Basi di Dati
Architettura dei DBMS: gestione della concorrenza

Contenuti - Roadmap

Lab (progettazione)	Corso di Teoria	Lab (SQL)
• Metodologie e modello Entità Associazioni • Progettazione concettuale e logica	• Modello Relazionale • Algebra relazionale • Ottimizzazione logica • Calcolo relazionale • La normalizzazione • Metodi di accesso e indici • **Gestione della concorrenza** • Gestione del ripristino	• Linguaggio SQL

Outline

- **Introduzione alla gestione della concorrenza**
- Inconsistenze derivanti dall'interfogliamento
- Serializzabilità
- Evitare la non serializzabilità
  - Meccanismo dei lock
  - 2PL

Esempio di transazioni

Esempio: due operazioni bancarie che aggiornano conti A e B.

In T_1 si trasferisce una somma di 50 dal conto A al conto B.

In T_2 si trasferisce il 10% del saldo del conto A al conto B.

T_1
read(A)
A := A – 50
write(A)
read(B)
B := B + 50
write(B)

T_2
read(A)
temp := 0.1 · A
A := A – temp
write(A)
read(B)
B := B + temp
write(B)

Proprietà di T_1 e T_2

Immaginiamo che tutti i vincoli della base di dati siano rispettati.

Si vede immediatamente che né T_1 né T_2 modificano la somma A+B (altrimenti si creerebbe o distruggerebbe denaro).
Per le due transazioni, cioè, A+B=3000 è un invariante.

Supponiamo ora che le due transazioni entrino entrambe nel sistema e vengano eseguite in successione e che all'inizio A contenga il valore 1000 e B il valore 2000.

Esecuzione di T_1T_2

1. Prima dell'inizio:
   - A = 1000, B = 2000 e A + B = 3000

2. La transazione T_1 toglie da A il valore 50 e lo aggiunge a B, quindi:
   - A = 950, B = 2050 e A + B = 3000

3. Infine, la transazione T_2 toglie il 10% dal nuovo saldo di A e lo aggiunge a B:
   - A = 855, B = 2145 e A + B = 3000

Esecuzione di T_2T_1

Consideriamo l'ordine inverso.

1. Prima dell'inizio:
   - A = 1000, B = 2000 e A + B = 3000

2. La transazione T_2 toglie il 10% da A e lo aggiunge a B
   - A = 900, B = 2100 e A + B = 3000

3. Infine, la transazione T_1 toglie da A il valore 50 e lo aggiunge a B, quindi alla fine avremo:
   - A = 850, B = 2150 e A + B = 3000

DBMS e isolamento

- Supponiamo che il DBMS riceva contemporaneamente le transazioni T_1 e T_2.
- Le transazioni T_1 e T_2 devono godere della proprietà di isolamento.
- Dal punto di vista del DBMS, l'esecuzione completa di T_1 (isolata) seguita dall'esecuzione completa di T_2 (sempre isolata) ha la proprietà di lasciare la base di dati in una situazione di consistenza.
- Se il DBMS riceve le transazioni contemporaneamente, la scelta di eseguire la sequenza T_1T_2 oppure T_2T_1 è **irrilevante**.

Sistema informativo e DBMS

- Se per il sistema informativo fosse importante l'ordine di esecuzione delle due transazioni, attenderebbe il termine della prima per sottoporre la seconda, oppure le includerebbe in una unica transazione (alcuni DBMS supportano transazioni annidate).

- Se però il sistema informativo sottopone in parallelo le due transazioni al DBMS, allora il DBMS può eseguirle nell'ordine che ritiene opportuno.

Esecuzione concorrente

- L'esecuzione in sequenza delle transazioni è **inefficiente**.
- Dal punto di vista del DBMS, una esecuzione in sequenza lascia **tempi morti** di elaborazione molto lunghi per via dei frequenti accessi alle periferiche (read e write).
- Tutti i DBMS sono stati sviluppati in modo da mandare avanti **concorrentemente** operazioni provenienti da transazioni diverse.
- L'esecuzione in parallelo di due transazioni richiede di **interfogliare** (interleave) le attività delle transazioni.
- Problema: l'interfogliamento può causare effetti indesiderati.

Outline

- Introduzione alla gestione della concorrenza
- **Inconsistenze derivanti dall'interfogliamento**
- Serializzabilità
- Evitare la non serializzabilità
  - Meccanismo dei lock
  - 2PL

Esempio

Supponiamo che il DBMS esegua il seguente interfogliamento

T_1
read(A)
A := A – 50
write(A)
read(B)
B := B + 50
write(B)

