

Tecnologia dei DB server (centralizzati e distribuiti)

Enrico Bacis
enrico.bacis@unibg.it

Affidabilità e Concorrenza

- Affidabilità
 - Resistenza ai guasti
- Concorrenza
 - Efficienza: più transazioni contemporanee
 - Senza introdurre fenomeni indesiderati

Transazione

- Unità elementare di lavoro
- Ben formata: Sequenza di operazioni tra inizio (begin) e fine (commit o abort)
- Necessaria la concorrenza per avere transazioni in parallelo.

Lost update

Transazione t1

$r1(x)$

$x = x + 1$

$w1(x)$

commit

Transazione t2

$r2(x)$

$x = x + 1$

$w2(x)$

commit

Dirty read

Transazione t1

$r1(x)$

$x = x + 1$

$w1(x)$

abort

Transazione t2

$r2(x)$

commit

Lecture inconsistent

Transazione t1

$r1(x)$

$r1(x)$

commit

Transazione t2

$r2(x)$

$x = x + 1$

$w2(x)$

commit

Aggiornamento fantasma

Transazione t1

$r1(x)$

$r1(y)$

commit

Transazione t2

$r2(x)$

$r2(y)$

$x = x + 1$

$y = y - 1$

$w2(x)$

$w2(y)$

commit

A. Ricerca di anomalie

Indicare se le sequenze di operazioni possono produrre anomalie;
i simboli c_i e a_i indicano l'esito
(commit o abort) della transazione i .

A.1 $r_1(x)$ $w_1(x)$ $r_2(x)$ $w_2(y)$ a_1 c_2

Lettura Sporca:

$r_1(x)$ **$w_1(x)$** **$r_2(x)$** $w_2(y)$ **a_1** c_2

t_2 legge uno stato che non dovrebbe essere esposto, poiché t_1 si conclude con un abort.

A.2 $r_1(x)$ $w_1(x)$ $r_2(y)$ $w_2(y)$ a_1 c_2

Nessuna anomalia

A.3 $r_1(x)$ $r_2(x)$ $r_2(y)$ $w_2(y)$ $r_1(z)$ a_1 c_2

Nessuna anomalia

A.4 $r_1(x)$ $r_2(x)$ $w_2(x)$ $w_1(x)$ c_1 c_2

Perdita di aggiornamento

$r_1(x)$ $r_2(x)$ **$w_2(x)$** **$w_1(x)$** c_1 c_2

L'effetto di t_2 è sovrascritto e annullato da t_1 , ma nessuna delle due transazioni legge valori inconsistenti

A.5 $r_1(x)$ $r_2(x)$ $w_2(x)$ $r_1(y)$ c_1 c_2

Nessuna anomalia

A.6 $r_1(x)$ $w_1(x)$ $r_2(x)$ $w_2(x)$ c_1 c_2

Nessuna anomalia

Controllo di concorrenza

- Schedule: Sequenza di operazioni r/w
- Scheduler: Accetta gli schedule che non generano errori e rifiuta gli altri
- Ipotesi [forte]: sappiamo già il risultato delle operazioni e rimuoviamo gli abort (*commit-proiezione*)

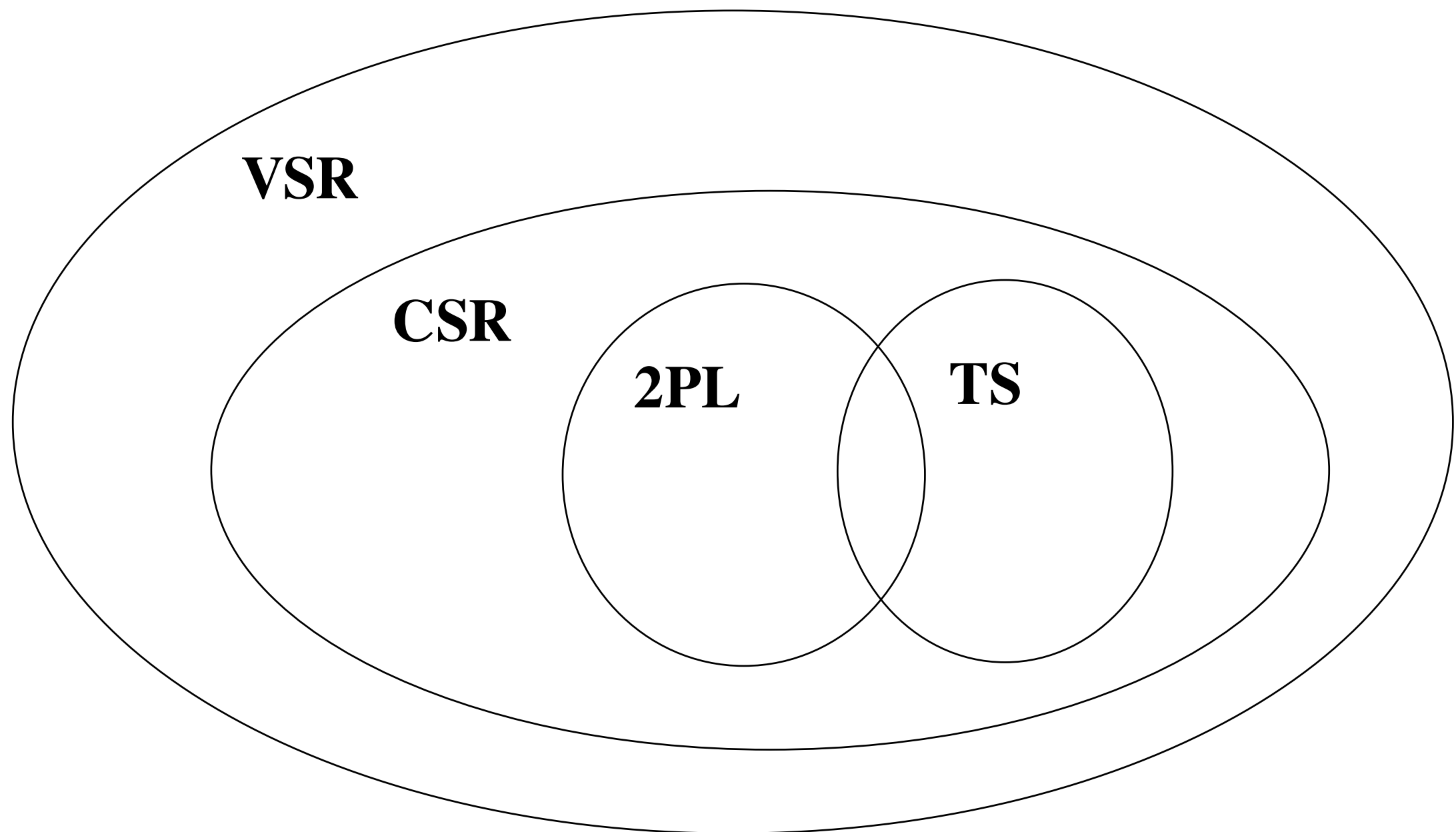
Schedule seriali e serializzabili

- Schedule seriali: azioni di ciascuna transazione in sequenza
- Schedule corretto se ha lo stesso risultato di uno schedule seriale: serializzabile.

Metodologie di controllo

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

Metodologie di controllo



View-Serializzabile VSR

Stesse **scritture finali e legge-da** di un qualsiasi altro schedule seriale.

