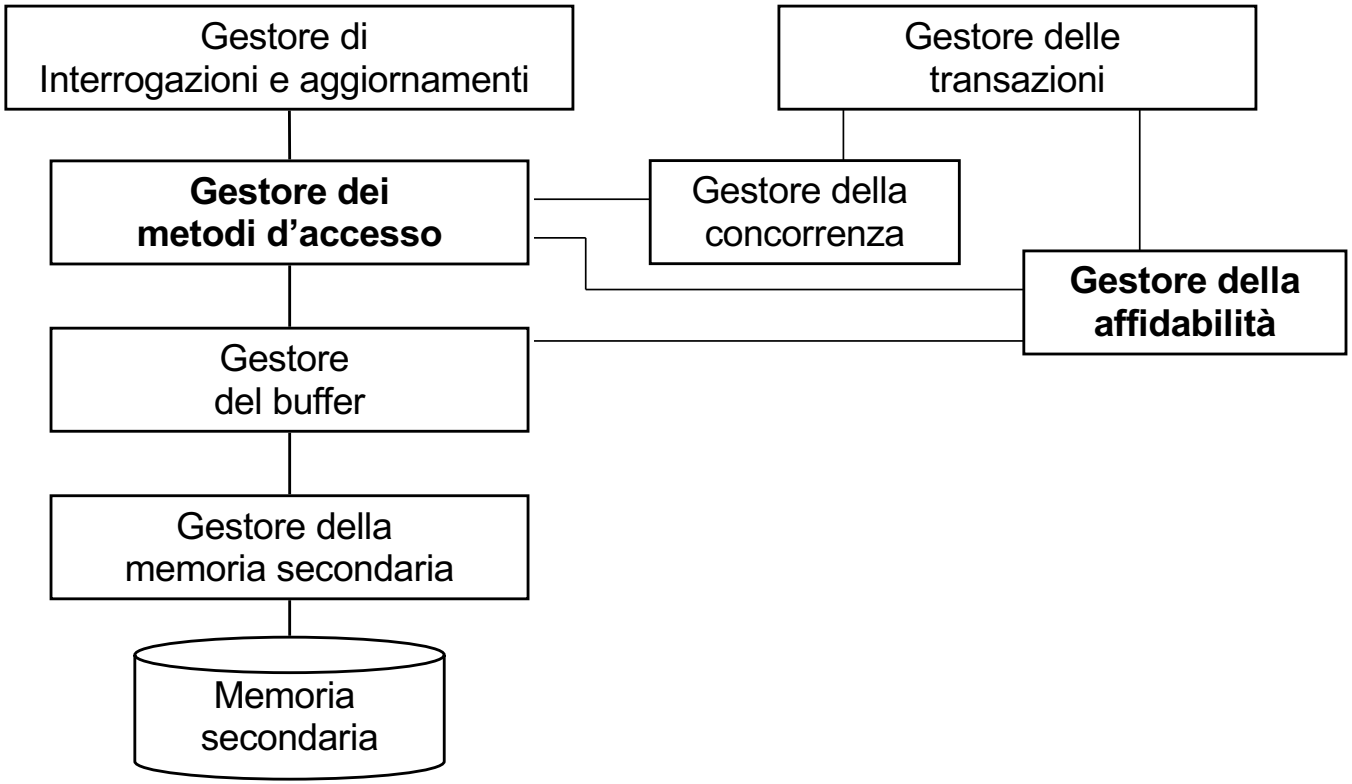
Gestore degli accessi e delle interrogazioni

Gestore delle transazioni



Controllo di concorrenza

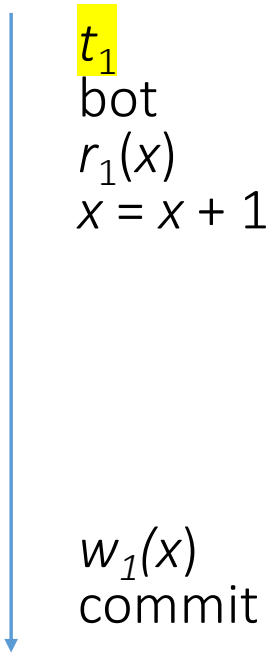
- Scopo:
  - garantire l'integrità/consistenza della base di dati in presenza di accessi concorrenti da parte di più utenti/applicazioni
  - necessità di sincronizzare le transazioni in esecuzione concorrente per garantire la proprietà di isolamento

Controllo di concorrenza

- Modello di riferimento
  - Operazioni di input/output su oggetti astratti
    - Es: T1: $r_1(x)$, $w_1(x)$,$w_1(y)$,$r_1(y)$
      T2: $w_2(x)$, $r_2(z)$,$w_2(z)$,$w_2(y)$
- Per garantire l'isolamento, una soluzione è eseguire le transazioni in modo seriale
  - T1, T2 -> $r_1(x)$, $w_1(x)$,$w_1(y)$,$r_1(y)$, $w_2(x)$, $r_2(z)$,$w_2(z)$,$w_2(y)$
  - Spreco di risorse, degrado della presazioni
- La concorrenza è fondamentale:
  - decine o centinaia di transazioni al secondo, non possono essere seriali
    - Esempi: banche, prenotazioni aeree
- Problema
  - Anomalie causate dall'esecuzione concorrente, che quindi va governata

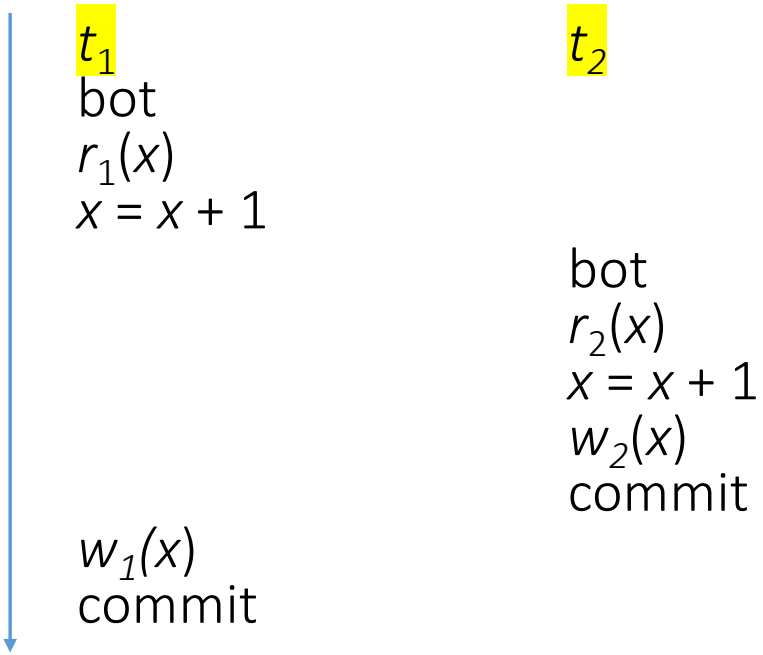
Anomalia: Perdita di aggiornamento

- Due transazioni identiche:
  - $t\ 1 : r_1(x),\ x = x + 1,\ w_1(x)$
  - $t\ 2 : r_2(x),\ x = x + 1,\ w_2(x)$
- Inizialmente x=2; dopo un'esecuzione seriale x=4



Anomalia: Perdita di aggiornamento

- Due transazioni identiche:
  - $t\ 1 : r_1(x),\ x = x + 1,\ w_1(x)$
  - $t\ 2 : r_2(x),\ x = x + 1,\ w_2(x)$
- Inizialmente x=2; dopo un'esecuzione seriale x=4
- Un'esecuzione concorrente:



- Un aggiornamento viene perso: x=3

Anomalia: Lettura sporca

t_1
bot
$r_1(x)$
$x = x + 1$
$w_1(x)$

abort

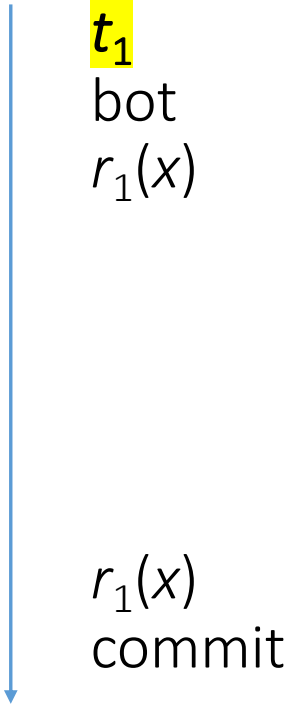
Anomalia: Lettura sporca

t_1	t_2
bot	
$r_1(x)$	
$x = x + 1$	
$w_1(x)$	
	bot
	$r_2(x)$
abort	
	$x = x + 1$
	$w_2(x)$
	commit

- Aspetto critico: t_2 ha letto uno stato intermedio ("sporco") e lo può comunicare all'esterno

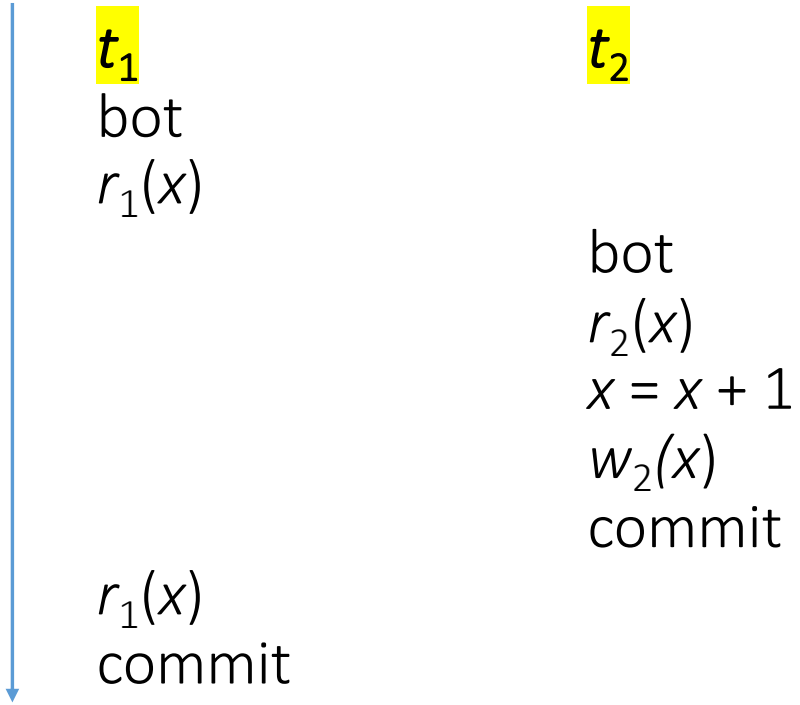
Anomalia: Letture inconsistenti

- t_1 legge due volte:



Anomalia: Letture inconsistenti

- t_1 legge due volte:



- t_1 legge due valori diversi per x!

