

Lezione 26– Il meccanismo di lock – Lock binario

Prof.ssa Maria De Marsico
demarsico@di.uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA

lock : privilegio di **accesso** ad un **singolo** item
realizzato mediante una **variabile** associata all'item
(variabile **lucchetto**) il cui valore descrive lo **stato**
dell'item rispetto alle operazioni che **possono essere**
effettuate su di esso

Nella sua forma più semplice, un lock

- viene **richiesto** da una transazione mediante un'operazione di *locking*: se il valore della variabile è **unlocked** la transazione **può accedere** all'item e alla variabile viene assegnato il valore *locked*
- viene **rilasciato** da una transazione mediante un'operazione di *unlocking* che assegna alla variabile il valore *unlocked*

Quindi:

il locking agisce come **primitiva di sincronizzazione**, cioè se una transazione richiede un lock su un item **su cui un'altra transazione mantiene un lock**, la transazione **non può** procedere finchè il lock **non viene rilasciato** dalla prima transazione

Fra l'esecuzione di un'operazione di locking su un certo item X e l'esecuzione di un'operazione di unlocking su X la transazione ***mantiene un lock su X***

Uno **schedule** è detto *legale* se
una transazione effettua un locking **ogni volta** che deve
leggere o scrivere un item
ciascuna transazione **rilascia ogni lock** che ha ottenuto

La forma più semplice: Lock binario



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Un lock **binario** può assumere solo due valori **locked** e **unlocked**

Le transazioni fanno uso di due operazioni

- **lock(X)** per **richiedere** l'accesso all'item X
- **unlock(X)** per **rilasciare** l'item X **consentendone** l'accesso ad altre transazioni

L'insieme degli item **letti** e quello degli item **scritti** da una transazione **coincidono**



Risolviamo il **primo** dei problemi visti, cioè la perdita di aggiornamento



T_1

read(X)
X:=X-N
write(X)
read(Y)
Y:=Y+N
write(Y)

T_1 può essere interpretata
come il trasferimento
di una somma di denaro N
dal conto corrente X al conto corrente Y

T_2

read(X)
X:=X+M
write(X)

T_2 può essere interpretata
come l'accredito sul conto
corrente X
di una somma di denaro M

T_1	T_2
$read(X)$ $X := X - N$	$read(X)$ $X := X + M$
$write(X)$ $read(Y)$	
$Y := Y + N$ $write(Y)$	$write(X)$

Consideriamo il seguente schedule di T_1 e T_2

Se il valore iniziale di X è X_0 al termine dell'esecuzione dello schedule il valore di X è $X_0 + M$ invece di $X_0 - N + M$

L'**aggiornamento** di X prodotto da T_1 viene **perso**



Riscriviamo le
transazioni
utilizzando le
primitive di
lock binario

T_1	T_2
<i>lock(X)</i>	<i>lock(X)</i>
<i>read(X)</i>	<i>read(X)</i>
$X := X - N$	$X := X + M$
<i>write(X)</i>	<i>write(X)</i>
<i>unlock(X)</i>	<i>unlock(X)</i>
<i>lock(Y)</i>	
<i>read(Y)</i>	
$Y := Y + N$	
<i>write(Y)</i>	
<i>unlock(Y)</i>	



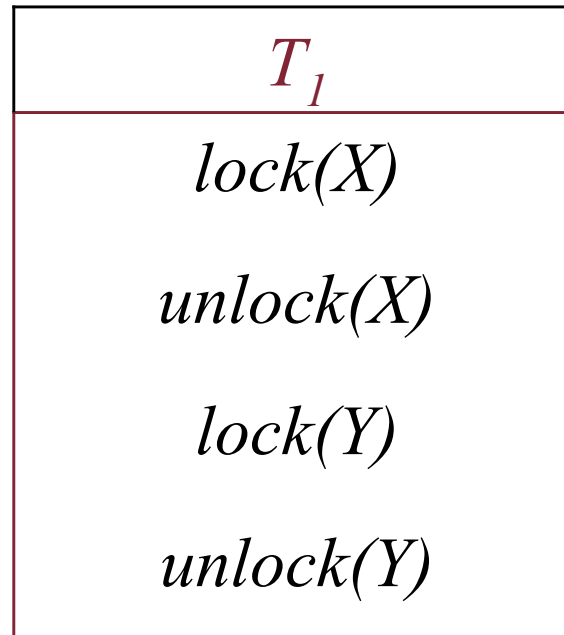
T_1	T_2
$lock(X)$ $read(X)$ $X := X - N$ $write(X)$ $unlock(X)$	$lock(X)$ $read(X)$ $X := X + M$ $write(X)$ $unlock(X)$
$lock(Y)$ $read(Y)$ $Y := Y + N$ $write(Y)$ $unlock(Y)$	

