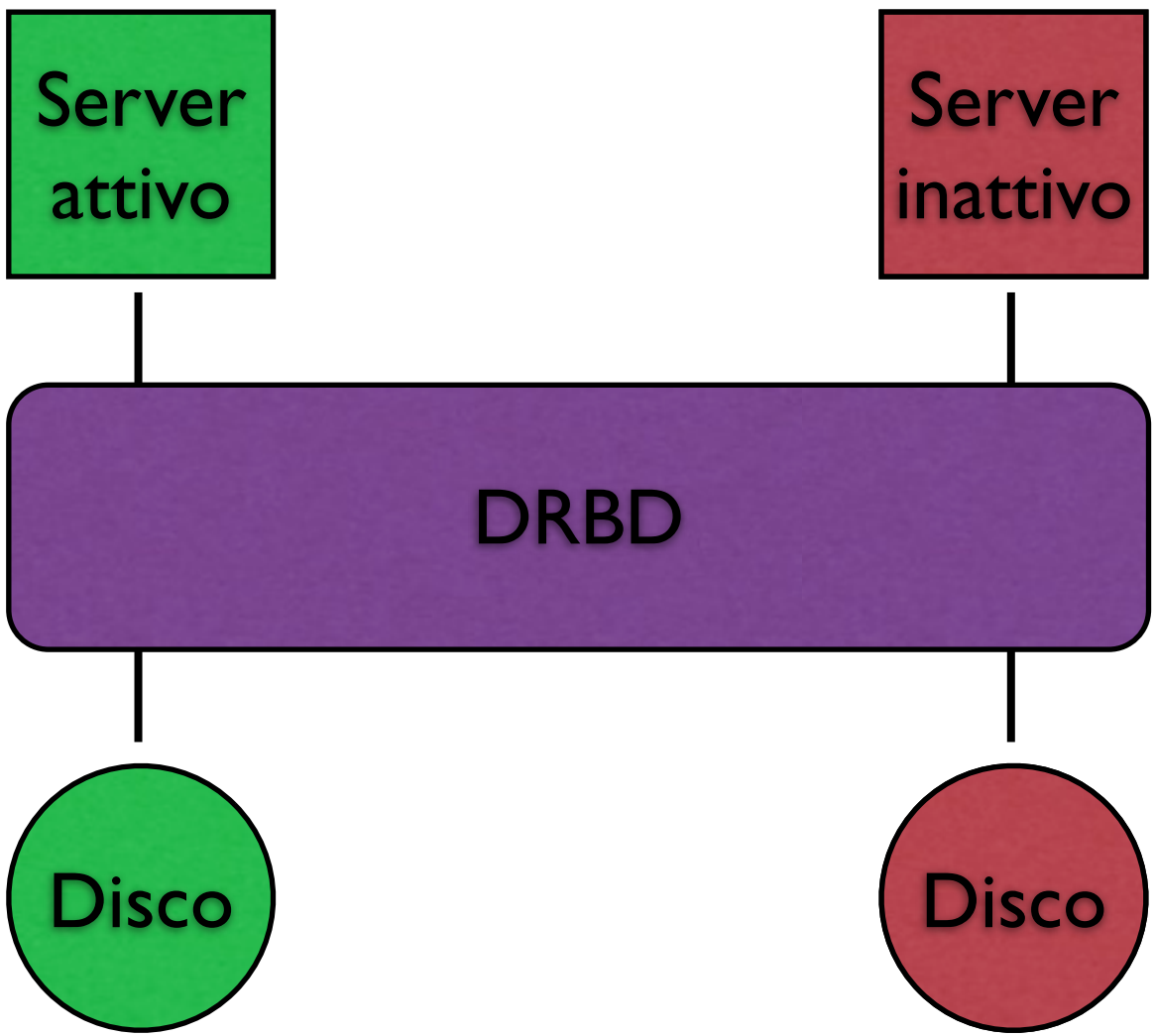
Shared Disk Failover



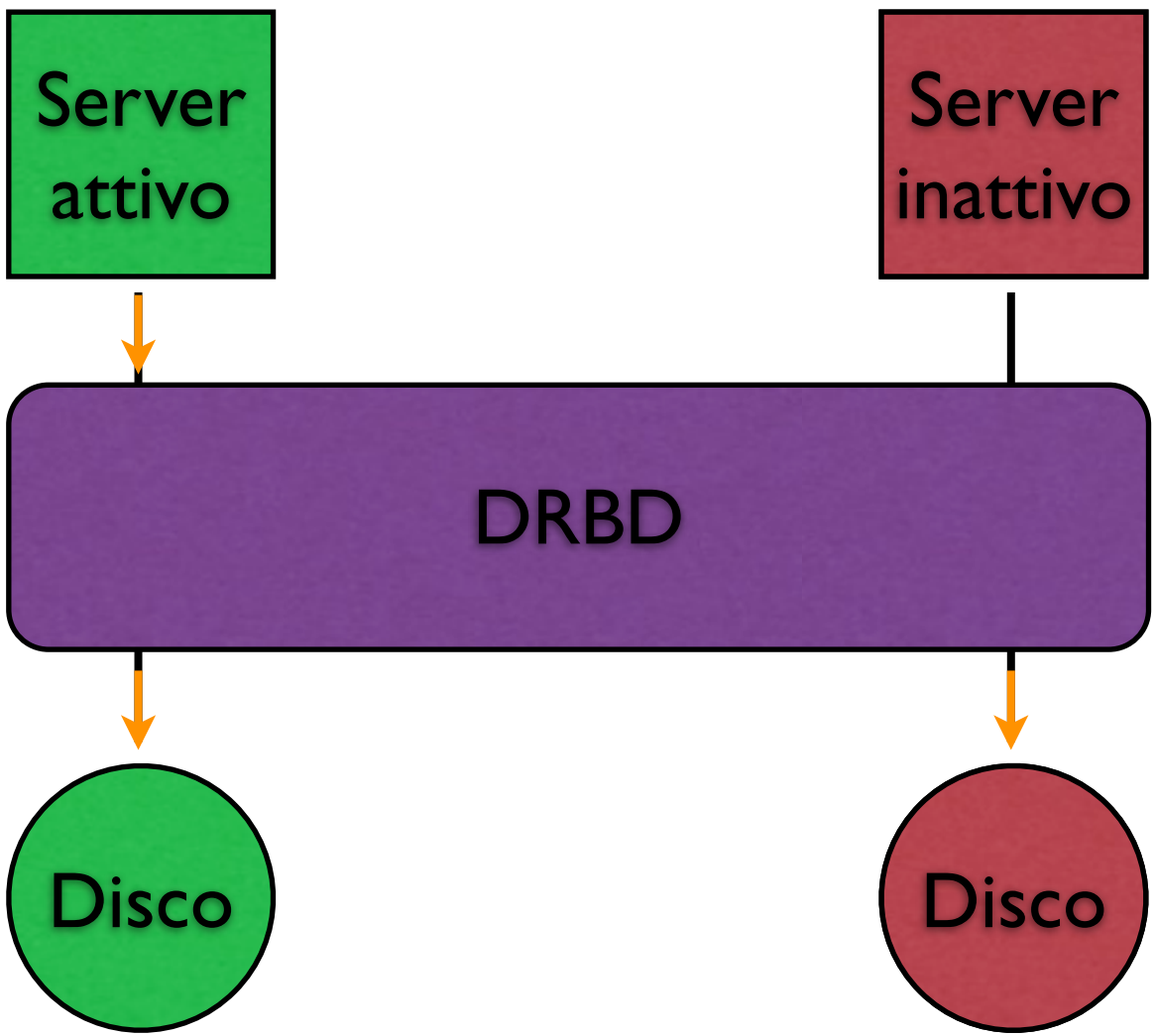
Shared Disk Failover



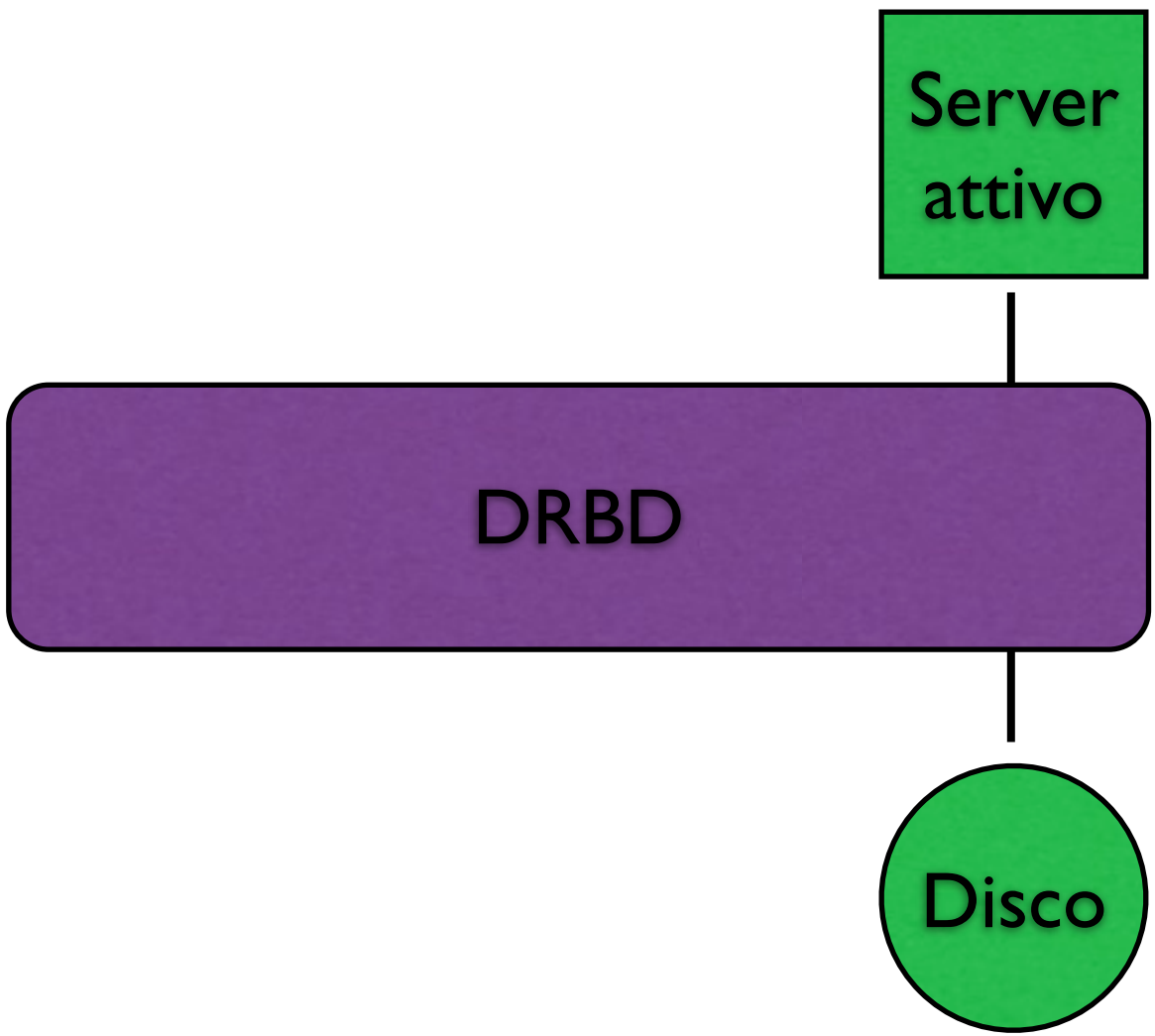
Shared Disk Failover



Shared Disk Failover



Shared Disk Failover



Warm Standby Server

Un'altra soluzione di alta affidabilità è data dalla preparazione di server inattivi fino al momento di necessità.

Il server master funziona in *archiving mode* e gli standby server funzionano costantemente in *recovery mode*. Il master salva tutte le operazioni nel Write Ahead Log (WAL) che viene inviato in modo asincrono alle repliche. In caso di rottura, i dati non inviati sono persi.

