

# Data Bases II

## Schedules and Concurrency Control

Michele Beretta

[michele.beretta@unibg.it](mailto:michele.beretta@unibg.it)



## Exercise S.1

Classify the following schedule (CSR/VSR)

$$r_1(X)r_4(X)w_4(X)r_1(Y)r_4(Z)w_4(Z)w_3(Y) \\ w_3(Z)w_2(T)w_2(Z)w_1(T)w_5(T)$$

If possible, add/remove/move one action in order to change the class of the schedule.

How to check if a schedule is CSR:

1. Split operations by resource
2. Identify conflicts
  - $w_x \dots w_y \Rightarrow x$  conflicts with  $y$
  - $r_x \dots w_y \Rightarrow x$  conflicts with  $y$
  - $w_x \dots r_y \Rightarrow x$  conflicts with  $y$
3. Build the conflict graph
4. The schedule is CSR if and only if there are no cycles

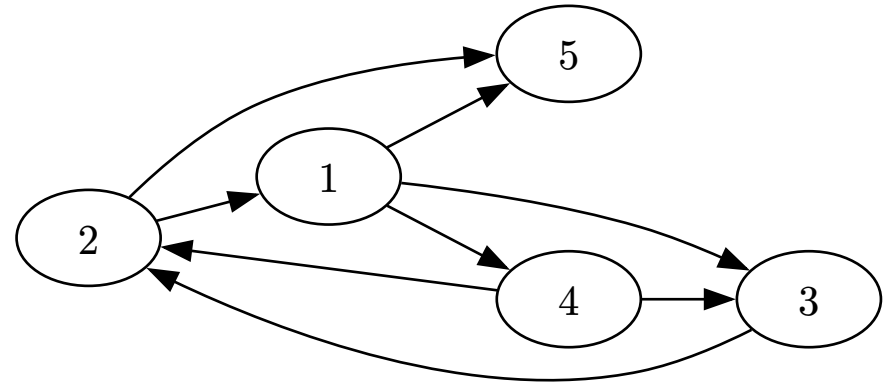
$$r_1(X)r_4(X)w_4(X)r_1(Y)r_4(Z)w_4(Z)w_3(Y)w_3(Z)w_2(T)w_2(Z)w_1(T)w_5(T)$$

**T:**  $w_2w_1w_5$

**X:**  $r_1r_4w_4$

**Y:**  $r_1w_3$

**Z:**  $r_4w_4w_3w_2$



We can see there is a cycle (more than one).

Hence, the schedule is **not** CSR. It could be VSR.

How to check if a schedule is VSR:

1. Split operations by resource
2. Try and find a *serial* schedule that has
  - the same *read-from* relations ( $w_x \dots r_y \Rightarrow y$  reads from  $x$ )
  - the same *last writes*
3. If found, the schedule is VSR

Alternatively, build a graph showing all the necessary relations

1. *read-from* relations
2. *last writes* relations
3. moreover,  $r_x \dots w_y$  means  $y$  must come after  $x$ , otherwise a new *read-from* relation would be born out of nowhere

## Method 1

$$r_1(X)r_4(X)w_4(X)r_1(Y)r_4(Z)w_4(Z)w_3(Y)w_3(Z)w_2(T)w_2(Z)w_1(T)w_5(T)$$

T:  $w_2w_1w_5$

X:  $r_1r_4w_4$

Y:  $r_1w_3$

Z:  $r_4w_4w_3w_2$

It is VSR:  $t_1, t_4, t_3, t_2, t_5$  is view-equivalent.

There are no read-froms. Last-writes are immediate (e.g.,  $t_5$  must follow  $t_1$  and  $t_2$  for T, and so on).

**Note:** we can swap  $w_2$  and  $w_1$  in T without modifying anything. This is because  $w_5$  always overwrites T, and there are no read-froms on T.