Anomalia: Aggiornamento fantasma

- Assumere ci sia un vincolo $y + z = 1000$;

t_1
bot
$r_1(y)$

$r_1(z)$
$s = y + z$
commit

Anomalia: Aggiornamento fantasma

- Assumere ci sia un vincolo $y + z = 1000$;

t_1	t_2
bot	
$r_1(y)$	
	bot
	$r_2(y)$
	$y = y - 100$
	$r_2(z)$
	$z = z + 100$
	$w_2(y)$
	$w_2(z)$
	commit
$r_1(z)$	
$s = y + z$	
commit	

- $s = 1100$: il vincolo sembra non soddisfatto, t_1 vede un aggiornamento non coerente

Inserimento **fantasma**

t_1	t_2
`bot`	
"legge gli stipendi degli impiegati del dip A e calcola la media"	
	`bot`
	"inserisce un impiegato in A"
	`commit`
"legge gli stipendi degli impiegati del dip A e calcola la media"	
`commit`	

Esercizio: Ricerca di anomalie

- Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):

- S1: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1\ c_2$

- S2: $r_1(x)\ w_1(x)\ r_2(y)\ w_2(y)\ a_1\ c_2$

- S3: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(z)\ a_1\ c_2$

- S4: $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$

- S5: $r_1(x)\ r_2(x)\ w_2(x)\ r_1(y)\ c_1\ c_2$

- S6: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_1\ c_2$

Esercizio: Ricerca di anomalie

- **Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):**
- S1: $r_1(x)$ $w_1(x)$ $r_2(x)$ $w_2(y)$ a_1 c_2 /* lettura sporca: T2 legge uno stato di x che non dovrebbe esserci dato l'abort di T1
- S2: $r_1(x)$ $w_1(x)$ $r_2(y)$ $w_2(y)$ a_1 c_2
- S3: $r_1(x)$ $r_2(x)$ $r_2(y)$ $w_2(y)$ $r_1(z)$ a_1 c_2
- S4: $r_1(x)$ $r_2(x)$ $w_2(x)$ $w_1(x)$ c_1 c_2
- S5: $r_1(x)$ $r_2(x)$ $w_2(x)$ $r_1(y)$ c_1 c_2
- S6: $r_1(x)$ $w_1(x)$ $r_2(x)$ $w_2(x)$ c_1 c_2

Esercizio: Ricerca di anomalie

- Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):

- S1: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1\ c_2$

- S2: $r_1(x)\ w_1(x)\ r_2(y)\ w_2(y)\ a_1\ c_2$ /* nessuna anomalia

- S3: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(z)\ a_1\ c_2$

- S4: $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$

- S5: $r_1(x)\ r_2(x)\ w_2(x)\ r_1(y)\ c_1\ c_2$

- S6: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_1\ c_2$

Esercizio: Ricerca di anomalie

- Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):

- S1: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1\ c_2$

- S2: $r_1(x)\ w_1(x)\ r_2(y)\ w_2(y)\ a_1\ c_2$

- S3: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(z)\ a_1\ c_2$ /* nessuna anomalia

- S4: $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$

- S5: $r_1(x)\ r_2(x)\ w_2(x)\ r_1(y)\ c_1\ c_2$

- S6: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_1\ c_2$

Esercizio: Ricerca di anomalie

- Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):

- S1: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1\ c_2$

- S2: $r_1(x)\ w_1(x)\ r_2(y)\ w_2(y)\ a_1\ c_2$

- S3: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(z)\ a_1\ c_2$

- S4: $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$ /* Perdita di aggiornamento, le modifiche di T2 sono sovrascritte da T1

- S5: $r_1(x)\ r_2(x)\ w_2(x)\ r_1(y)\ c_1\ c_2$

- S6: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_1\ c_2$

Esercizio: Ricerca di anomalie

- Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):

- S1: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1\ c_2$

- S2: $r_1(x)\ w_1(x)\ r_2(y)\ w_2(y)\ a_1\ c_2$

- S3: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(z)\ a_1\ c_2$

- S4: $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$

- S5: $r_1(x)\ r_2(x)\ w_2(x)\ r_1(y)\ c_1\ c_2$ /* nessuna anomalia

- S6: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_1\ c_2$

Esercizio: Ricerca di anomalie

- Indicare se ci sono anomalia in queste sequenze (c_i, a_i, indicano commit e abort della transazione T_i):

- S1: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(y)\ a_1\ c_2$

- S2: $r_1(x)\ w_1(x)\ r_2(y)\ w_2(y)\ a_1\ c_2$

- S3: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(z)\ a_1\ c_2$

- S4: $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$

- S5: $r_1(x)\ r_2(x)\ w_2(x)\ r_1(y)\ c_1\ c_2$

- S6: $r_1(x)\ w_1(x)\ r_2(x)\ w_2(x)\ c_1\ c_2$ /* nessuna anomalia

Controllo di concorrenza: Scheduler

- Dato un insieme di transazioni da eseguire (T_0,T_1,T_2,T_3,…,T_n), il gestore della concorrenza (scheduler) deve decidere la sequenza di ordine (schedule) con cui eseguire le operazioni delle transazioni in modo concorrente
- Per non avere anomalie: *Schedule seriale,* le transazioni sono separate, ed eseguite una alla volta

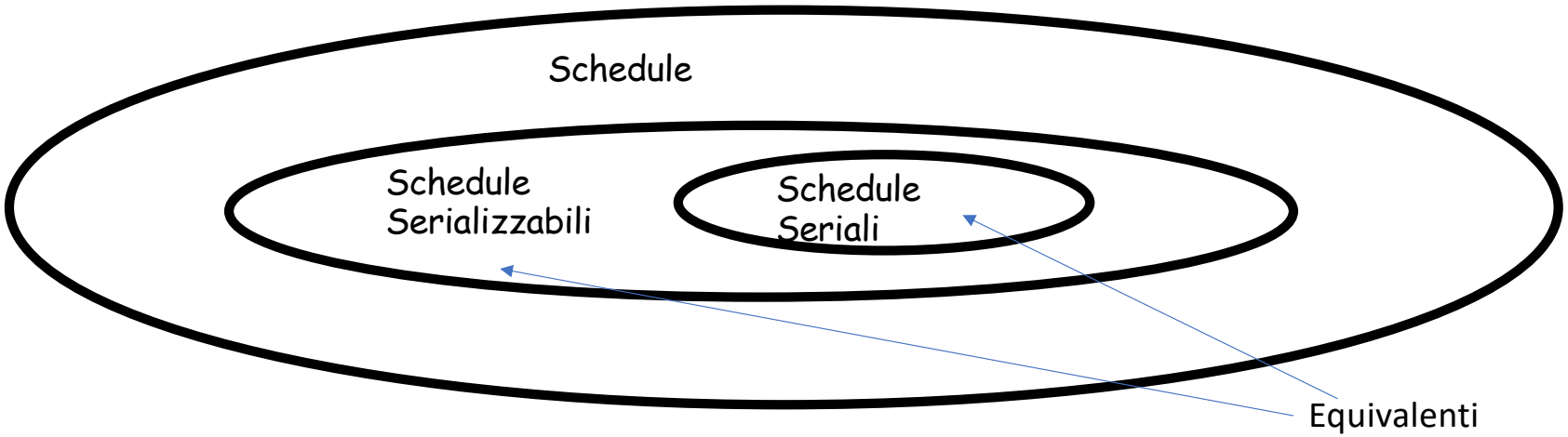
$$S : r_0(x)\ r_0(y)\ w_0(x)\ r_1(y)\ r_1(x)\ w_1(y)\ r_2(x)\ r_2(y)\ r_2(z)\ w_2(z)\ldots\ldots\ldots$$

non efficiente!

- Lo scheduler sceglie un schedule con transazioni sovrapposte, MA con lo stesso risultato di uno schedule seriale: schedule serializzabile
  - È necessaria una definizione di equivalenza tra schedule

Metodi teorici: Idea base

- Individuare classi di schedule serializzabili che siano sottoclassi degli schedule possibili e la cui proprietà di serializzabilità sia verificabile a costo basso



- Ipotesi semplificativa (commit-proiezione)

  Assumiamo che tutte le transazioni vadano in commit, ignoriamo quindi le transazioni che vanno in abort, rimuovendo tutte le loro azioni dallo schedule

rimuoveremo in futuro questa ipotesi, in quanto non accettabile in pratica

Metodologie di controllo

- Teorici:
  - View-serializzabilità (VSR)
  - Conflict-serializzabilità (CSR)
- Pratici:
  - Locking a due fasi (2PL)
  - Timestamp (TS Singolo, TS Multi)

View-Serializzabilità

- Definizioni preliminari:
  - in uno schedule S , $r_i(x)$ e $w_j(x)$ sono in relazione *legge-da* *(i.e.,* $r_i(x)$ legge da $w_j(x)$*)* se $w_j(x)$ precede $r_i(x)$ in S e non c'è $w_k(x)$ fra $r_i(x)$ e $w_j(x)$ in S
  - $w_i(x)$ in uno schedule S è *scrittura finale su x se* è l'ultima scrittura dell'oggetto x in S
- Schedule *view-equivalenti* ($S_i \approx_V S_j$): includono le stesse transazioni e relative operazioni e hanno
  - le stesse relazioni *legge-da*
  - le stesse scritture finali
- Uno schedule S è *view-serializzabile* se è view-equivalente ad un qualche schedule seriale
- L'insieme degli schedule view-serializzabili è indicato con VSR

View serializzabilità: esempi

- S_3 : $w_0(x)\ r_2(x)\ r_1(x)\ w_2(x)\ w_2(z)$

  S_4 : $w_0(x)\ r_1(x)\ r_2(x)\ w_2(x)\ w_2(z)$

  S_5 : $w_0(x)\ r_1(x)\ w_1(x)\ r_2(x)\ w_1(z)$

  S_6 : $w_0(x)\ r_1(x)\ w_1(x)\ w_1(z)\ r_2(x)$

  - S_3 è view-equivalente allo schedule seriale S_4 e quindi è view-serializzabile
  - S_5 è view-equivalente allo schedule seriale S_6 e quindi è view-serializzabile

Esercizio: controllo VSR

Indicare se i seguenti schedule sono VSR

1. r1(x) r2(y) w1(y) r2(x) w2(x)
2. r1(x) r2(y) w1(x) w1(y) r2(x) w2(x)
3. r1(x),r1(y) r2(y) w2(z) w1(z) w3(z) w3(x)

Esercizio: controllo VSR

Indicare se i seguenti schedule sono VSR

1. r1(x) r2(y) w1(y) r2(x) w2(x)

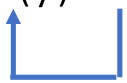
Per determinare se è VSR, verifico se è view-equivalente ad uno schedule seriale di T1 e T2, dove

T1: r1(x) w1(y)

T2: r2(y) r2(x) w2(x)