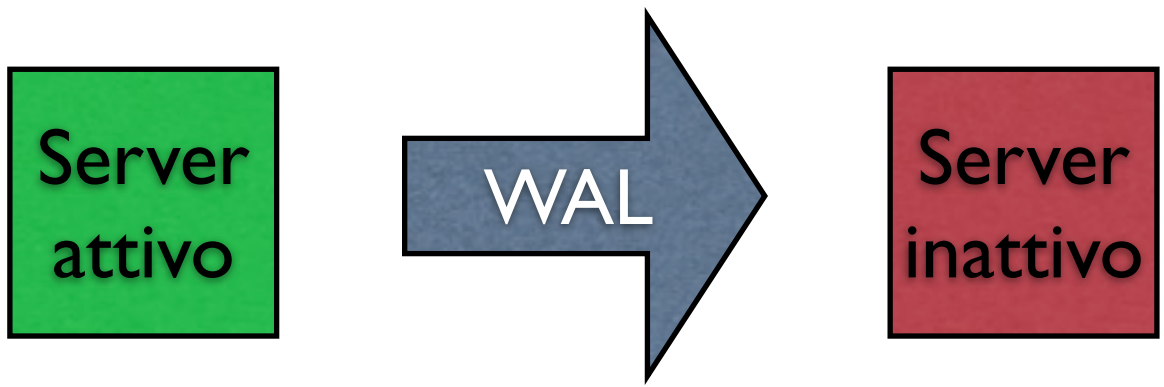
Warm Standby Server

Server sincronizzati

Warm Standby Server

Server non sincronizzati

Warm Standby Server



Server sincronizzati

Warm Standby Server

Server non sincronizzati

Warm Standby Server

Server non sincronizzati

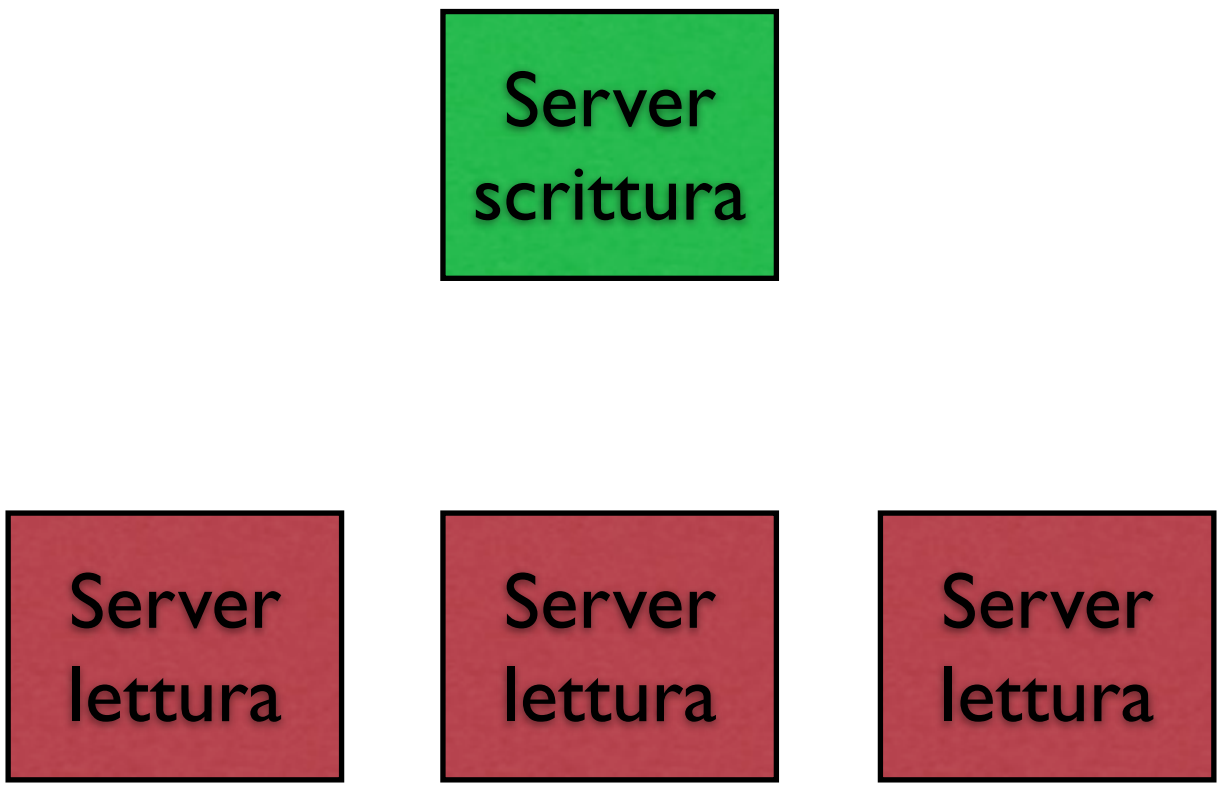
Perdita dei dati non trasferiti

Master-Slave replication

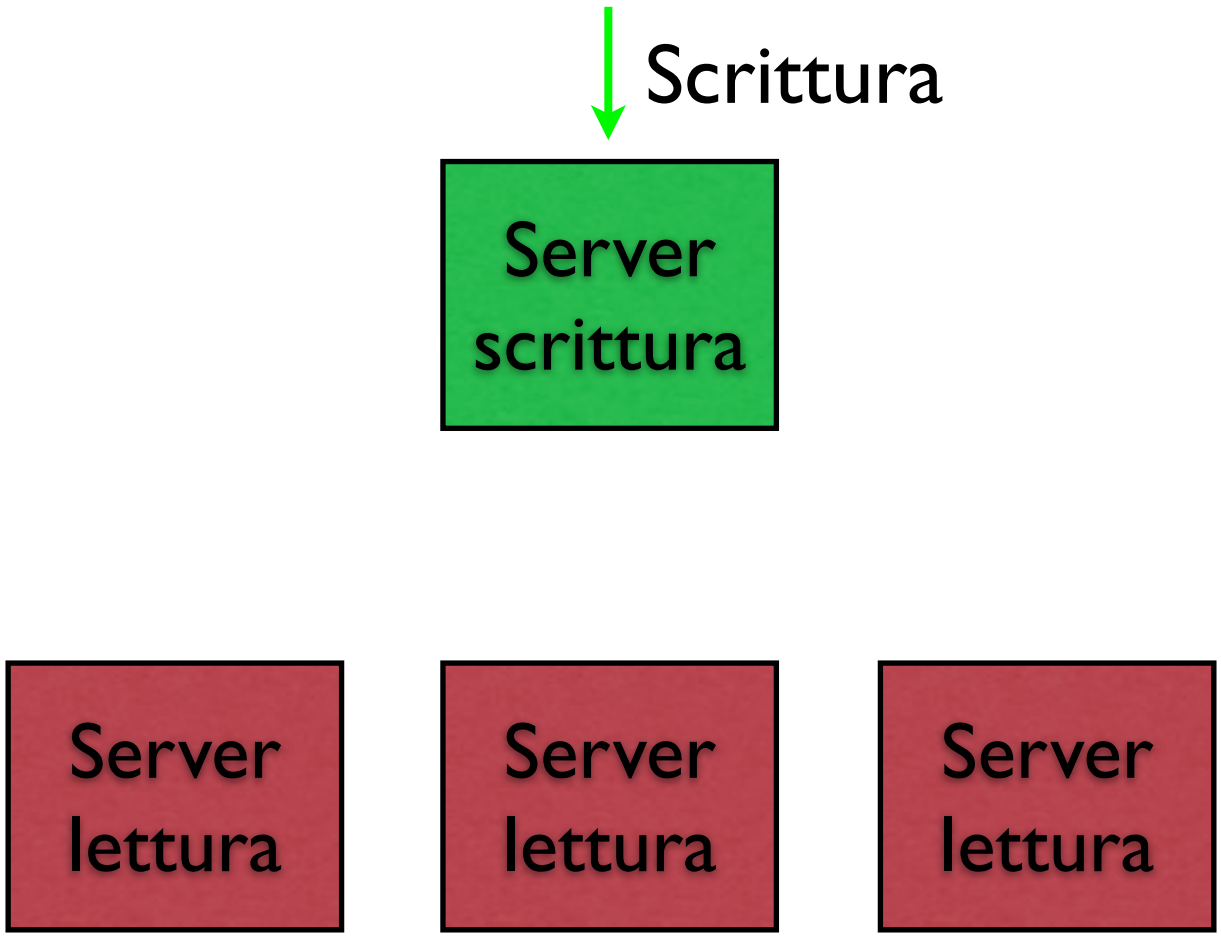
La replica master-slave è particolarmente utile negli scenari in cui le interrogazioni in lettura sono molto maggiori di quelle in scrittura.

Slony-I offre un sistema di replica basato su trigger. Poiché le modifiche DDL, le operazioni sui large objects e le richieste TRUNCATE non possono essere associate a trigger, queste non possono essere replicate.