$w_0(x)$ $r_2(x)$ $r_1(x)$ **$w_2(x)$** **$w_2(z)$**

$w_0(x)$ $r_1(x)$ $r_2(x)$ **$w_2(x)$** **$w_2(z)$**

B.1 Classificare il seguente schedule
come non-VSR, VSR o CSR:

$r_1(x)$	$r_2(y)$	$w_3(y)$	$r_5(x)$	$w_5(u)$	$w_3(s)$
$w_2(u)$	$w_3(x)$	$w_1(u)$	$r_4(y)$	$w_5(z)$	$r_5(z)$

Consideriamo le operazioni
relative a ogni risorsa
separatamente

S: w_3

U: $w_5 \quad w_2 \quad w_1$

X: $r_1 \quad r_5 \quad w_3$

Y: $r_2 \quad w_3 \quad r_4$

Z: $w_5 \quad r_5$

Costruzione Grafo Conflitti CSR

- Ogni nodo è una transazione
- Un arco orientato per ogni conflitto di tipo:
 - read / write
 - write / write
 - write / read
- I CONFLITTI SI GENERANO ANCHE SE LE AZIONI NON SONO CONSECUTIVE

Consideriamo le operazioni
relative a ogni risorsa
separatamente

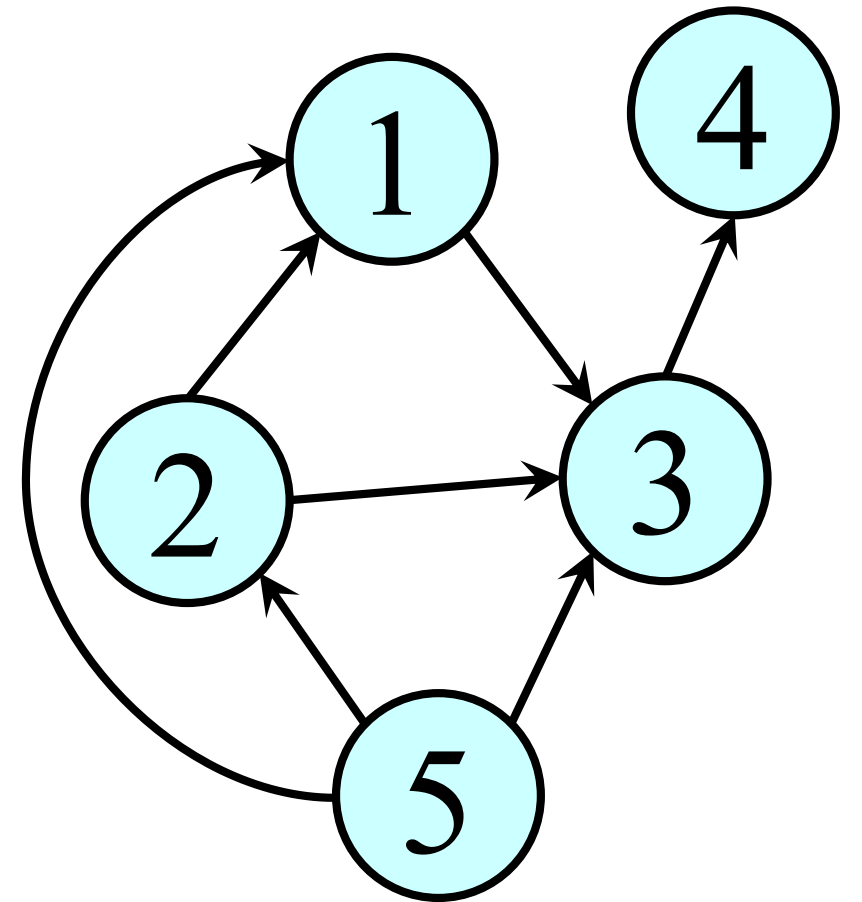
S: w_3

U: w_5 w_2 w_1

X: r_1 r_5 w_3

Y: r_2 w_3 r_4

Z: w_5 r_5



Si verifica dapprima
l'appartenenza a CSR:
il grafo dei conflitti è
aciclico: lo schedule è CSR
(e quindi anche VSR)

Supponendo di aggiungere lo schedule $r_2(u) w_2(s)$ come prefisso oppure come suffisso dello schedule precedente, classificare i due schedule risultanti.

Aggiungiamolo come prefisso:

$r_2(u) w_2(s) r_1(x) r_2(y) w_3(y) r_5(x)$
 $w_5(u) w_3(s) w_2(u) w_3(x) w_1(u) r_4(y)$
 $w_5(z) r_5(z)$

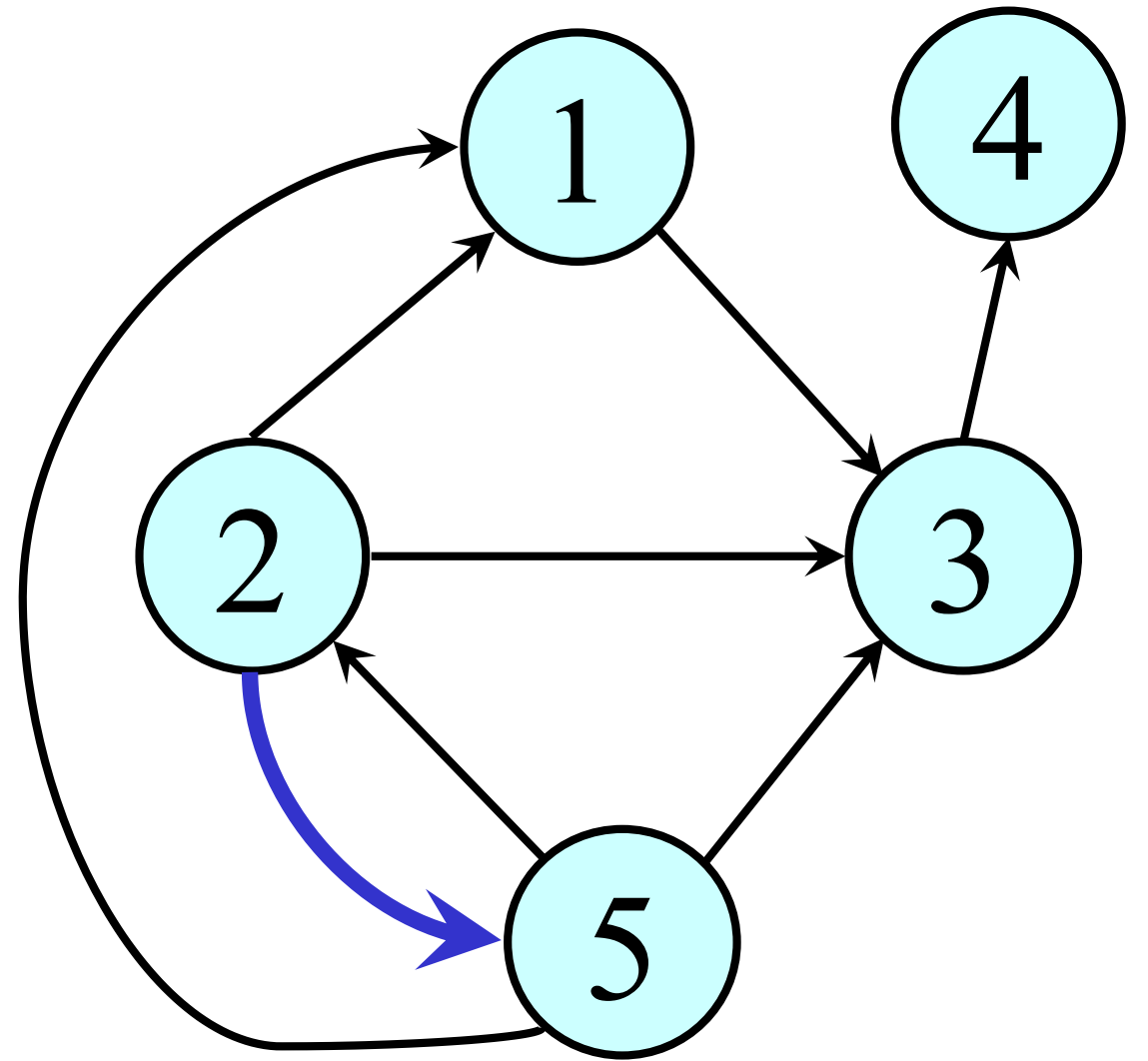
S: w_2 w_3

U: r_2 w_5 w_2 w_1

X: r_1 r_5 w_3

Y: r_2 w_3 r_4

Z: w_5 r_5



Il prefisso introduce 4 nuovi conflitti, dei quali uno solo causa un nuovo arco nel grafo, che diventa ciclico.


Lo schedule NON è CSR. Sarà VSR?

S: w_2 w_3


U: r_2 w_5 w_2 w_1

X: r_1 r_5 w_3

Y: r_2 w_3 r_4



Z: w_5 r_5



Lo schedule (seriale) $S' = t_2, t_5, t_1, t_3, t_4$
ha le stesse *legge-da* e *scritture finali*,
quindi lo schedule è VSR.