Possibili schedule seriali T1, T2 o T2,T1

T1,T2 s1: r1(x) w1(y) r2(y) r2(x) w2(x)



T2,T1 s2: r2(y) r2(x) w2(x) r1(x) w1(y)

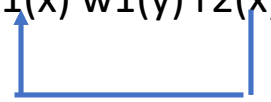
Non sono VSR

Hanno entrambi una differente relazione LEGGE – DA

Esercizio: controllo VSR

Indicare se i seguenti schedule sono VSR

2. r1(x) r2(y) w1(x) w1(y) r2(x) w2(x)



Per determinare se è VSR, verifico se è view-equivalente ad uno schedule seriale di T1 e T2, dove

T1: r1(x) w1(x) w1(y)

T2: r2(y) r2(x) w2(x)

Possibili schedule seriali T1, T2 o T2,T1

T1,T2 s1: r1(x) w1(x) w1(y) r2(y) r2(x) w2(x)

T2,T1 s2: r2(y) r2(x) w2(x) r1(x) w1(x) w1(y)

Non sono VSR

Hanno entrambi una differente relazione LEGGE – DA

Esercizio: controllo VSR

Indicare se i seguenti schedule sono VSR

3. r1(x) r1(y) r2(y) w2(z) w1(z) w3(z) w3(x)

Per determinare se è VSR, verifico se è view-equivalente ad uno schedule seriale di T1 e T2, dove

T1: r1(x) r1(y) w1(z)

T2: r2(y) w2(z)

T3: w3(z) w3(x)

Possibili schedule seriali dalle combinazioni di T1, T2,T3

T1,T2,T3 s1: r1(x) r1(y) w1(z) r2(y) w2(z) w3(z) w3(x)

E' VSR
Stesse relazione LEGGE – DA
Stesse scritte finali

View serializzabilità

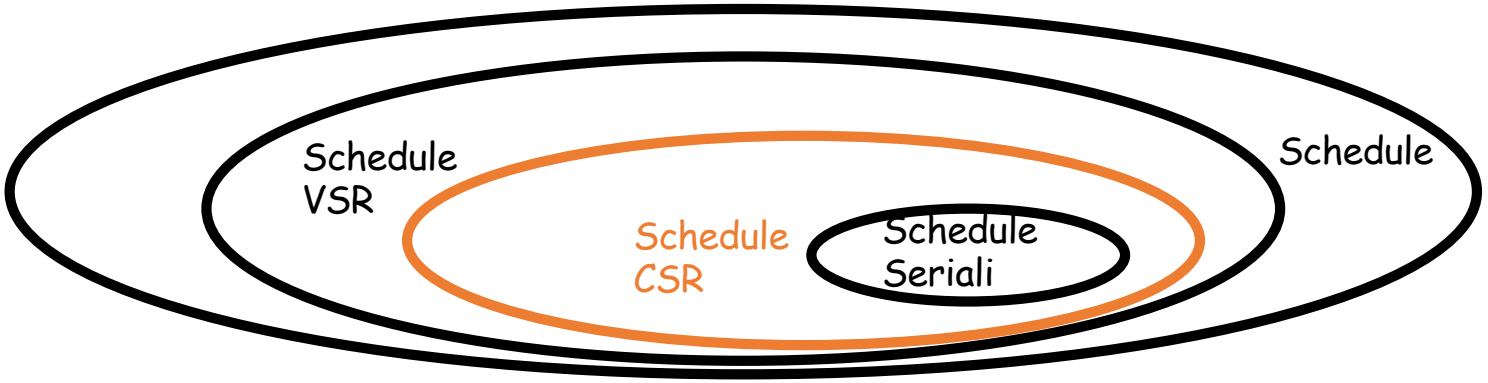
- Complessità:
  - la verifica della view-equivalenza di due dati schedule:
    - polinomiale
  - decidere sulla view serializzabilità di uno schedule:
    - problema NP-completo, perchè date n transazioni ci sono n! possibili schedule seriali.
- Non è utilizzabile in pratica

Conflict-serializzabilità

- Definizione preliminare:
  - Un'azione a_i è in *conflitto* con a_j ($i \neq j$), se operano sullo stesso oggetto e almeno una di esse è una scrittura. Due casi:
    - conflitto *read-write* (*rw* o *wr*)
    - conflitto *write-write* (*ww*).
- *Schedule conflict-equivalenti* ($S_i \approx_C S_j$): includono le stesse transazioni e relative operazioni e ogni coppia di operazioni in conflitto compare nello stesso ordine in entrambi
- Uno schedule è *conflict-**serializzabile*** se esiste uno schedule seriale ad esso conflict-equivalente
- L'insieme degli schedule conflict-serializzabili è indicato con CSR

CSR e VSR

- Ogni schedule conflict-serializzabile è view-serializzabile, ma non necessariamente viceversa
  - Non lo dimostriamo



Verifica di conflict-serializzabilità

- Per mezzo del **grafo dei conflitti**:
  - un nodo per ogni transazione t_i
  - un arco (orientato) da t_i a t_j se c'è almeno un conflitto fra un'azione a_i e un'azione a_j tale che a_i precede a_j
- Teorema
  - **Uno schedule è in CSR se e solo se il grafo è aciclico** (non dimostrazione)

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

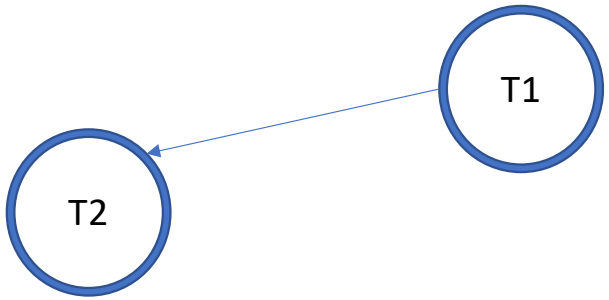
r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