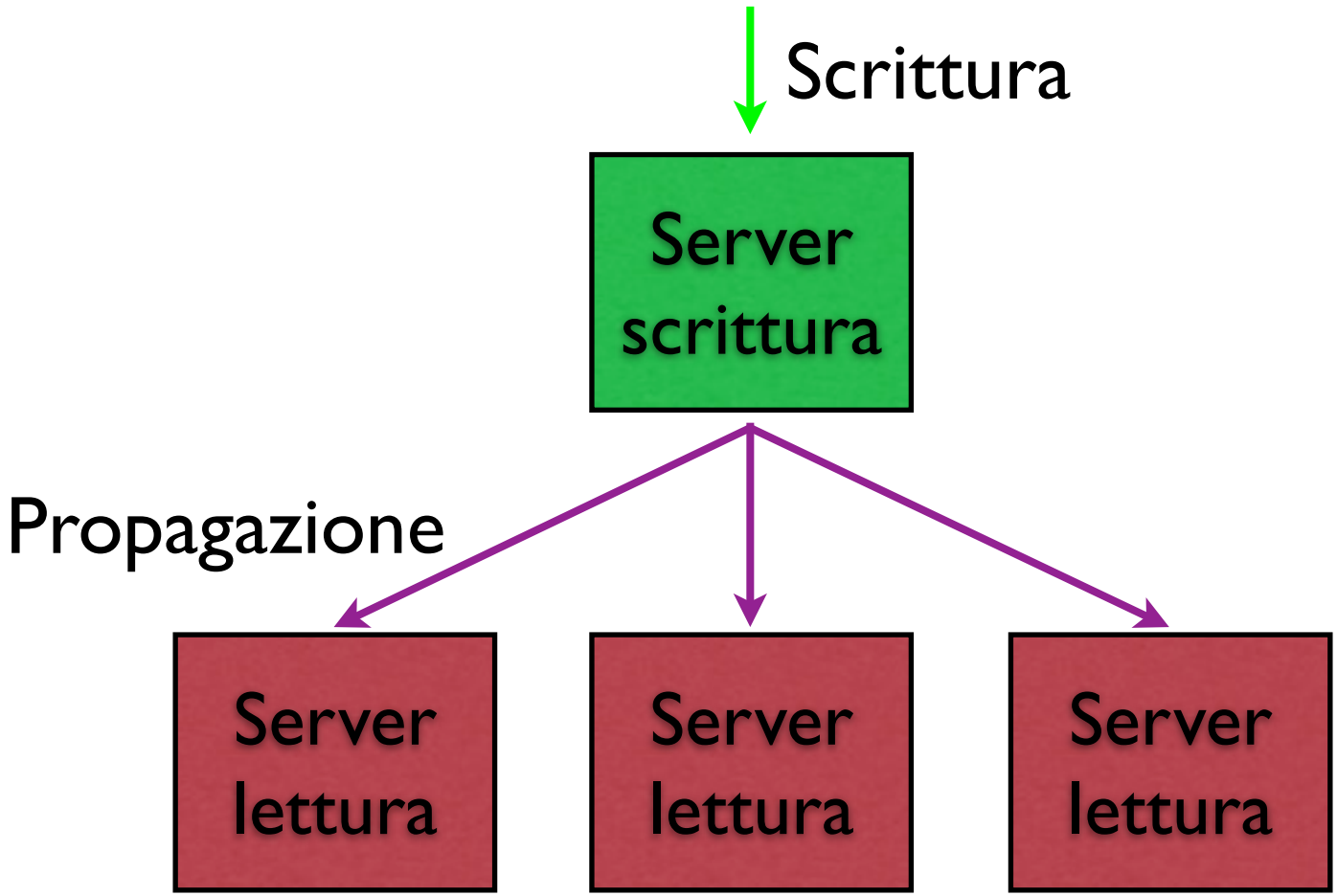
Master-Slave replication



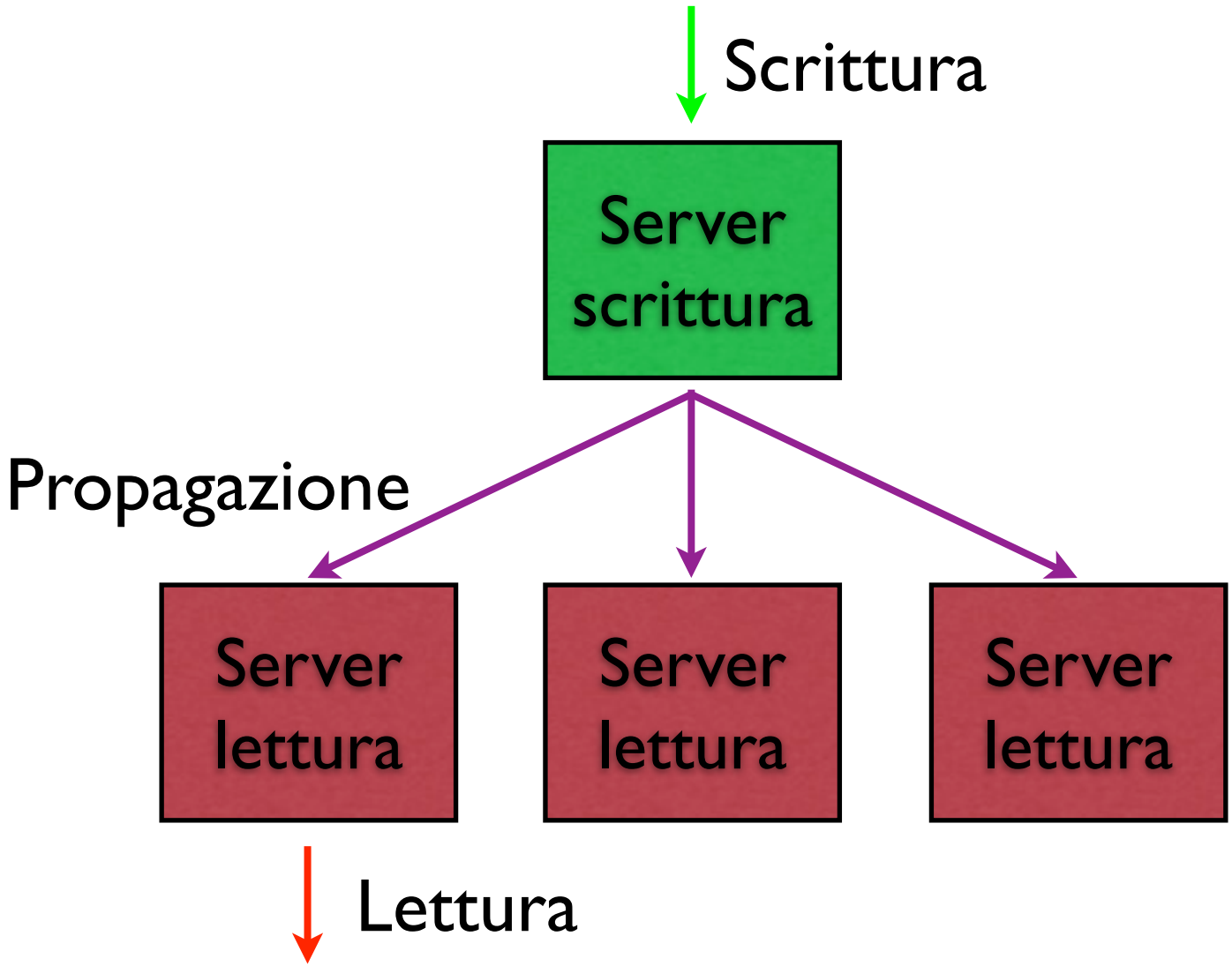
Master-Slave replication



Master-Slave replication



Master-Slave replication

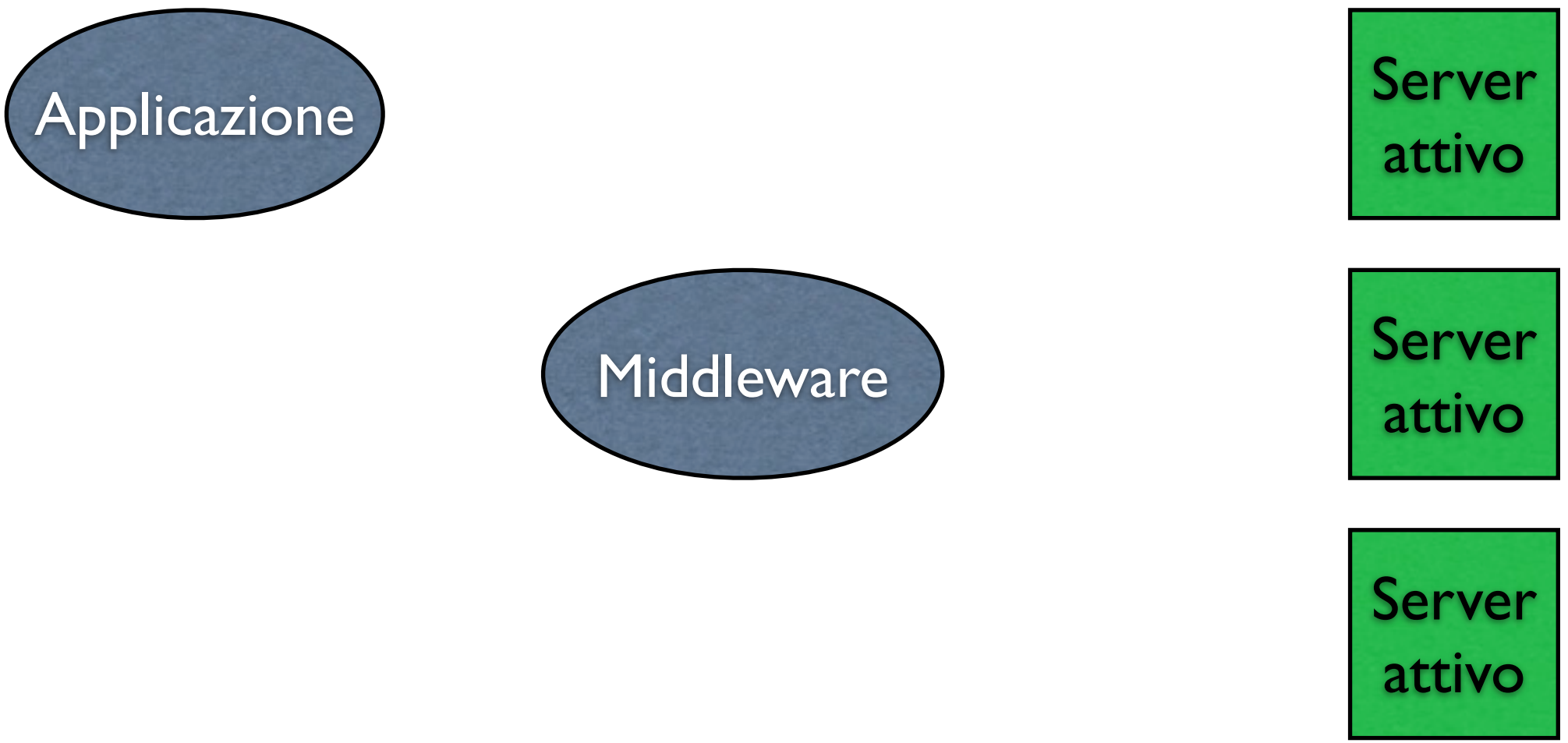


Statement-Based Replication Middleware

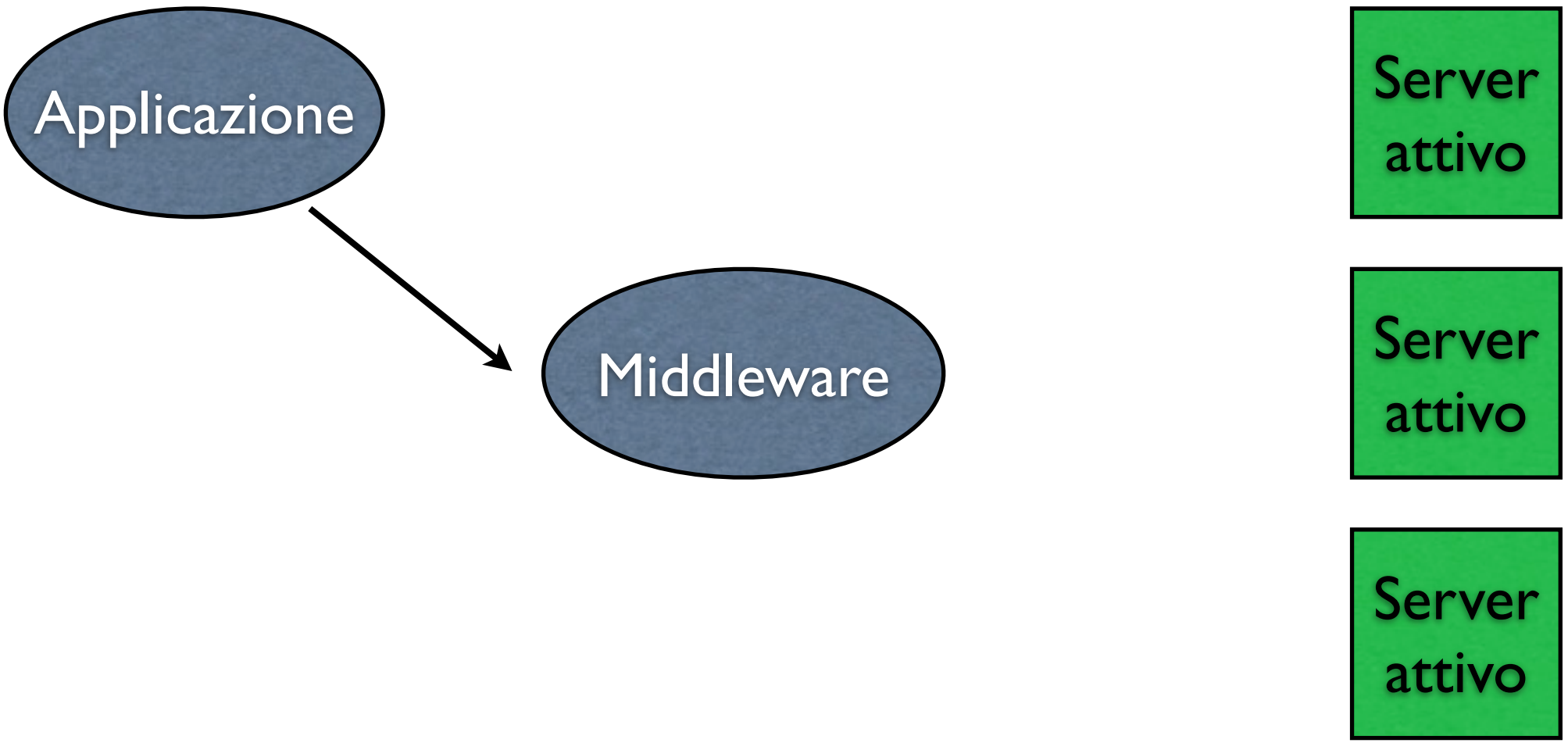
Un'altra soluzione di replica ed alta affidabilità è data dall'inserimento di uno strato di software che replica le query su tutti i server.

Un esempio di middleware è PgPool, che offre replica, load balancing ed esecuzione parallela delle query. Il limite di questo software è che alcuni valori non possono essere correttamente replicati, es: sequenze e timestamp e simili.

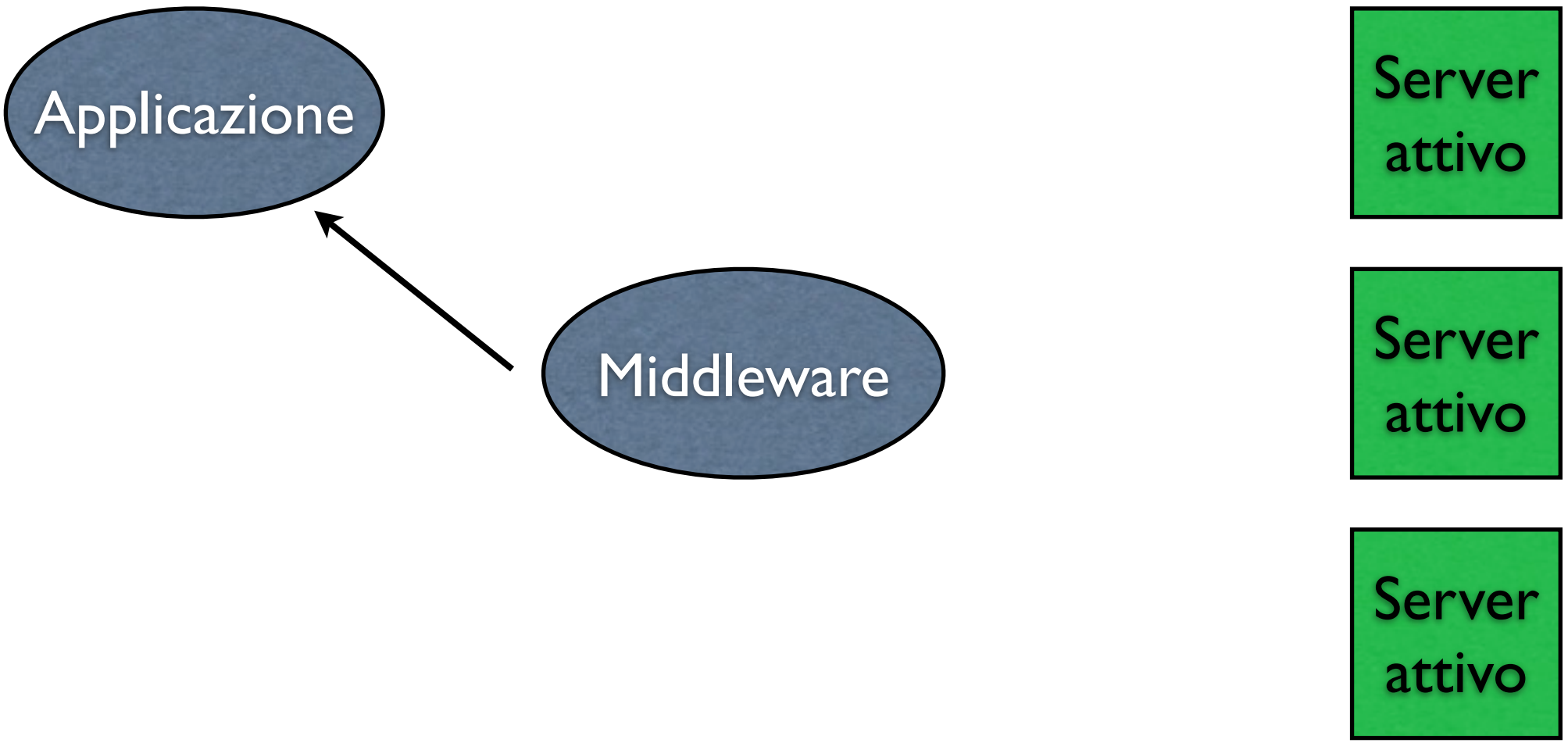
Statement-Based Replication Middleware



Statement-Based Replication Middleware



Statement-Based Replication Middleware



Asynchronous multimaster replication

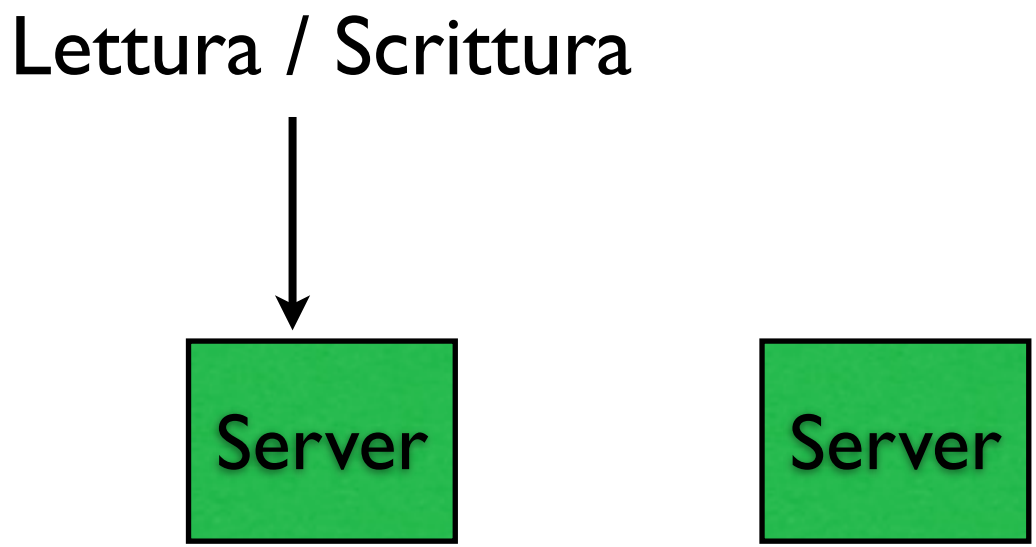
Nella replica multimaster asincrona tutte le operazione di lettura e scrittura sono effettuate su un singolo nodo e successivamente vengono propagate negli altri elementi del cluster.

Questo permette di rispondere rapidamente all'applicazione ma espone a rischi di conflitti.

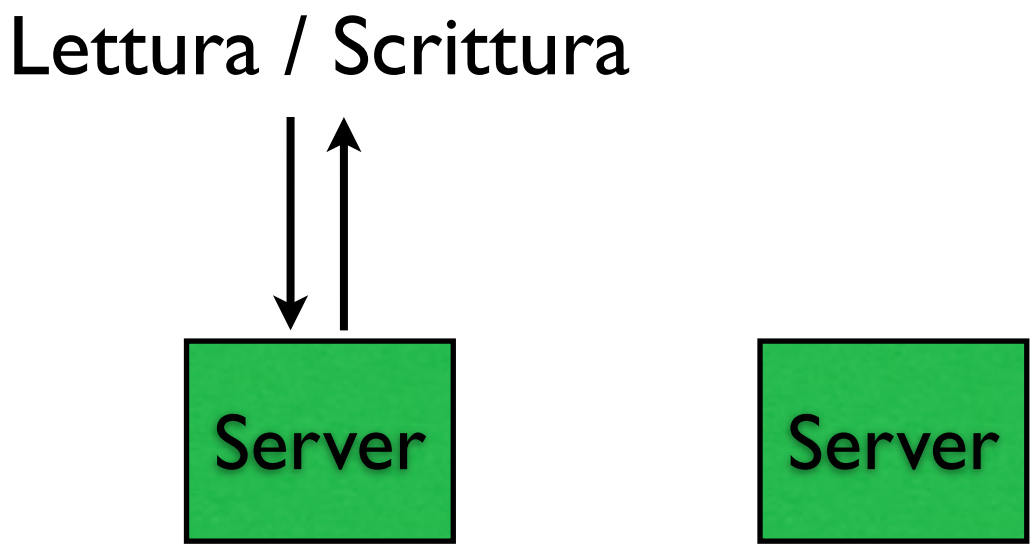
Asynchronous multimaster replication

Lettura / Scrittura

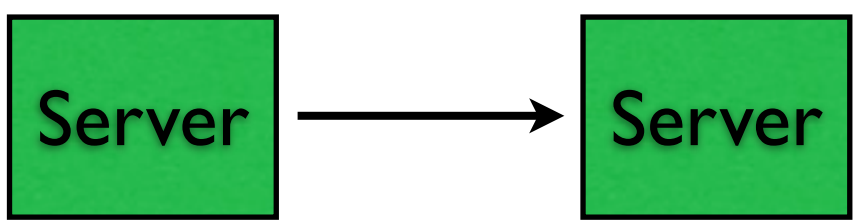
Asynchronous multimaster replication



Asynchronous multimaster replication



Asynchronous multimaster replication



Replica asincrona

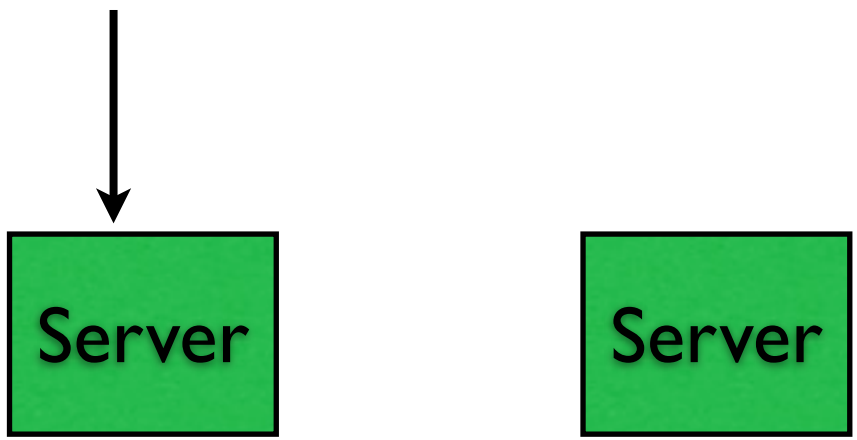
Asynchronous multimaster replication

Asynchronous multimaster replication

drop table tabella;

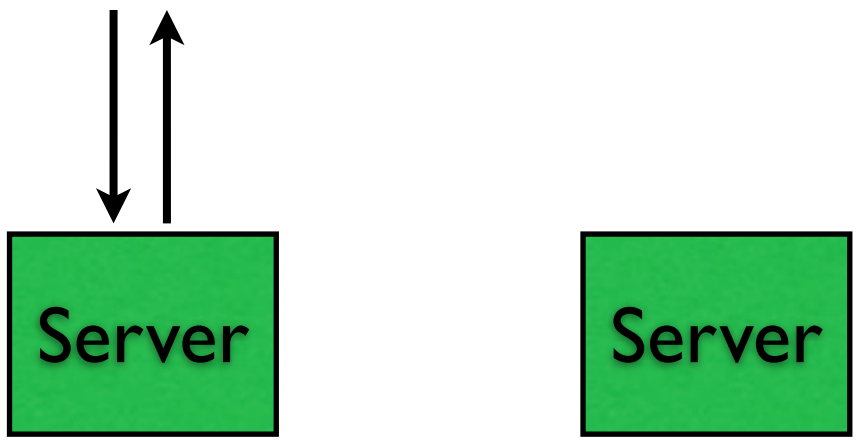
Asynchronous multimaster replication

drop table tabella;



Asynchronous multimaster replication

drop table tabella;



Asynchronous multimaster replication

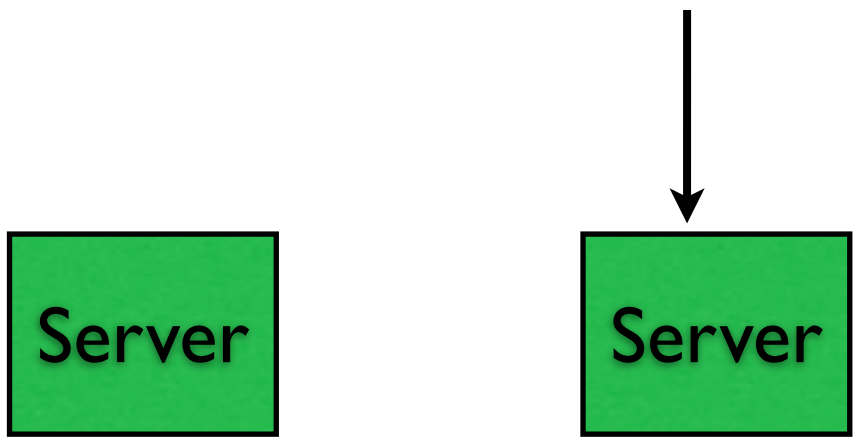
drop table tabella;

update tabella ...;

Asynchronous multimaster replication

drop table tabella;

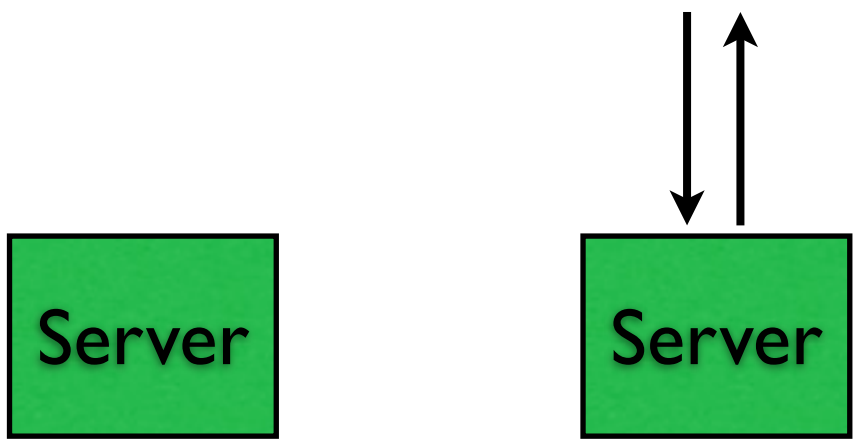
update tabella ...;



Asynchronous multimaster replication

drop table tabella;

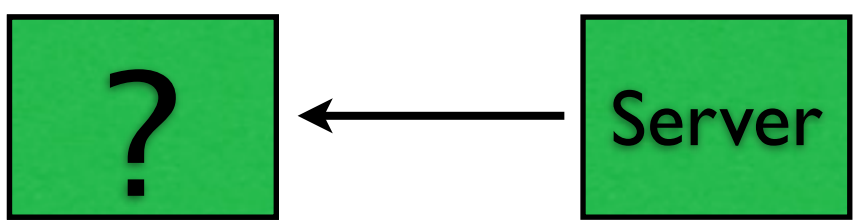
update tabella ...;



Asynchronous multimaster replication

drop table tabella;

update tabella ...;



conflitto!!!