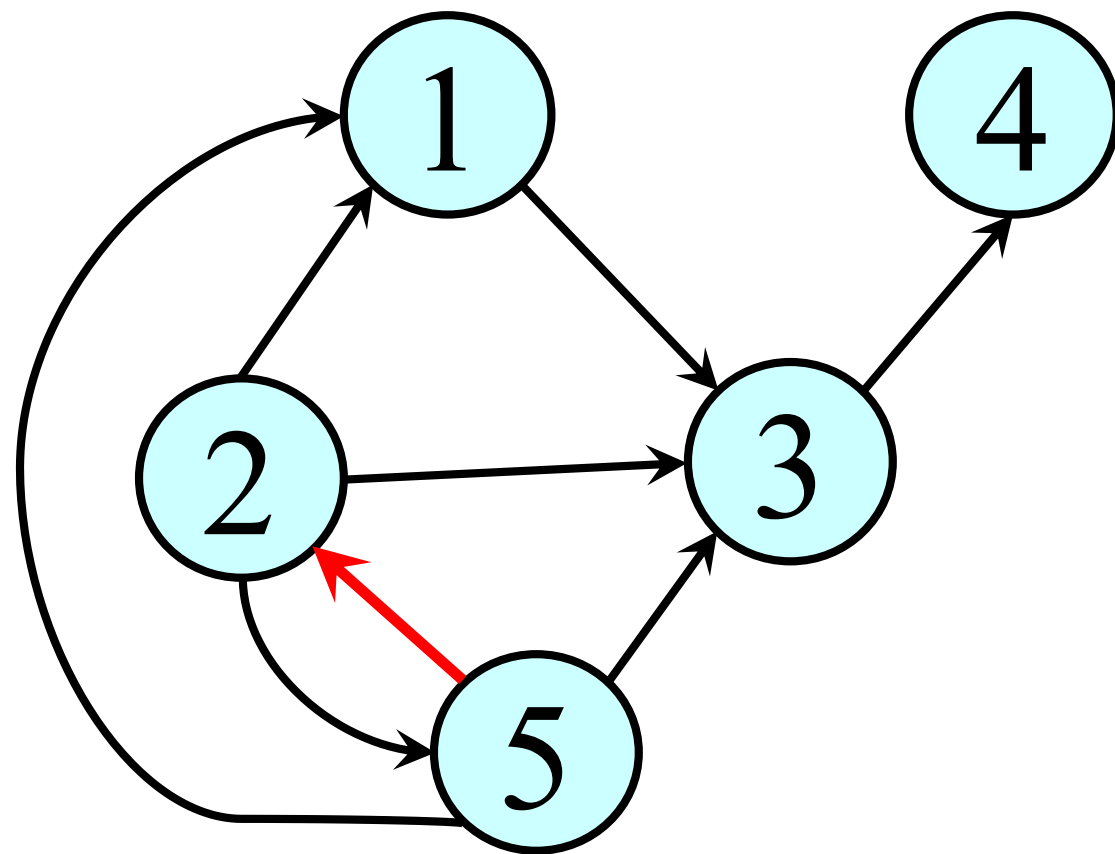
Ma come individuare S' “a occhio” ?

Definizione: una scrittura $w_i(X)$ si dice *cieca* se non è l'ultima azione sulla risorsa X e l'azione successiva su X è una scrittura $[w_j(X)]$

Proprietà: ogni schedule S tale che $S \in \text{VSR}$ ma $S \notin \text{CSR}$ ha, nel grafo dei conflitti, cicli a cui concorrono archi che corrispondono a coppie di scritture cieche.

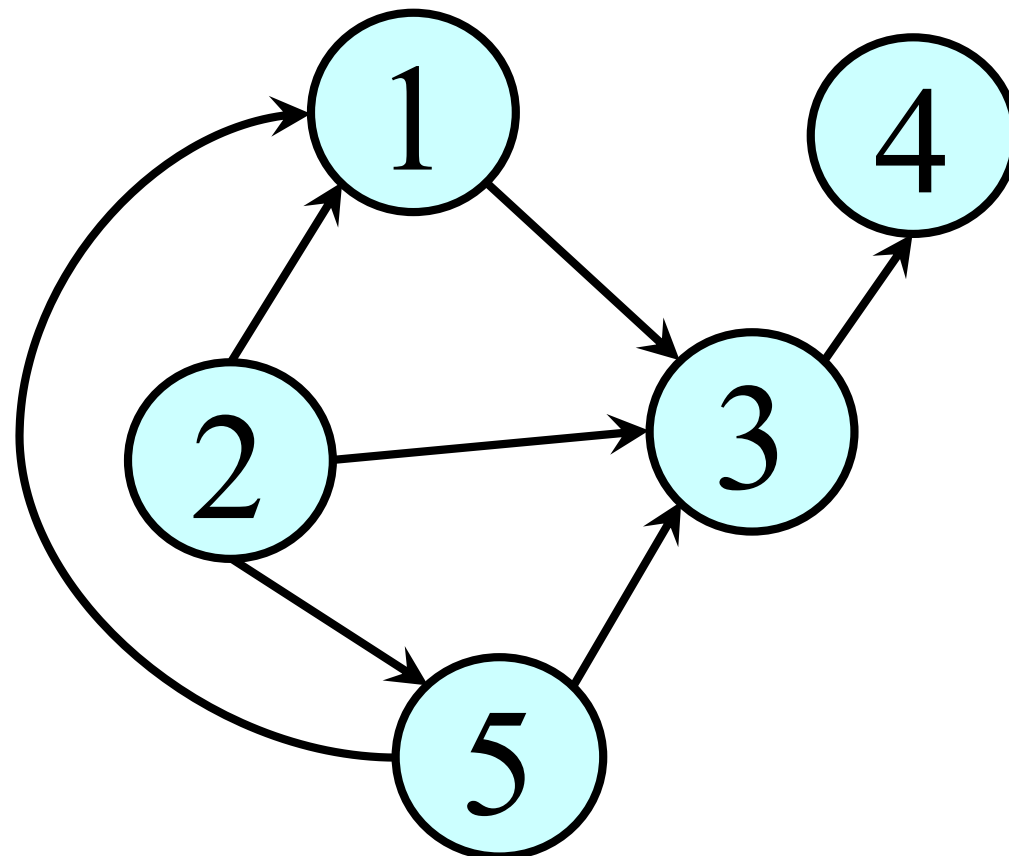
Tali scritture possono essere invertite senza impatto sulla relazione legge-da e sulle scritture finali (ma con l'effetto di invertire un arco nel grafo dei conflitti). Con questi scambi si può rendere aciclico il grafo, e si può ispezionarlo per individuare uno schedule seriale view-equivalente a quello iniziale.

S: w_2 w_3
 U: r_2 **w_5** **w_2** w_1
 X: r_1 r_5 w_3
 Y: r_2 w_3 r_4
 Z: w_5 r_5



Il grafo precedente può essere reso aciclico operando uno scambio tra due *scritture cieche*

S: w_2 w_3
 U: r_2 **w_2** **w_5** w_1
 X: r_1 r_5 w_3
 Y: r_2 w_3 r_4
 Z: w_5 r_5



Lo schedule seriale $S' = t_2, t_5, t_1, t_3, t_4$,
 indicato in precedenza, si può
 riconoscere corrispondente ad un
 cammino che tocca tutti i nodi del grafo
 dei conflitti dello schedule modificato.