## Method 2

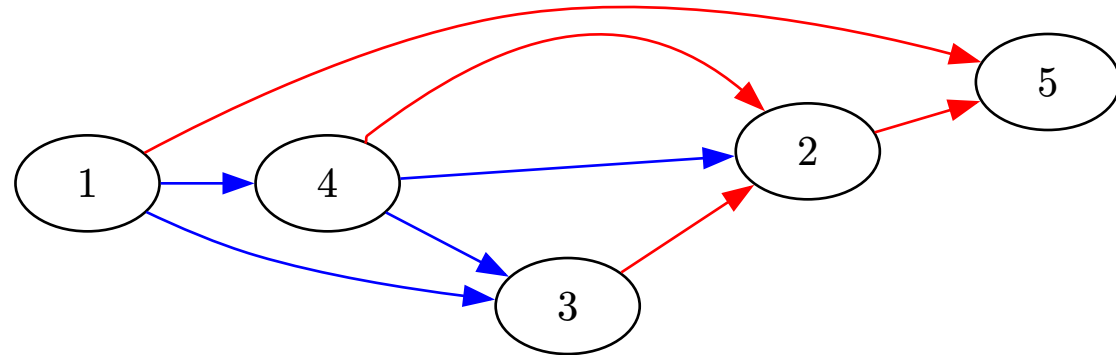
$$r_1(X)r_4(X)w_4(X)r_1(Y)r_4(Z)w_4(Z)w_3(Y)w_3(Z)w_2(T)w_2(Z)w_1(T)w_5(T)$$

**T:**  $w_2w_1w_5$

**X:**  $r_1r_4w_4$

**Y:**  $r_1w_3$

**Z:**  $r_4w_4w_3w_2$



In *red*, last writes constraints. In *blue*, to prevent new read-froms (no existing read-froms). There is no cycle, so it is VSR.

**If possible, add/remove/move one action in order to change the class of the schedule.**

If we add  $w_1(Y)$  at the end, the schedule is no longer VSR.

Swapping  $w_1(T)$  and  $w_2(T)$  (a couple of *blind writes*) we build a schedule that is in CSR. Note that this swapping does not modify the reads-from and final write relationships. This confirms that the initial schedule was VSR.

## Exercise S.2

Classify the following schedule

$$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)$$
$$w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$$

$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$

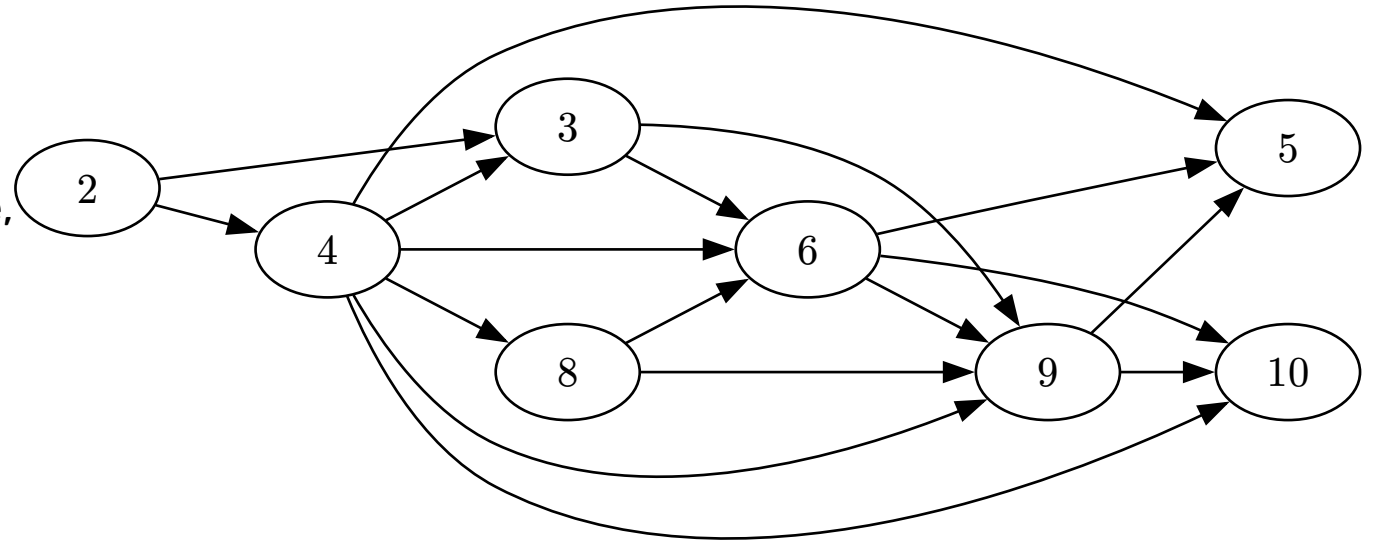
Is it CSR?