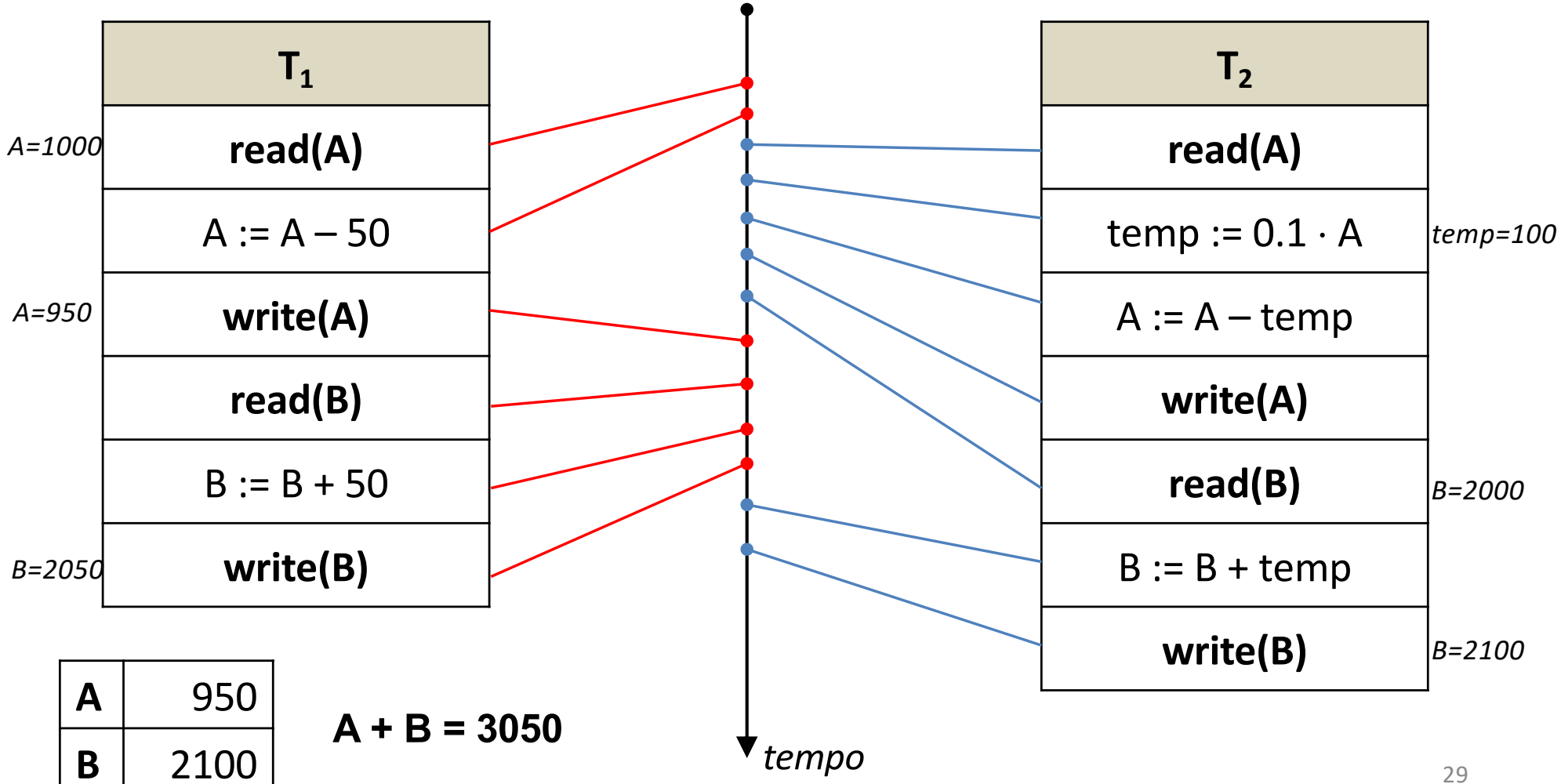
T_2
read(A)
temp := 0.1 · A
A := A – temp
write(A)
read(B)
B := B + temp
write(B)

A	1000
B	2000

A + B = 3000

Esempio

Supponiamo che il DBMS esegua il seguente interfogliamento



A	950
B	2100

A + B = 3050

Inconsistenza

C’è un’inconsistenza perché, dato che A+B=3050, abbiamo «creato denaro»: l’interfogliamento ha interferito con le due transazioni.

Non è detto che l’interfogliamento di due transazioni sia sempre nocivo: alcuni interfogliamenti possono lasciare la base di dati in uno stato consistente.

Esempio

Supponiamo che il DBMS esegua il seguente interfogliamento

T_1
read(A)
A := A – 50
write(A)
read(B)
B := B + 50
write(B)

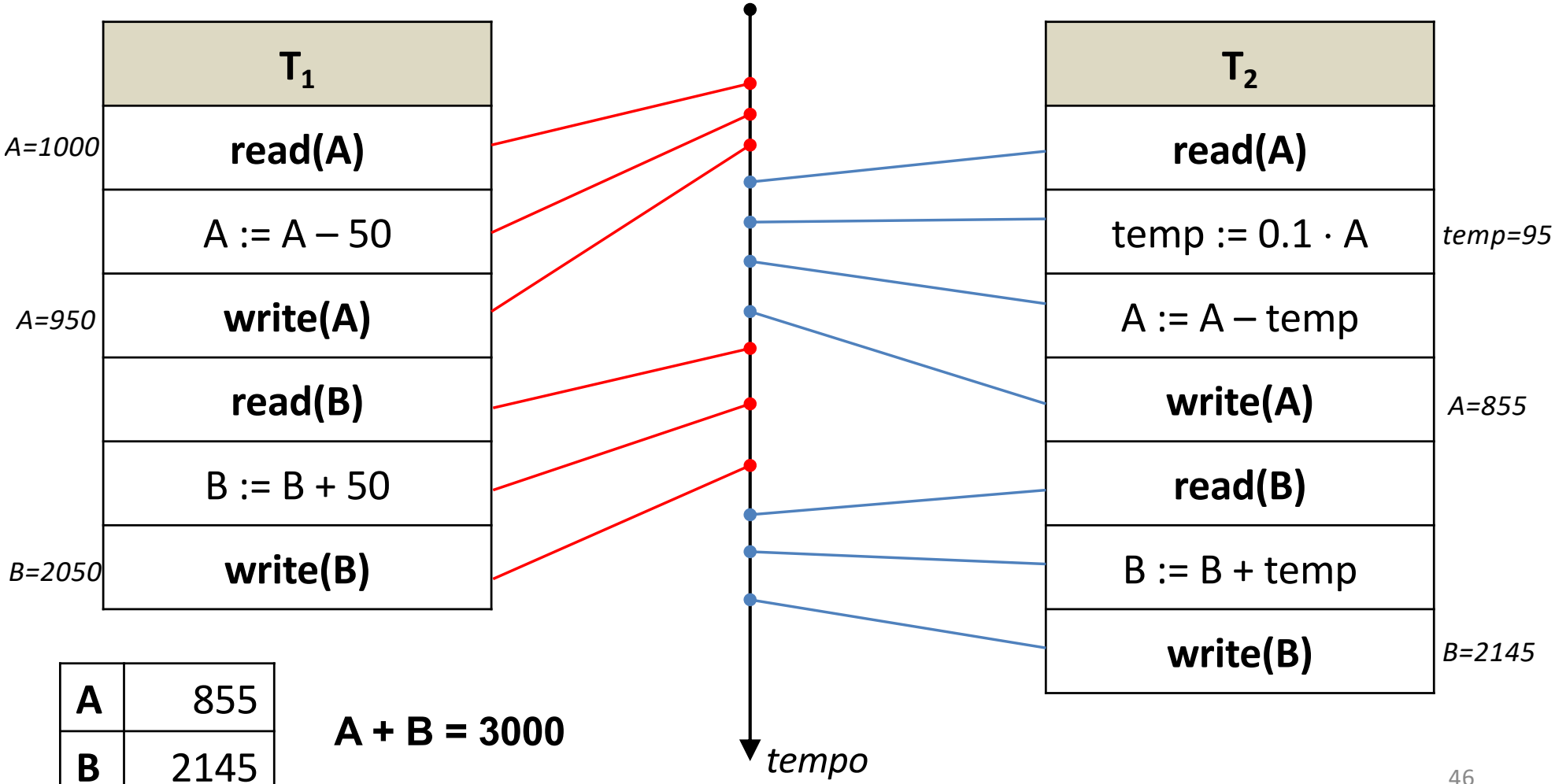
T_2
read(A)
temp := 0.1 · A
A := A – temp
write(A)
read(B)
B := B + temp
write(B)