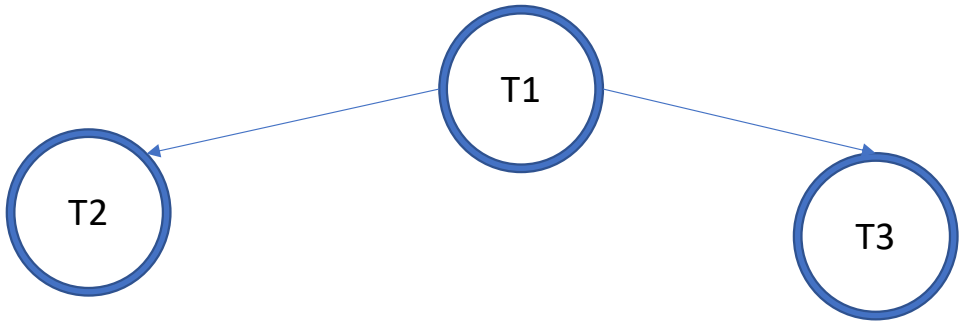


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

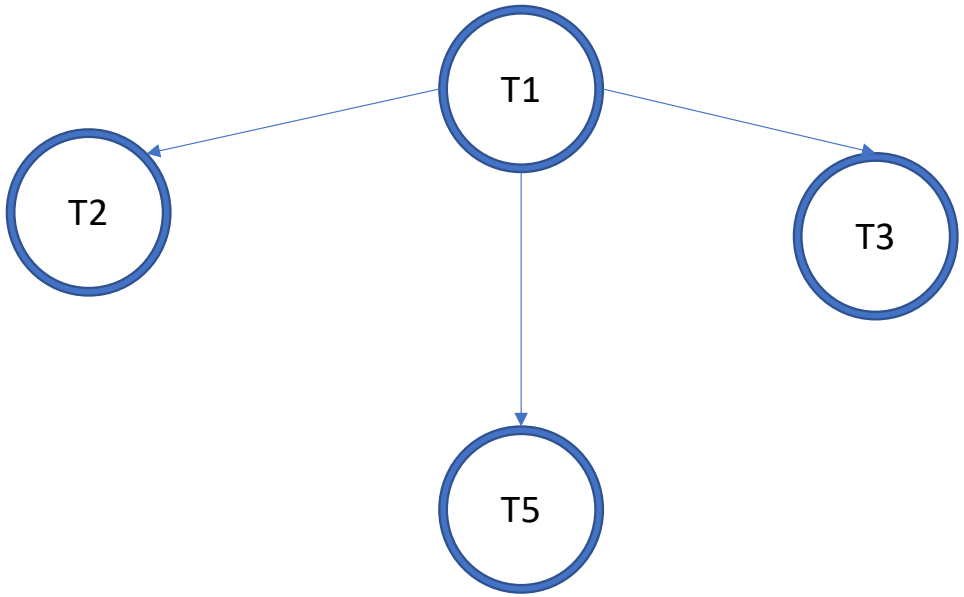


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

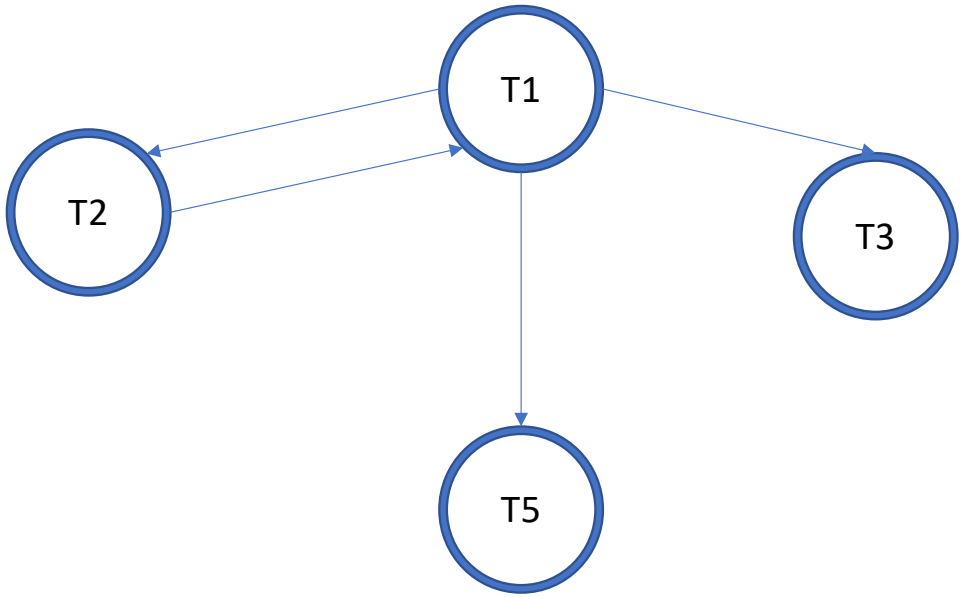


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

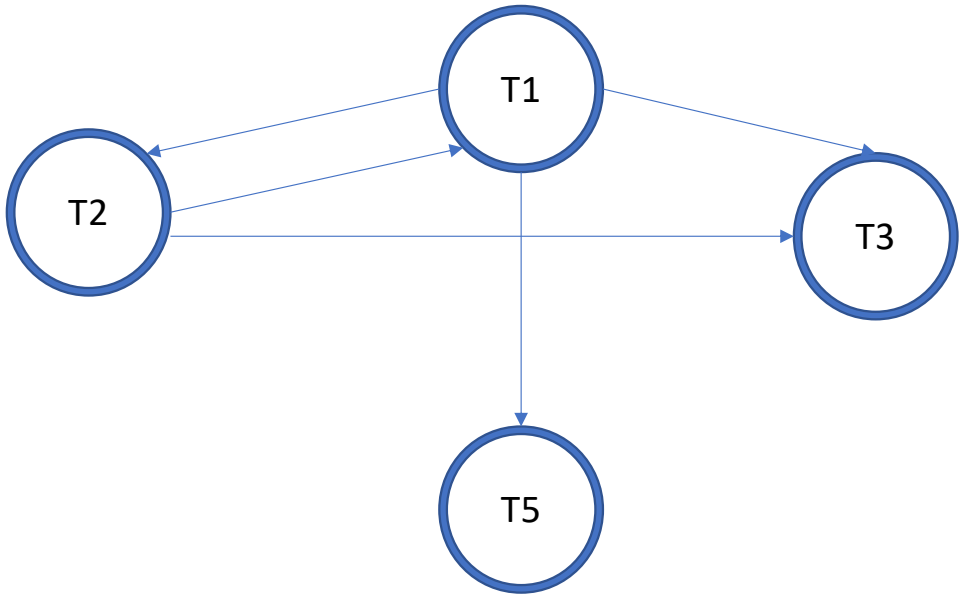


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

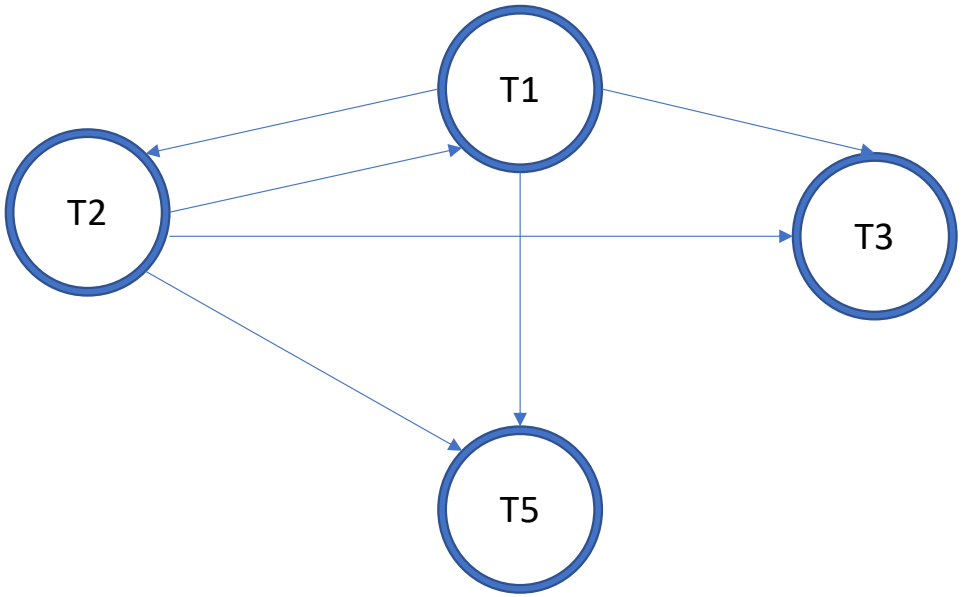


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

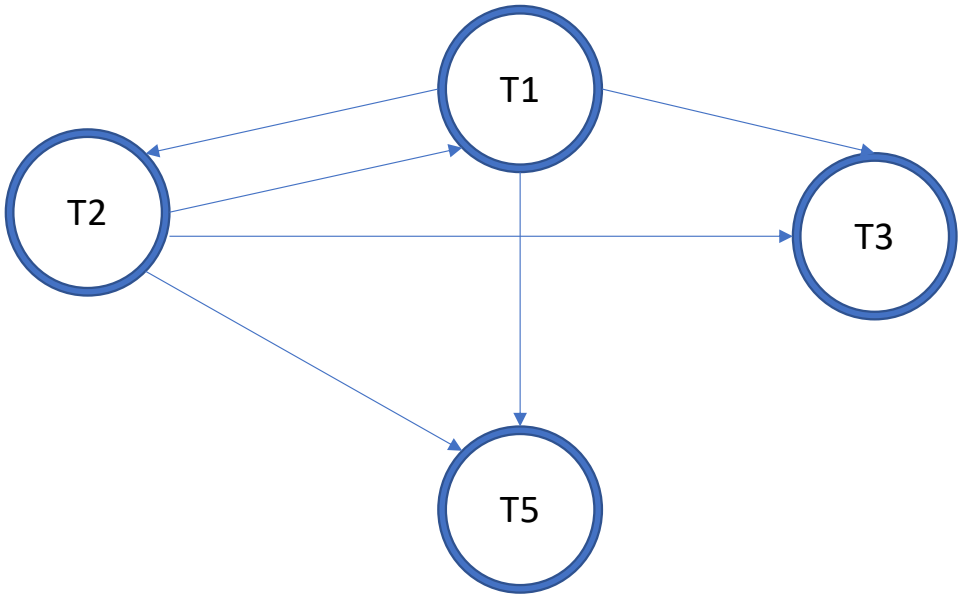


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

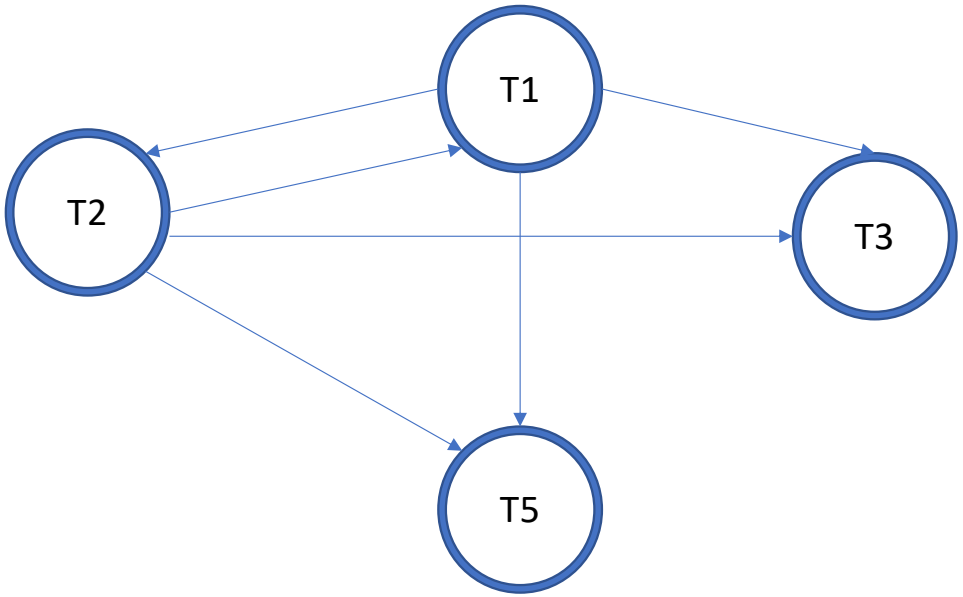


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

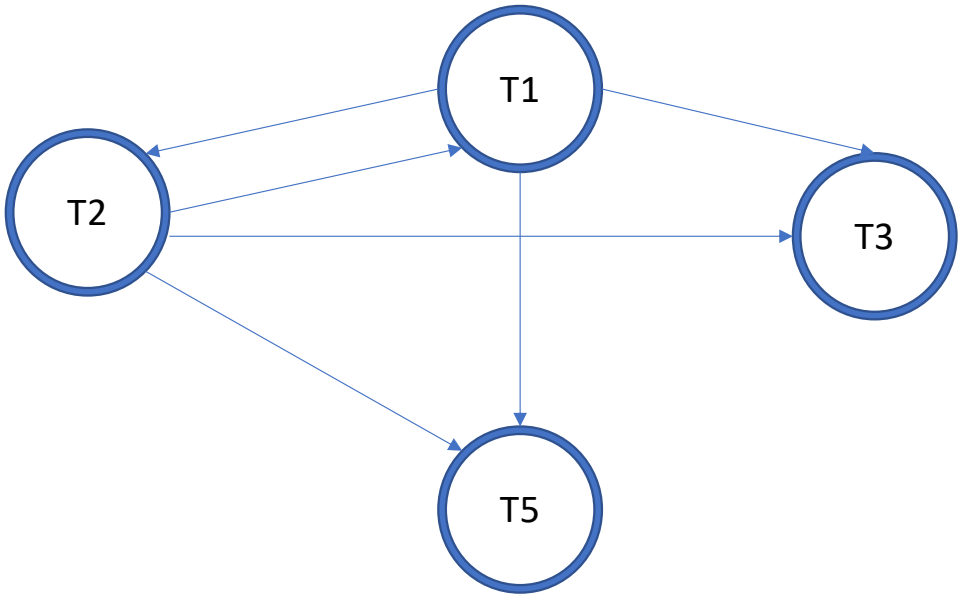