**X:**  $r_4r_2w_4w_3$

**Y:**  $w_2w_4r_3$

**Z:**  $w_4r_3r_6r_8w_6w_9r_5r_{10}$

Is there any cycle? **No.** Therefore,  
*it is CSR (and VSR too).*

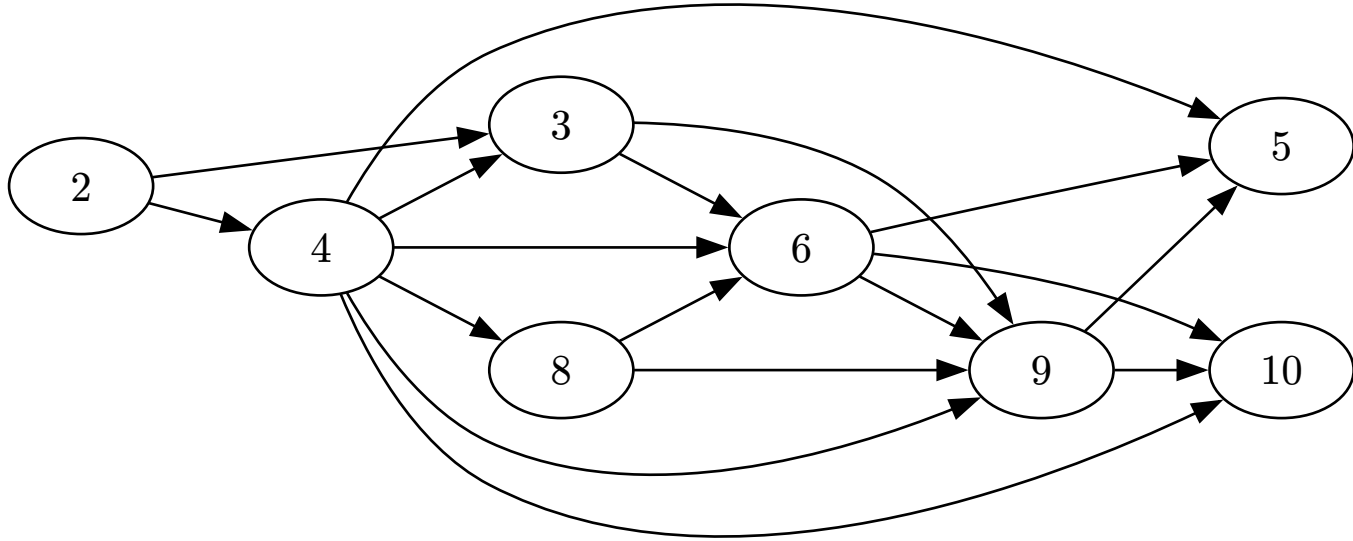


Quick interlude: how do we check for acyclicity?