Asynchronous multimaster replication

Le modifiche asincrone possono portare a conflitti che possono essere risolti o dall'applicazione o dal software di replica.

Le applicazioni che funzionano con un sistema di replica asincrona devono essere in grado di gestire i fallimenti delle transazioni e i relativi rollback.

SW replica asincrona: PostgreSQL Replicator

Synchronous multimaster replication

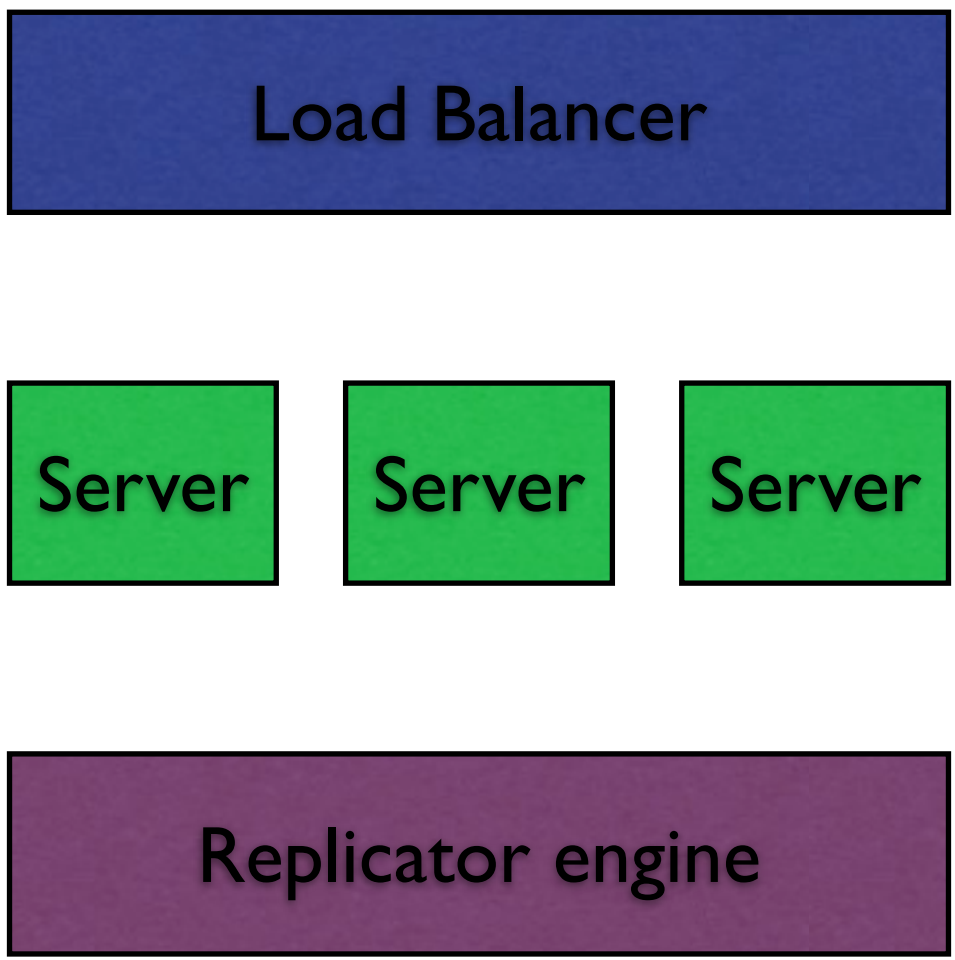
La replica sincrona multimatser permette di parallelizzare i server e farli funzionare come se fossero uno.

Ovviamente, poiché le operazioni di lettura e scrittura sono eseguite su ogni server sarà necessario più tempo per eseguirle.

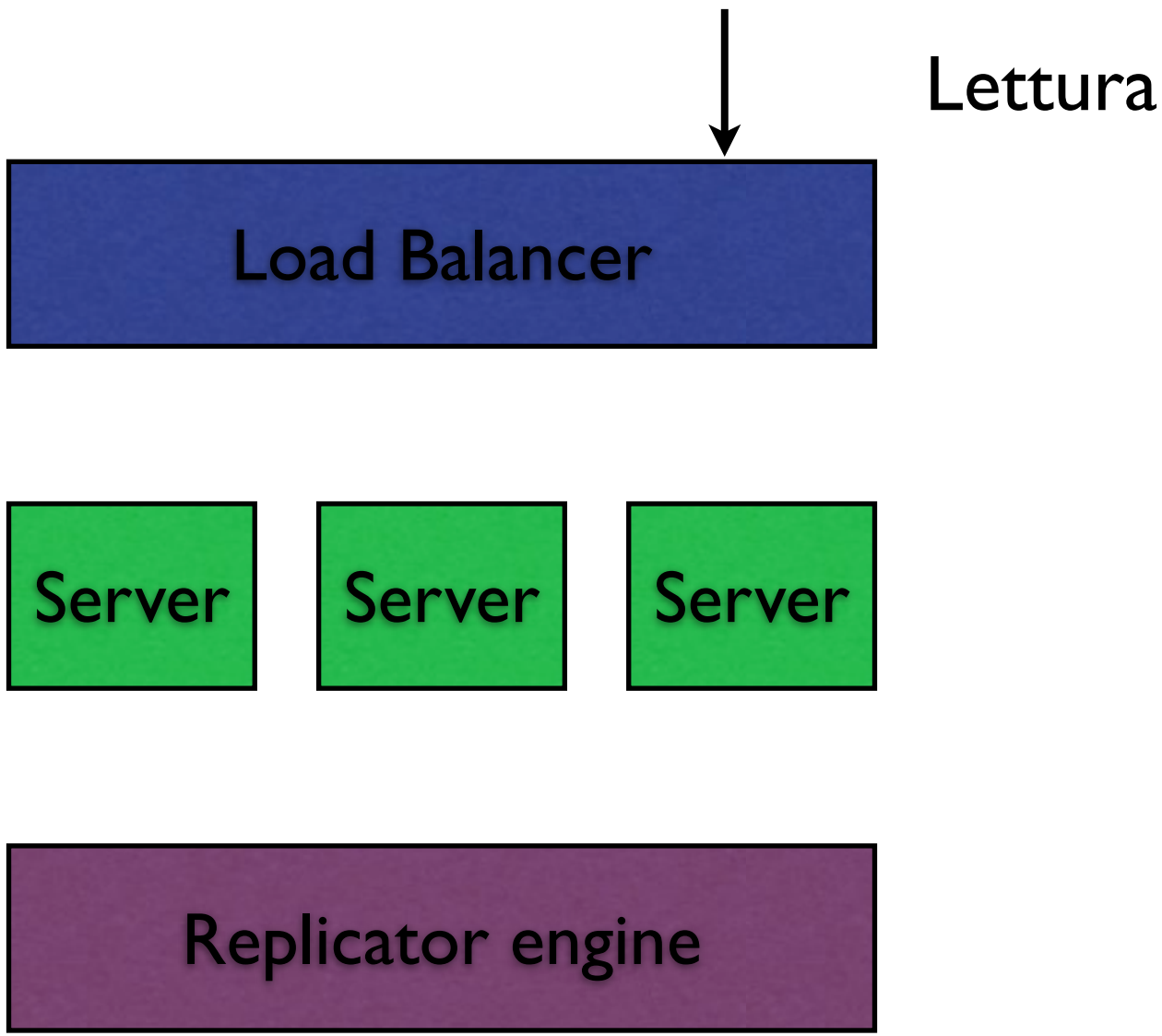
Il vantaggio è che il cluster appare come un solo computer.