Schedule legale di T_1 e T_2

Il problema
dell'**aggiornamento perso**
è **risolto!**



- Vedremo che la proprietà di **equivalenza** degli schedule **dipende** dal **protocollo** di locking adottato
- Vediamo nel caso semplice del lock binario (a due valori) come formalizzare questo concetto
- Prima di tutto adottiamo un modello delle transazioni **che astrae dalle specifiche operazioni** e si basa su quelle **rilevanti** per valutare le **sequenze degli accessi**, cioè in questo caso **lock e unlock**



Una transazione è una **sequenza di operazioni di *lock* e *unlock***

- ogni *lock(X)* implica la **lettura** di X
- ogni *unlock(X)* implica la **scrittura** di X

T_1
$lock(X)$
$unlock(X) f_1(X)$
$lock(Y)$
$unlock(Y) f_2(X, Y)$

In corrispondenza di una scrittura viene associato un nuovo valore all'item coinvolto che viene calcolato da una **funzione**

- che è associata in modo **univoco** ad ogni **coppia lock-unlock**
- che ha per **argomenti tutti** gli item letti (locked) dalla transazione **prima** dell'operazione di unlock (perché magari i loro valori hanno contribuito all'aggiornamento dell'item corrente)

Due schedule sono *equivalenti* se le formule che danno i valori finali per ciascun item sono le stesse

Le formule devono essere uguali per tutti gli item!



Uno schedule è **serializzabile** se è **equivalente** ad uno **schedule seriale** (basta trovarne uno)

Consideriamo le due transazioni

T_1	T_2
$lock(X)$	$lock(Y)$
$unlock(X) f_1(X)$	$unlock(Y) f_3(Y)$
$lock(Y)$	$lock(X)$
$unlock(Y) f_2(X, Y)$	$unlock(X) f_4(X, Y)$

e lo schedule ...



	T_1	T_2
legge X_0 scrive $f_1(X_0)$	$lock(X)$ $unlock(X)$	$lock(Y)$ $unlock(Y)$ legge Y_0 scrive $f_3(Y_0)$
legge $f_3(Y_0)$ scrive $f_2(X_0, f_3(Y_0))$	$lock(Y)$ $unlock(Y)$	$lock(X)$ $unlock(X)$ legge $f_1(X_0)$ scrive $f_4(f_1(X_0), Y_0)$

X_0 valore iniziale di X
 Y_0 valore iniziale di Y

$f_4(f_1(X_0), Y_0)$ valore finale di X

Esempio: i possibili schedule seriali



	T_1	T_2
legge X_0	$lock(X)$	
scrive $f_1(X_0)$	$unlock(X)$	
legge Y_0	$lock(Y)$	
scrive $f_2(X_0, Y_0)$	$unlock(Y)$	
		$lock(Y)$
		$unlock(Y)$
		$lock(X)$
		$unlock(X)$

Consideriamo lo schedule
seriale T_1, T_2

legge $f_2(X_0, Y_0)$

scrive $f_3(f_2(X_0, Y_0))$

legge $f_1(X_0)$

scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$

$f_4(f_1(X_0), f_2(X_0, Y_0))$ valore finale di X

prodotto dallo schedule
seriale T_1, T_2



Consideriamo lo
schedule seriale T_2, T_1

legge $f_4(X_0, Y_0)$
scrive $f_1(f_4(X_0, Y_0))$
legge $f_3(Y_0)$
scrive $f_2(f_4(X_0, Y_0), f_3(Y_0))$