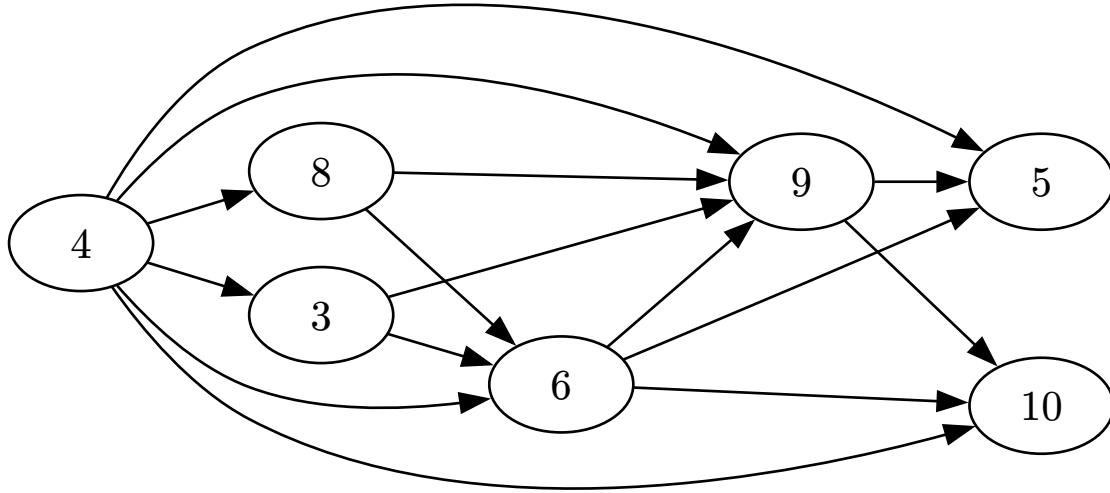
A node can be part of a cycle *if and only if* it has **both** incoming and outgoing edges.

Nodes with only incoming or outgoing arcs **cannot** be part of a cycle, and can be ignored.

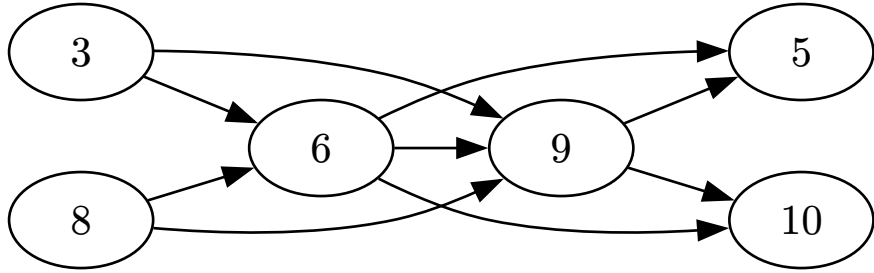
Let's see an example with the previous graph.



Node 2 has no incoming edges: it can be deleted.



Now node 4 can be removed.



The process can be repeated.

If no node remains, there is no cycle.

## Exercise S.2 - 2PL

$$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X) \\ w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$$

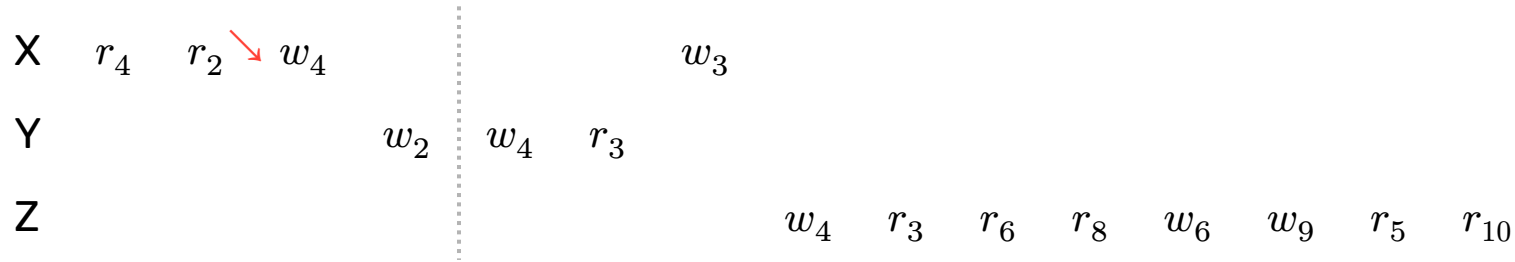
Back on track: is the schedule **2PL-strict**?