Synchronous multimaster replication

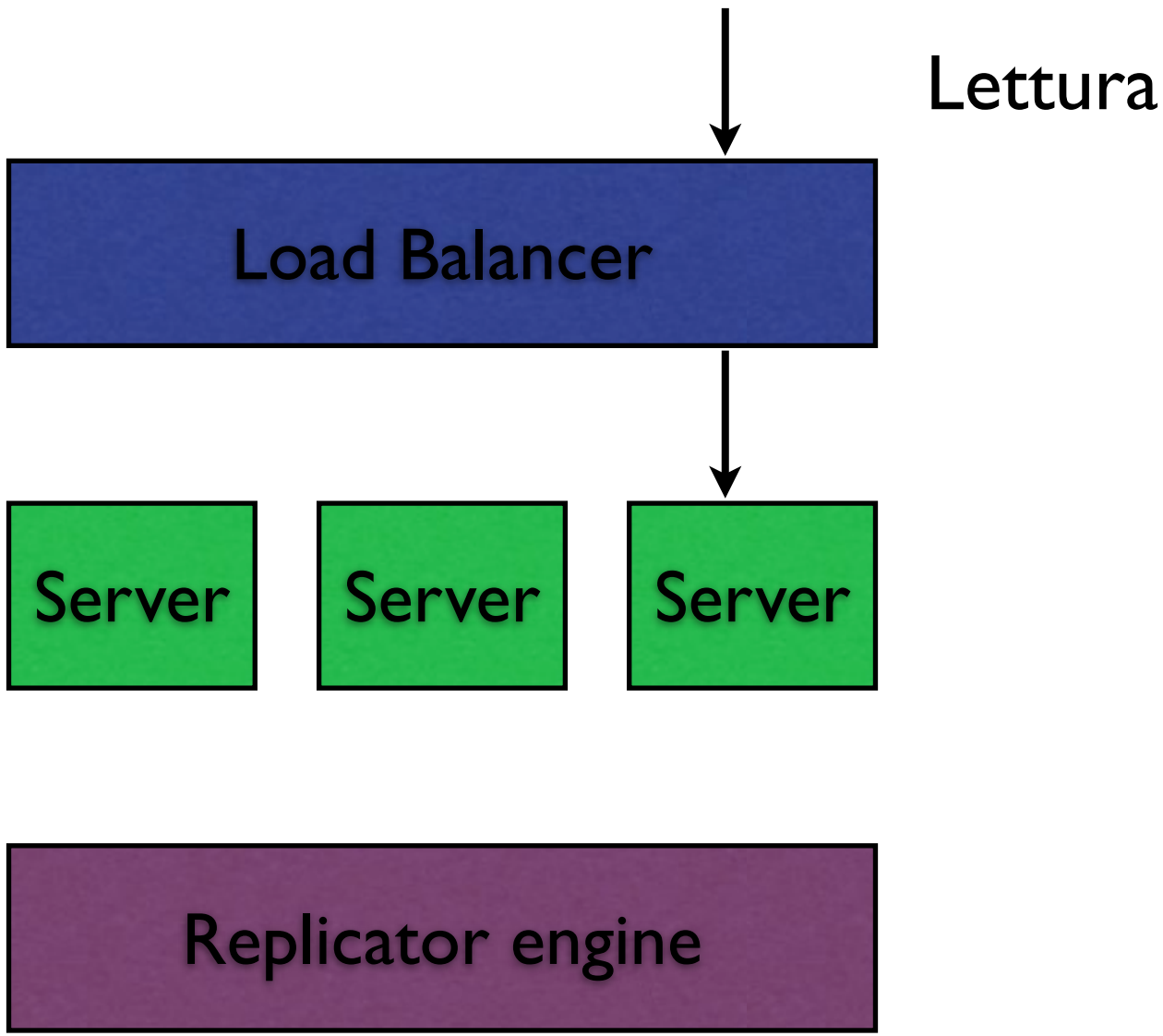
Lettura



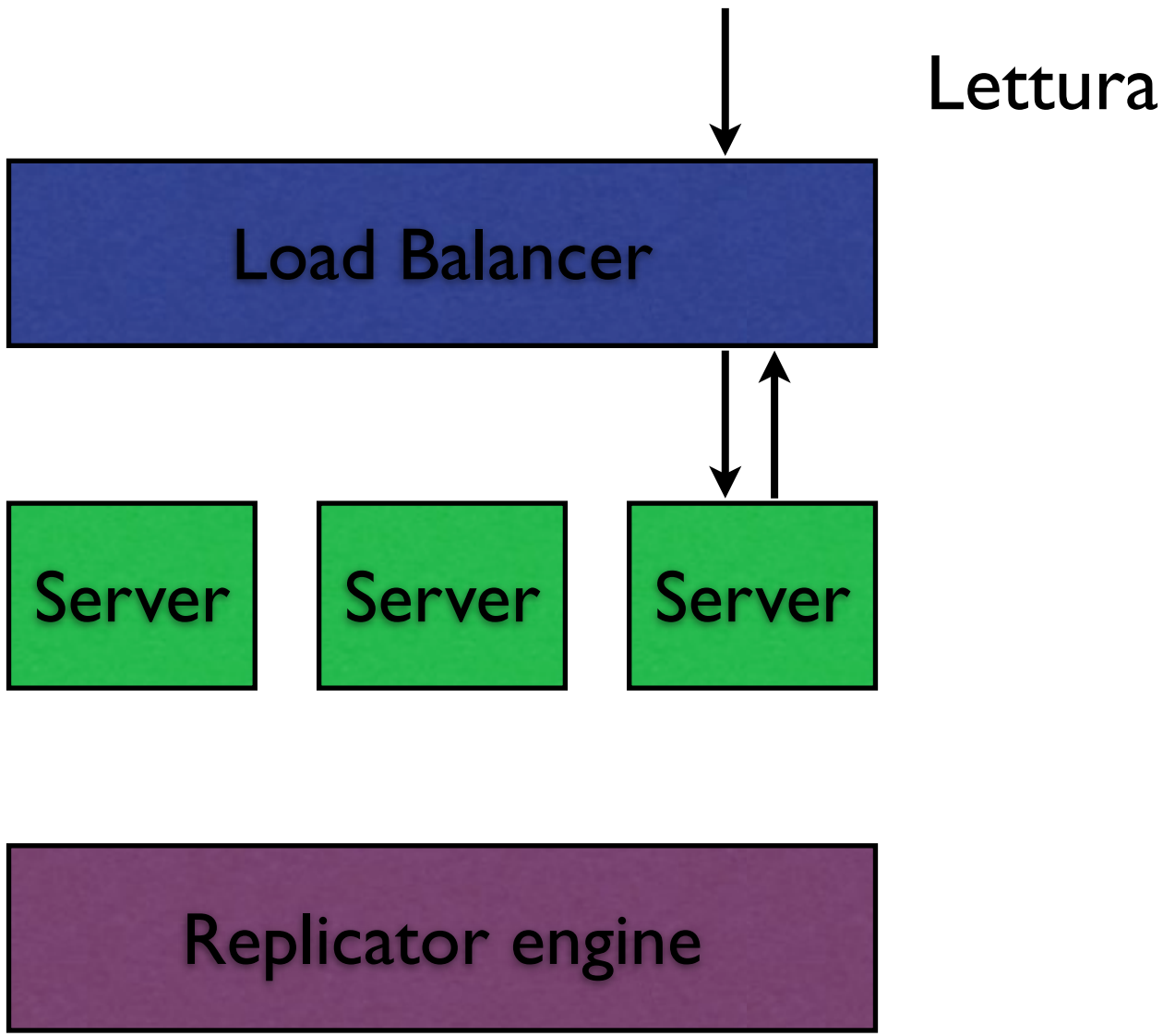
Synchronous multimaster replication



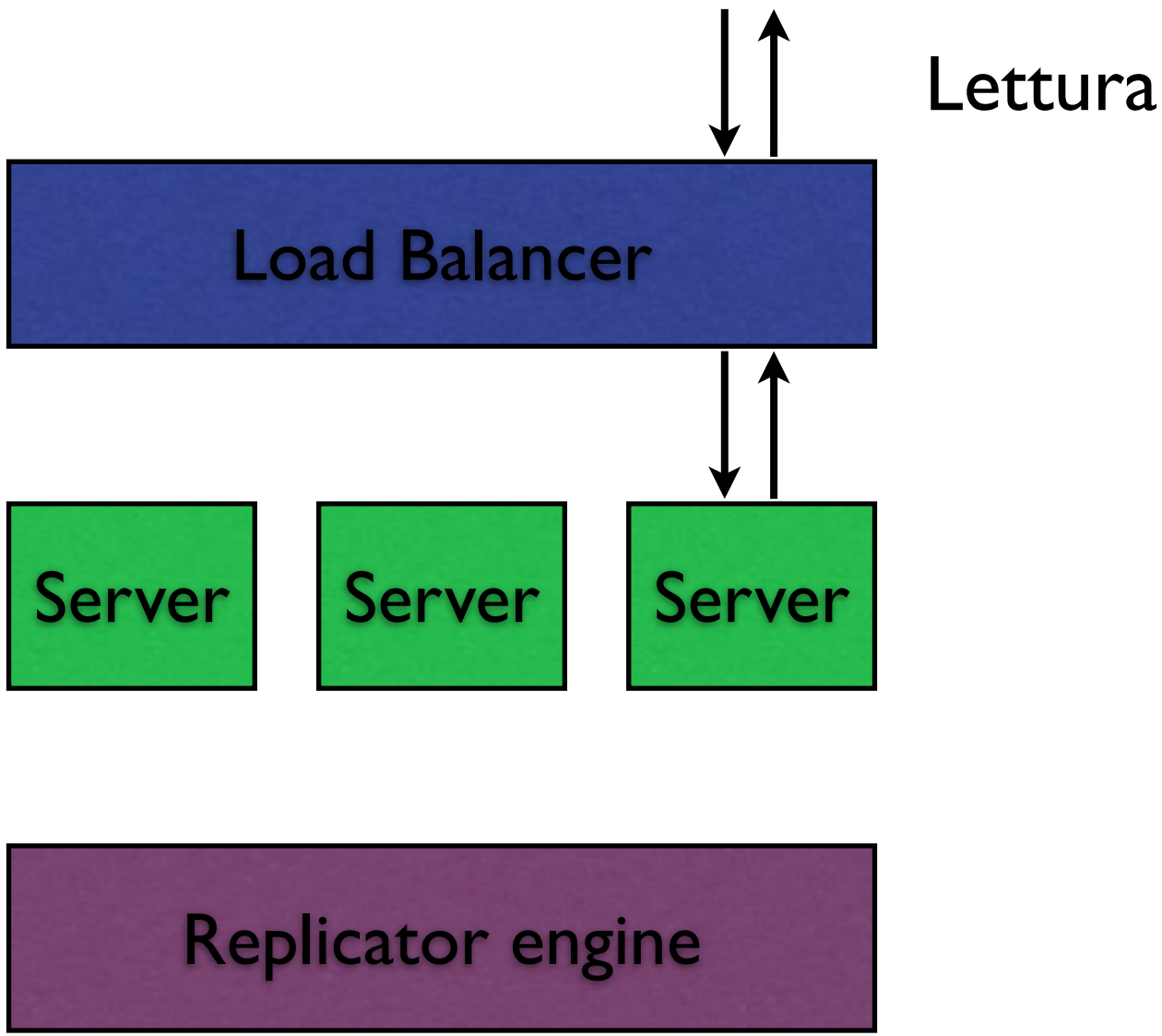
Synchronous multimaster replication



Synchronous multimaster replication

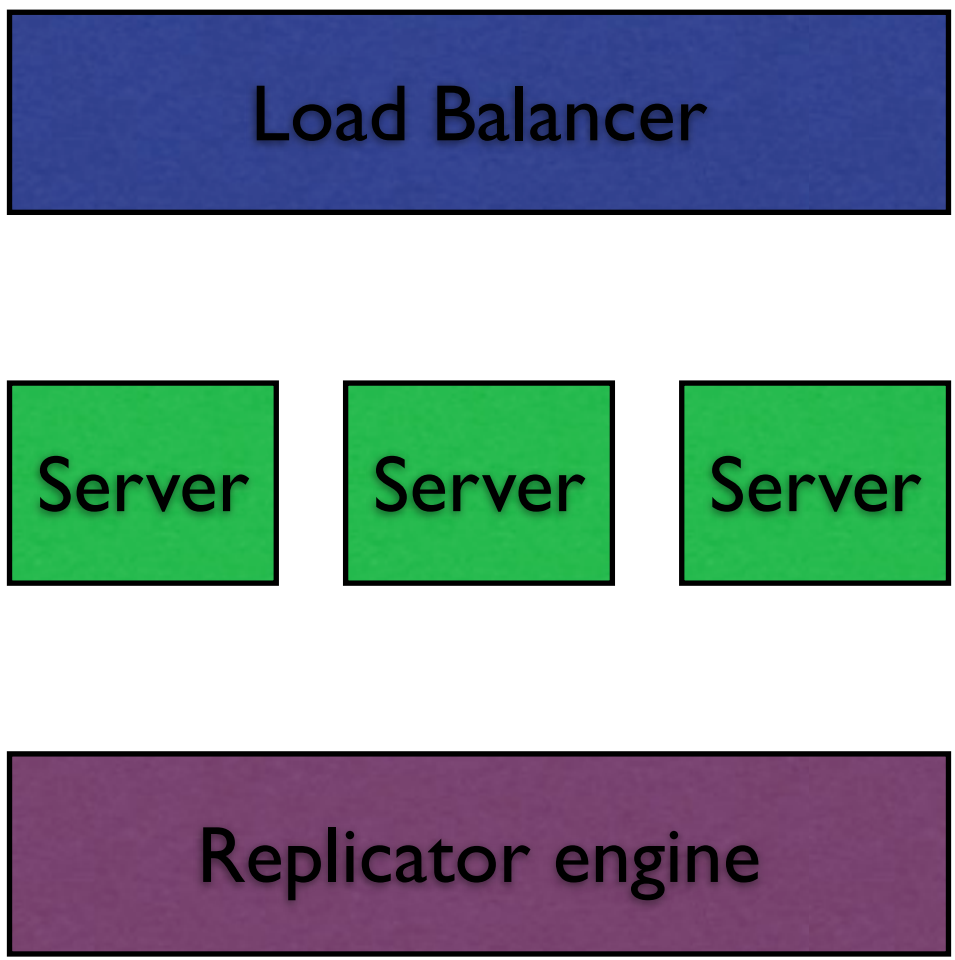


Synchronous multimaster replication

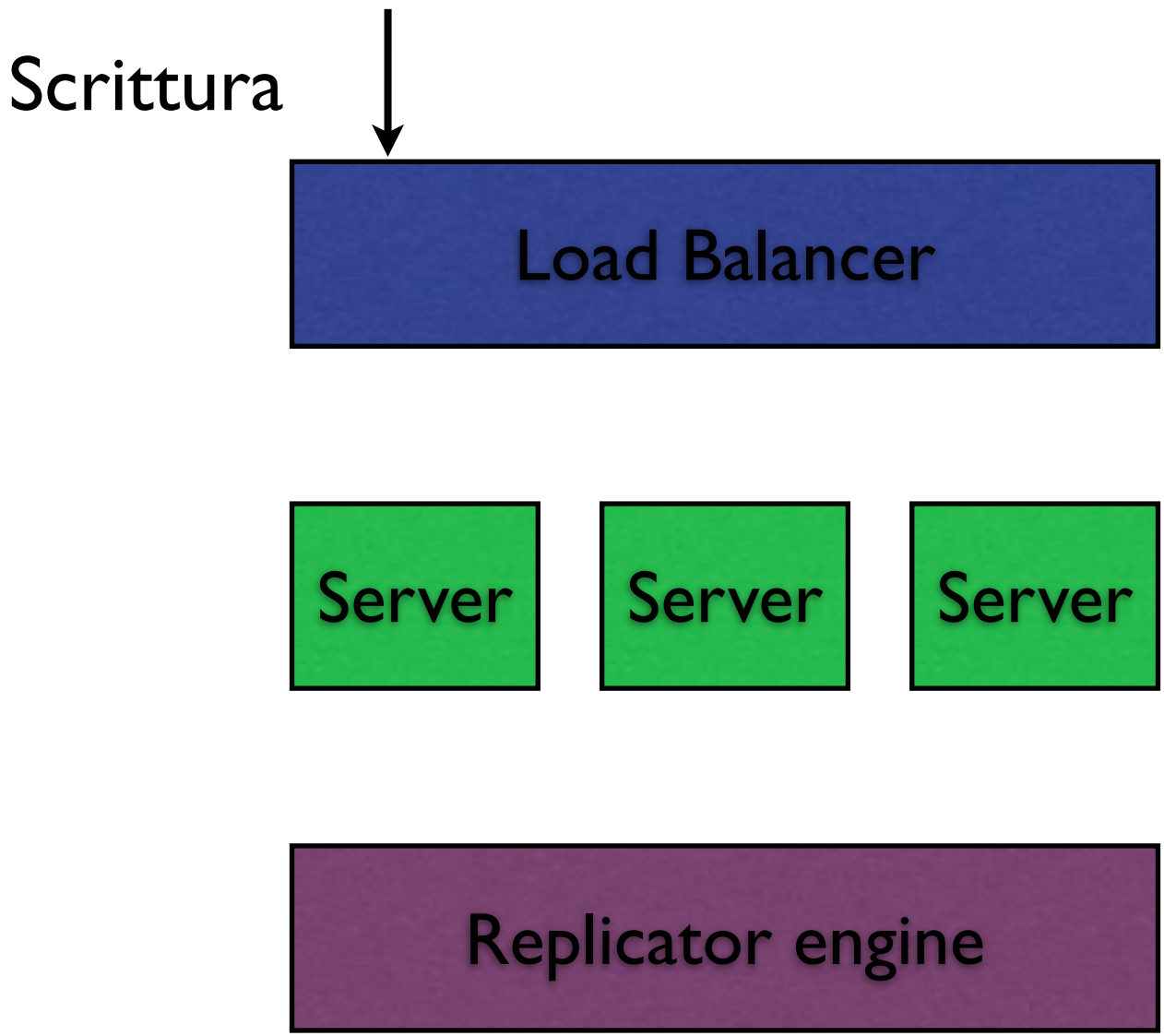


Synchronous multimaster replication

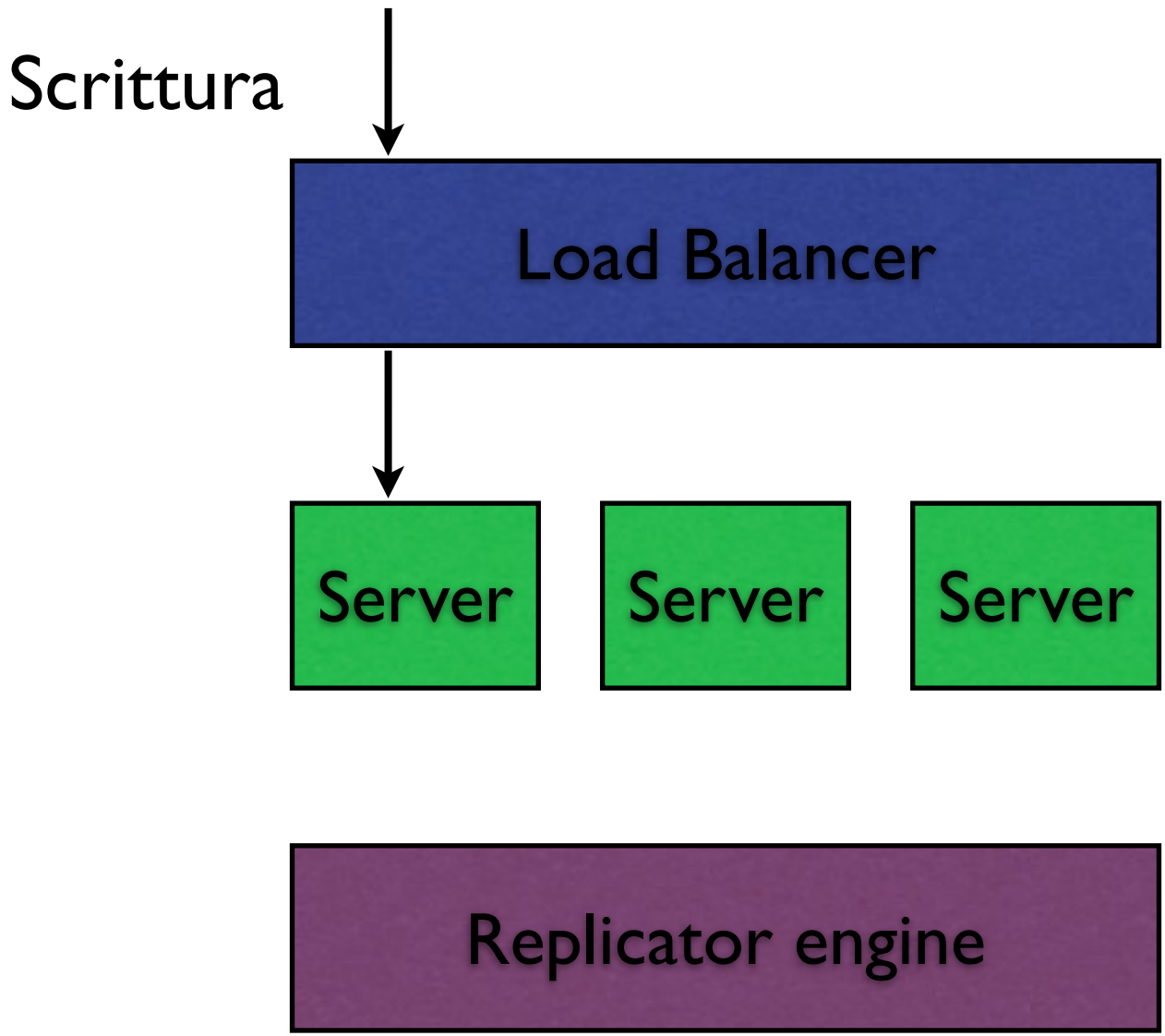
Scrittura



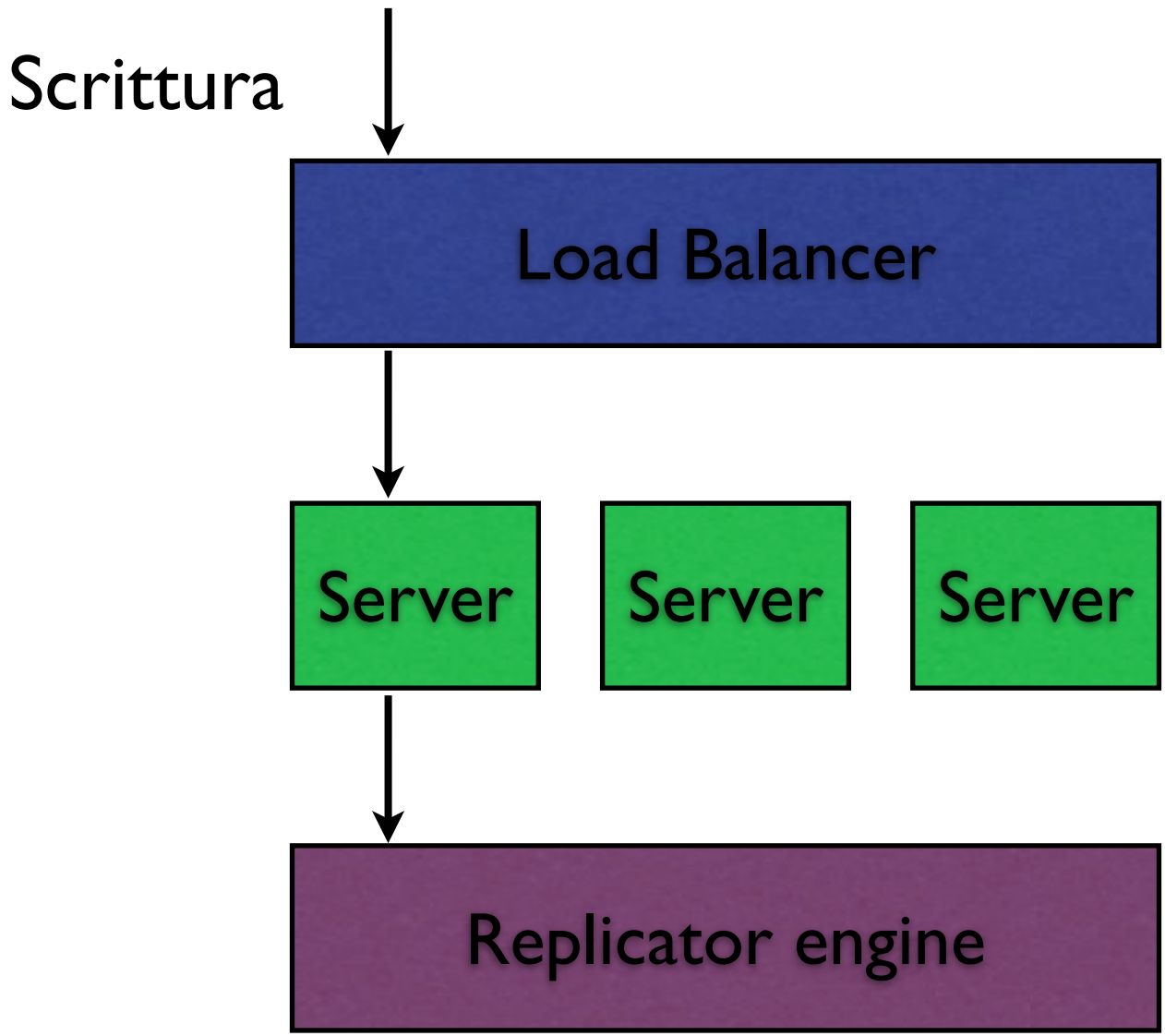
Synchronous multimaster replication



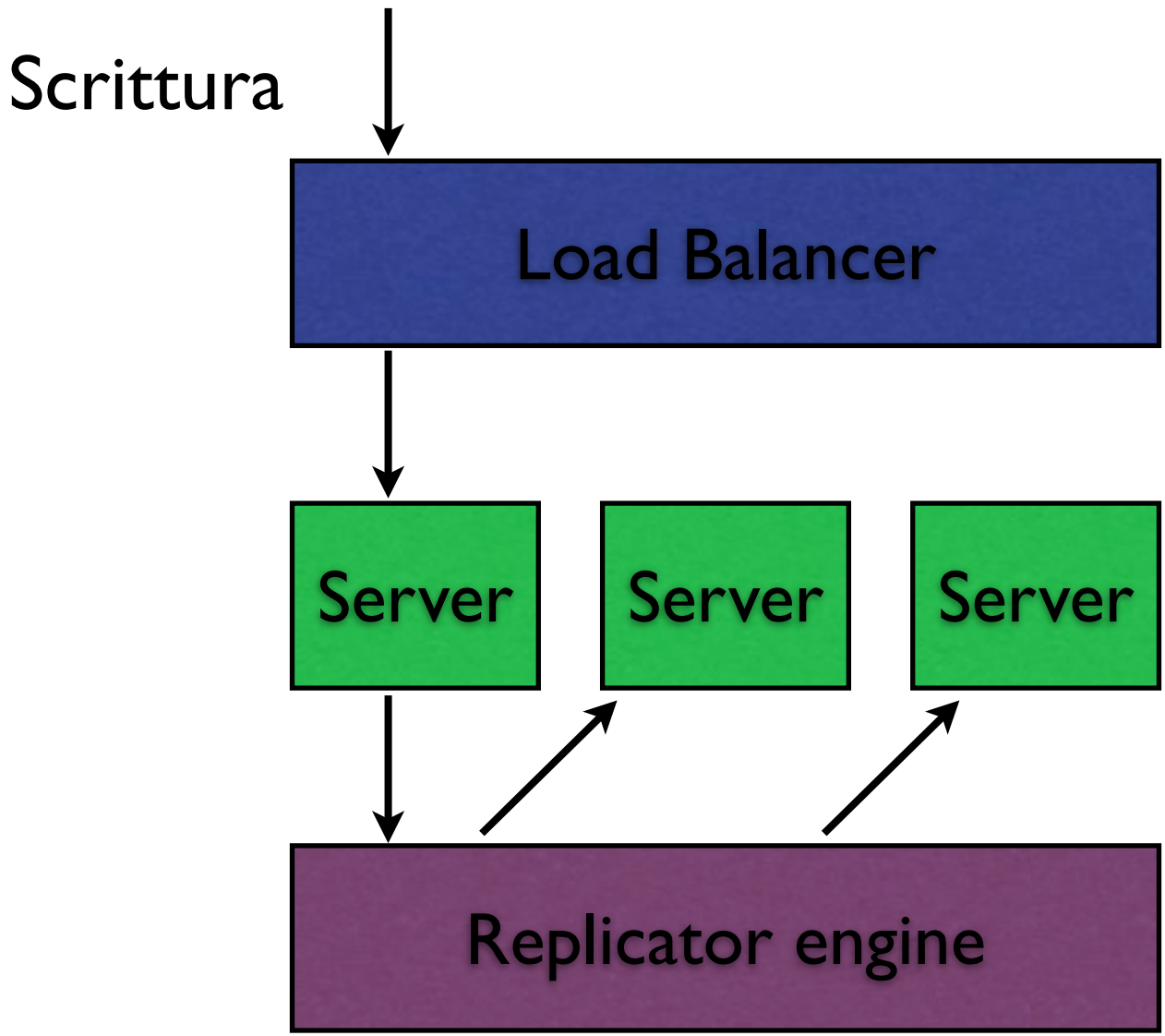