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

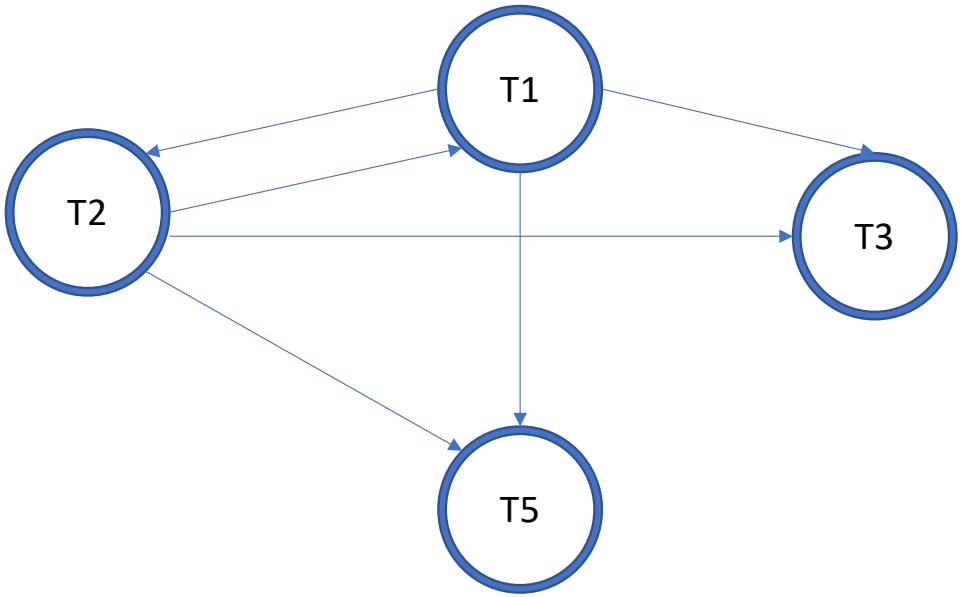


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

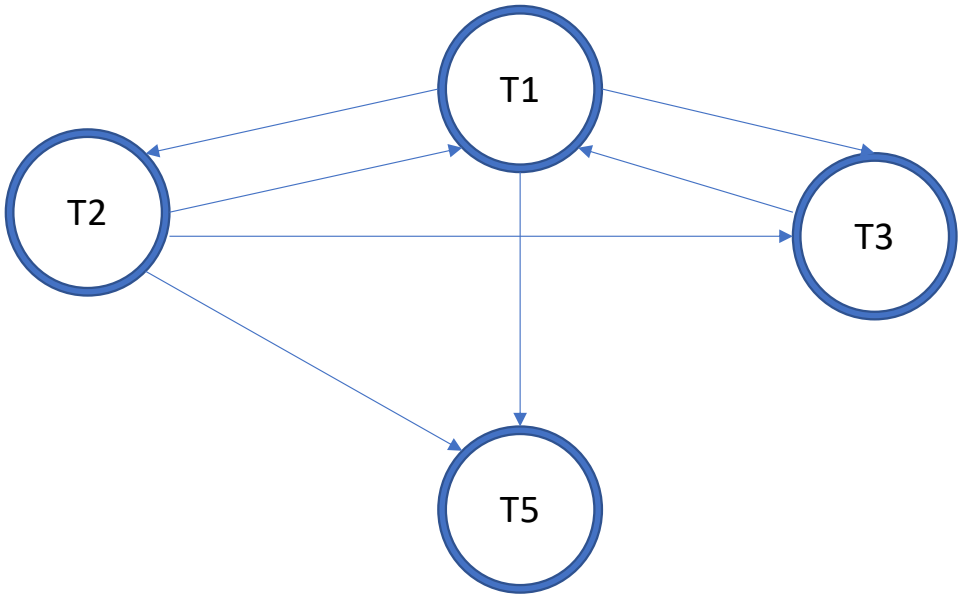


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

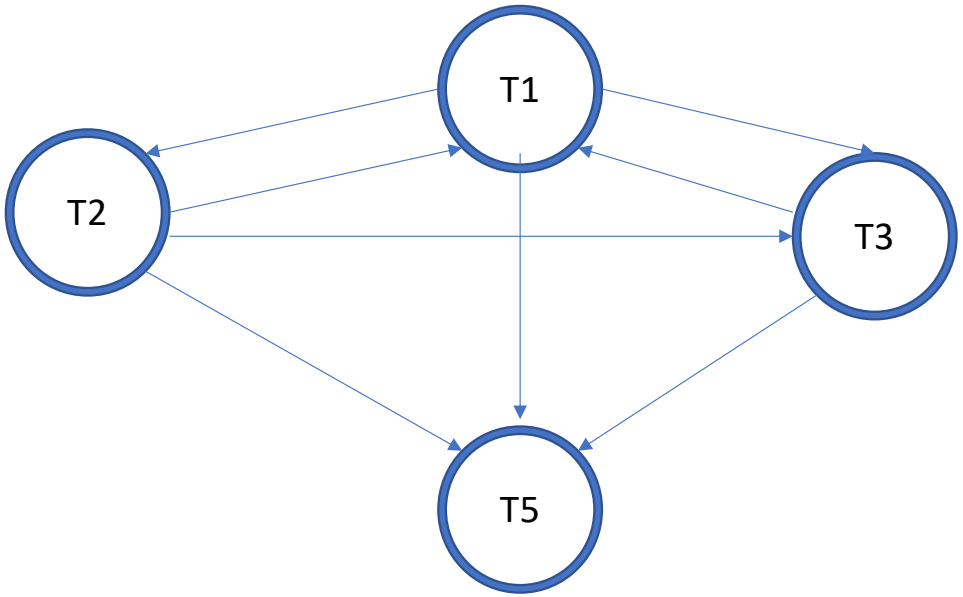


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

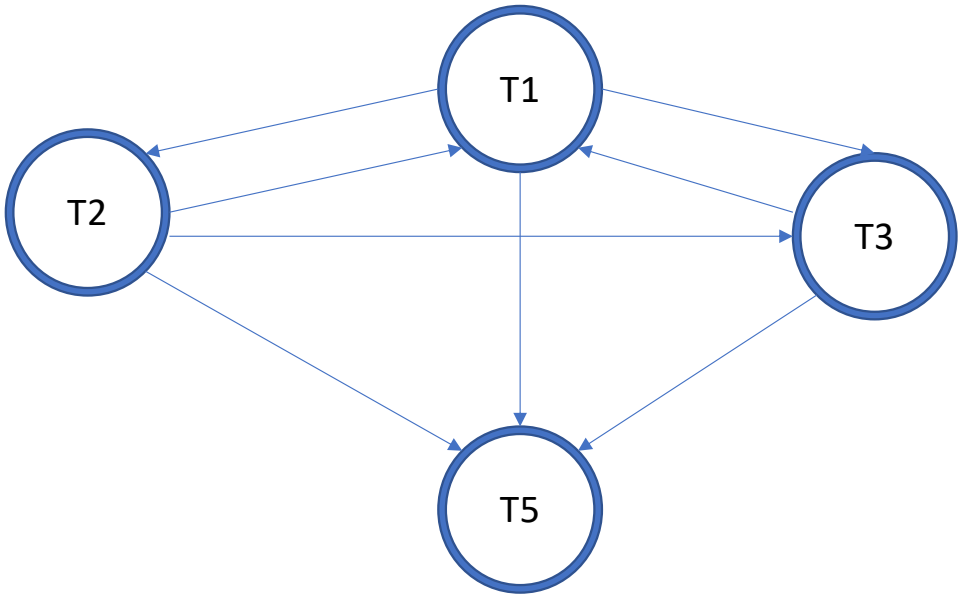


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

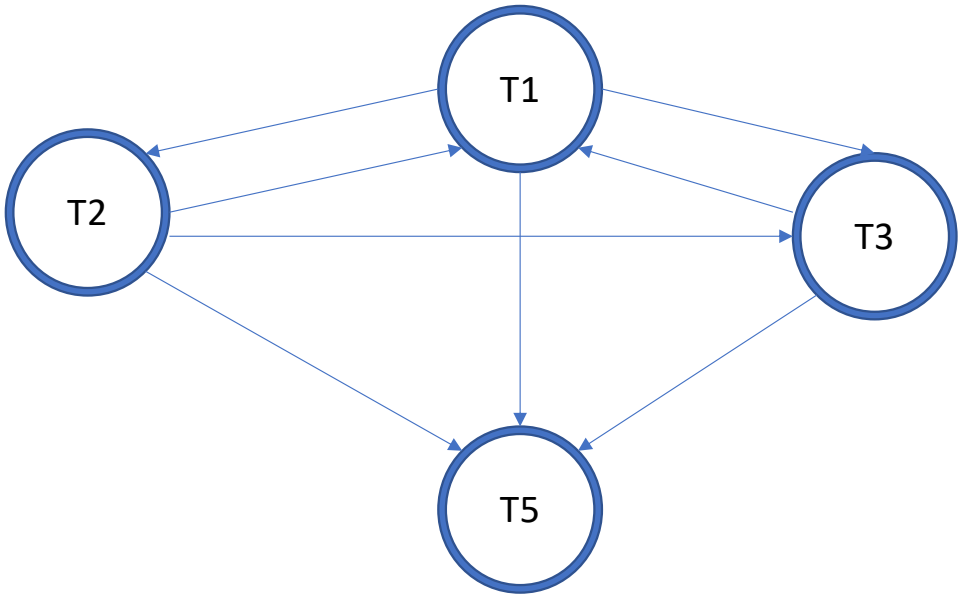


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

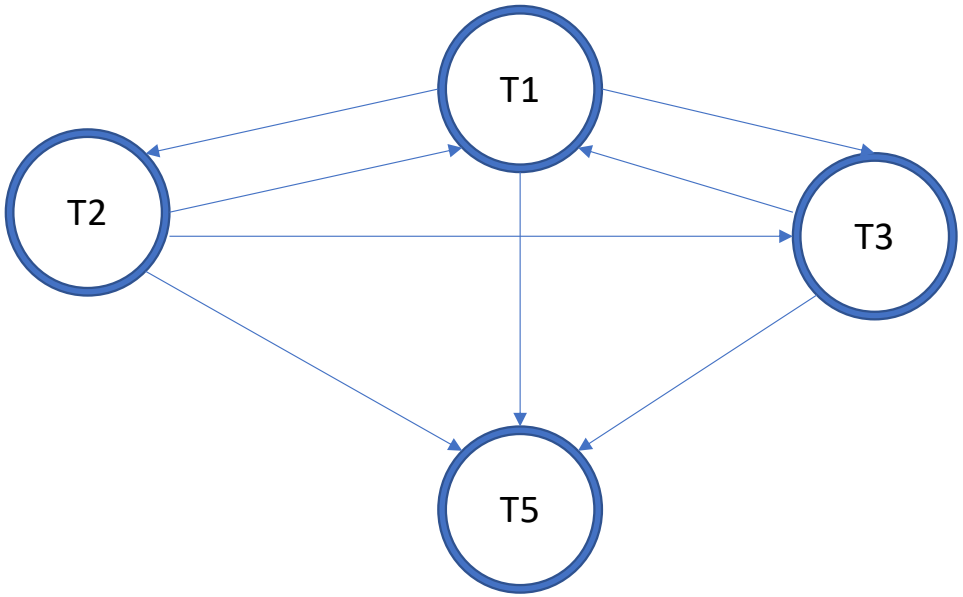


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

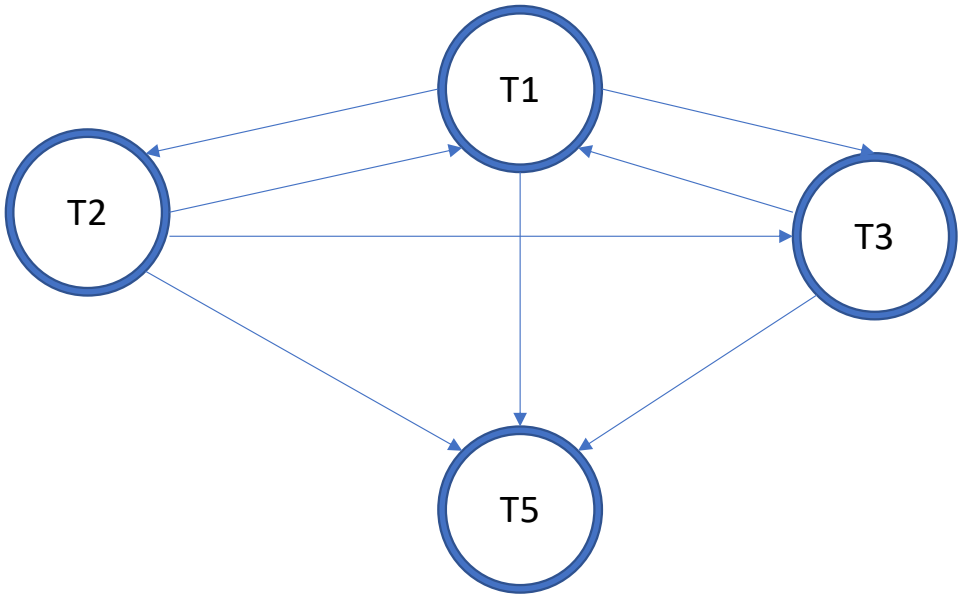


Considero oggetto x:
r1,r2, w2,r3,w1,w3,w5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