A	1000
B	2000

A + B = 3000

Esempio

Supponiamo che il DBMS esegua il seguente interfogliamento

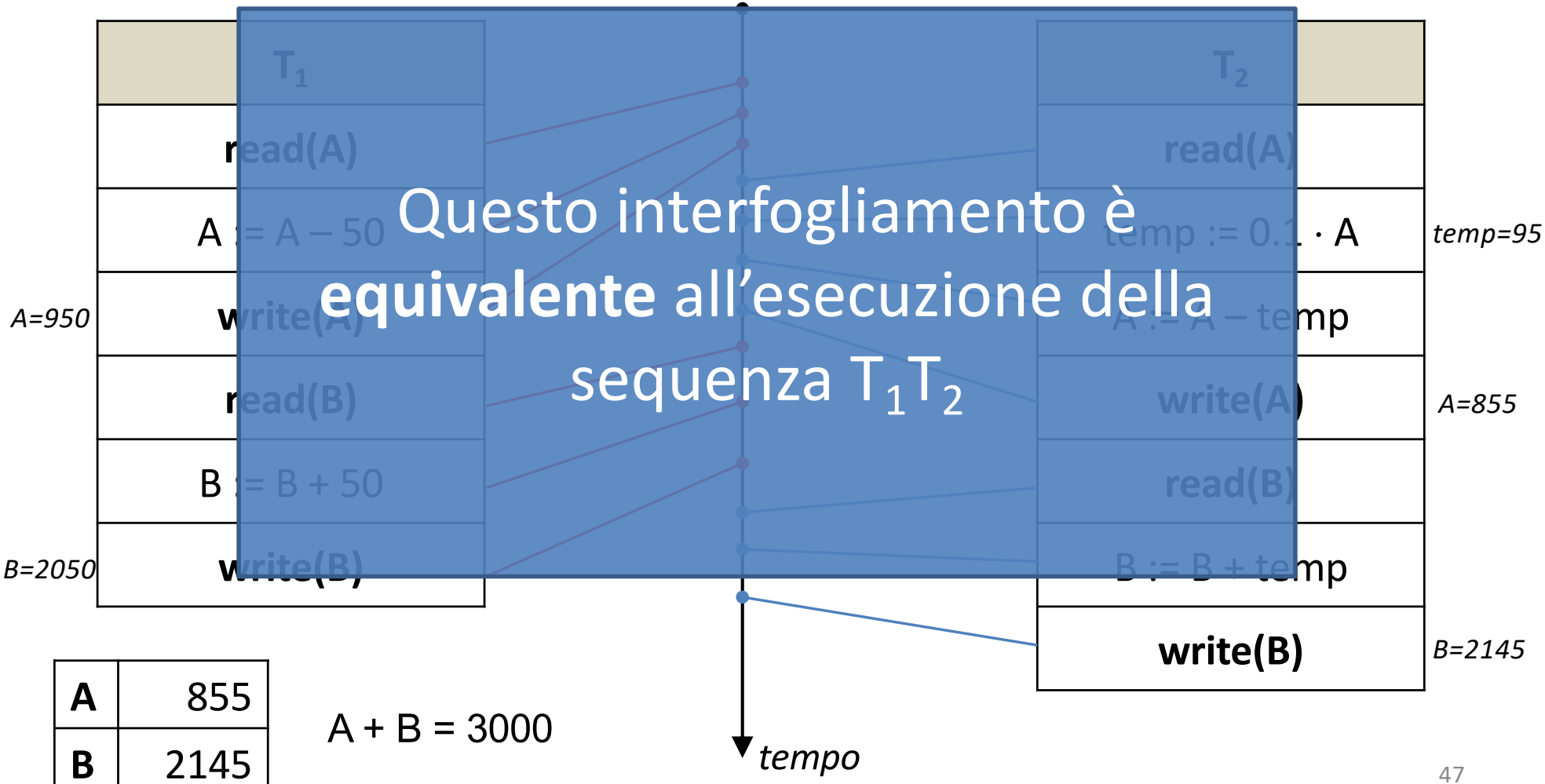


A	855
B	2145

A + B = 3000

Esempio

Supponiamo che il DBMS esegua il seguente interfogliamento



A	855
B	2145

A + B = 3000

Outline

- Introduzione alla gestione della concorrenza
- Inconsistenze derivanti dall'interfogliamento
- **Serializzabilità**
- Evitare la non serializzabilità
  - Meccanismo dei lock
  - 2PL

Storie

Uno specifico interfogliamento delle transazioni si chiama ***schedulazione*** o ***storia***.

Una **storia**, cioè, è la sequenza di azioni eseguite dal DBMS a fronte delle richieste da parte delle transazioni.

Storie: esempi

Consideriamo le transazioni T_1 e T_2

T_1
read(A)
A := A – 50
write(A)
read(B)
B := B + 50
write(B)

T_2
read(A)
temp := 0.1 · A
A := A – temp
write(A)
read(B)
B := B + temp
write(B)

Storie: esempi

Consideriamo, come fa un DBMS, solo le azioni compiute sulla base di dati e ignoriamo le modifiche alle variabili interne alle transazioni.