Synchronous multimaster replication



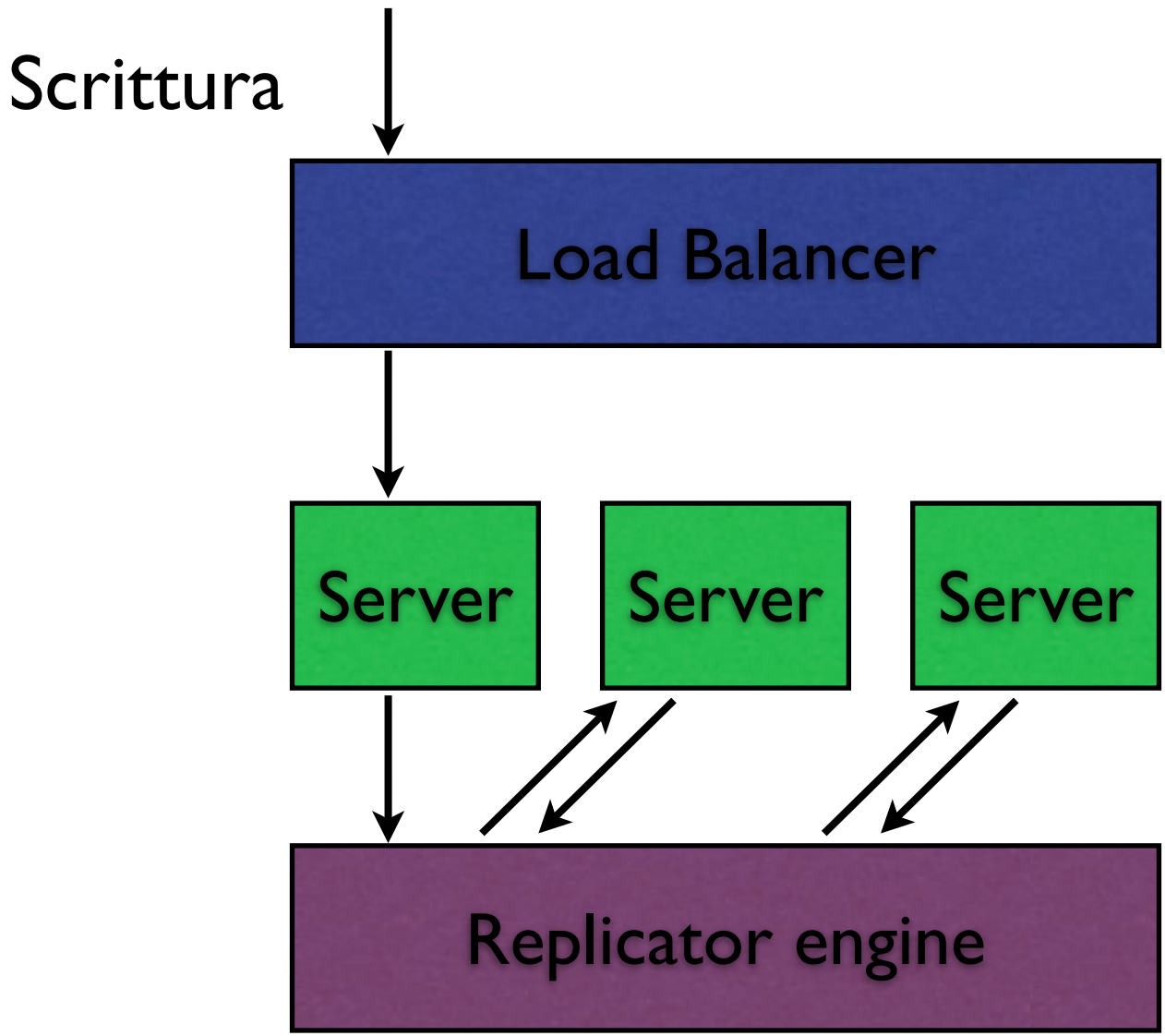
Synchronous multimaster replication



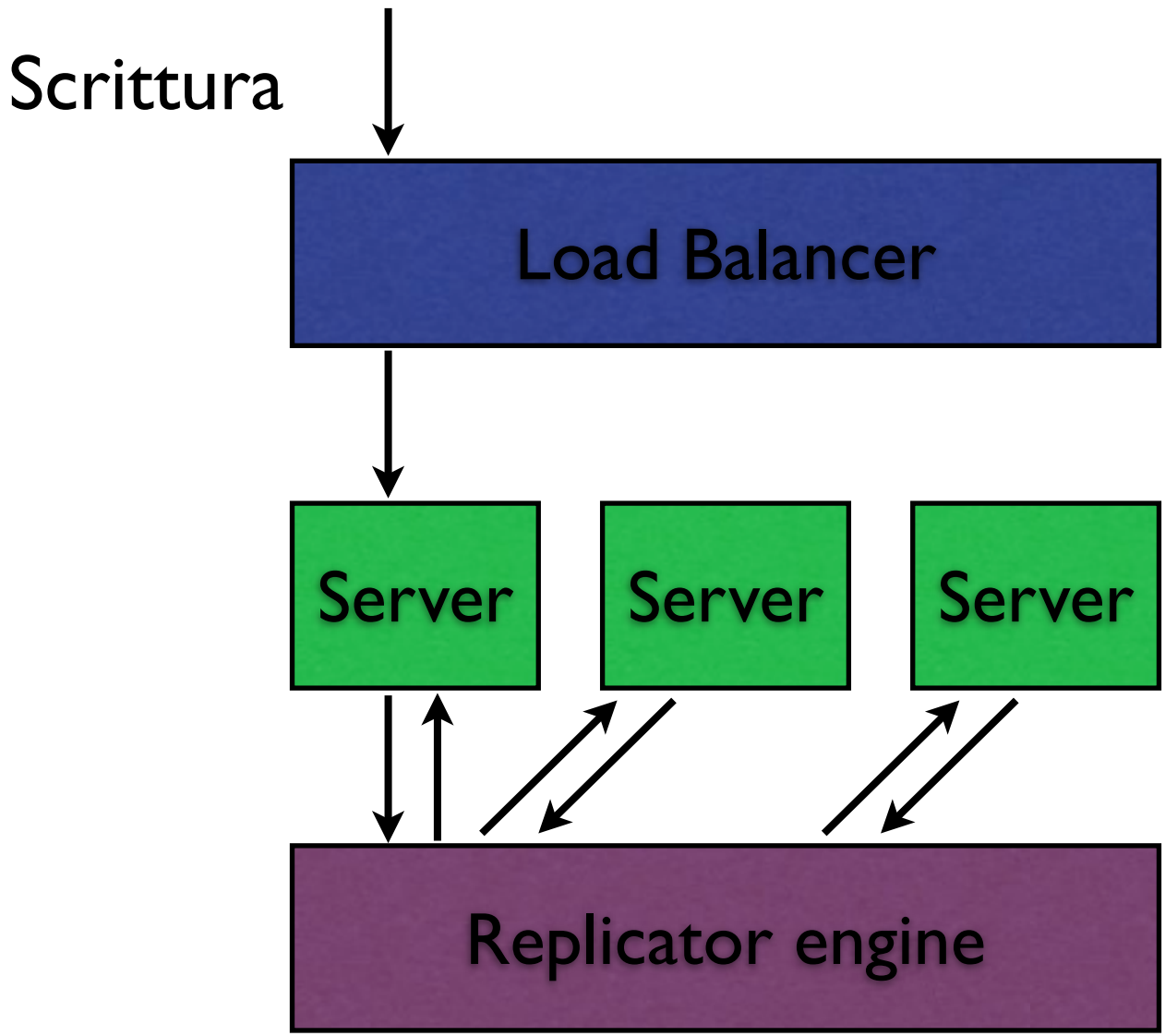
Synchronous multimaster replication



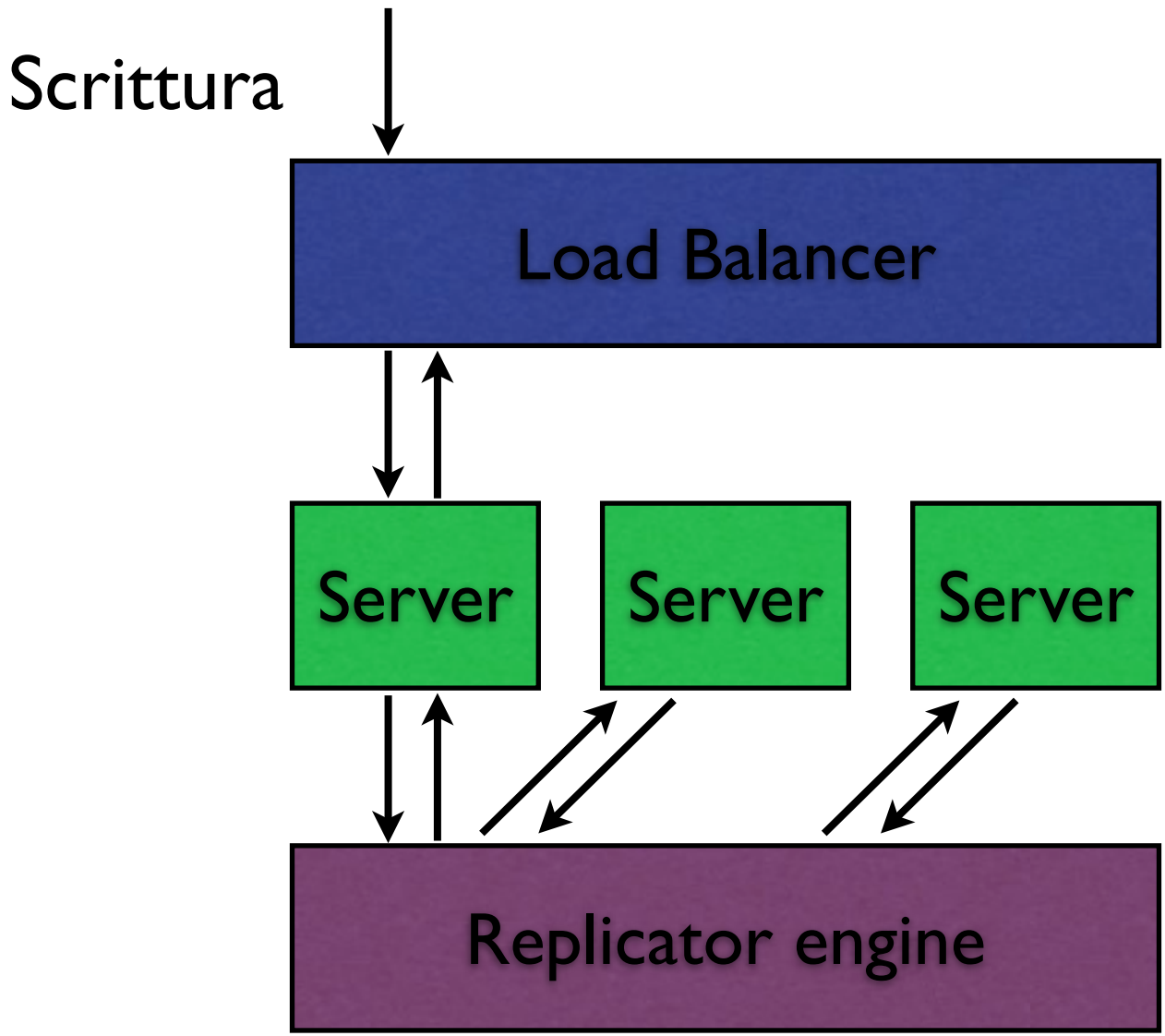
Synchronous multimaster replication



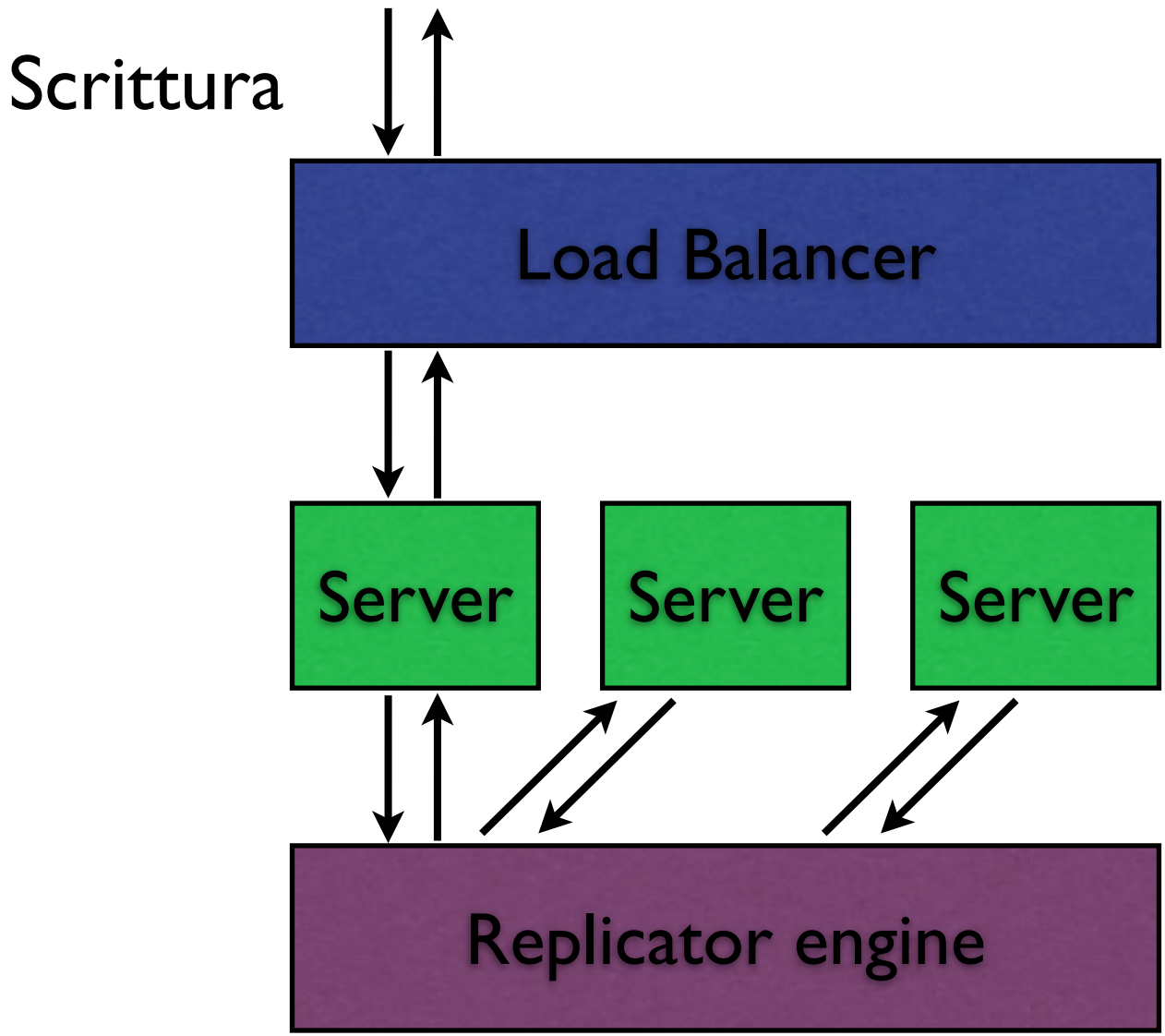
Synchronous multimaster replication



Synchronous multimaster replication



Synchronous multimaster replication



Synchronous multimaster replication

La replica sincrona è spesso considerata la soluzione per tutti i problemi, sia sistemistici che di programmazione.