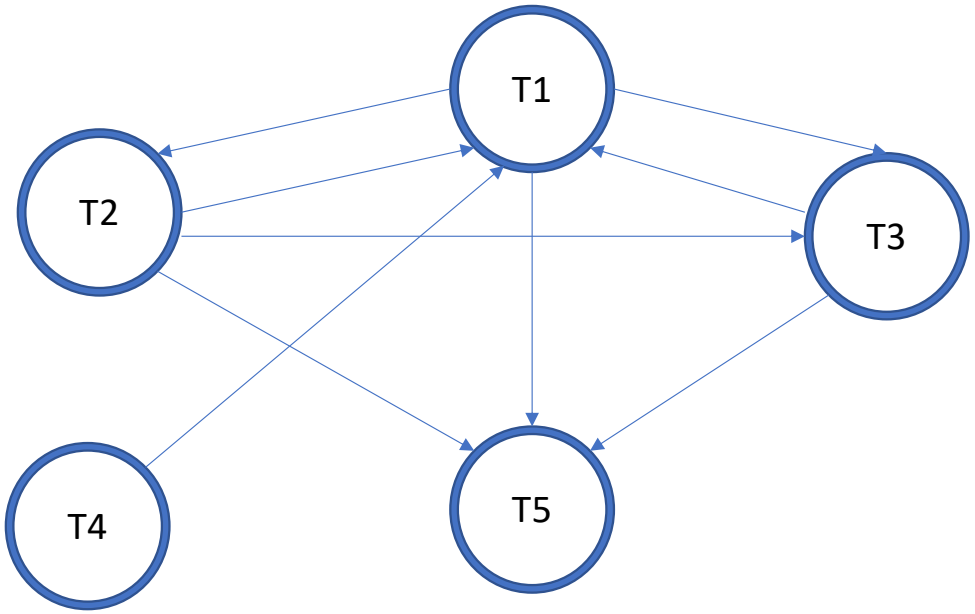


Considero oggetto z:
r4,w1,r5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)

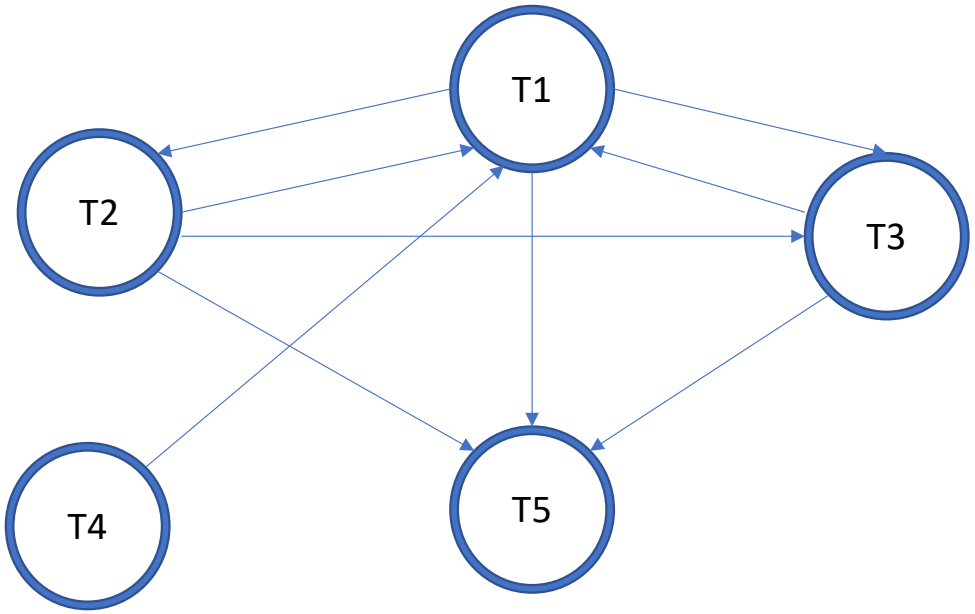


Considero oggetto z:
r4,w1,r5

Esercizio: controllo CSR

- Indicare se il seguente schema è CSR

r1(x) r2(x) w2(x) r3(x) r4(z) w1(x) w3(y) w3(x) w1(y) w5(x) w1(z) w5(y) r5(z)



Considero oggetto y:
w3,w1,w5

Non è CSR

Controllo della concorrenza in pratica

- Anche la conflict-serializabilità, anche se più rapidamente verificabile è inutilizzabile in pratica
- La tecnica sarebbe efficiente se potessimo conoscere il grafo dall'inizio, ma così non è:
  - uno scheduler deve operare "incrementalmente"
- Inoltre, la tecnica si basa sull'ipotesi di commit-proiezione
- In pratica, si utilizzano tecniche che
  - garantiscono la conflict-serializzabilità senza dover costruire il grafo
  - non richiedono l'ipotesi della commit-proiezione

Tecniche di Locking

- Tecniche di locking evitano conflitti, garantendo che le transazioni accedano ai dati in mutua esclusione
- Regole:
  - Ogni operazione di lettura è preceduta da un *r_lock* e seguita da *unlock.*
    - *Il r_lock è di tipo condiviso, in quanto possono esistere più r_lock contemporaneamente*
  - Ogni operazione di scrittura è preceduta da un *w_lock* e seguita da unlock.
    - *Il w_lock è di tipo esclusivo, in quanto NON possono esistere altri lock contemporaneamente*
- Le transazioni che rispettano queste regole sono ben formate rispetto al locking
- Quando una transazione prima legge e poi scrive sullo stesso oggetto, può:
  - richiedere subito un lock esclusivo (w_lock)
  - chiedere prima un lock condiviso (r_lock) e poi uno esclusivo (*lock escalation*)
- Il *lock manager* (gestore della concorrenza) riceve queste richieste dalle transazioni e le accoglie o rifiuta, sulla base della tavola dei conflitti

Gestione dei lock

- Tavola dei conflitti

Stato della risorsa

RICHIESTA	LIBERO	R_locked	W_locked
r_locked	Ok / r_locked	Ok / r_locked	NO / w_locked
w_locked	Ok / w_locked	NO / r_locked	NO / w_locked
Unlock	Error	Ok / dipende	OK / libero

Un contatore tiene conto del numero di "lettori"; la risorsa è rilasciata quando il contatore scende a zero

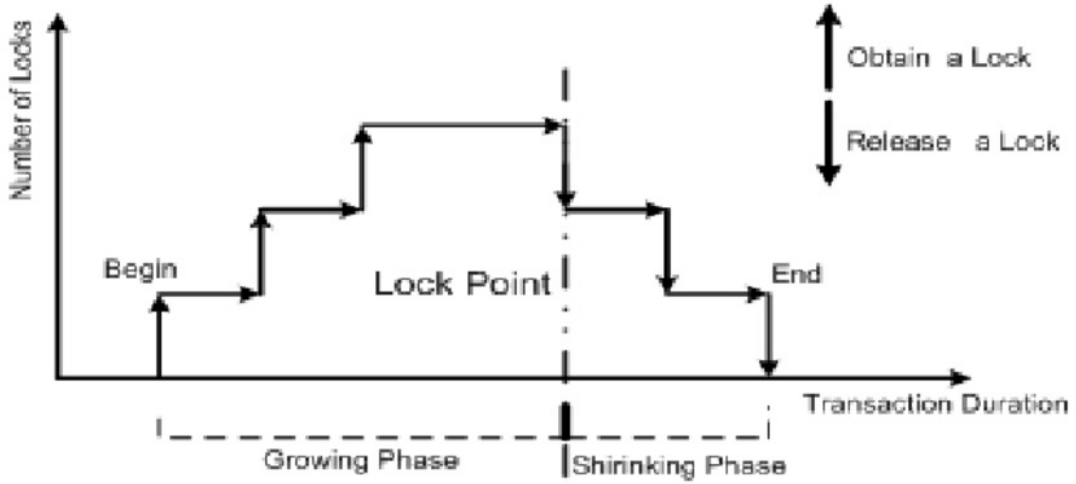
- Se la risorsa non è concessa, la transazione richiedente è posta in attesa (eventualmente in coda), fino a quando la risorsa non diventa disponibile
- Il lock manager gestisce una tabella dei lock, per ricordare la situazione

2PL - Locking **a due fasi**

- la tabella dei conflitti, ci garantisce che le transazioni accedano ai dati in mutua esclusione
- Per garantire che le transazioni seguano un schedule serializzabile è necessario porre un'ulteriore restrizione:

Locking a due fasi *(Two Phase Locking - 2PL):*
una transazione, dopo aver rilasciato un lock, non può acquisirne altri

- Una transazione 2PL ha due fasi
  - Prima fase in cui si acquisiscono i lock per le risorse a cui si deve accedere
  - Seconda fase in cui i lock acquisti vengono rilasciati