T_1	T_2
	$lock(Y)$
	$unlock(Y)$
	$lock(X)$
	$unlock(X)$
$lock(X)$	
$unlock(X)$	
$lock(Y)$	
$unlock(Y)$	

legge Y_0
scrive $f_3(Y_0)$
legge X_0
scrive $f_4(X_0, Y_0)$

$f_1(f_4(X_0, Y_0))$ valore finale di X

prodotto dallo schedule
seriale T_2, T_1



T_1	T_2
$lock(X)$ $unlock(X)$	$lock(Y)$ $unlock(Y)$
$lock(Y)$ $unlock(Y)$	$lock(X)$ $unlock(X)$

Pertanto lo schedule **non** è **serializzabile** in quanto produce per X un valore finale ($f_4(f_1(X_0), Y_0)$) diverso sia da quello ($f_4(f_1(X_0), f_2(X_0, Y_0))$) prodotto dallo schedule seriale T_1, T_2 sia da quello ($f_1(f_4(X_0, Y_0))$) prodotto dallo schedule seriale T_2, T_1

Notiamo che lo stesso vale anche per Y



	T_1	T_2
legge X_0	$lock(X)$	
scrive $f_1(X_0)$	$unlock(X)$	
		$lock(Y)$
		$unlock(Y)$
legge $f_3(Y_0)$	$lock(Y)$	
scrive $f_2(X_0, f_3(Y_0))$	$unlock(Y)$	
		$lock(X)$
		$unlock(X)$
		legge $f_1(X_0)$
		scrive $f_4(f_1(X_0), Y_0)$

X_0 valore iniziale di X
 Y_0 valore iniziale di Y

$f_4(f_1(X_0), Y_0)$ valore finale di X
 $f_2(X_0, f_3(Y_0))$ valore finale di Y

Esempio: i possibili schedule seriali su Y



	T_1	T_2
legge X_0	$lock(X)$	
scrive $f_1(X_0)$	$unlock(X)$	
legge Y_0	$lock(Y)$	
scrive $f_2(X_0, Y_0)$	$unlock(Y)$	
		$lock(Y)$
		$unlock(Y)$
		$lock(X)$
		$unlock(X)$

Consideriamo lo schedule
seriale T_1, T_2

legge $f_2(X_0, Y_0)$
scrive $f_3(f_2(X_0, Y_0))$
legge $f_1(X_0)$
scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$

$f_4(f_1(X_0), f_2(X_0, Y_0))$ valore finale di X
 $f_3(f_2(X_0, Y_0))$ valore finale di Y

prodotti dallo schedule
seriale T_1, T_2

Esempio: i possibili schedule seriali su Y



Consideriamo lo
schedule seriale T_2, T_1

legge $f_4(X_0, Y_0)$

scrive $f_1(f_4(X_0, Y_0))$

legge $f_3(Y_0)$

scrive $f_2(f_4(X_0, Y_0), f_3(Y_0))$

T_1	T_2
	$lock(Y)$
	$unlock(Y)$
	$lock(X)$
	$unlock(X)$
$lock(X)$	
$unlock(X)$	
$lock(Y)$	
$unlock(Y)$	

legge Y_0

scrive $f_3(Y_0)$

legge X_0

scrive $f_4(X_0, Y_0)$

$f_1(f_4(X_0, Y_0))$ valore finale di X

$f_2(f_4(X_0, Y_0), f_3(Y_0))$ valore finale di Y

prodotti dallo schedule
seriale T_2, T_1

- Basta che le formule siano **diverse anche per un solo item** per concludere che gli schedule **non sono equivalenti**
- Quindi per verificare che uno schedule NON è serializzabile, possiamo fermarci appena troviamo un item le cui formule finali sono diverse da quelle di OGNI schedule seriale
- Per verificare che uno schedule È serializzabile occorre verificare che le formule finali di **TUTTI** gli item coincidono con quelle di uno (stesso) schedule seriale.



Consideriamo le due transazioni

T_1
$lock(X)$
$unlock(X) f_1(X)$
$lock(Y)$
$unlock(Y) f_2(X, Y)$

T_2
$lock(X)$
$unlock(X) f_3(X)$
$lock(Y)$
$unlock(Y) f_4(X, Y)$

e lo schedule ...