Di fatto realizza un sistema MVCC distribuito e trasparente all'utente. Si paga in termini di velocità persa.

Due pacchetti: PgCluster e CyberCluster

Prestazioni

Ma è veramente lento come dicono?

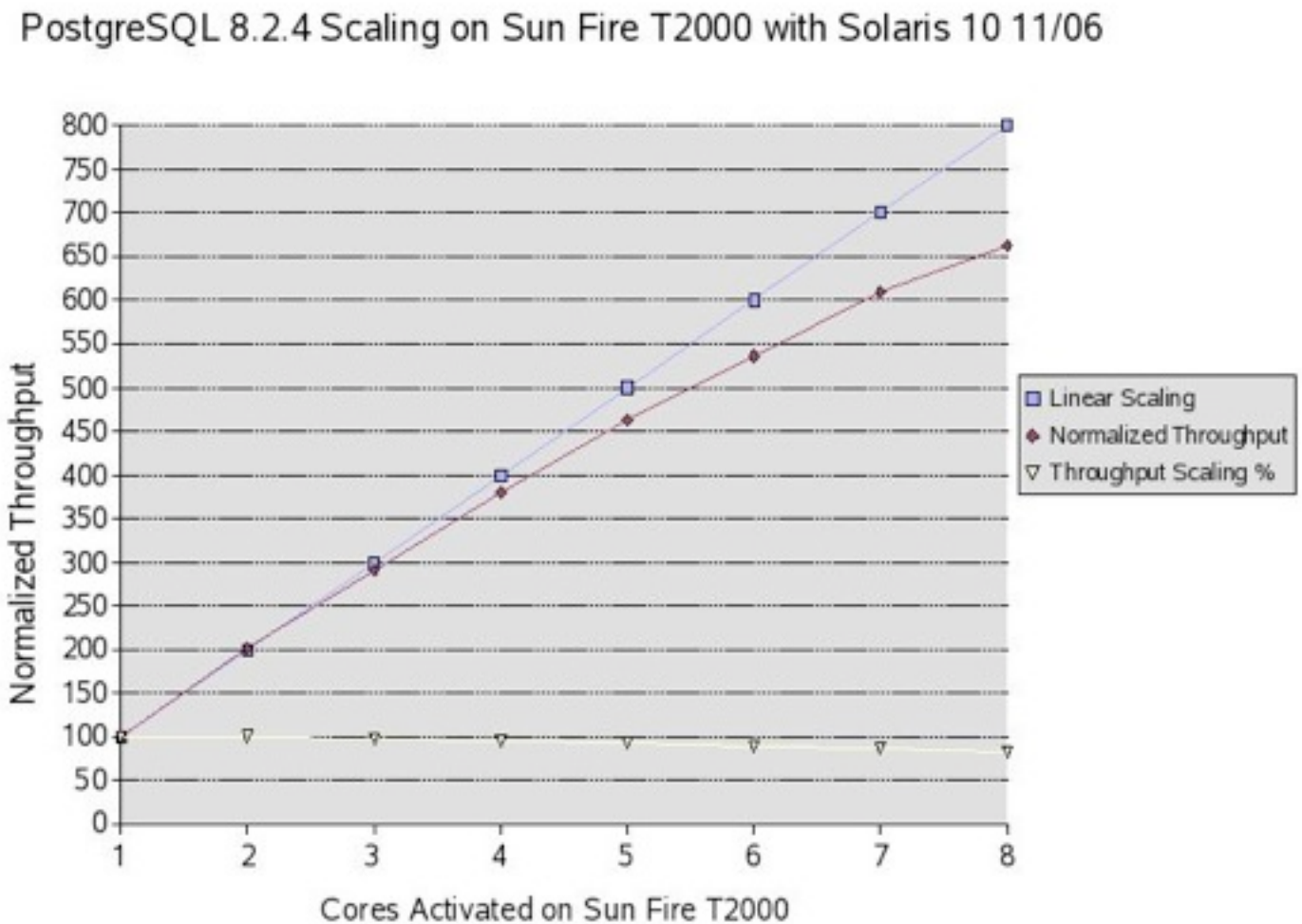
Confronto con altri database

Nel passato PostgreSQL aveva fama di essere più lento di altre basi di dati, opensource e non.

I benchmark 2007 hanno messo in luce come PostgreSQL abbia una velocità maggiore o uguale a quella di MySQL e comparabile con Oracle.

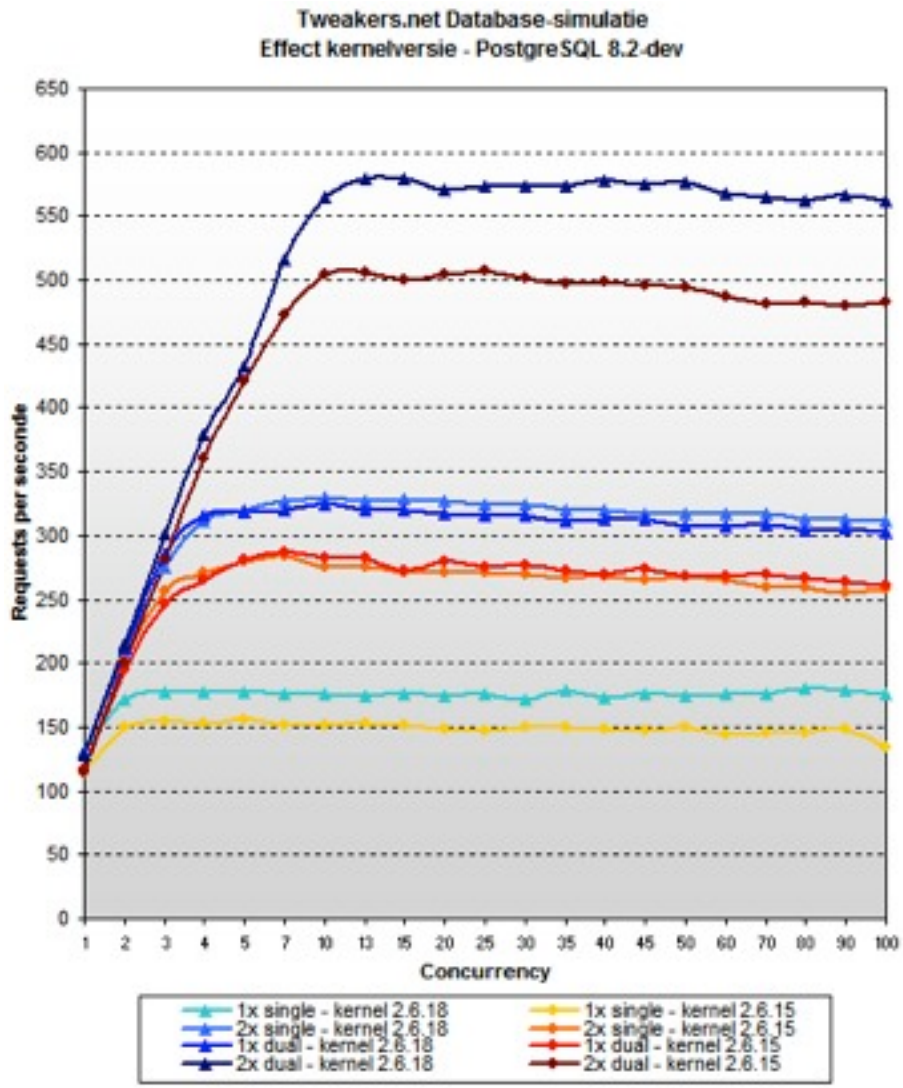
Fonte: PostgreSQL publishes first real benchmark