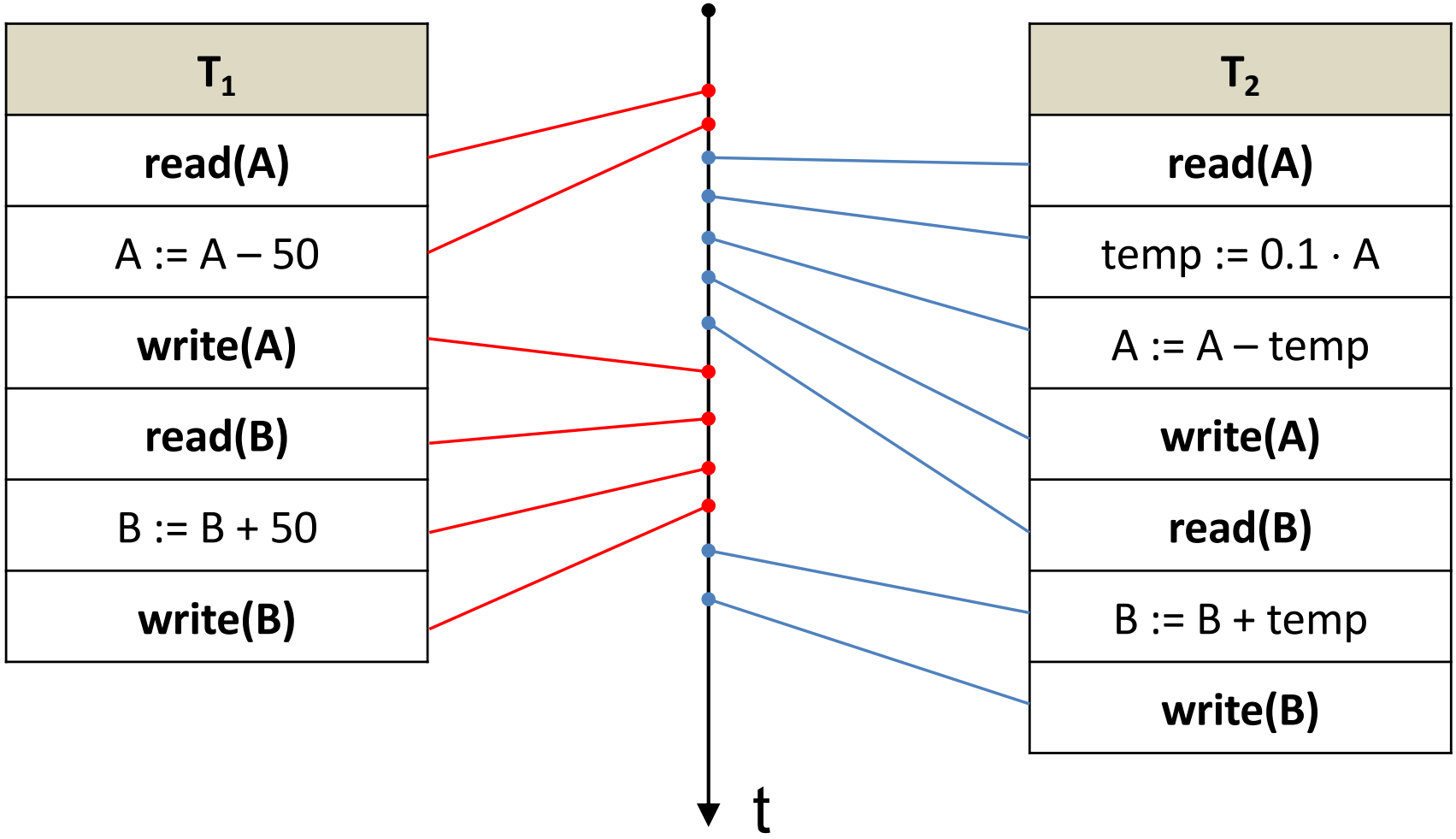
Descriveremo le storie T_1T_2 e T_2T_1 con questa notazione:

- T_1T_2: $r_1(A),w_1(A),r_1(B),w_1(B),r_2(A),w_2(A),r_2(B),w_2(B)$
- T_2T_1: $r_2(A),w_2(A),r_2(B),w_2(B),r_1(A),w_1(A),r_1(B),w_1(B)$

Consideriamo ora le storie delle transazioni interfogliate.