We have to

1. Split operations per resource, and organize by *time*
2. For each transaction
  1. Find where it ends: it can release locks only after this point
  2. If it cannot acquire all locks before this point, or release them after, it's not 2PL-strict

$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$

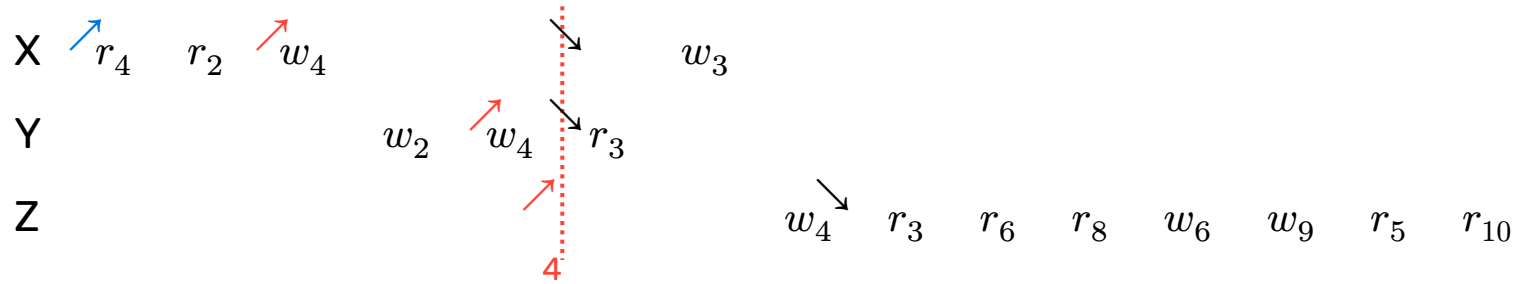
Let's check  $t_2$ .



$t_4$  at time 3 must acquire an exclusive lock (XL) on  $X$ , i.e.,  $t_2$  must release  $X$ . But,  $t_2$  finishes later at time 4. Hence, the schedule **is not** 2PL-strict.

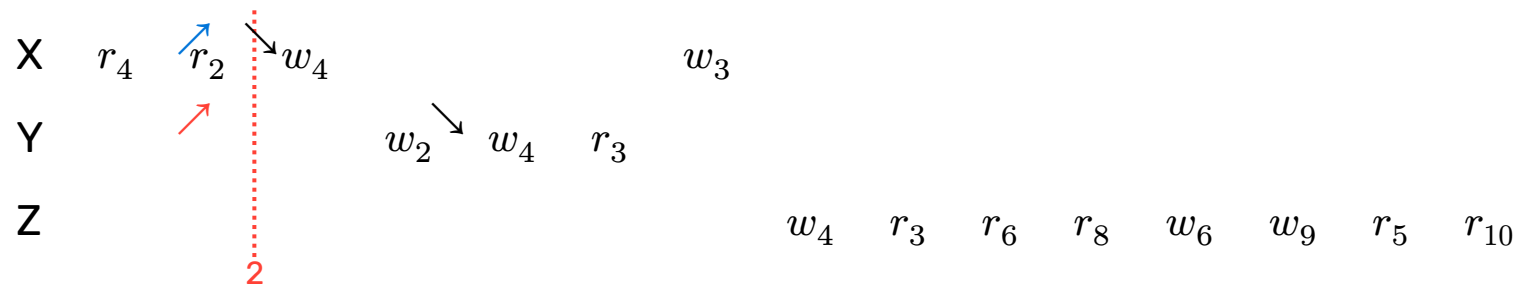
$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$

Is it **2PL**? Let's check  $t_4$ . In blue shared locks, in red exclusive locks.