	T_1	T_2	
legge X_0	$lock(X)$		
scrive $f_1(X_0)$	$unlock(X)$		
		$lock(X)$	legge $f_1(X_0)$
		$unlock(X)$	scrive $f_3(f_1(X_0))$
legge Y_0	$lock(Y)$		
scrive $f_2(X_0, Y_0)$	$unlock(Y)$		
		$lock(Y)$	legge $f_2(X_0, Y_0)$
		$unlock(Y)$	scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$



	T_1	T_2
legge X_0	$lock(X)$	
scrive $f_1(X_0)$	$unlock(X)$	
legge Y_0	$lock(Y)$	
scrive $f_2(X_0, Y_0)$	$unlock(Y)$	
		$lock(X)$
		$unlock(X)$
		$lock(Y)$
		$unlock(Y)$

Consideriamo lo schedule
seriale T_1, T_2

legge $f_1(X_0)$

scrive $f_3(f_1(X_0))$

legge $f_2(X_0, Y_0)$

scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$



T_1	T_2
$lock(X)$ $unlock(X)$	$lock(X)$ $unlock(X)$
$lock(Y)$ $unlock(Y)$	$lock(Y)$ $unlock(Y)$

Pertanto lo schedule è **serializzabile** in quanto produce
sia per X che per Y gli stessi valori finali prodotti dallo
schedule seriale T_1, T_2



Uno schedule è **serializzabile** se esiste uno schedule seriale tale che

- **per ogni item l'ordine** in cui le varie transazioni **fanno un lock** su quell'item **coincide** con quello dello **schedule seriale**

Algoritmo 1

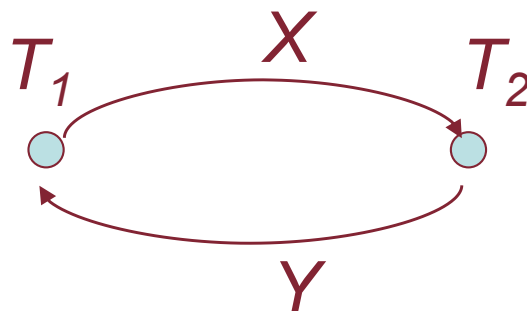
Dato uno schedule S

- **Passo 1**
- crea un grafo **diretto** G (*grafo di serializzazione*)
 - nodi: transazioni
 - archi: $T_i \rightarrow T_j$ (con etichetta X) se in S T_i esegue un *unlock(X)* e T_j esegue il successivo *lock(X)*

non **UN** successivo ma **IL** successivo, cioè T_j è la **prima** transazione che effettua il lock di X dopo che T_i ha effettuato l'unlock, anche se le due operazioni non sono di seguito



T_1	T_2
$lock(X)$ $unlock(X)$	
	$lock(Y)$ $unlock(Y)$
$lock(Y)$ $unlock(Y)$	
	$lock(X)$ $unlock(X)$





T_1	T_2
$lock(X)$ $unlock(X)$	$lock(X)$ $unlock(X)$
$lock(Y)$ $unlock(Y)$	$lock(Y)$ $unlock(Y)$