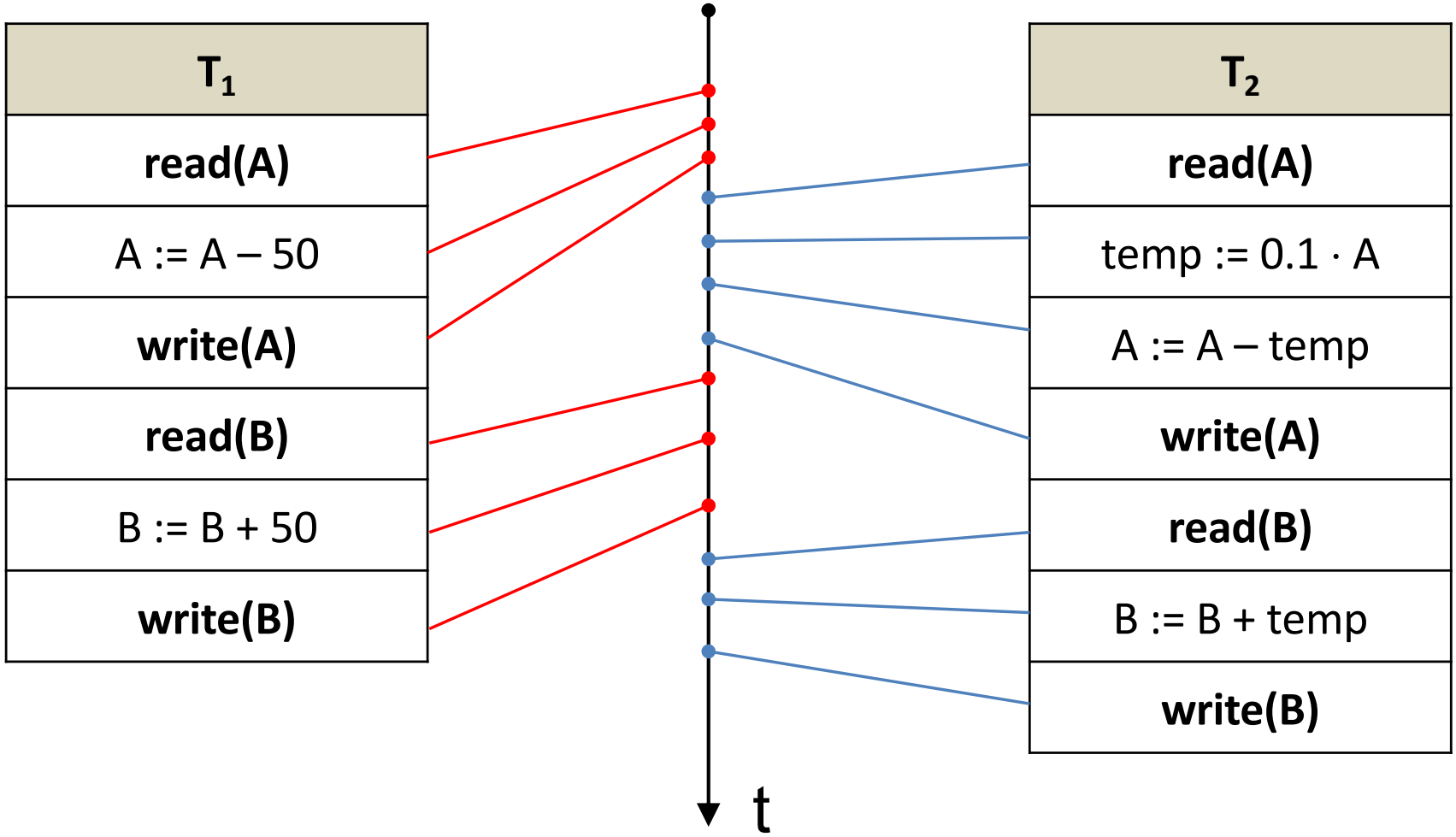
Storie: esempi

Storia S_1: $r_1(A), r_2(A), w_2(A), r_2(B), w_1(A), r_1(B), w_1(B), w_2(B)$



Storie: esempi

Storia S_2: $r_1(A), w_1(A), r_2(A), w_2(A), r_1(B), w_1(B), r_2(B), w_2(B)$



Criterio di serializzabilità

Quali sono le storie che non causano inconsistenze?

Dato che assumiamo che l'esecuzione seriale delle transazioni sia consistente, sono consistenti tutti gli interfogliamenti **equivalenti** alle storie seriali.

Il **criterio di serializzabilità** dice quindi che
una storia S è corretta se è equivalente a una qualsiasi storia seriale delle transazioni coinvolte da S.

N.B.: date **n** transazioni esistono **n!** storie seriali.

Outline

- Introduzione alla gestione della concorrenza
- Inconsistenze derivanti dall'interfogliamento
- Serializzabilità
- **Evitare la non serializzabilità**
  - **Meccanismo dei lock**
  - 2PL

Problema

- Cosa potrebbe fare un DBMS se, durante l'esecuzione di una storia, scoprisse che non è serializzabile?
- Dovrebbe fare abortire alcune delle transazioni che partecipano alla storia e ripeterle in seguito in modo da garantire la serializzabilità.
- Questo causerebbe un grande spreco di risorse, quindi nei DBMS commerciali si adotta un approccio diverso: si definisce un **protocollo** che garantisca la serializzabilità **a priori**, cioè un insieme di regole da seguire che prevengono che venga generata una storia non serializzabile.
- Presenteremo il **metodo basato sui lock** perché è il più diffuso.

Lock

Le transazioni hanno a disposizione i seguenti comandi per richiedere l'autorizzazione a compiere azioni sull'oggetto X:

- LS(X): **lock shared** sull'oggetto X da richiedere prima di una lettura
- LX(X): **lock exclusive** sull'oggetto X da richiedere prima di una scrittura
- UN(X): **unlock** sull'oggetto X

Possiamo assimilare l'oggetto X a un record.

Di solito LS, LX e UN vengono invocati implicitamente dal gestore della concorrenza e non dalla transazione

Lock shared

Quando una transazione T_i vuole eseguire un'azione di **lettura** $r_i(X)$, prima di effettuare la lettura deve avere acquisito almeno il lock shared sull'oggetto X, ovvero il permesso per leggere l'oggetto X.

Il **lock shared** (condiviso) si chiama così perché transazioni diverse possono acquisire un lock condiviso sul medesimo oggetto.

Il DBMS può quindi concedere il lock shared sullo stesso oggetto contemporaneamente a più transazioni.

Esempio: la transazione T_j può chiedere e ottenere il LS sull'oggetto X già ottenuto dalla transazione T_i.

Lock exclusive

Il **lock exclusive** richiede un accesso esclusivo all'oggetto X da parte di una transazione T_i.

La richiesta di lock exclusive è di norma fatta da una transazione per **modificare** un oggetto X: questa richiesta deve sempre precedere una $w_i(X)$.

Quando T_i ha acquisito l'autorizzazione esclusiva a scrivere l'oggetto X, nessuna altra transazione può acquisire lock (né LS né LX) sullo stesso oggetto.

La transazione T_i, quindi, diventa «proprietaria» esclusiva di X.

Gestione del lock

- Quando il DBMS non può concedere il lock, la transazione richiedente va in **wait** (stato di attesa).

- Quando una transazione non ha più bisogno di leggere o scrivere l'oggetto X, può rilasciare il lock (shared o exclusive) attraverso **l'unlock**.

- Il DBMS tiene traccia dei lock concessi con una tabella dei lock.