Ora mettiamo $r_2(u) w_2(s)$ come suffisso:

$r_1(x) \quad r_2(y) \quad w_3(y) \quad r_5(x) \quad w_5(u) \quad w_3(s)$
 $w_2(u) \quad w_3(x) \quad w_1(u) \quad r_4(y) \quad w_5(z) \quad r_5(z)$
 $r_2(u) \quad w_2(s)$

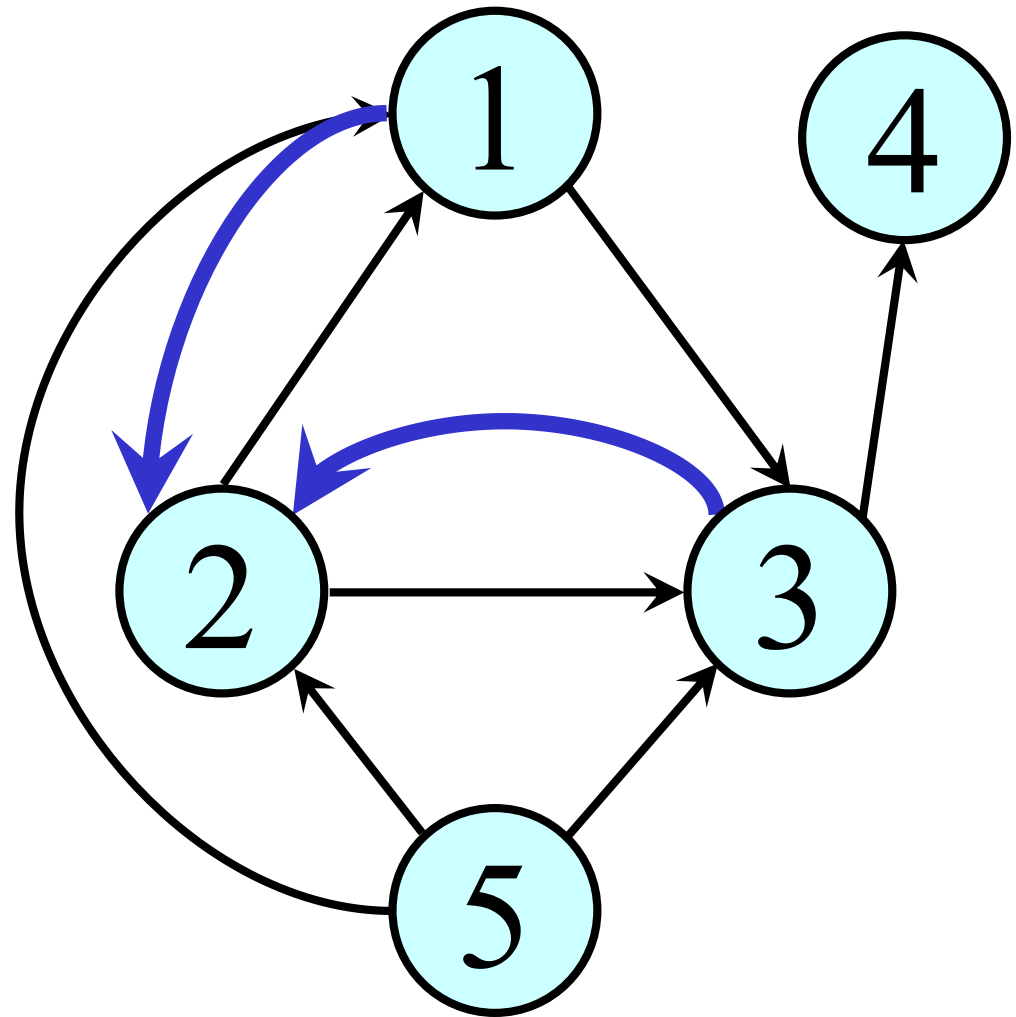
S: w_3 w_2

U: w_5 w_2 w_1 r_2

X: r_1 r_5 w_3

Y: r_2 w_3 r_4

Z: w_5 r_5



Il nuovo grafo è ciclico:
lo schedule NON è CSR
Sarà VSR?

S: w_3 w_2

U: w_5 w_2 w_1 r_2

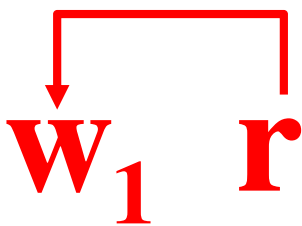
X: r_1 r_5 w_3

Y: r_2 w_3 r_4

Z: w_5 r_5

Non è VSR – infatti è piuttosto evidente che non è possibile decidere se t_2 debba precedere o seguire t_3 : nel primo caso $w_2(s)$ non sarebbe più finale, nel secondo caso si introdurrebbe una relazione $r_2(y)$ *legge-da* $w_3(y)$ non presente nello schedule da classificare.

S: w_3 w_2
 U: w_5 w_2 w_1 r_2
 X: r_1 r_5 w_3
 Y: r_2 w_3 r_4
 Z: w_5 r_5



Oppure basta considerare il sotto-schedule
 $w_2(u)$ $w_1(u)$ $r_2(u)$: nessuno schedule seriale
 potrà mai preservare la relazione $r_2(u)$ *legge-*
da $w_1(u)$ evidenziata dalla freccia, poiché
 ogni schedule seriale avrà $w_2(u)$ e $r_2(u)$
 adiacenti.

B.2 Classificare il seguente schedule
come non-VSR, VSR o CSR:

$r_1(x)$ $r_4(x)$ $w_4(x)$ $r_1(y)$ $r_4(z)$ $w_4(z)$
 $w_3(y)$ $w_3(z)$ $w_2(t)$ $w_2(z)$ $w_1(t)$ $w_5(t)$

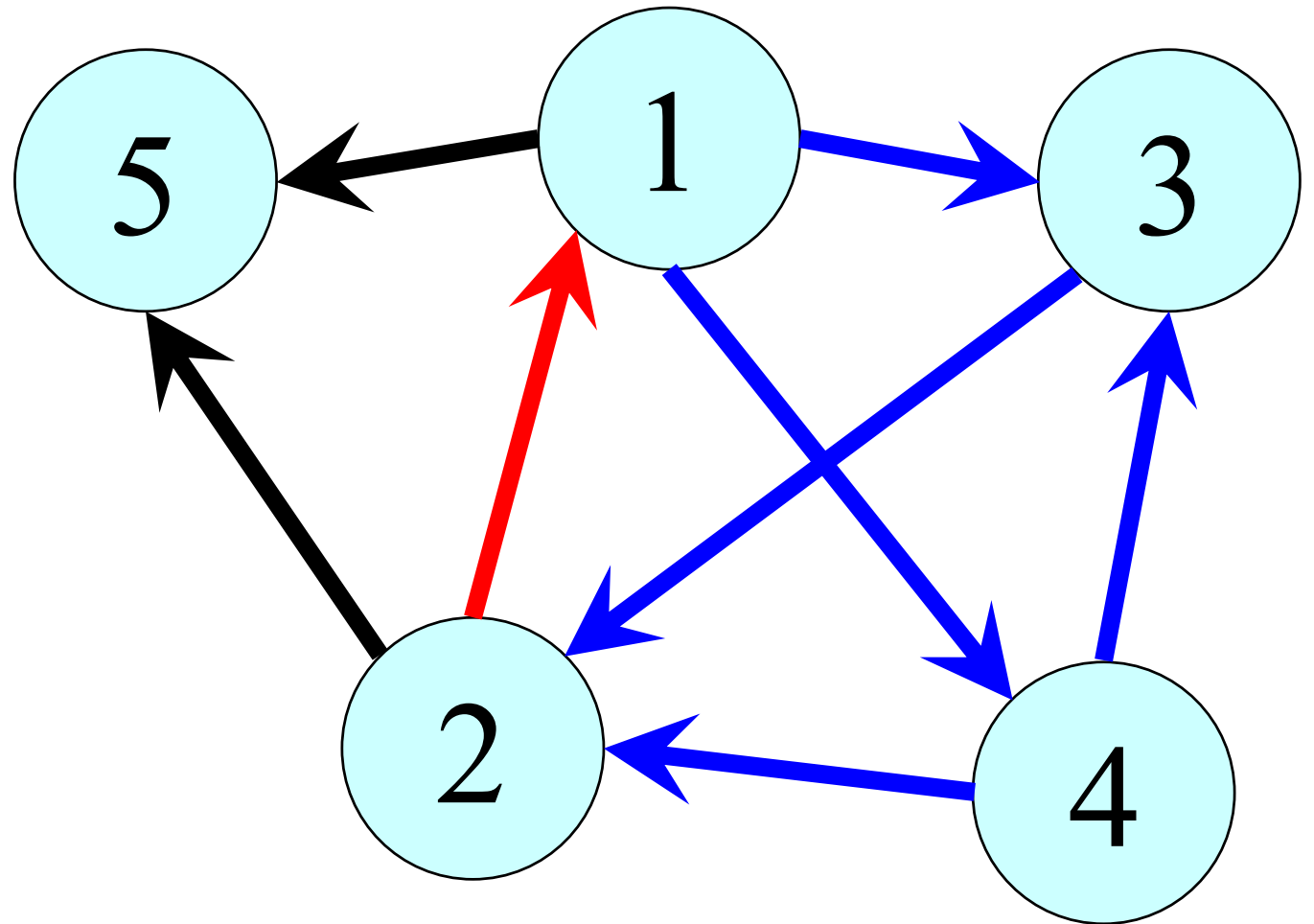
Se possibile, aggiungere e togliere una
azione in modo da far cambiare classe
allo schedule.