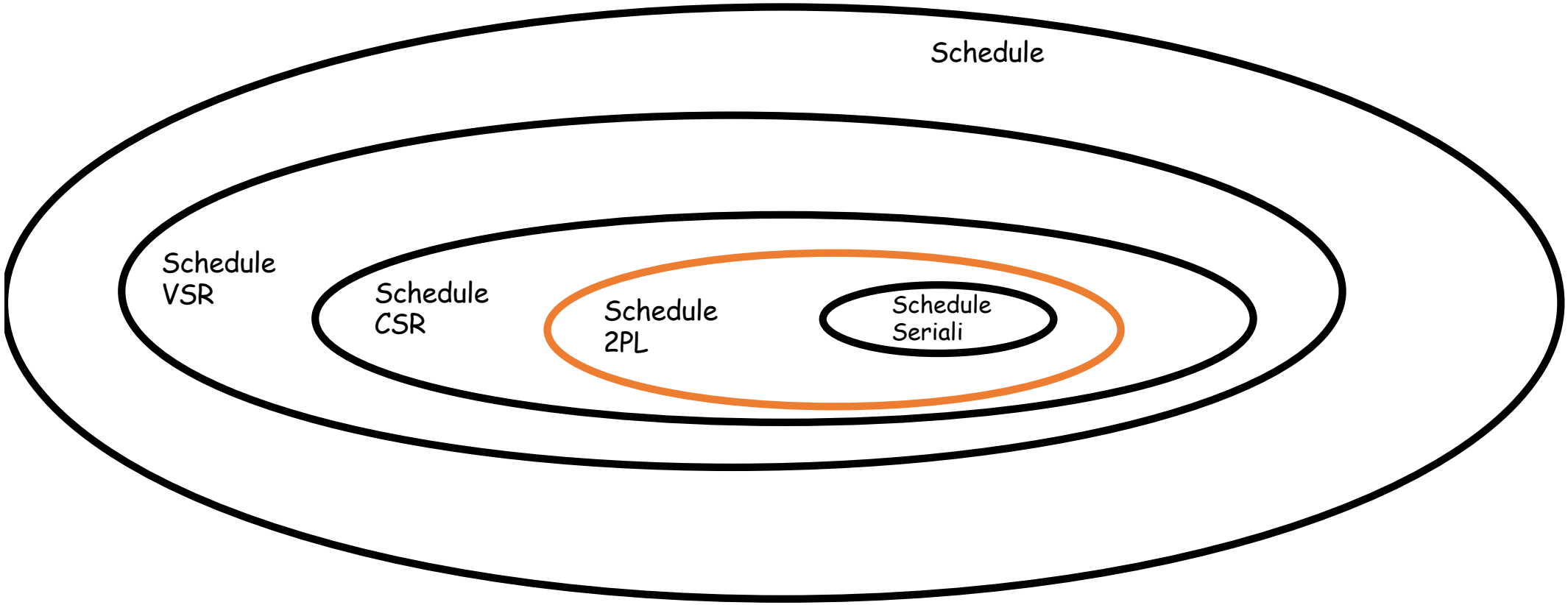


Locking a due fasi

- Un sistema in cui
  - le transazioni sono ben formate rispetto ai locking
  - il lock manager rispetta la politica descritta dalla tabella dei conflitti
  - le transazioni seguono il principio del locking in due fasi (2PL)
- è un sistema transazionale caratterizzato dalla serializzabilità delle proprie transazioni

- Ogni schedule 2PL è anche conflict serializzabile, ma non necessariamente viceversa
  - Non lo dimostriamo

CSR, VSR e 2PL


Schedule
Schedule VSR
Schedule CSR
Schedule 2PL
Schedule Seriali

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T3 richiede r_lock(y) ma lo ha già T1!

T1 non può fare unlock di y, perché non potrebbe più chiedere lock su x (per R1(x))

Può anticipare lock su x cosi da iniziare la fase di unlocking?

Si, perché su X non ci sono altre operazioni che lavorano su x prima di R1

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T3 ora può richiede r_lock(y)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T3 ora può richiede r_lock(y)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		
		r_lock(y)
		R3(y)

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		
		r_lock(y)
		R3(y)
	r_Lock(y)	
	R2(y)	

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		
		r_lock(y)
		R3(y)
	r_Lock(y)	
	R2(y)	
R1(x)		
Unlock(x)		

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		
		r_lock(y)
		R3(y)
	r_Lock(y)	
	R2(y)	
R1(x)		
Unlock(x)		
	R_lock(z)	
	R2(z)	

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

R2(x) W1(y) W1(z) R3(y) R2(y) R1(x) R2(z) W3(x)

T1: W1(y) W1(z) R1(x)

T2: R2(x) R2(y) R2(z)

T3: R3(y) W3(x)

Lo schedule può essere eseguito in 2PL, quindi è anche CSR

T1	T2	T3
	r_lock(x)	
	R2(x)	
w_lock(y)		
W1(y)		
w_lock(z)		
W1(z)		
r_lock(x)		
Unlock(y)		
Unlock(z)		
		r_lock(y)
		R3(y)
	r_Lock(y)	
	R2(y)	
R1(x)		
Unlock(x)		
	R_lock(z)	
	R2(z)	
		w_lock(x)
		W3(x)

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

T1	T2	T3
	W_Lock(x)	
	W2(x)	

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

T1	T2	T3
	W_Lock(x)	
	W2(x)	

T1 chiede w_lock(x), il lock lo ha T2

T2 può anticipare il lock di y per W2(y) cosi da fare unlock di x?

Si, perché tra W1(x) e W2(y) non ci sono altre operazioni che lavorano su y

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

T1	T2	T3
	W_Lock(x)	
	W2(x)	
	W_Lock_(y)	
	Unlock(x)	

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

T1	T2	T3
	W_Lock(x)	
	W2(x)	
	W_Lock_(y)	
	Unlock(x)	
W_Lock_(x)		
W1(x)		

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

T1	T2	T3
	W_Lock(x)	
	W2(x)	
	W_Lock_(y)	
	Unlock(x)	
W_Lock_(x)		
W1(x)		

T3 chiede w_lock(x), il lock lo ha T1

T1 può anticipare il lock di y per W1(y) cosi da fare unlock di x?

No, perché tra W3(x) e W1(y) c'è W2(y) che lavorare su y, quindi non si può anticipare il lock

Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

T1: W1(x) W1(y)

T2: W2(x) W2(y)

T3: W3(x) W3(y)

T1	**T2**	**T3**
	W_Lock(x)	
	W2(x)	
	W_Lock_(y)	
	Unlock(x)	
W_Lock_(x)		
W1(x)		
W_Lock(y) //negato		

Non è in 2PL!

È CSR?

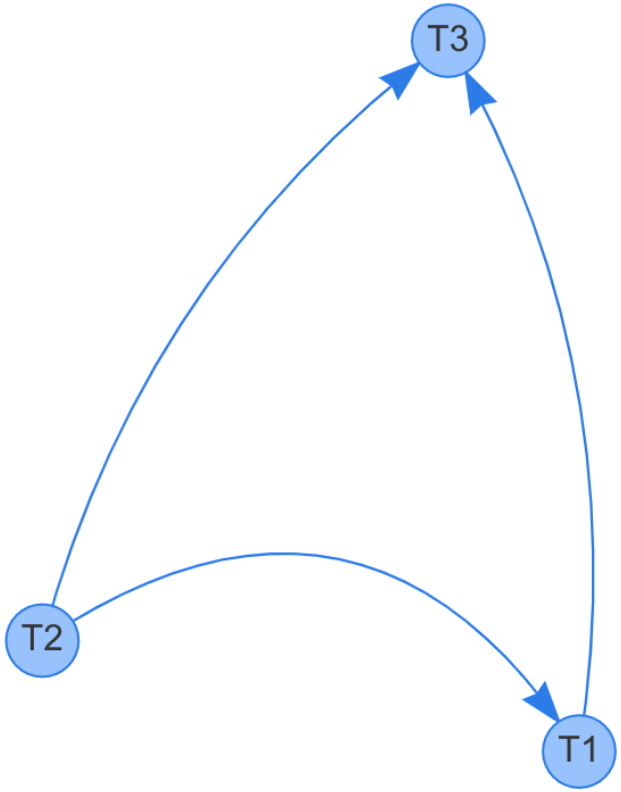
Esercizio: controllo 2PL

- Indicare se il seguente schedule è in 2PL

W2(x) W1(x) W3(x) W2(y) W1(y) W3(y)

x: w2,w1,w3,

y: w2,w1,w3

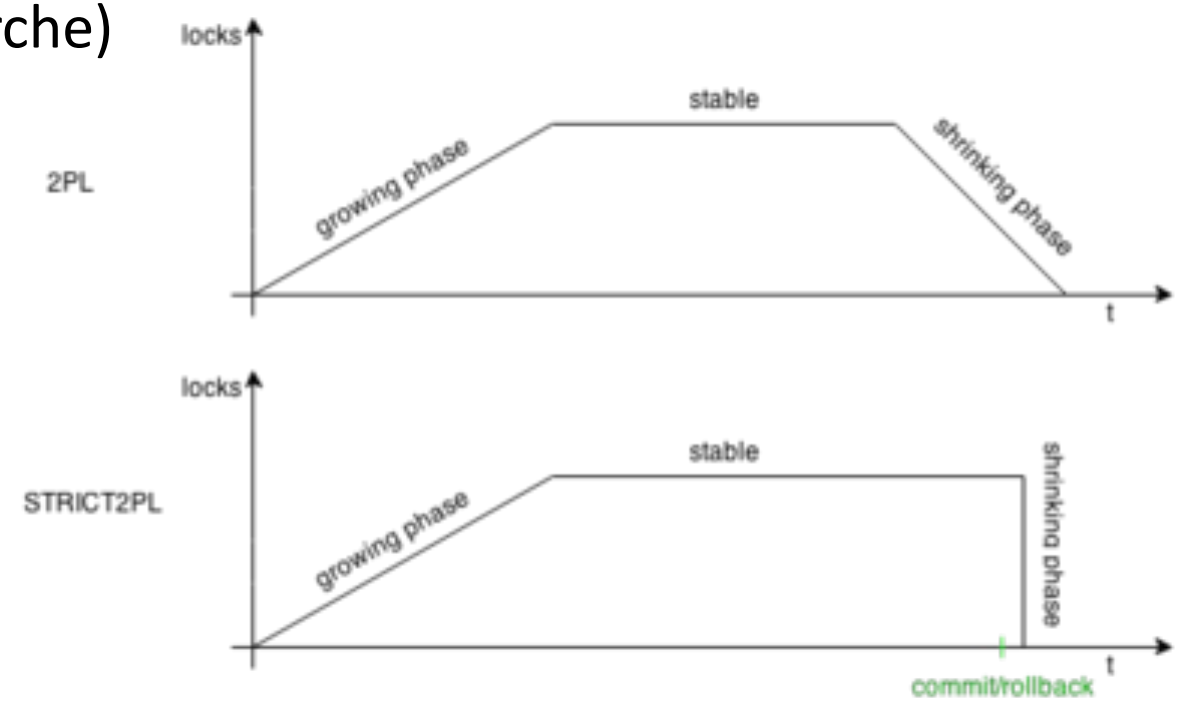


Non è in 2PL!

