Controllo di concorrenza basato su *timestamp*

- Tecnica alternativa al 2PL
- ***Timestamp***:
  - identificatore che definisce un ordinamento totale sugli eventi di un sistema
- Ogni transazione ha un *timestamp* che rappresenta l'**istante di inizio della transazione**
- Uno schedule è accettato solo se riflette **l'ordinamento seriale** delle transazioni **indotto dai timestamp**

Dettagli

- Lo scheduler ha **due contatori** $RTM(x)$ e $WTM(x)$ **per ogni oggetto** x
- Lo scheduler riceve **richieste** di **letture** e **scritture** (con indicato il timestamp della transazione):
  - `read`(x, ts):
    - se $ts < WTM(x)$ allora la **richiesta è respinta** e la transazione viene **uccisa**
    - altrimenti, la **richiesta viene accolta** e $RTM(x)$ è posto uguale al maggiore fra $RTM(x)$ e ts
  - `write`(x,ts):
    - se $ts < WTM(x)$ o $ts < RTM(x)$ allora la **richiesta è respinta** e la transazione viene **uccisa**
    - altrimenti, la **richiesta viene accolta** e $WTM(x)$ è posto uguale a ts
- Vengono **uccise molte transazioni**

Esempio

$$RTM(x) = 0$$
$$WTM(x) = 0$$

Richiesta	Risposta	Nuovo valore
$read(x, t_1)$	OK	$RTM(x) = 1$
$write(x, t_1)$	OK	$WTM(x) = 1$
$read(x, t_2)$	OK	$RTM(x) = 2$
$read(x, t_1)$	OK	
$write(x, t_1)$	No, t_1 aborted	
$read(x, t_2)$	OK	
$write(x, t_2)$	OK	$WTM(x) = 2$

Esempio

$$RTM(x) = 7$$
$$WTM(x) = 4$$

Richiesta	Risposta	Nuovo valore
$read(x, t_6)$	OK	
$read(x, t_8)$	OK	$RTM(x) = 8$
$read(x, t_9)$	OK	$RTM(x) = 9$
$write(x, t_8)$	NO, t_8 aborted	
$write(x, t_{11})$	OK	$WTM(x) = 11$
$read(x, t_{10})$	NO, t_{10} aborted	

Risoluzione del deadlock

- L'**ordine seriale** delle transazioni è **fissato** prima che le operazioni vengano richieste, tutti gli altri ordinamenti non sono accettati
- Quando T_1 comincia prima di T_2, potrebbe essere abilitato uno schedule 2PL o CSR equivalente ad uno seriale $T_2\ T_1$; col TS non è possibile, al limite T_1 viene **abortita e poi fatta ripartire** dopo T_2
- In 2PL le transazioni sono poste in attesa quando non è possibile acquisire un lock, in TS uccise e rilanciate
  - Le **ripartenze** sono di solito **più costose** delle **attese**: conviene il 2PL
- 2PL può causare ***deadlock***, TS no
  - mediamente si uccide **una transazione ogni due conflitti**, ma la probabilità di insorgenza di deadlock è molto minore della probabilità di un conflitto: conviene il 2PL

2PL vs TS

- Gli schedule TS sono automaticamente CSR:
  - corrispondono a **una** esecuzione seriale (quella in cui le transazioni sono eseguite nell'ordine in cui sono iniziate)
- Ma **2PL e TS sono incomparabili**:
  - Schedule **in TS** ma **non in 2PL:**