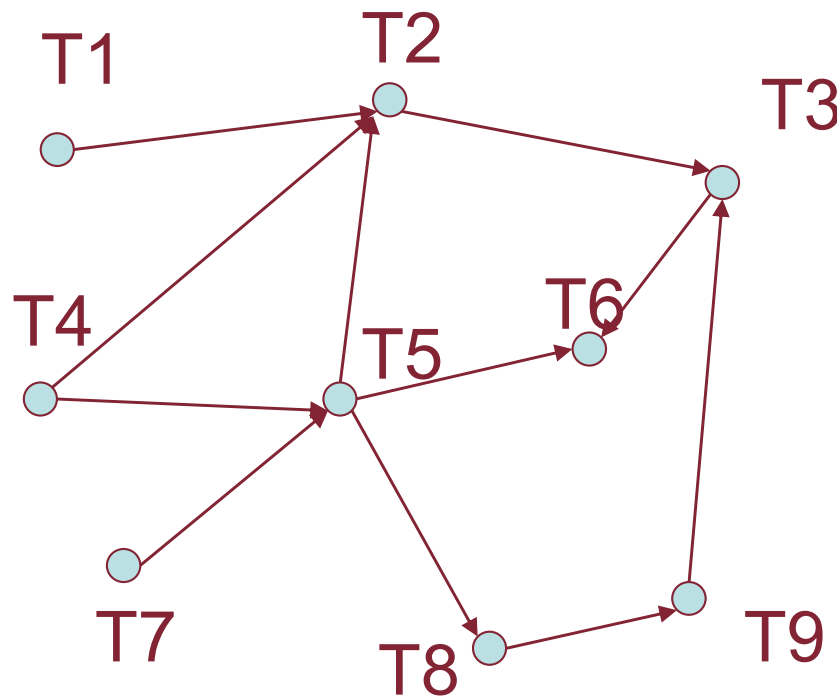




- ***Passo 2***
- Se G ha un ciclo allora S non è serializzabile; altrimenti applicando a G l'*ordinamento topologico* si ottiene uno schedule **seriale** S' equivalente ad S

Si ottiene eliminando ricorsivamente un nodo che non ha archi entranti, insieme ai suoi archi uscenti

Notare che un grafo può ammettere più ordinamenti topologici



Attenzione al verso degli archi!

T4 T7 T5 T1 T8 T9 T2 T3 T6



T_1	T_2
$lock(X)$ $unlock(X)$ $lock(Y)$ $unlock(Y)$	 $lock(X)$ $unlock(X)$ $lock(Y)$ $unlock(Y)$



T_1 T_2

Teorema (correttezza dell'Algoritmo del grafo di serializzazione)



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Uno schedule S è **serializzabile** se e solo se il suo grafo di **serializzazione** è **aciclico**

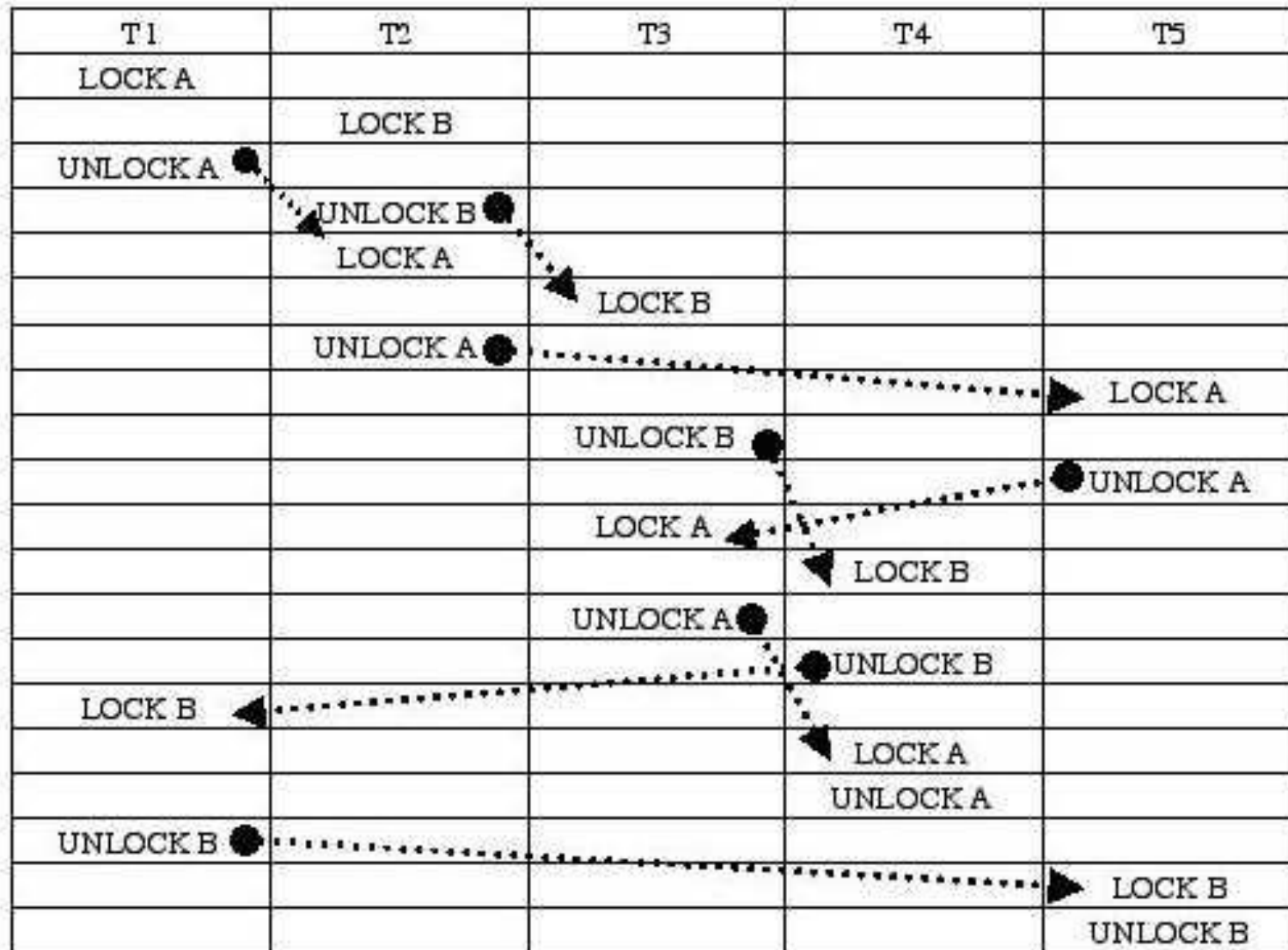
Prendiamo questo schedule di 5 transazioni