Gestione del lock

Azioni	Oggetto X	Oggetto Y
(stato iniziale)	libero	libero
T_1: $LS_1(X)$	LS a T_1	libero
T_1: $LX_1(Y)$	LS a T_1	LX a T_1
T_2: $LX_2(X)$	LS a T_1 WAIT(LX) a T2	LX a T_1
T_3: $LS_3(X)$	LS a T_1 e T_3 WAIT(LX) a T_2	LX a T_1
T_3: $LS_3(Y)$	LS a T1 e T_3 WAIT(LX) a T_2	LX a T_1 WAIT(LS) a T_3
T_4: $LS_4(X)$	LS a T_1, T_3 e T_4 WAIT(LX) a T2	LX a T_1 WAIT(LS) a T_3
T_4: $LX_4(Y)$	LS a T_1, T_3 e T_4 WAIT(LX) a T_2	LX a T_1 WAIT(LS) a T_3 WAIT(LX) a T_4

tempo

Gestione dei lock

T_1 termina o rilascia gli oggetti (unlock).

Assumiamo che le code delle transazioni in attesa vengano gestite con politiche FIFO.

tempo

Azioni	**X**	**Y**
(situazione precedente)	LS a **T_1** , T_3 e T_4 WAIT(LX) a T_2	LX a **T_1** WAIT(LS) a T_3 WAIT(LX) a T_4
T_1: UN_1(X,Y)	LS a ~~T_1~~, T_3 e T_4 WAIT(LX) a T_2	~~LX a T_1~~ WAIT(LS) a T_3 WAIT(LX) a T_4
…	LS a T_3 e T_4 WAIT(LX) a T_2	LS a T_3 WAIT(LX) a T_4
T_3: UN_3(X,Y)	LS a T_4 WAIT(LX) a T_2	LX a T_4
T_4: UN(X,Y)	LX a T_2	libero

Gestione del lock

Transazioni diverse possono lanciare attività parallele su medesimi oggetti.

Come viene amministrata dal DBMS la compresenza di più lock?

- Il DBMS si avvale della **tabella di compatibilità**

possesso \ richiesta	LS	LX
LS	concede	nega
LX	nega	nega

Ad esempio, se T_i possiede il lock sull'oggetto X e T_j ne fa richiesta, il DBMS concede o nega il lock a seconda dei casi elencati nella tabella di compatibilità.

ATTENZIONE: È possibile **aggiornare** il lock da LS a LX se una stessa transazione ne fa richiesta ed è l'unica che possiede il LS.

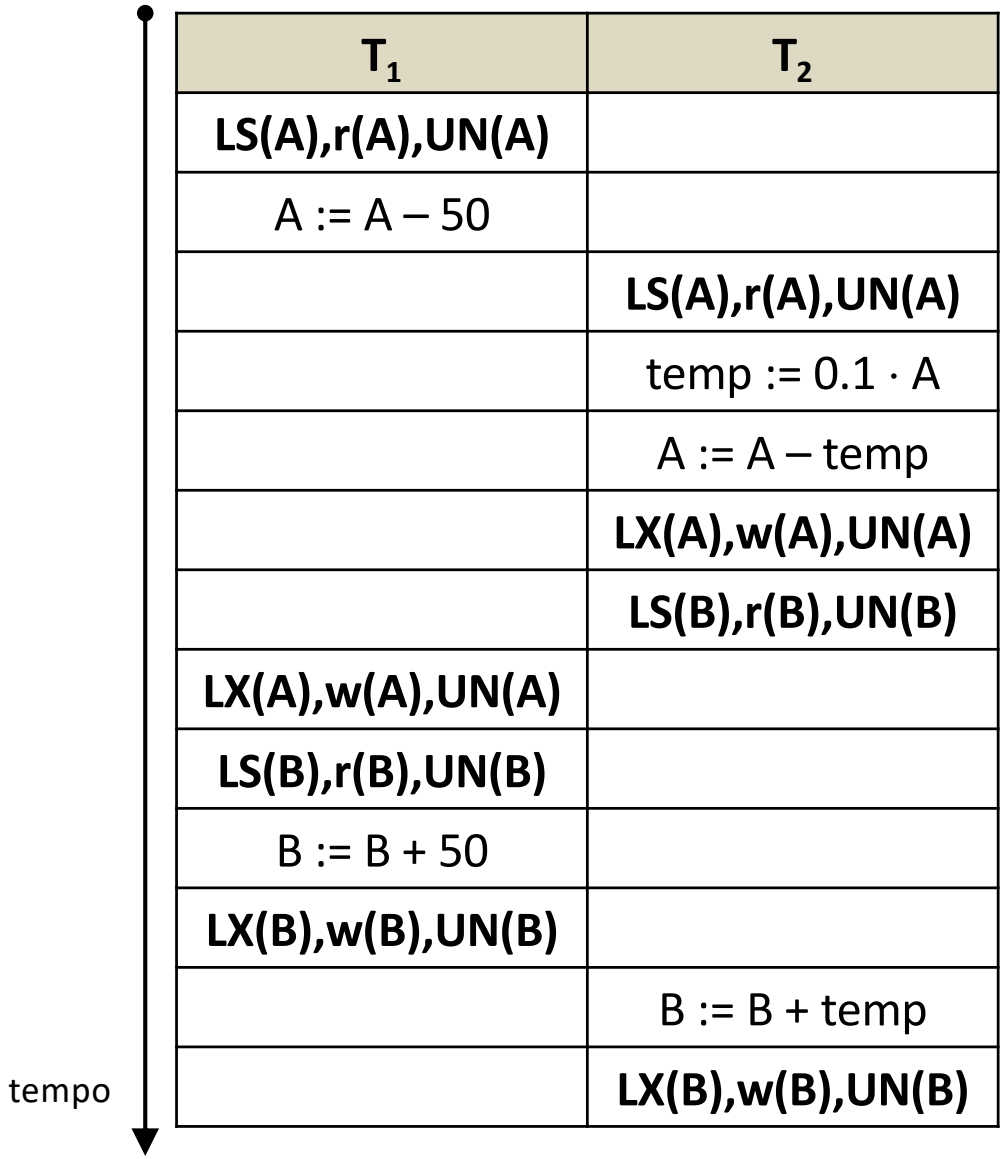
Lock e concorrenza

Il sistema dei lock è stato introdotto per gestire la concorrenza.

Ci chiediamo ora se questo sistema sia sufficiente per costruire delle storie serializzabili.

La risposta è **no** e si può vedere sull'esempio S_1.