$$r_1(x)w_1(x)r_2(x)w_2(x)r_0(y)w_1(y)$$

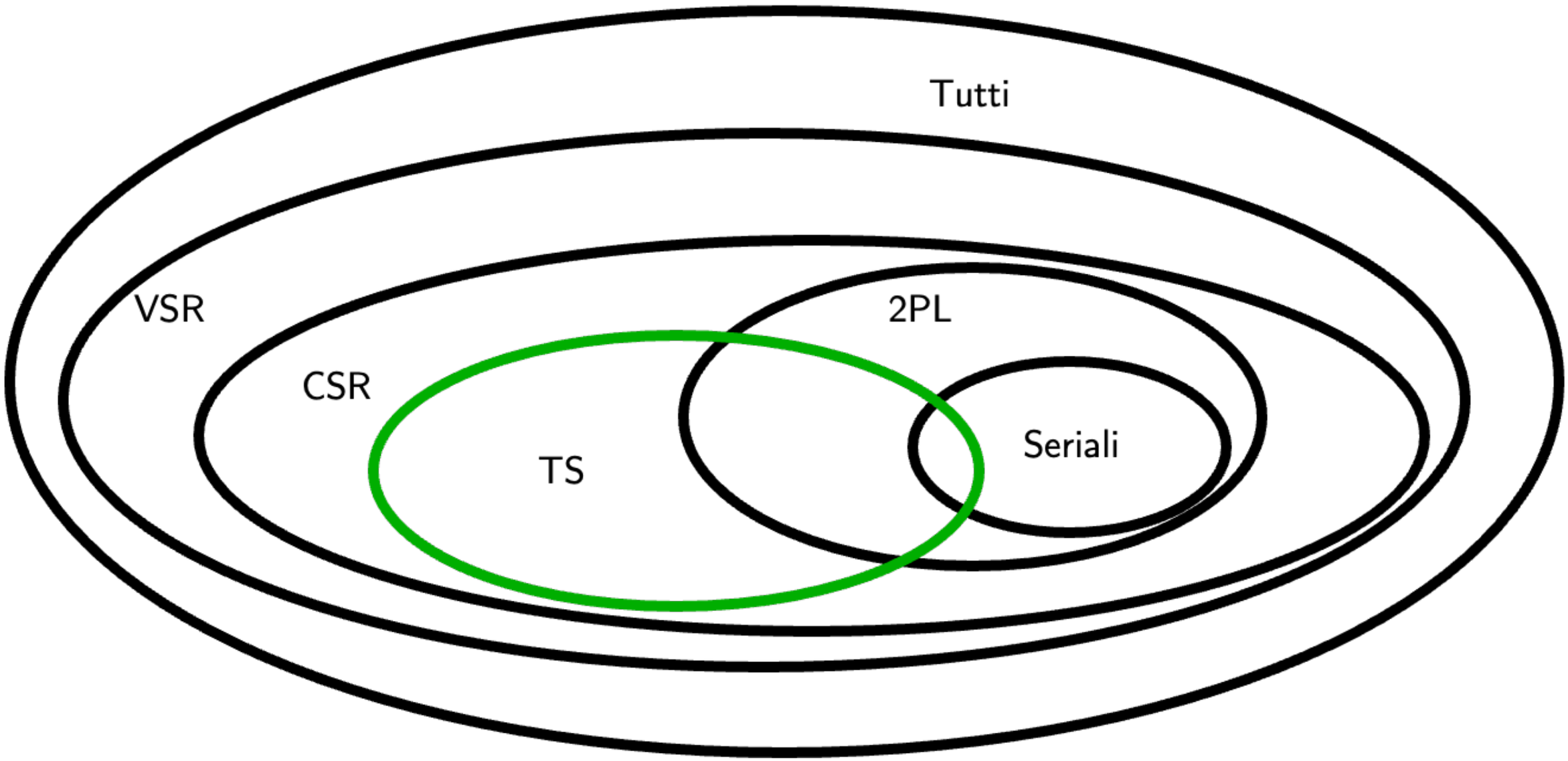
  - Schedule **in 2PL** ma **non in TS:**

$$r_2(x)w_2(x)r_1(x)w_1(x)$$

  - Schedule **in TS** e **in 2PL**:

$$r_1(x)r_2(y)w_2(y)w_1(x)r_2(x)w_2(x)$$

VSR, *CSR*, 2PL, TS



Grafo delle attese

- Se uno scheduler, come 2PL, permette dei *deadlock*, esso ha bisogno di **meccanismi per rilevare i *deadlock***
- In genere si usa il **grafo delle attese**
  - I **nodi** sono le **transazioni attive**
  - Un **arco** (T_i, T_j) indica che T_i **attende** che T_j **rilasci** un lock di cui ha bisogno
- Un **ciclo** in questo grafo corrisponde a un ***deadlock***

Tecniche di risoluzione dei *deadlock*

- **Tre tecniche** di risoluzione:
  1. ***Timeout***
     - Le transazioni rimangono in **attesa** di una risorsa **per un tempo prefissato**
     - Se, trascorso tale tempo, la **risorsa non** è ancora stata **concessa**, alla richiesta di lock viene data **risposta negativa**
     - In tal modo una transazione in potenziale stato di deadlock viene tolta dallo stato di attesa e di norma **abortita**
     - **Tecnica molto semplice**, usata dalla gran parte dei sistemi commerciali
     - **Problema**: scelta dell'intervallo
  2. **Rilevamento dello stallo**
     - **Ricerca di cicli** nel grafo delle attese
  3. **Prevenzione dello stallo**
     - **Uccisione di transazioni** "**sospette**"