T1	T2	T3	T4	T5
LOCK A				
	LOCK B			
UNLOCK A				
	UNLOCK B			
	LOCK A			
		LOCK B		
	UNLOCK A			
				LOCK A
		UNLOCK B		
				UNLOCK A
		LOCK A		
			LOCK B	
		UNLOCK A		
			UNLOCK B	
LOCK B				
			LOCK A	
			UNLOCK A	
UNLOCK B				
				LOCK B
				UNLOCK B

Esempio



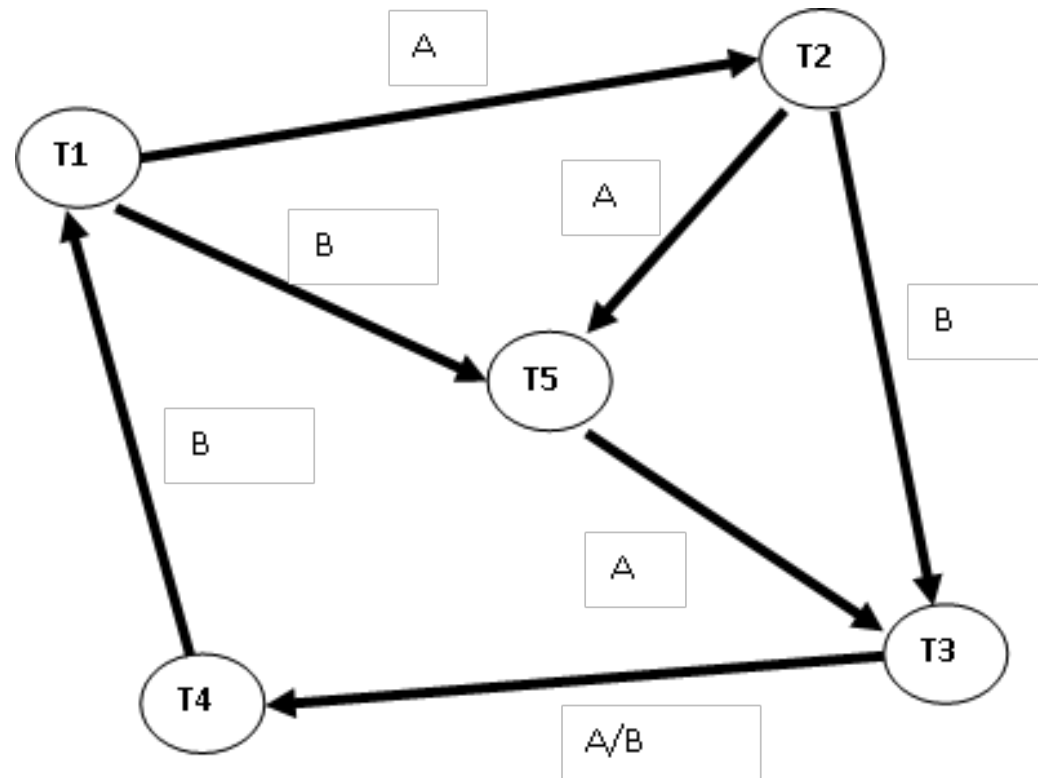
Applichiamo l'algoritmo e segniamo sulla tabella le relazioni tra le transazioni che produrranno archi nel grafo