Esempio: storia S_1



T_1	T_2
LS(A),r(A),UN(A)	
A := A – 50	
	LS(A),r(A),UN(A)
	temp := 0.1 · A
	A := A – temp
	LX(A),w(A),UN(A)
	LS(B),r(B),UN(B)
LX(A),w(A),UN(A)	
LS(B),r(B),UN(B)	
B := B + 50	
LX(B),w(B),UN(B)	
	B := B + temp
	LX(B),w(B),UN(B)

La storia S_1 è inconsistente nonostante usi correttamente i lock (ogni richiesta di lock su una risorsa viene soddisfatta perché la risorsa è già stata liberata).

Per avere benefici dal meccanismo del lock occorre fare un passo ulteriore…

Outline

- Introduzione alla gestione della concorrenza
- Inconsistenze derivanti dall'interfogliamento
- Serializzabilità
- Evitare la non serializzabilità
  - Meccanismo dei lock
  - **2PL**

Politica del locking a due fasi

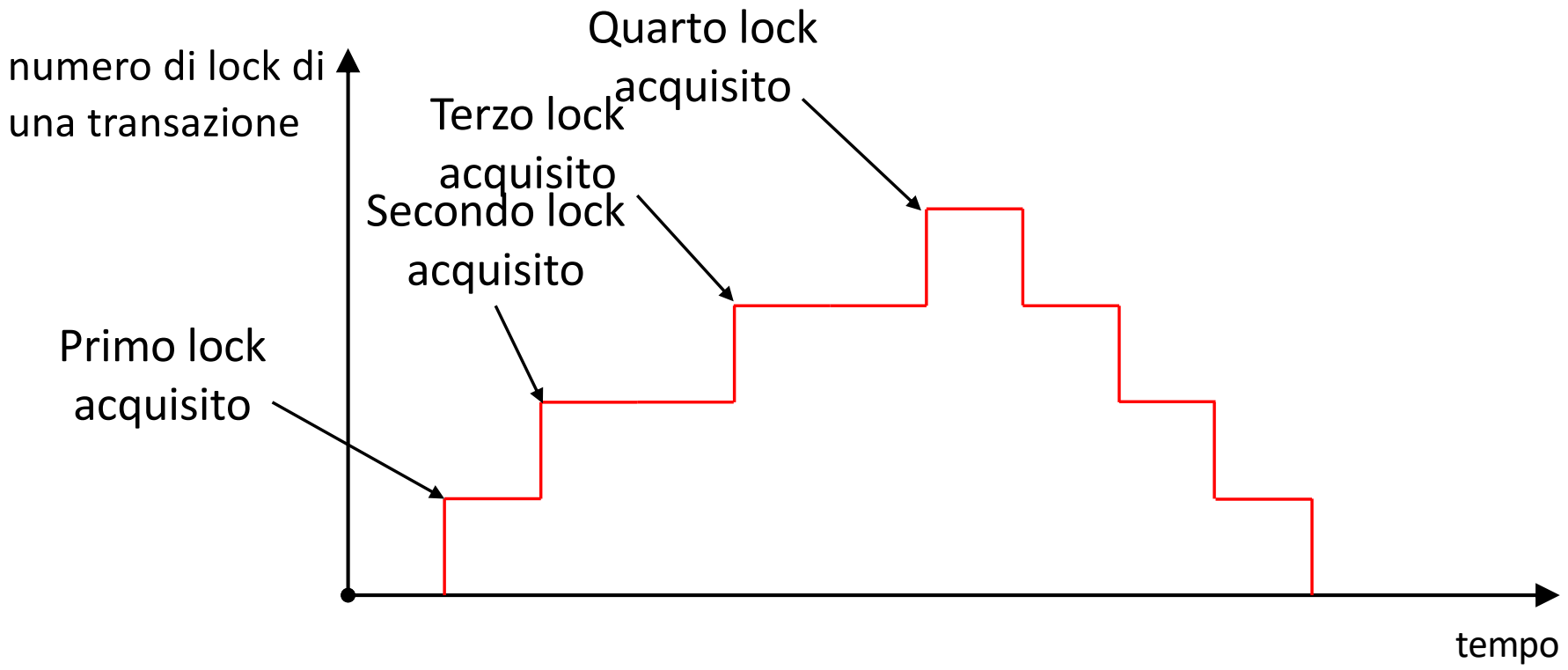
Se si impone alle transazioni un vincolo nell'utilizzo degli **unlock**, allora la politica dei lock diventa la soluzione nella gestione della concorrenza.

Il vincolo è stabilito dalla **politica di lock delle transazioni a due fasi** detta anche two-phase lock (**2PL**).

Nella politica di lock a due fasi si ha **una fase di acquisizione dei lock** seguita da una **fase di rilascio dei lock**.