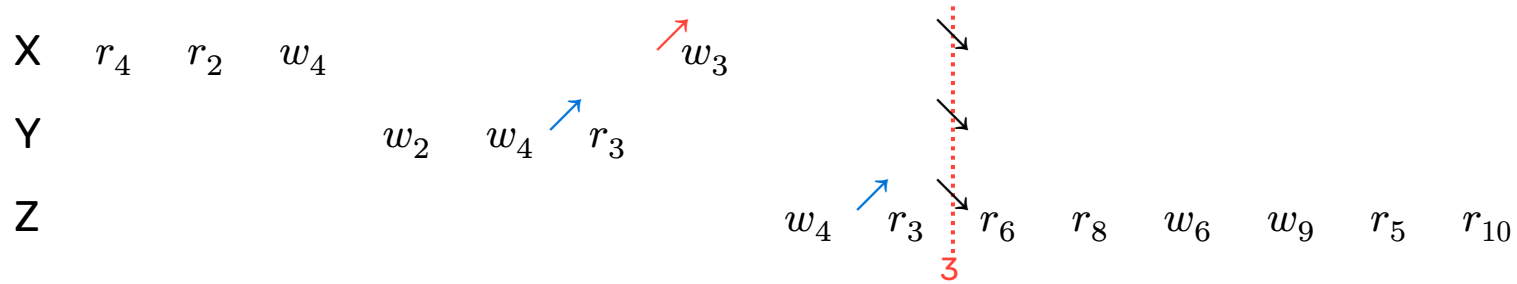
$$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$$

Is it **2PL**? Let's check  $t_2$ .



$$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$$

Is it **2PL**? Let's check  $t_3$ .



$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$

Is it **2PL**? **Yes**.

Other transactions pose no problem.

## Exercise S.2 - TS

$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$

Is it **TS-mono**? How to check:

1. Build a table with RTM and WTM for each resource
2. For each action
  - $r_i(X)$ : if  $i < \text{WTM}(X)$  reject, else update RTM with max
  - $w_i(X)$ : if  $i < \text{RTM}(X) \vee i < \text{WTM}(X)$  reject, else update WTM
3. If there is a reject, it's not TS-mono

	X		Y		Z		Killed?
	RTM	WTM	RTM	WTM	RTM	WTM	
(begin)	0	0	0	0	0	0	
$r_4(X)$	4	0	0	0	0	0	
$r_2(X)$	4	0	0	0	0	0	
$w_4(X)$	4	4	0	0	0	0	
$w_2(Y)$	4	4	0	2	0	0	
$w_4(Y)$	4	4	0	4	0	0	
$r_3(Y)$	4	4	0	4	0	0	yes, $3 < \text{WTM}(Y)$

There has been a kill, so it's **not TS-mono**.

$r_4(X)r_2(X)w_4(X)w_2(Y)w_4(Y)r_3(Y)w_3(X)w_4(Z)r_3(Z)r_6(Z)r_8(Z)w_6(Z)w_9(Z)r_5(Z)r_{10}(Z)$

Is it **TS-multi**? How to check:

1. Build a table with RTM and WTM for each resource and keep *multiple written versions*
2. For each action
  - $r_i(X)$ : ok, read the correct version, update RTM as before
  - $w_i(X)$ : reject if  $i < \text{RTM}(X)$ , otherwise update WTM adding the version
3. If there is a reject, it's not TS-multi

	X		Y		Z		Killed?
	RTM	WTM	RTM	WTM	RTM	WTM	
(begin)	0	0	0	0	0	0	
$r_4(X)$	$4_0$	0	0	0	0	0	
$r_2(X)$	$4_0$	0	0	0	0	0	
$w_4(X)$	$4_0$	0, 4	0	0	0	0	
$w_2(Y)$	$4_0$	0, 4	0	0, 2	0	0	
$w_4(Y)$	$4_0$	0, 4	0	0, 2, 4	0	0	
$r_3(Y)$	$4_0$	0, 4	$3_2$	0, 2, 4	0	0	
$w_3(X)$	$4_0$	0, 4	$3_2$	0, 2, 4	0	0	yes, $3 < \text{RTM}$

As before, there has been a kill: it's **not TS-multi**.

### Exercise S.3

Is the following schedule CSR or VSR?