Esempio



Applichiamo l'algoritmo e segniamo sulla tabella le relazioni tra le transazioni che produrranno archi nel grafo



Il grafo presenta il ciclo T1-T2 - T3 - T4. Possiamo quindi concludere che lo schedule dato non è serializzabile.

Attenzione! T1-T2 -T5 e T2 -T5 -T3 **NON** sono cicli in quanto i sensi delle frecce non sono descrivono cicli, mentre T1-T5-T3-T4-T1 **lo è**

Una transazione obbedisce al protocollo di locking a due fasi, o più semplicemente è **a due fasi**, se

- prima effettua **tutte** le operazioni di *lock* (**fase di locking**) e
- poi **tutte** le operazioni di *unlock* (**fase di unlocking**)

Attenzione! Da non confondere con il lock a due valori!
Il fatto di essere **a due fasi** è una caratteristica in più,
ma ci sono protocolli a due fasi ... e tre valori.



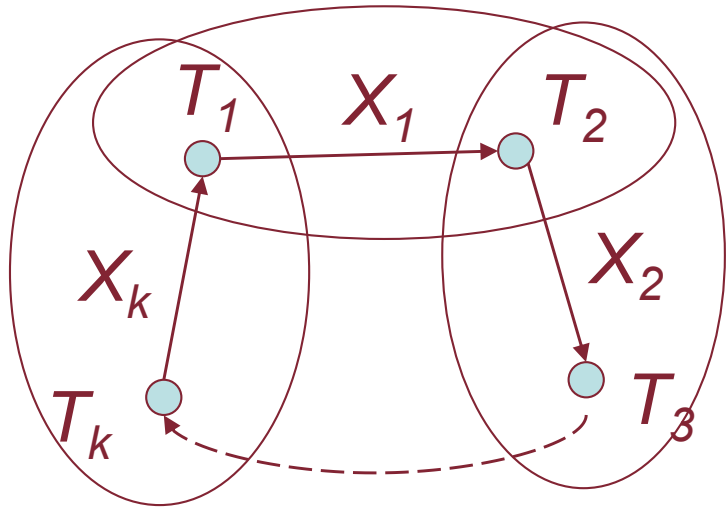
Sia T un insieme di transazioni.