T: w_2 w_1 w_5

X: r_1 r_4 w_4

Y: r_1 w_3

Z: r_4 w_4 w_3 w_2



Il grafo è ciclico $(1,4,2 - 1,3,2 - 1,4,3,2)$.
 $2,1$ è l'unico arco comune a tutti e tre i cicli

Lo schedule non è CSR – sarà VSR?

T: w_2 w_1 w_5

X: r_1 r_4 w_4

Y: r_1 w_3

Z: r_4 w_4 w_3 w_2

Si riconosce agevolmente che lo schedule (seriale) t_1, t_4, t_3, t_2, t_5 è view-equivalente allo schedule dato, che quindi è VSR.

Lo schedule seriale si intuisce “a occhio”, oppure si costruisce, ad esempio, notando che lo scambio tra $w_2(t)$ e $w_1(t)$ rende aciclico il grafo dei conflitti.

Togliendo l'ultima operazione dello schedule $[w_5(t)]$ si ottiene l'effetto di renderlo non-VSR (infatti $w_1(t)$ diventa finale per t , e non si può più agire sul grafo per derivarne uno aciclico, relativo a uno schedule diverso ma view-equivalente a quello dato).

Alternativamente si può aggiungere, ad esempio, $r_5(t)$ subito prima di $w_1(t)$, ottenendo lo stesso effetto.

Togliendo $w_1(t)$, invece, lo schedule diventa CSR.

B.3 Classificare il seguente schedule come non-VSR, VSR o CSR:

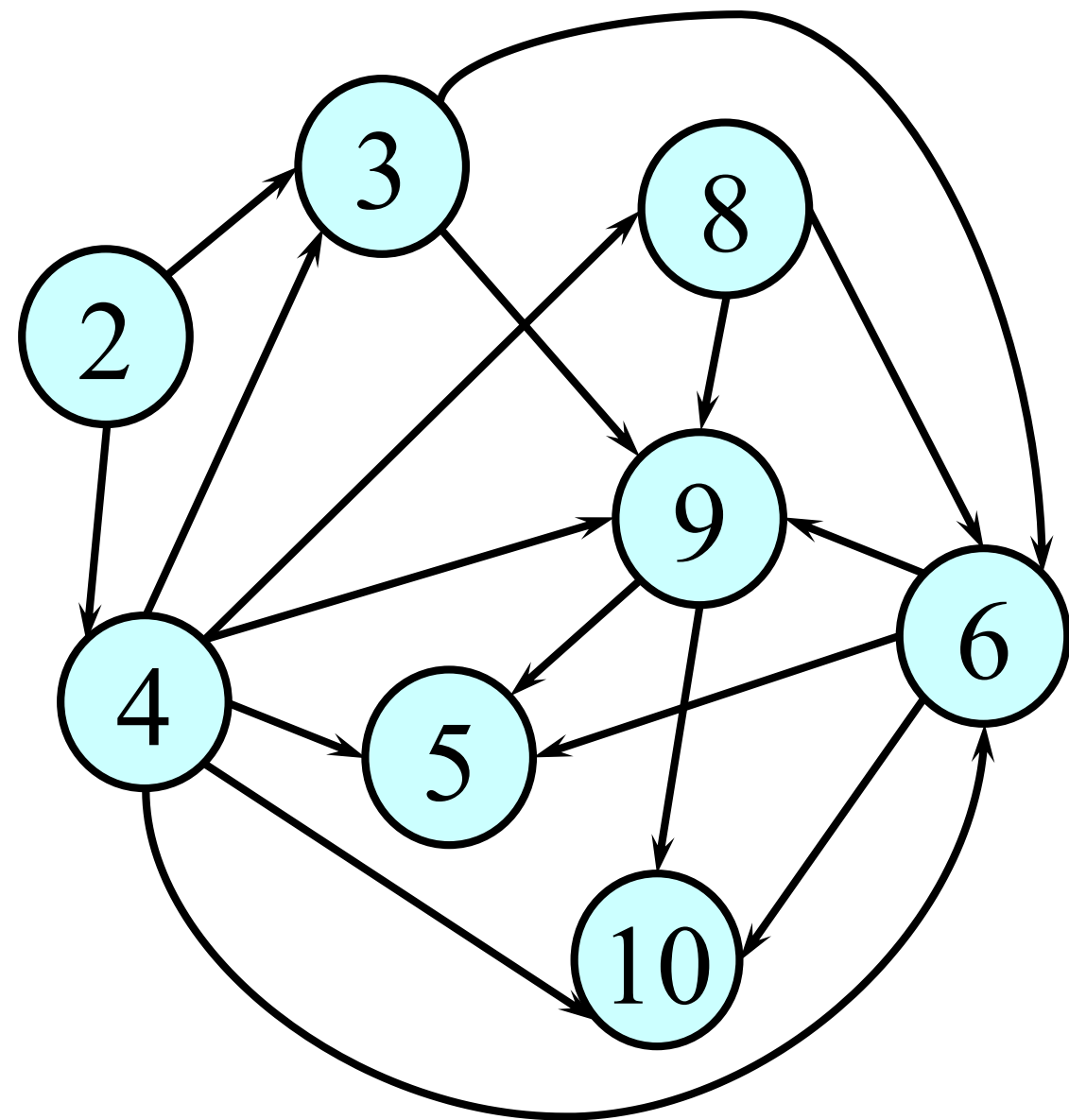
$r_4(x)$ $r_2(x)$ $w_4(x)$ $w_2(y)$ $w_4(y)$
 $r_3(y)$ $w_3(x)$ $w_4(z)$ $r_3(z)$ $r_6(z)$ $r_8(z)$
 $w_6(z)$ $w_9(z)$ $r_5(z)$ $r_{10}(z)$

Costruiamo il grafo dei conflitti

X: r_4 r_2 w_4 w_3

Y: w_2 w_4 r_3

Z: w_4 r_3 r_6 r_8 w_6 w_9 r_5 r_{10}



Il grafo è aciclico: lo schedule è CSR (e quindi anche VSR). Il grafo suggerisce lo schedule seriale 2,4,3,8,6,9,5,10 (costruito scegliendo ed eliminando via via i nodi senza archi entranti).

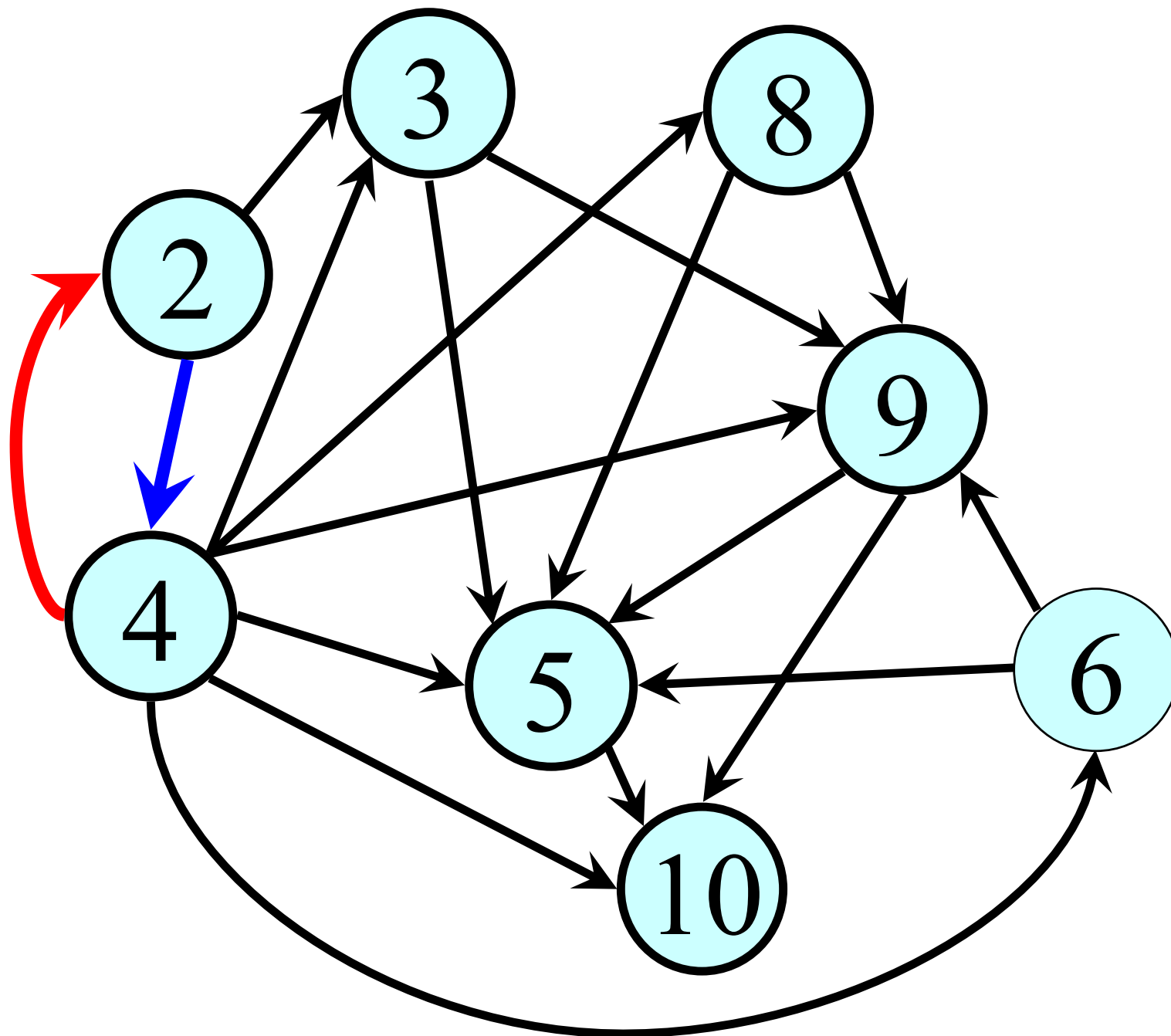
B.4 Classificare il seguente schedule
come non-VSR, VSR o CSR:

$w_4(x)$	$r_2(x)$	$w_2(y)$	$w_4(y)$	$w_3(x)$	$w_4(z)$
$r_3(z)$	$r_6(z)$	$r_8(z)$	$w_9(z)$	$w_5(z)$	$r_{10}(z)$

Costruiamo il grafo dei conflitti

X: w_4 r_2 w_3 Y: w_2 w_4

Z: w_4 r_3 r_6 r_8 w_9 w_5 r_{10}



Il grafo è ciclico
(4, 2): lo schedule
non è CSR.

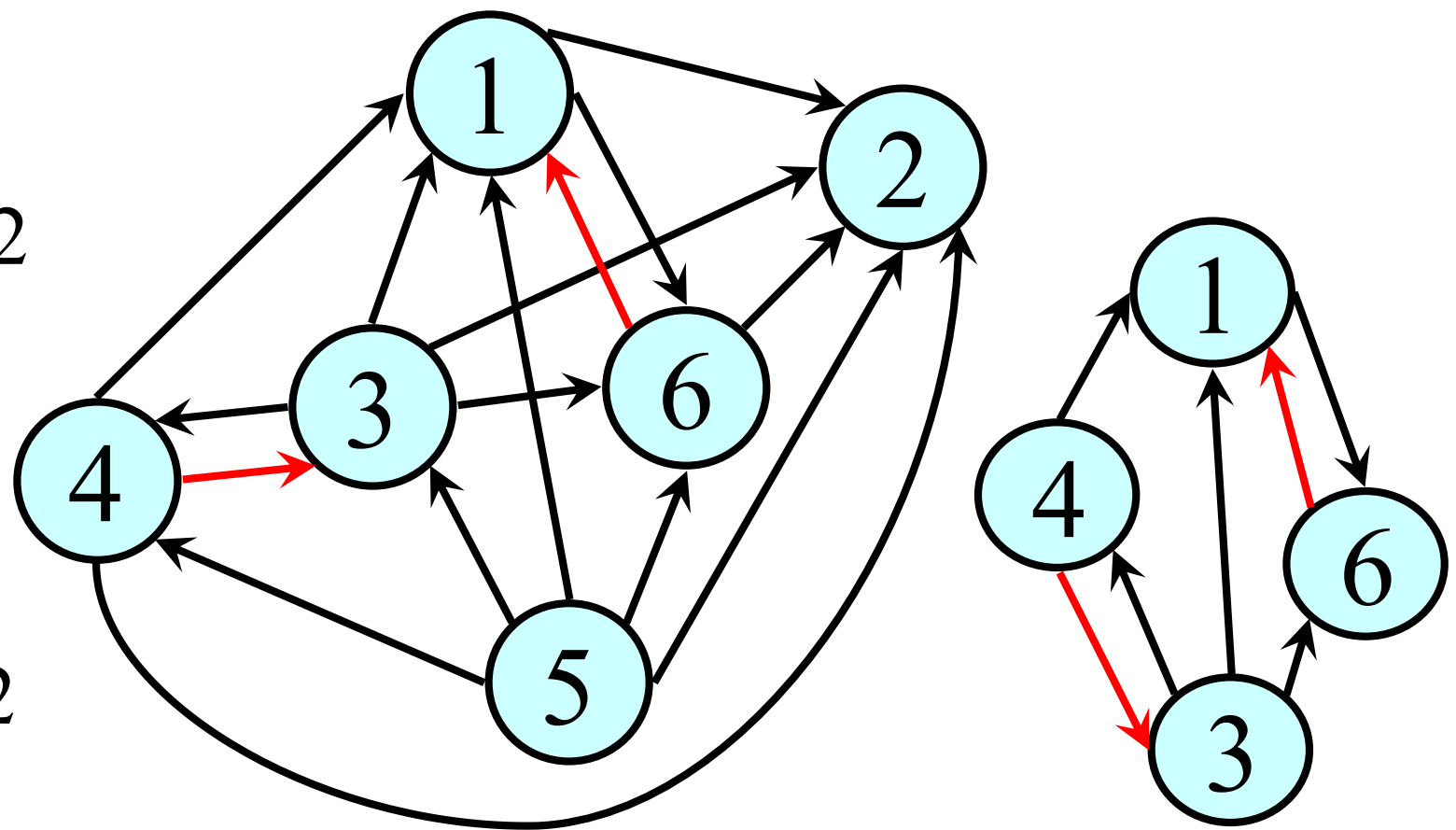
Non è nemmeno
VSR (2 e 4 non
sono serializzabili:
 $r_2(x)$ legge-da
 $w_4(x)$, ma $w_4(y)$ è
finale).

Del resto non
esistono scritture
cieche scambiabili. 53

B.5 Classificare il seguente schedule
come non-VSR, VSR o CSR:

$r_5(x) \quad r_3(y) \quad w_3(y) \quad r_6(t) \quad r_5(t) \quad w_5(z) \quad w_4(x)$
 $r_3(z) \quad w_1(y) \quad r_6(y) \quad w_6(t) \quad w_4(z) \quad w_1(t)$
 $w_3(x) \quad w_1(x) \quad r_1(z) \quad w_2(t) \quad w_2(z)$

$T: \mathbf{r}_6 \ r_5 \ \mathbf{w}_6 \ \mathbf{w}_1 \ w_2$
 $X: r_5 \ \mathbf{w}_4 \ \mathbf{w}_3 \ w_1$
 $Y: r_3 \ \mathbf{w}_3 \ w_1 \ r_6$
 $Z: w_5 \ r_3 \ w_4 \ r_1 \ w_2$



Lo schedule non è CSR: sfrondando il grafo dai nodi 2 e 5 (che hanno solo archi in ingresso e in uscita rispettivamente) si vede bene che ci sono due cicli (6-1 e 4-3). Il conflitto 4-3 può essere rimosso se si considera lo schedule view-equivalente a quello dato ottenibile per scambio di $w_4(x)$ e $w_3(x)$. Non è invece eliminabile il conflitto 6-1: lo scambio di $w_6(t)$ e $w_1(t)$ **non elimina il conflitto dovuto a $\mathbf{r}_6(t)$** . Quindi lo schedule non appartiene neanche a VSR.