E' CSR

Locking a due fasi stretto **(STRICT 2PL)**

- Condizione aggiuntiva:
  - **I lock possono essere rilasciati solo dopo il commit o abort**
- Supera la necessità dell'ipotesi di commit-proiezione (ed elimina il rischio di letture sporche)



Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3
r_lock(y)		
R1(y)		

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T2 chiede w_lock(x). T1 può rilasciare lock perché non ha altre operazione di lock

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
	w_lock(x)	
	W2(x)	

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T3 chiede w_lock(x). Lock su x lo ha T2

T2 anticipa la richiesta di lock su y, cosi da iniziare unlock

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	
Unlock(y)		
	w_lock(y)	
	Unlock(x)	
		W_lock(x)
		W3(x)

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	
Unlock(y)		
	w_lock(y)	
	Unlock(x)	
		W_lock(x)
		W3(x)
	W2(y)	
	Unlock(y)	
		Unlock(x)

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

E’ 2PL strict?

Con 2PL strict gli unlock avvengono solo quando si fa il commit:

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	
Unlock(y)		
	w_lock(y)	
	Unlock(x)	
		W_lock(x)
		W3(x)
	W2(y)	
	Unlock(y)	
		Unlock(x)

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

E' 2PL strict?

Con 2PL strict gli unlock avvengono solo quando si fa il commit:

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
Unlock(y)		

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

E' 2PL strict?

Con 2PL strict gli unlock avvengono solo quando si fa il commit:

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
Unlock(y)		
	w_lock(x)	
	W2(x)	

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

E' 2PL strict?

Con 2PL strict gli unlock avvengono solo quando si fa il commit:

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
Unlock(y)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	

Esercizio: controllo 2PL

- R1(y) R3(z) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(z) W3(x)

E' 2PL strict?

Con 2PL strict gli unlock avvengono solo quando si fa il commit:

- T3 chiede il w_lock(x) ma lo ha T2 che non può rilasciarlo fino al commit dopo W2(y)

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(z)
		R3(z)
r_lock(x)		
R1(x)		
Unlock(x)		
Unlock(y)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	
		w_lock(x) /negato

Esercizio: controllo 2PL

- R1(y) R3(y) R1(x) W2(x) R2(y) W3(x) W2(y)

T1: R1(y) R1(x)

T2: W2(x) R2(y) W2(y)

T3: R3(y) W3(x)

E' 2PL strict?

Con 2PL strict gli unlock avvengono solo quando si fa il commit:

- T3 chiede il w_lock(x) ma lo ha T2 che non può rilasciarlo fino al commit dopo W2(y)

Non è 2PL strict

T1	T2	T3
r_lock(y)		
R1(y)		
		r_lock(y)
		R3(y)
r_lock(x)		
R1(x)		
Unlock(x)		
Unlock(y)		
	w_lock(x)	
	W2(x)	
	r_lock(y)	
	R2(y)	
		w_lock(x) /negato

Esercizi: controllo concorrenza

- Determinare se il seguente schedule è VSR e/o CSR.
  - r1(x) r2(y) w3(y) r5(x) w5(u) w3(s) w2(u) w3(x) w1(u) r4(y) w5(z) r5(z)

Costruisco il grafo dei conflitti:

Considero le operazioni sui singoli oggetti

S: w3(s)

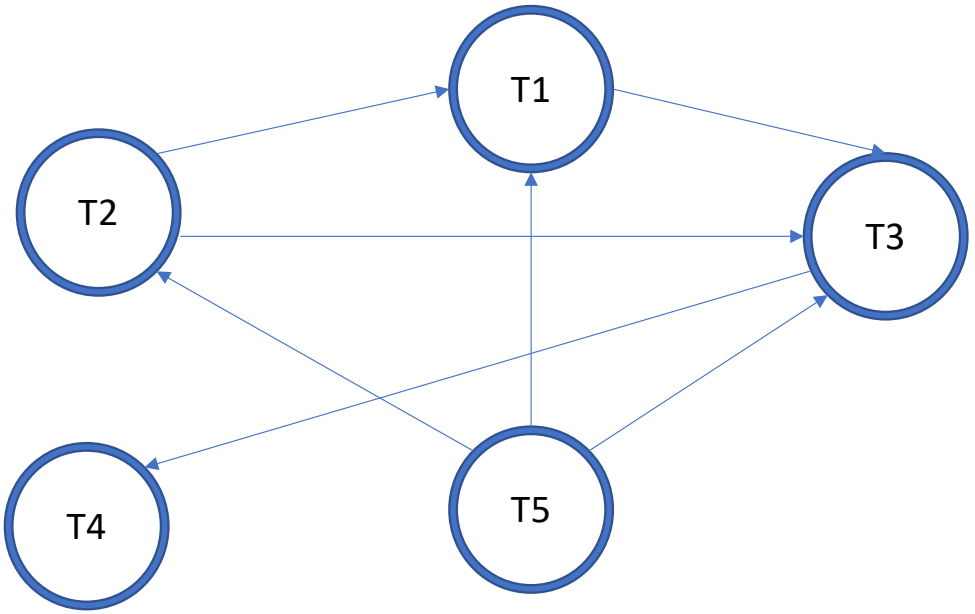
U: w5(u), w2(u), w1(u)

X: r1(x), r5(x), w3(x)

Y: r2(y), w3(y), r4(y)

Z: w5(z), r5(z)

È CSR, quindi anche VSR



Esercizi: controllo concorrenza

- Determinare se il seguente schedule è VSR e/o CSR.
- r1(x) w1(x) r2(z) r1(y) w1(y) r2(x) w2(x) w2(z)

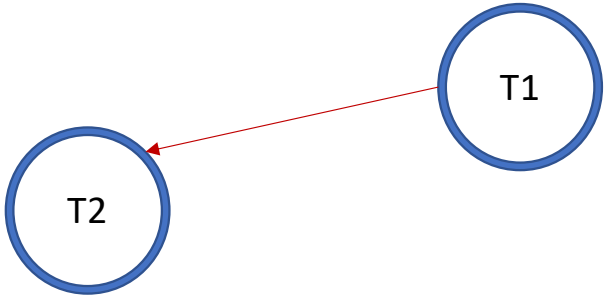
Costruisco il grafo dei conflitti:

Considero le operazioni sui singoli oggetti

X: r1(x), w1(x), r2(x), w2(x)

Y: r1(y), w1(y)

Z: r2(z), w2(z)



È CSR, quindi anche VSR

Esercizi: ripresa a caldo

B(T1)
B(T2)
U(T2, O2, B1, A1)
U(T1, O1, B2, A2)
C(T2)
C(T1)
B(T3)
U(T3, O4, B3, A3)
U(T3, O2, B4, A4)

Assumere che il gestore dell'affidabilità faccia un checkpoint dopo l'ultimo update:

- Cosa contiene il record di CK da inserire nel log?

- Al checkpoint quali oggetti vengono traferiti in memoria secondaria?

Esercizi: ripresa a caldo

B(T1)
B(T2)
U(T2, O2, B1, A1)
U(T1, O1, B2, A2)
C(T2)
C(T1)
B(T3)
U(T3, O4, B3, A3)
U(T3, O2, B4, A4)
CK(T3)

Assumere che il gestore dell'affidabilità faccia un checkpoint dopo l'ultimo update:

- Cosa contiene il record di CK da inserire nel log?
  - CK(T3)

- Al checkpoint quali oggetti vengono traferiti in memoria secondaria?

Esercizi: ripresa a caldo

B(T1)
B(T2)
U(T2, O2, B1, A1)
U(T1, O1, B2, A2)
C(T2)
C(T1)
B(T3)
U(T3, O4, B3, A3)
U(T3, O2, B4, A4)
CK(T3)

Assumere che il gestore dell'affidabilità faccia un checkpoint dopo l'ultimo update:

- Cosa contiene il record di CK da inserire nel log?
  - CK(T3)

- Al checkpoint quali oggetti vengono traferiti in memoria secondaria?
  - Al checkpoint si trasferisce in memoria secondaria tutti gli oggetti modificati dalle transazioni andati in commit. In questo log, T1 e T2 hanno fatto il commit, viene quindi trasferito in memoria secondaria gli oggetto O1 e O2

Esercizi: ripresa a caldo

B(T1)
B(T2)
U(T2, O2, B1, A1)
U(T1, O1, B2, A2)
C(T2)
C(T1)
B(T3)
U(T3, O4, B3, A3)
U(T3, O2, B4, A4)
CK(T3)
B(T4)
U(T4, O1, B5, A5)
D(T4, O6, B6)
U(T3, O1, B7, A7)
C(T3)
B(T5)
U(T5, O4, B8, A8)
A(T4)
U(T5, O1, B9, A9)

Ipotizzando un failure dopo l'ultima istruzione inserita nel log, calcolare l'insieme delle transazioni per cui è necessario fare undo (UNDO_SET) e redo (REDO_SET) in caso di ripresa a caldo:

Esercizi: ripresa a caldo

B(T1)
B(T2)
U(T2, O2, B1, A1)
U(T1, O1, B2, A2)
C(T2)
C(T1)
B(T3)
U(T3, O4, B3, A3)
U(T3, O2, B4, A4)
CK(T3)
B(T4)
U(T4, O1, B5, A5)
D(T4, O6, B6)
U(T3, O1, B7, A7)
C(T3)
B(T5)
U(T5, O4, B8, A8)
A(T4)
U(T5, O1, B9, A9)

Ipotizzando un failure dopo l'ultima istruzione inserita nel log, calcolare l'insieme delle transazioni per cui è necessario fare undo (UNDO_SET) e redo (REDO_SET) in caso di ripresa a caldo:

Esercizi: ripresa a caldo

B(T1)
B(T2)
U(T2, O2, B1, A1)
U(T1, O1, B2, A2)
C(T2)
C(T1)
B(T3)
U(T3, O4, B3, A3)
U(T3, O2, B4, A4)
CK(T3)
B(T4)
U(T4, O1, B5, A5)
D(T4, O6, B6)
U(T3, O1, B7, A7)
C(T3)
B(T5)
U(T5, O4, B8, A8)
A(T4)
U(T5, O1, B9, A9)

Ipotizzando un failure dopo l'ultima istruzione inserita nel log, calcolare l'insieme delle transazioni per cui è necessario fare undo (UNDO_SET) e redo (REDO_SET) in caso di ripresa a caldo:

UNDO_set= T4, T5
REDO_set= T3

Esercizi: ripresa a caldo

B(T1)	
B(T2)	
U(T2, O2, B1, A1)	
U(T1, O1, B2, A2)	
C(T2)	
C(T1)	
B(T3)	
U(T3, O4, B3, A3)	5 Redo
U(T3, O2, B4, A4)	6 Redo
CK(T3)	
B(T4)	
U(T4, O1, B5, A5)	4 Undo O1=B5
D(T4, O6, B6)	3 Undo
U(T3, O1, B7, A7)	7 Redo O1=A7
C(T3)	
B(T5)	
U(T5, O4, B8, A8)	2 Undo
A(T4)	
U(T5, O1, B9, A9)	1 Undo O1=B9

Che valore ha l'oggetto O1 dopo la ripresa a caldo:

UNDO_set= T4, T5
REDO_set= T3