Se ogni transazione in T è a due fasi allora ogni schedule di T è serializzabile

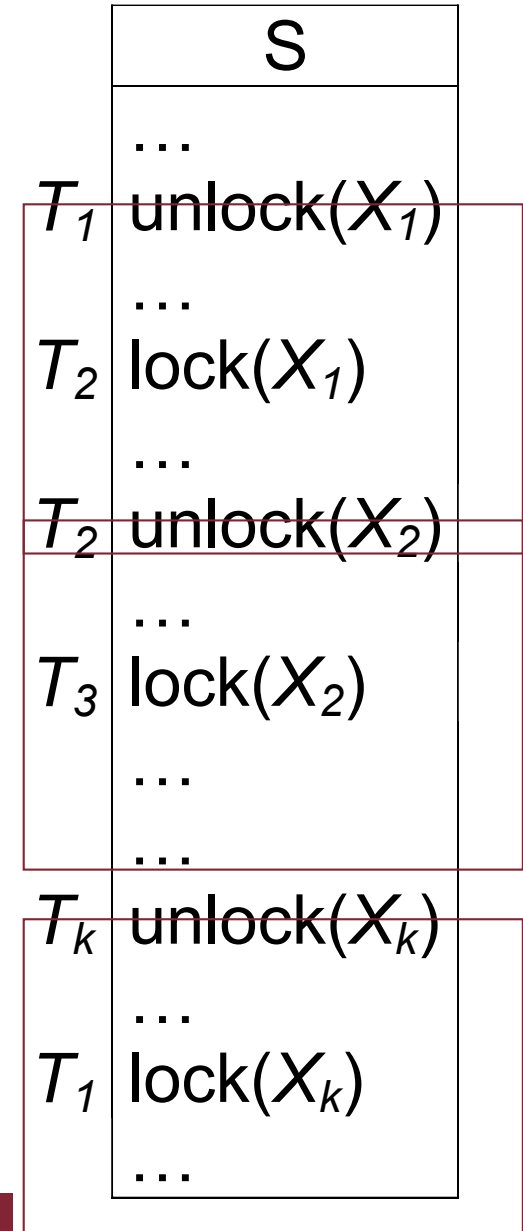
Dimostrazione Teorema 2



Per assurdo: ogni transazione in S è a due fasi
ma nel grafo di serializzazione c'è un ciclo



La transazione T_1 non è a due fasi
(**contraddizione**)



Per comodità abbiamo chiamato T_1 la **prima** transazione del ciclo che compare nello schedule con una operazione che genera un arco del ciclo

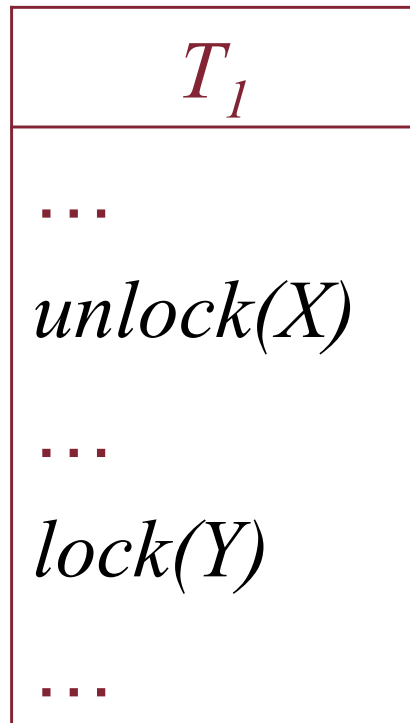
Pur non essendo T_1 e T_2 a due fasi, lo schedule

T_1	T_2
$lock(X)$ $unlock(X)$	$lock(X)$ $unlock(X)$
$lock(Y)$ $unlock(Y)$	$lock(Y)$ $unlock(Y)$

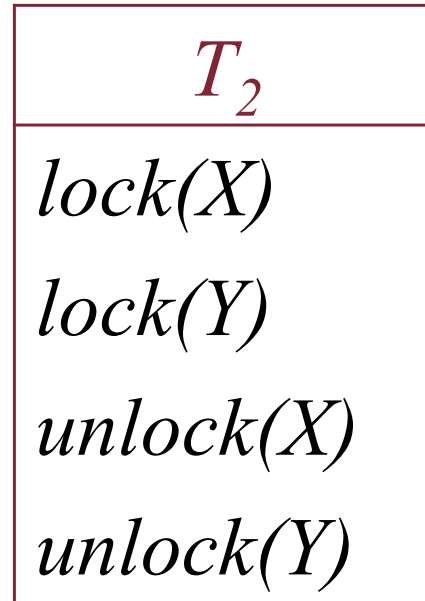


è serializzabile. D'altra parte...

Se una transazione non è a due fasi...

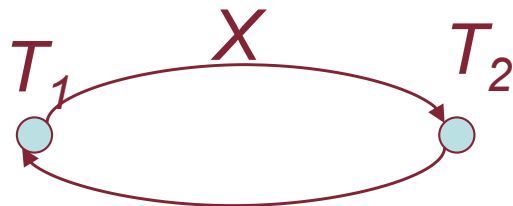


...esiste sempre una transazione a due fasi...



... e uno schedule delle due transazioni...

T_1	T_2
...	
<i>unlock(X)</i>	
...	
	<i>lock(X)</i>
	<i>lock(Y)</i>
	<i>unlock(X)</i>
	<i>unlock(Y)</i>
...	
<i>lock(Y)</i>	
...	



Y

... che non è serializzabile

Solo se **tutte** le transazioni sono a due fasi possiamo avere la certezza che **ogni** schedule è serializzabile

Nota importante: **TUTTI** i protocolli di lock a due fasi (a prescindere **dal numero** di valori di lock) **risolvono** il problema **dell'aggregato non corretto**