C.1

Verificare se la seguente sequenza di operazioni è **compatibile** con il protocollo di **locking a due fasi**:

$r_1(A)$ $r_2(B)$ $w_1(C)$ $r_2(A)$ $r_1(B)$ $w_2(C)$
 $r_3(C)$ $w_2(B)$ $r_3(B)$ $w_1(A)$ $w_3(A)$

- Vincoli temporali sulle richieste di lock/unlock
(quando si rendano necessarie delle attese)
- Numeriamo progressivamente gli istanti in cui avvengono le *azioni*

Otteniamo un sistema di disequazioni

→ Approccio “necessario”: il 2PL non strict non permette di conoscere con precisione i momenti di rilascio (è invece possibile con il 2PL strict)

Istante	A	B	C
1	r1		
2		r2	
3			w1
4	r2		
5		r1	
6			w2
7			r3
8		w2	
9		r3	
10	w1		
11	w3		

Diagrammiamo
lo schedule
separando le
azioni in base
alla risorsa a cui
si riferiscono

Istante	A	B	C
1	r1		
2		r2	
3			w1
4	r2		
5		r1	
6			w2
7			r3
8		w2	
9		r3	
10	w1		
11	w3		

Dal comportamento di t_2
derivano i seguenti vincoli

$$4 < U^{r_2}_2^A < L^{w_1}_1^A$$

$$U^{r_1}_1^B < L^{w_2}_2^B < 8$$

Interpretiamoli...

Istante	A	B	C
1	r1		
2		r2	
3			w1
4	r2		
5		r1	
6			w2
7		r3	
8		w2	
9		r3	
10	w1		
11	w3		

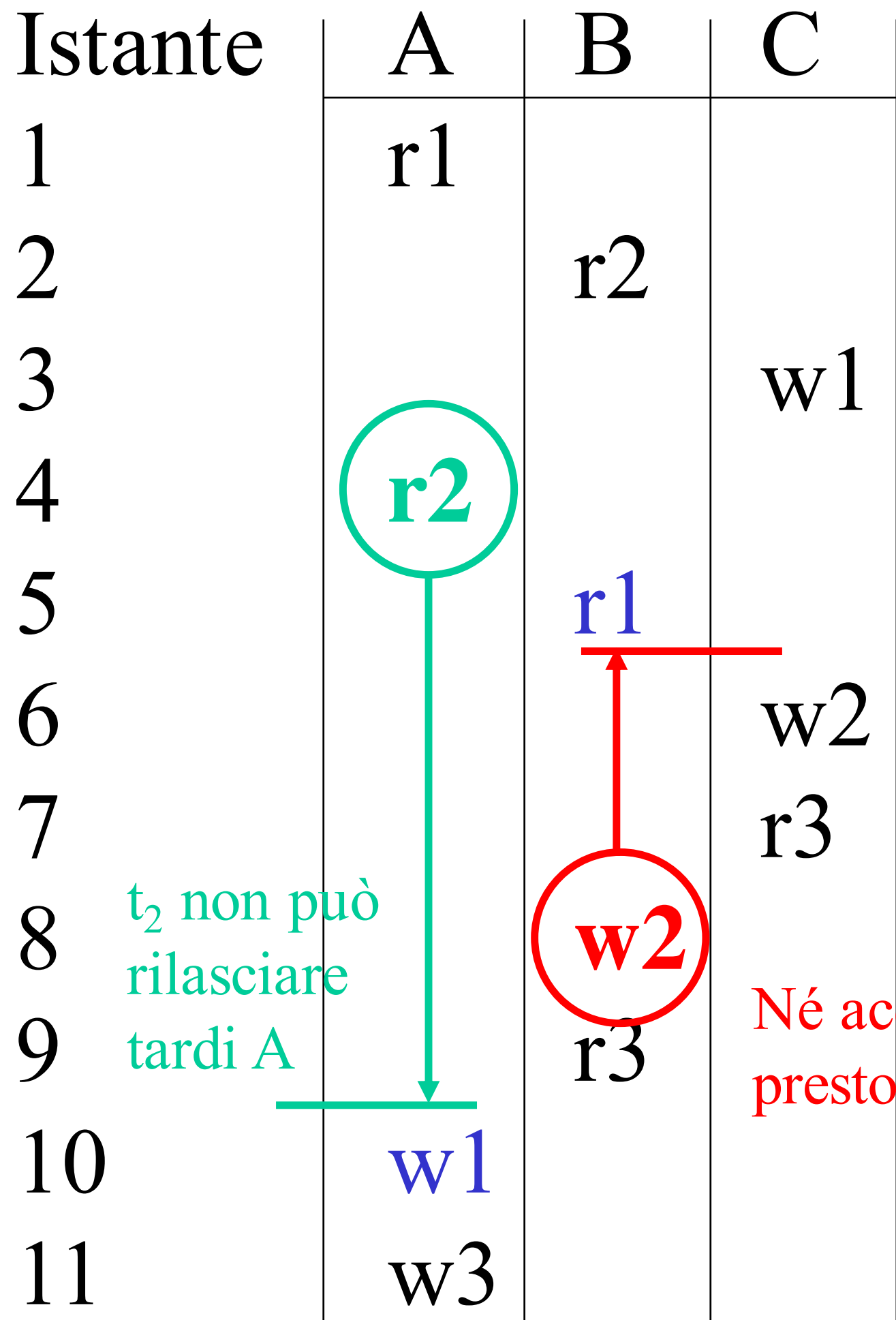
Dal comportamento di t_2
derivano i seguenti vincoli

$$4 < U_{r_2}^A < L_{w_1}^A$$

$$U_{r_1}^B < L_{w_2}^B < 8$$

t_1 non può
acquisire
presto A

Né rilasciare
tardi B



Dal comportamento di t_2 derivano i seguenti vincoli

$$4 < \mathbf{U}^{\mathbf{r}_2^A} < \mathbf{L}^{\mathbf{w}_1^A}$$

$$\mathbf{U}^{\mathbf{r}_1^B} < \mathbf{L}^{\mathbf{w}_2^B} < 8$$

Istante	A	B	C
1	r1		
2		r2	
3			w1
4	r2		
5		r1	
6			w2
7			r3
8		w2	
9		r3	
10	w1		
11	w3		

$$4 < U_2^r{}^A < L_1^w{}^A$$

$$U_1^r{}^B < L_2^w{}^B < 8$$

Inoltre il 2PL impone di **non acquisire** alcun lock **dopo il primo rilascio:**

$$L_1^w{}^A < U_1^r{}^B$$

$$L_2^w{}^B < U_2^r{}^A$$

Componendo i vincoli:

$$U^A < L^A < U^B < L^B < U^A$$

Non esiste alcun assegnamento che li soddisfi: lo schedule **non** è compatibile con il 2PL.

Istante	A	B	C
1	r1		
2		r2	
3			w1
4	r2		
5		r1	
6			w2
7			r3
8		w2	
9		r3	
10	w1		
11	w3		

DEL RESTO
LO SCHEDULE
NON È NEMMENO
SERIALIZZABILE !

Sia iniziando con t_1
sia iniziando con t_2 si
introducono legge-da
non presenti nello
schedule iniziale.

C.2

Verificare se la seguente sequenza di operazioni è compatibile con il protocollo di locking a due fasi:

$r_1(A), r_2(B), w_1(C), r_2(A), r_1(B), w_2(C),$
 $r_3(C), w_2(B), r_3(B), w_2(A), w_3(A)$

Istante	A	B	C
1	r1		
2		r2	
3			w1
4	r2		
5		r1	
6			w2
7			r3
8		w2	
9		r3	
10	w2		
11	w3		

t_1 può acquisire tutti i lock all'inizio e rilasciare ogni risorsa subito dopo l'uso.
 t_3 può acquisire ogni risorsa subito prima dell'uso e rilasciarle tutte alla fine.
 t_2 deve rilasciare (C) già tra 6 e 7, e non può acquisire (B) prima di 6, ma questo è possibile.