Scelta del *timeout*

- Un **valore troppo elevato** tende a **risolvere tardi** i blocchi critici, dopo che le transazioni coinvolte hanno **trascorso diverso tempo in attesa**
- Un **valore troppo basso** rischia di interpretare come blocchi anche situazioni in cui una transazione sta attendendo la disponibilità di una risorsa destinata a liberarsi, **uccidendo la transazione** e **sprecando il lavoro già svolto**

Scelta della transazione da abortire

- **Politiche interrompenti**: un conflitto può essere risolto uccidendo la **transazione che possiede la risorsa** (in tal modo, essa rilascia la risorsa che può essere concessa ad un'altra transazione)
  - Criterio aggiuntivo: uccidere le transazioni che hanno svolto **meno lavoro** (si spreca meno)
- **Politiche non interrompenti**: una transazione può essere uccisa **solo nel momento** in cui effettua una **nuova richiesta**

Starvation

- Una transazione, all'inizio della propria elaborazione, accede ad un oggetto richiesto da molte altre transazioni, così è **sempre in conflitto** con altre transazioni e, essendo all'inizio del suo lavoro, viene **ripetutamente uccisa**
  - **Non** c'è ***deadlock***, **ma** ***starvation***
- **Possibile soluzione**: mantenere invariato il tempo di partenza delle transazioni abortite e fatte ripartire, dando in questo modo priorità alle **transazioni più "anziane"**

Esempio

- $S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$
- Mostrare l'esecuzione delle operazioni in S quando:
  - Si applica il 2PL stretto
  - Si applica il protocollo basato su timestamp

Esempio

$$S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$$

$$
\begin{aligned}
r_1(y) &\rightarrow RL_1(y)\\
w_3(z) &\rightarrow WL_3(z)\\
r_1(z) &\rightarrow T_1 \text{ waiting for } T_3\\
r_2(z) &\rightarrow T_2 \text{ waiting for } T_3\\
w_3(x) &\rightarrow WL_3(x)\\
r_3(y) &\rightarrow RL_3(y)\\
commit(T_3) &\rightarrow UL(y), UL(z), UL(x), \text{ release } T_1 \text{ and } T_2\\
r_1(z) &\rightarrow RL_1(z)\\
r_2(z) &\rightarrow RL_2(z)\\
w_1(x) &\rightarrow WL_1(x)\\
commit(T_1) &\rightarrow UL(z), UL(x)\\
w_2(x) &\rightarrow WL_2(x)\\
commit(T_2) &\rightarrow UL(z), UL(x)
\end{aligned}
$$

Schedule eseguito: $r_1(y)w_3(z)w_3(x)r_3(y)r_1(z)r_2(z)w_1(x)w_2(x)$

Esempio

$$S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$$

$$
\begin{array}{lcl}
r_1(y) & \rightarrow & RTM(y) = 1 \\
w_3(z) & \rightarrow & WTM(z) = 3 \\
r_1(z) & \rightarrow & \text{abort } T_1\text{, restart now as } T_4 \\
r_4(y) & \rightarrow & RTM(y) = 4 \\
r_4(z) & \rightarrow & RTM(z) = 4 \\
r_2(z) & \rightarrow & \text{abort } T_2\text{, restart now as } T_5 \\
r_5(z) & \rightarrow & RTM(z) = 5 \\
w_3(x) & \rightarrow & WTM(x) = 3 \\
w_4(x) & \rightarrow & WTM(x) = 4 \\
w_5(x) & \rightarrow & WTM(x) = 5 \\
r_3(y) & \rightarrow & RTM(y) = 4
\end{array}
$$

Esempio

$$S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$$

$$
\begin{array}{lcl}
r_1(y) & \rightarrow & RTM(y) = 1 \\
w_3(z) & \rightarrow & WTM(z) = 3 \\
r_1(z) & \rightarrow & \text{abort } T_1\text{, restart now as } T_4 \\
r_4(y) & \rightarrow & RTM(y) = 4 \\
r_4(z) & \rightarrow & RTM(z) = 4 \\
r_2(z) & \rightarrow & \text{abort } T_2\text{, restart now as } T_5 \\
r_5(z) & \rightarrow & RTM(z) = 5 \\
w_3(x) & \rightarrow & WTM(x) = 3 \\
w_4(x) & \rightarrow & WTM(x) = 4 \\
w_5(x) & \rightarrow & WTM(x) = 5 \\
r_3(y) & \rightarrow & RTM(y) = 4
\end{array}
$$

Schedule eseguito: $w_3(z)r_4(y)r_4(z)r_5(z)w_3(x)w_4(x)w_5(x)r_3(y)$