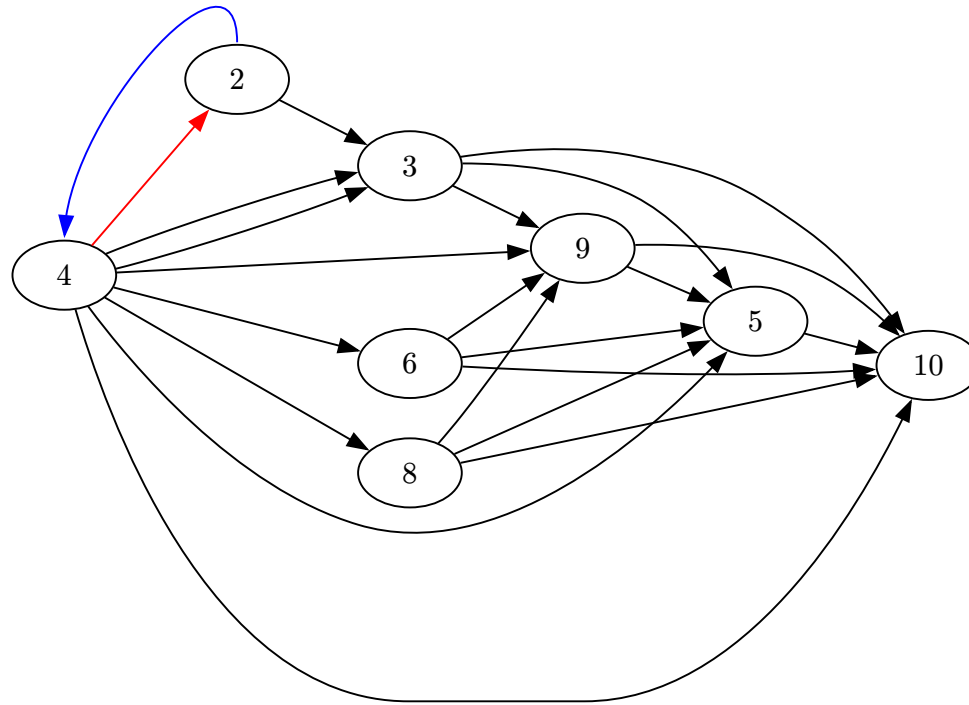
$$w_4(X)r_2(X)w_2(Y)w_4(Y)w_3(X)w_4(Z)$$
$$r_3(Z)r_6(Z)r_8(Z)w_9(Z)w_5(Z)r_{10}(Z)$$

Let's build the conflicts graph

X:  $w_4 r_2 w_3$

Y:  $w_2 w_4$

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



There is a cycle: **not CSR.**

Is it VSR? **No.**

We must have  $t_4 \rightarrow t_2$ , because otherwise we would introduce a new read-from on  $X$ .

But we must also have  $t_2 \rightarrow t_4$ , otherwise the last-write on  $Y$  would change.

Hence, the schedule cannot be VSR.

## Exercise S.4 – A lesson about blind writes

Classify the following schedule in CSR and/or VSR

$$r_5(X)r_3(Y)w_3(Y)r_6(T)r_5(T)w_5(Z)w_4(X)r_3(Z)w_1(Y)$$
$$r_6(Y)w_6(T)w_4(Z)w_1(T)w_3(X)w_1(X)r_1(Z)w_2(T)w_2(Z)$$