Timestamp Mono-version

- The scheduler has two counters: $RTM(x)$ and $WTM(x)$ for each object
- The scheduler receives read and write requests with timestamps:
 - $read(x, ts)$:
 - If $ts < WTM(x)$ the request is rejected and the transaction killed
 - Else, the request is granted and $RTM(x)$ is set to $\max(RTM(x), ts)$
 - $write(x, ts)$:
 - If $ts < WTM(x)$ or $ts < RTM(x)$ the request is rejected and the transaction killed
 - Else, the request is granted and $WTM(x)$ is set to ts
- Many transactions are killed
- To work w/o the commit-projection hypothesis, it needs to "buffer" write operations until commit, which introduces waits

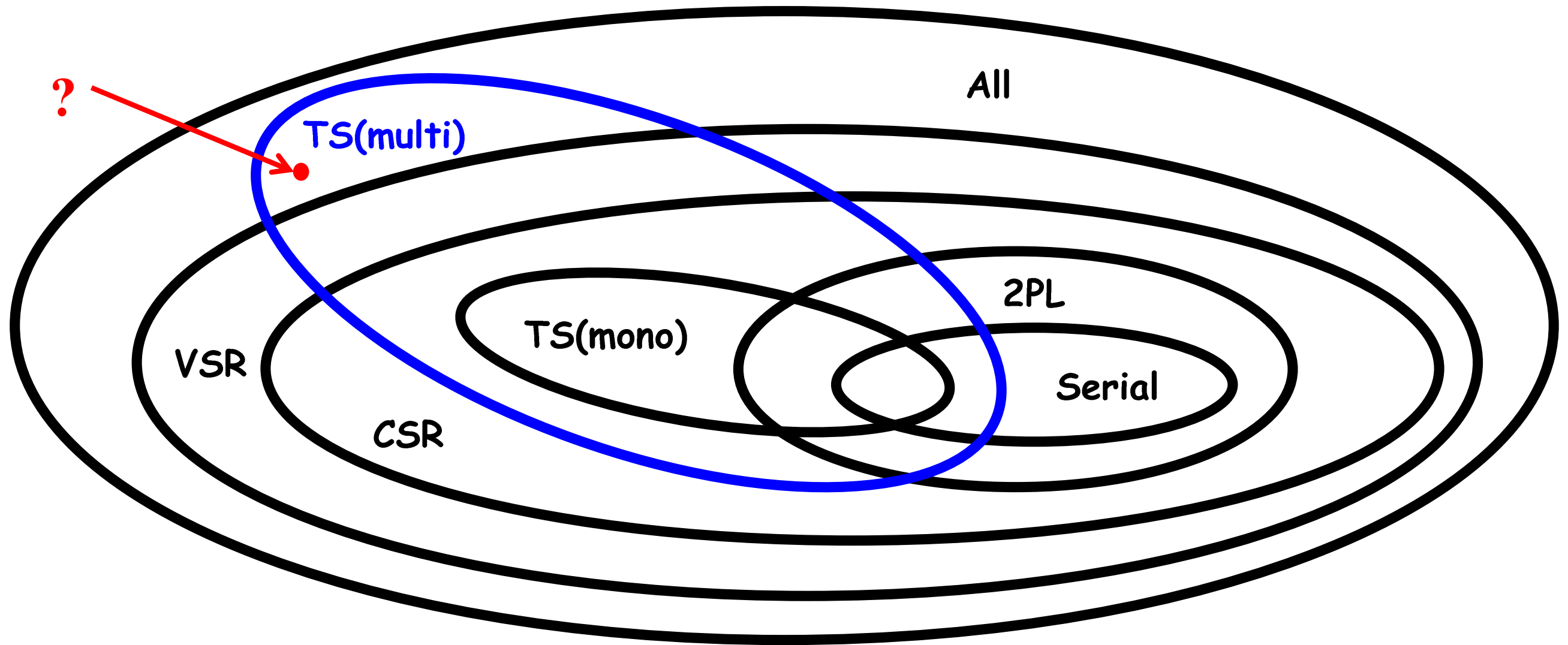
Timestamp Multi-version

- Idea: writes generate new copies, reads access the "right" copy
 - The one they would see if serially executed in timestamp order
 - Reads are **always** accepted
- Writes generate new copies, each one with a new WTM. Each object x always has $N > 1$ active copies with $WTM_N(x)$. There is a unique global RTM(x)

Timestamp Multi-version

- **Rules:**

- *read*(x, ts) is always accepted. A copy x_k is selected for reading such that:
 - If $ts > WTM_N(x)$, then $k = N$
 - Else take k such that $WTM_k(x) < ts < WTM_{k+1}(x)$
- *write*(x, ts):
 - If $ts < RTM(x)$ the request is rejected
 - Else a new version is created (N is incremented) with $WTM_N(x) = ts$



A schedule that is TS-multi but NOT VSR?

w1(X) w2(X) r1(X)

C.3 Si consideri il seguente schedule, in cui l'identificativo di una transazione rappresenta il timestamp della stessa:

$r_4(x)$ $r_2(x)$ $w_4(x)$ $w_2(y)$ $w_4(y)$ $r_3(y)$ $w_3(x)$
 $w_4(z)$ $r_3(z)$ $r_6(z)$ $r_8(z)$ $w_6(z)$ $w_9(z)$ $r_5(z)$ $r_{10}(z)$

Si assuma inizialmente $RTM = WTM = 0$ per ogni risorsa. Discutere il comportamento del sistema con controllo di concorrenza basato su timestamp mono-versione (a) e multi-versione (b). Infine, si classifichi lo schedule.

Rappresentiamo l'evoluzione del sistema con una tabella che mostri:

(a) – monoversione: Il valore dei due indicatori associati a ciascuna risorsa e le transazioni uccise

(b) – multiversione: Il valore di RTM e tutti i valori di WTM_i di ogni risorsa, oltre alle transazioni uccise. Per ogni lettura indichiamo anche (tra parentesi) il timestamp della versione effettivamente letta.

	X		Y		Z		T uccisa
	RTM	WTM	RTM	WTM	RTM	WTM	
R4(x)	4	0	0	0	0	0	
R2(x)	4						
W4(x)		4					
W2(y)				2			
W4(y)				4			
R3(y)							3 ($3 < \text{WTM}(y)=4$)
W3(x)							(già uccisa)
W4(z)						4	
R3(z)							(già uccisa)
R6(z)					6		
R8(z)					8		
W6(z)							6 ($6 < \text{RTM}(z)=8$)
W9(z)						9	
R5(z)							5 ($5 < \text{WTM}(z)=9$)
R10(z)					10		70

	X		Y		Z		T uccisa
	RTM	WTM(i)	RTM	WTM(i)	RTM	WTM(i)	
R4(x)	4 (0)	0	0	0	0	0	
R2(x)	4 (0)						
W4(x)		0, 4					
W2(y)				0, 2			
W4(y)				0, 2, 4			
R3(y)			3 (2)				
W3(x)							3 (3 < RTM(x)=4)
W4(z)						0, 4	
R3(z)							(già uccisa)
R6(z)					6 (4)		
R8(z)					8 (4)		
W6(z)							6 (6 < RTM(z)=8)
W9(z)						0, 4, 9	
R5(z)					8 (4)		
R10(z)					10(9)		71

C.4 Si consideri il seguente schedule, in cui l'identificativo di una transazione rappresenta il timestamp della stessa:

$w_4(x)$ $r_2(x)$ $w_2(y)$ $w_4(y)$ $w_3(x)$ $w_4(z)$ $r_3(z)$
 $r_6(z)$ $r_8(z)$ $w_9(z)$ $w_5(z)$ $r_{10}(z)$

Si assuma inizialmente $RTM = WTM = 0$ per ogni oggetto.

Discutere il comportamento del sistema con controllo di concorrenza basato su timestamp mono-versione e multi-versione.

	X		Y		Z		T uccisa
	RTM	WTM	RTM	WTM	RTM	WTM	
W4(x)	0	4	0	0	0	0	
R2(x)							2 ($2 < \text{WTM}(x)=4$)
W2(y)							(già uccisa)
W4(y)				4			
W3(x)							3 ($3 < \text{WTM}(x)=4$)
W4(z)						4	
R3(z)							(già uccisa)
R6(z)					6		
R8(z)					8		
W9(z)						9	
W5(z)							5 ($5 < \text{WTM}(z)=9$)
R10(z)					10		

	X		Y		Z		T uccisa
	RTM	WTM _i	RTM	WTM _i	RTM	WTM _i	
W4(x)	0	0, 4	0	0	0	0	
R2(x)	2 (0)						
W2(y)				0, 2			
W4(y)				0, 2, 4			
W3(x)		0, 3 , 4	[un sistema reale probabilmente ucciderebbe t3]				
W4(z)						0, 4	
R3(z)					3 (0)		
R6(z)					6 (4)		
R8(z)					8 (4)		
W9(z)						0, 4, 9	
W5(z)							5 (5< RTM(z)=8)
R10(z)					10(9)		