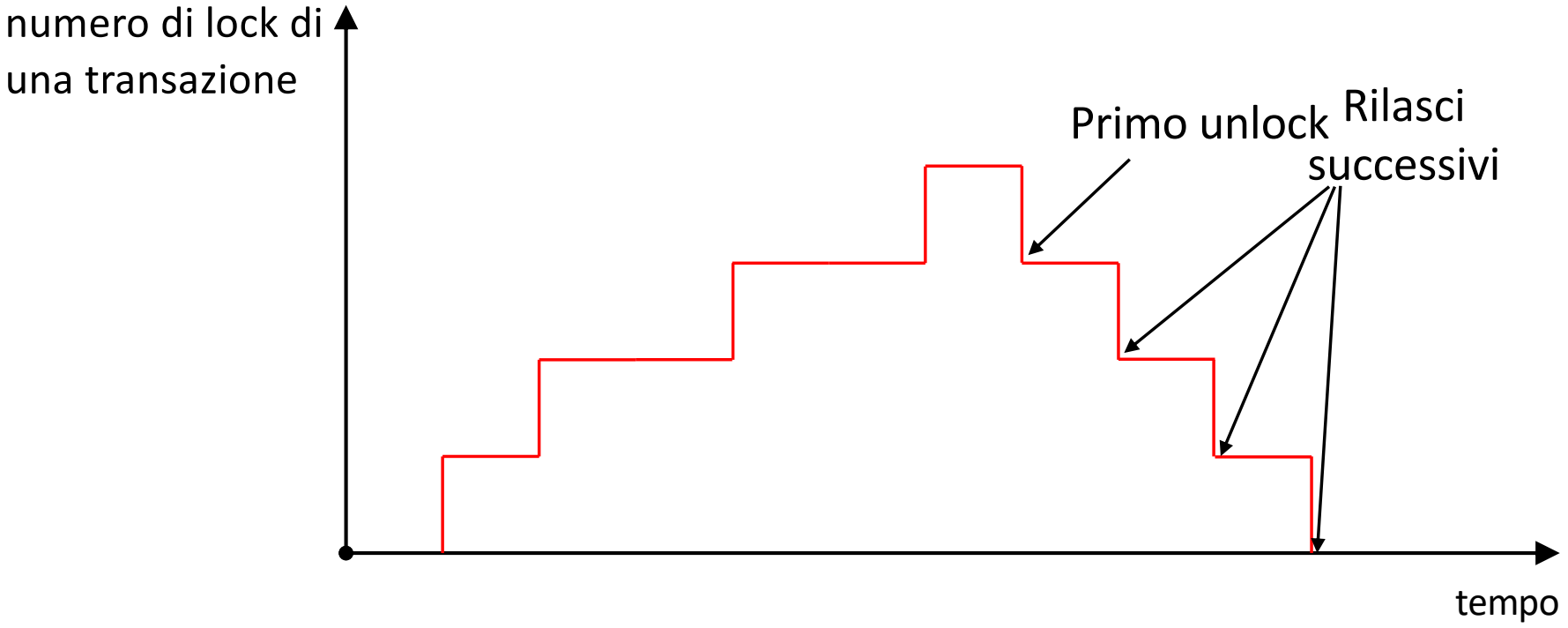
Lock a due fasi

Possiamo vedere il funzionamento in un grafico in cui sull'asse delle ascisse abbiamo il **tempo** e sull'asse delle ordinate abbiamo il **numero di lock** concessi dal DBMS a una transazione T_i



Lock a due fasi

A un certo punto la transazione incomincia a **rilasciare** dei lock: **quando una transazione inizia la fase di rilascio, da quel momento in poi non può più acquisirne**



Lock a due fasi

Si può dimostrare che la politica del lock a due fasi garantisce la serializzabilità delle storie che genera.

Esercizio 1

- La storia

  $S = r_1(x), r_1(y), r_2(x), w_3(z), w_1(y), r_2(y), w_2(x)$

  è compatibile con 2PL?

Esercizio 1

- La storia

  S = $r_1(x)$, $r_1(y)$, $r_2(x)$, $w_3(z)$, $w_1(y)$, $r_2(y)$, $w_2(x)$

  è compatibile con 2PL?

- Sì, infatti una possibile realizzazione, con i relativi lock e unlock, è la seguente:

  S = **$LS_1(x)$**, $r_1(x)$, **$LS_1(y)$**, $r_1(y)$, **$LS_2(x)$**, $r_2(x)$, **$LX_3(z)$**, $w_3(z)$, **$UN_3(z)$**, **$LX_1(y)$**, $w_1(y)$, **$UN_1(x)$**, **$UN_1(y)$**, **$LS_2(y)$**, $r_2(y)$, **$LX_2(x)$**, $w_2(x)$, **$UN_2(y)$**, **$UN_2(x)$**

- È 2PL perché:
  - è coerente con le operazioni (*lettura* → *LS o LX, scrittura* → *LX*)
  - nessuna transazione acquisisce nuovi lock dopo averne rilasciati.

Esercizio 2

- La storia

  S_2: $r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B), r_1(B), w_1(B)$

  è compatibile con 2PL?

Esercizio 2

- La storia

  S_2: $r_1(A), w_1(A), r_2(A), w_2(A), r_2(B), w_2(B), r_1(B), w_1(B)$

  è compatibile con 2PL?

- No:
  1. T_1 deve acquisire un lock LX su A per potere eseguire $r_1(A), w_1(A)$
  2. poi T_1 deve rilasciarlo per permettere a T_2 di acquisire il lock LX su A per eseguire $r_2(A)$ $w_2(A)$ e non può riacquisirne
  3. ma per eseguire $r_1(B)$ $w_1(B)$ T_1 deve acquisire il lock LX su B
  4. T_1 non può però acquisire il lock LX su B all'inizio della storia perché T_2 deve possedere il lock LX su B per eseguire $r_2(B), w_2(B)$.
- Quindi non c'è modo di aggiungere lock che siano 2PL.

Osservazione

Il protocollo del lock a due fasi è piuttosto rigido, infatti esistono storie serializzabili che non sono possibili con il lock a due fasi.

- Ad es. la storia S_2 è equivalente alla storia seriale T_1T_2 ma come abbiamo visto non è 2PL

  S_2: $r_1(A),w_1(A),r_2(A),w_2(A),r_2(B),w_2(B),r_1(B),w_1(B)$

- Il lock a due fasi permette interfogliamenti consistenti nelle letture, ma i lock exclusive sono bloccanti.
- Nello standard SQL è possibile abbassare il livello di isolamento tra le transazioni: si accettano possibili anomalie (per es. letture sporche) in cambio di maggiore parallelismo.

Granularità del lock

Abbiamo parlato in generale di lock di oggetti X, ma i DBMS possono avere diverse scelte:

- X può essere un record (soluzione più diffusa)
- X può essere un attributo (lock complessi da gestire, ma aumentano il parallelismo)
- X può essere una pagina (usati nei sistemi distribuiti: i lock diminuiscono il parallelismo, ma sono molto più facili da gestire)
- X può essere un intero file (lock usati di solito nella gestione degli indici)

Limiti dei lock

- Le politiche di lock hanno un grande overhead, perché ogni operazione di lettura e scrittura deve essere preceduta da una richiesta di lock.
- Inoltre ci possono essere starvation e deadlock (per cui esistono soluzioni, per es. basate su timeout/priorità).
- Una soluzione alternativa al meccanismo dei lock è basata sui timestamp e sul loro ordinamento.