Scalabilità



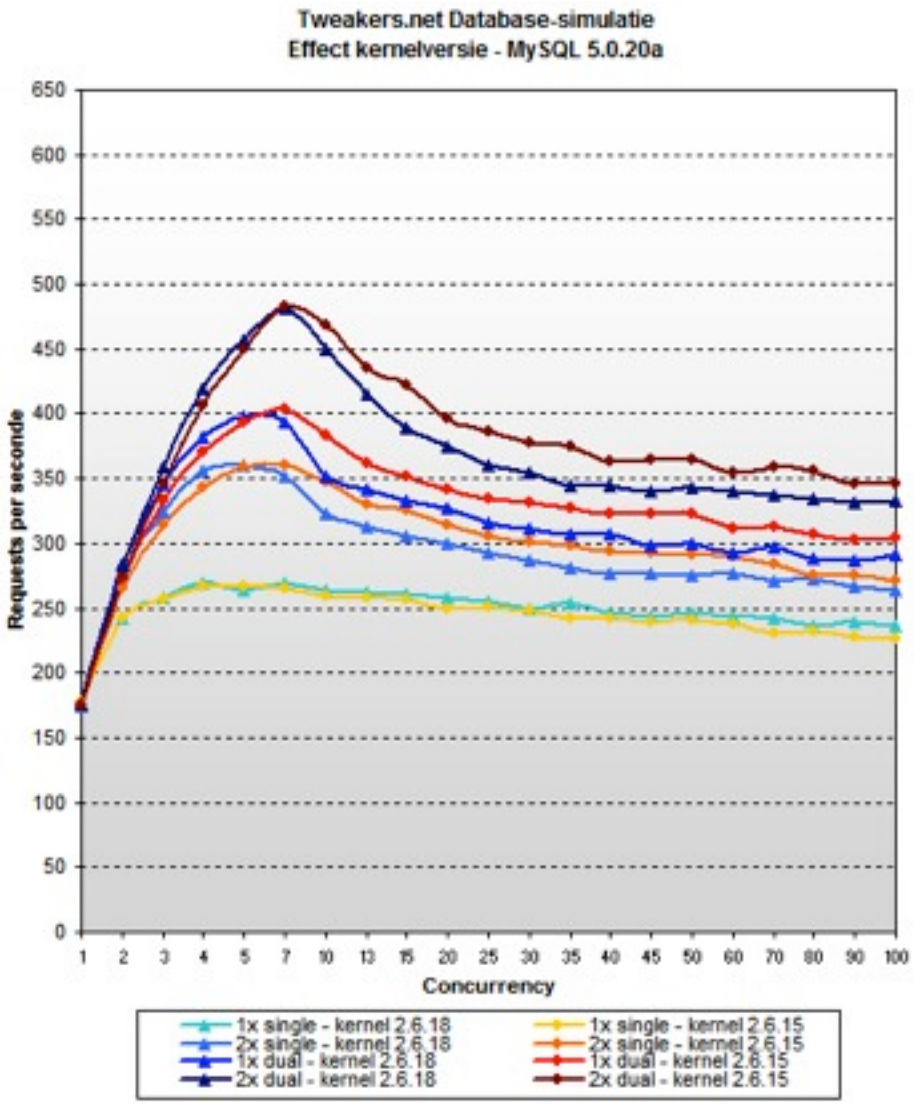
Fonte Jignesh Shah's Weblog

Scalabilità



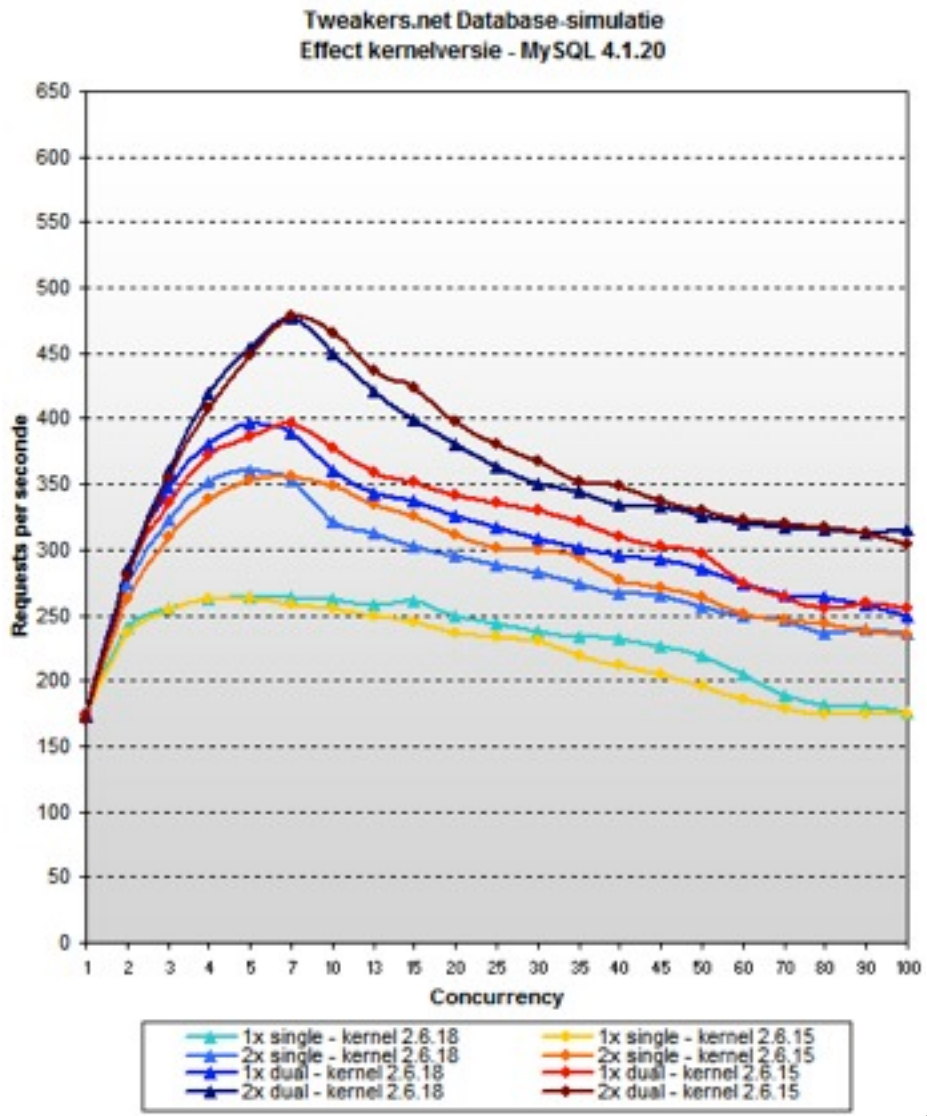
Fonte: Database test: dual Intel Xeon 5160

Scalabilità



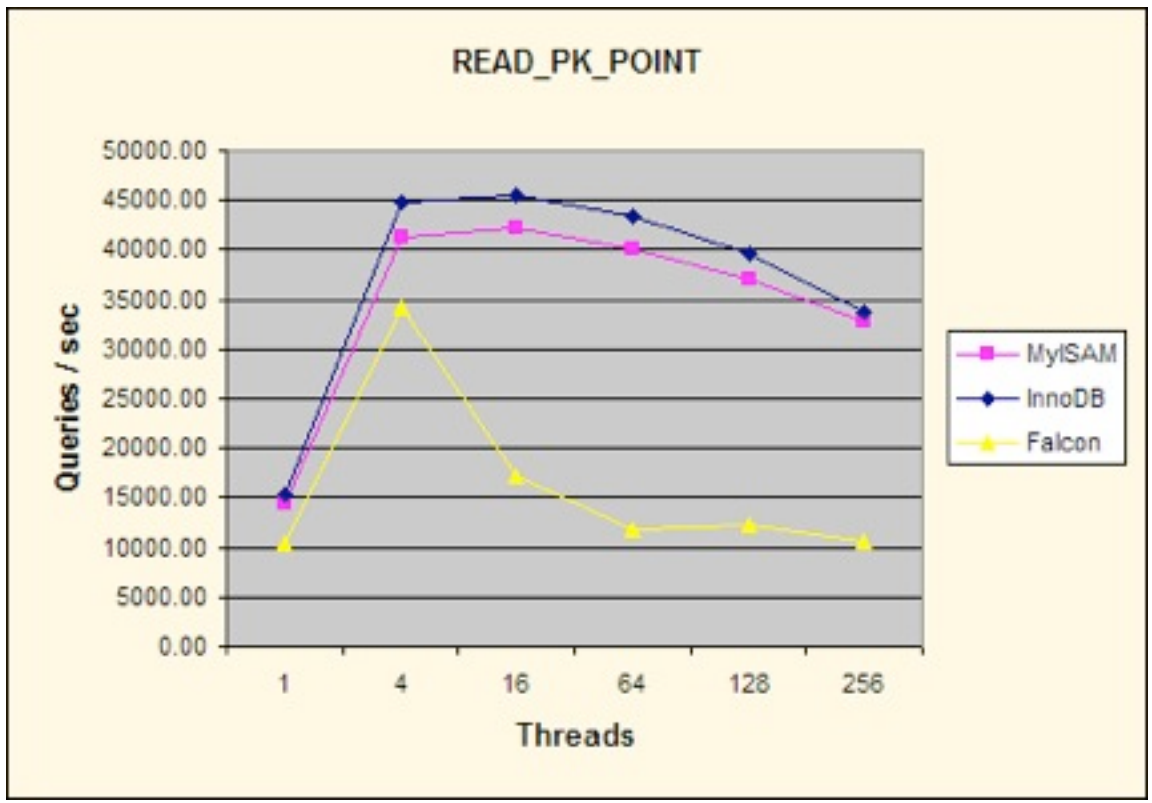
Fonte: Database test: dual Intel Xeon 5160

Scalabilità



Fonte: Database test: dual Intel Xeon 5160

Confronto con il futuro MVCC di mysql...



Fonte innodb-vs-myisam-vs-falcon-benchmarks

L’altra roba

- Performance tuning
- Stored procedures
- Planner e performance delle query
- Estensioni di PostgreSQL

Performance tuning

Le prestazioni di un database possono dipendere da molti aspetti, legati sia al server che al software.

- Controllo dell'attività del database;
- Controllo dello stato del database;
- Controllo della configurazione del server.

Raccolta statistiche

PostgreSQL ha un sistema di acquisizione statistiche che vengono raccolte in tabelle.

Alcune di queste tabelle sono:

- pg_stat_user_tables
- pg_stat_user_indexes
- pg_statio_user_tables
- pg_statio_user_indexes

Vacuum

Il comando VACUUM deve essere lanciato sui database con una certa regolarità per

- Recuperare lo spazio perduto in righe cancellate o modificate;
- aggiornare le statistiche usate dal planner;
- evitare perdite di dati dovuti a *transaction ID wraparound*.

Vacuum

Come abbiamo visto all'inizio della presentazione, il MVCC impedisce la cancellazione delle righe modificate o cancellate nelle tabelle.

Il comando VACUUM effettua una scansione delle tabella e marca le tuple obsolete affinché siano riutilizzate. Di conseguenza il database non viene compresso e l'operazione può essere effettuata senza interrompere il servizio.

Vacuum

Esiste una variante, VACUUM FULL, che non solo marca le tuple come obsolete, ma provvede immediatamente a compattare i file del database e lo spazio usato dalle righe obsolete viene istantaneamente liberato.

Poiché VACUUM FULL riscrive le tabelle, deve acquisire un lock esclusivo. Di conseguenza, deve essere eseguito quando il blocco non pregiudica il servizio.

Analyze

Durante l'operazione di vacuum è possibile aggiornare le statistiche sull'accesso alle tabelle.

Questo aggiornamento viene effettuato con il comando VACUMM (FULL) ANALYZE

Configurazione del server

Il file di configurazione postgresql.conf permette di modificare molti parametri di configurazione.

parametro	descrizione
shared_buffers	Dimensione del buffer. PostgreSQL fa uso aggressivo dei buffer del sistema operativo, oltre a un limite basso (128-256 MB) non fa cambiare le prestazioni.
max_fsm_pages	Dimensione della mappa dello spazio libero. Serve per mappare le tuple morte. Se la mappa contiene tutte le tuple morte, allora VACUUM FULL non serve.
sort_memory	Dimensione massima del buffer di ogni operazione di sort per ogni connessione.

Stored Procedures

Molti database prevedono la possibilità di scrivere delle funzioni direttamente nel database e di eseguirle al suo interno.

Vantaggi:

- risparmio di banda, perché si trasferiscono solo i dati e non le operazioni;
- riutilizzabilità della *business logic*.

Stored Procedures

In PostgreSQL sono disponibili diversi linguaggi per scrivere le stored procedures:

- PL/pgSQL
- PL/Tcl
- PL/Perl
- PL/Python

Ci sono inoltre C, Java, Lua, ...

Stored Procedures

Le stored procedures permettono assieme alla parte attiva del database di ridurre la mole di codice scritto.

Supponiamo per esempio di voler creare una tabella in cui salvare id, username e password degli utenti e di volerci assicurare che lo username sia unico e di voler creare una funzione che ritorni l'id del nuovo utente inserito o -1 in caso di errore.

Stored Procedures

Senza stored procedures e parte attiva del db la tabella può essere scritta come:

```
create table users (
        id serial,
        username text,
        password text);
```

Dove serial è il tipo di dato autoincrementante.

Stored Procedures

Esempio in python:

```
cursor = db.cursor ()
cursor.execute (“select id from users where username=‘%s’”; % username)

if not cursor.fetchone ():
  return -1

cursor.execute (“insert into users values (default, ‘%s’, ‘%s’);” %
                                                    (username, password))
cursor.execute (“select id from users where username=‘%s’”; % username)
result = cursor.fetchone()

if not result:
  return -1
db.commit ()
return result[0]
```

Stored Procedures

La parte attiva del db serve per imporre il vincolo di unicità dello username

```
create table users (
        id serial,
        username text,
        password text,
        contraint pk_users primary key (id));

create unique index idx_users_u on users(username);
```

Gli unique index possono essere definiti su tuple e coinvolgere condizioni logiche.

Stored Procedures

```
/** @brief select new_user (username, password) as result; */

create or replace function new_user (text, text) returns integer as
$$
declare
  var_username alias for $1;
  var_password alias for $2;
  return_id integer;
begin

  begin
    insert into users values (default, var_username, var_password);
    select into return_id currval ('users_id_seq');
  exception
    when unique_violation then
      return_id := -1;
  end

  return return_id;
end
$$ language plpgsql;
```

Stored Procedures

In questo caso il programma chiamerà solo:

```
cursor = db.cursor ()
cursor.execute (“select new_user (‘%s’,’%s’);” % (username,password))
return cursor.fetchone()[0]
```

Come si può vedere tutta la business logic rimane dentro al database e può essere usata da tutte le applicazioni. Il grande vantaggio è che in caso di bug basta correggerla una volta e non in tutte le applicazioni.

Planner e performance delle query

Capire perché una query sia lenta è una operazione complessa, soprattutto quando la query viene eseguita su delle join o su delle viste, che in postgresql sono sempre dematerializzate.

Quando viene eseguita una query, subito dopo il parsing del comando viene richiamato il planner che studia una strategia su come effettuare la ricerca.

Planner e performance delle query

Il modo per vedere cosa sceglie di fare il planner è possibile usare il comando EXPLAIN

Planner e performance delle query

```
create table users (id serial, username text, password text);
insert into users values (default, 'pippo', 'ppippo');
insert into users values (default, 'pluto', 'ppluto');

explain select * from users where username = ‘pluto’;
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on users  (cost=0.00..20.00 rows=4 width=68)
   Filter: (username = 'pluto'::text)
(2 rows)

create index idx_users_u on users (username);

explain select * from users where username = 'pluto';
                                QUERY PLAN
---------------------------------------------------------------------------
 Index Scan using idx_users_u on users  (cost=0.00..4.10 rows=1 width=22)
   Index Cond: (username = 'pluto'::text)
(2 rows)
```

Planner e performance delle query

La creazione degli indici non è sempre semplice.

Infatti, per capire quali indici siano utili all'esecuzione delle query è necessario studiare l'output del planner per capire a quali colonne associarli.

Un uso eccessivo di indici porta a spreco di ram, disco e cpu.

Estensioni di PostgreSQL

Uno dei maggiori punti i forza di PostgreSQL è la sua estensibilità.

Molti plugin sono disponibili nel pacchetto postgresql-contrib, che vengono mantenuti dagli sviluppatori e che vengono inclusi nel core quando sono ritenuti pronti.

XML

In PostgreSQL 8.3 è stato inserito il tipo di dato XML. Nei campi xml è possibile salvare *document* o *content*, come definito dallo standard XML.

```
XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</title><chapter>...</chapter><book>')

XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')
```

Con XMLSERIALIZE è possibile convertire le informazioni XML in altri tipi di dati.

XPATH

XML da solo serve a poco, perché è solo un contenitore di dati.

PostgreSQL 8.3 mette a disposizione la funzioni di ricerca di XPATH per XML.

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>')
```

Sono inoltre disponibili funzioni che permettono di mappare il database in XML per poter operare con XPATH non solo sui dati xml.

Ltree

Non sempre i dati si prestano bene alla rappresentazione tabellare.

Ltree permette di raggruppare le righe di una tabella in struttura ad albero, per accedere alle informazioni gerarchiche senza dover implementare la logica del tour di visita dell'albero.

Full text search

Lo standard SQL prevede di effettuare il text matching con l'istruzione LIKE. Purtroppo, anche se si usano algoritmi come quello di Knuth-Morris-Pratt la complessità temporale è lineare.

Nel caso in cui la ricerca sia effettuata su parole è possibile velocizzare la ricerca utilizzando tsearch, che divide il testo in parole e le indicizza con strutture ad albero ottenendo una complessità logaritmica.

PostGIS

PostGIS permette di salvare nelle tabelle informazioni spaziali e di effettuare ricerche.

Poiché sono disponibili anche indici per questi tipi di dato, è possibile accedere efficientemente dai dati salvati. Per questo motivo PostgreSQL e PostGIS vengono spesso utilizzati per la realizzazione di sistemi informativi territoriali.

Cube

L'estensione cubo permette di gestire dati tridimensionali in uno spazio multidimensionale.

Le operazioni tra oggetti cubo prevedono la determinazione delle condizioni di sovrapposizione, inclusione e distanza.

Un'altra estensione per tipi di dato cubici è data da EFEU, che spesso viene utilizzata per ottenere funzionalità OLAP in PostgreSQL.

PgCrypto

L'estensione PgCrypto fornisce un insieme di funzioni per crittare i dati salvati nel database:

- AES
- Blowfish
- SHA1 , SHA256/384/512
- DES/3DES/CAST5
- PGP crittografia simmetrica e pubblica

DbLink

Nel caso sistemi complessi, capita che i dati non siano contenuti in un unico database.

Una soluzione fornita da PostgreSQL è DbLink, in cui è possibile collegare due database ed eseguire query su tabelle nel database remoto come se fosse locale.

Large Objects

In PostgreSQL < 7.1 era impossibile inserire righe di dimensioni maggiori di una pagina, il cui valore predefinito è 8192 byte. E' stato quindi necessario creare un meccanismo basato su trigger per salvare tali dati esternamente al db.

In PostgreSQL 7.1 è stato inserito un meccanismo *The Oversized Attribute Storage Technique* (TOAST) che divide in più righe fisiche ed opzionalmente può comprimere.