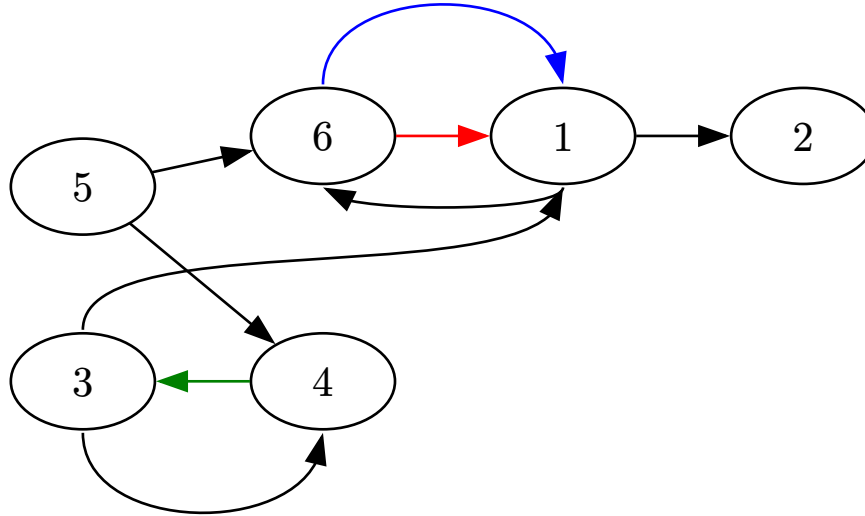
T:  $r_6 r_5 w_6 w_1 w_2$

X:  $r_5 w_4 w_3 w_1$

Y:  $r_3 w_3 w_1 r_6$

Z:  $w_5 r_3 w_4 r_1 w_2$

Let's draw a partial graph.



It is **not in CSR**. Cycle **3-4** can be broken (blind writes). However,  $w_6(T)w_1(T)$  are blind writes, but the cycle cannot be resolved (there is  $r_6(T)...w_1(T)$ ).

Hence, the schedule **cannot be in VSR**.

The lesson: *be careful when drawing conflict arcs*.

As an extra, what about TS?  $w_1(T)$  is sufficient to exclude that the schedule is both TS-mono and TS-multi. The reason is left as an exercise for the reader.

## Exercise S.5

The following schedule is in **VSR**. Classify it with respect to CSR, TS-mono, TS-multi, 2PL-strict

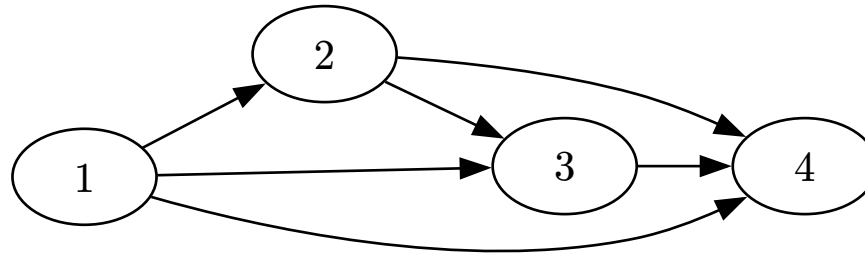
$$r_1(X)w_2(X)r_1(Z)w_1(Y)r_3(X)r_4(X)w_3(Z)w_2(Y)r_3(Y)w_4(X)w_4(Y)$$

Is it **CSR**? Yes.

**X:**  $r_1 w_2 r_3 r_4 w_4$

**Y:**  $w_1 w_2 r_3 w_4$

**Z:**  $r_1 w_3$



No cycles: **it is CSR.**

Is it **2PL-strict**? No.

	1	2	3	4	5	6	7	8	9	10	11
X	$r_1$	$w_2$			$r_3$	$r_4$				$w_4$	
Y				$w_1$				$w_2$	$r_3$		$w_4$
Z			$r_1$				$w_3$				

Focus on  $t_1$ . It has to release a lock on  $X$  before 2, but it doesn't end until 4. Hence, it cannot be 2PL-strict.

Is it **TS-mono** or **TS-multi**?

We *could* do the table, but note that, for each resource, all operations are in increasing index order.

Hence it is surely both **TS-mono** and **TS-multi**.

## To conclude

- It is clear that  $t_1$  must release the SL on  $X$  before 2 (for  $t_2$  to acquire XL on it), but this is not compatible with 2PL-strict, as the last operation of  $t_1$  occurs at 4.
- The conflict graph is acyclic, hence the schedule is in CSR.
- Eventually, it is easy to observe that the schedule is both in TS-mono and TS-multi, even without simulating the evolution of RTM and WTM counters, because transaction indexes occur in strictly increasing order in the operations for to each resource.

## Exercise S.6

Verify whether the following schedule is compatible with a 2PL system (non strict):

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

Let's use another technique to check: **temporal constraints**.

It is necessary to impose temporal constraints upon the lock and unlock requests, when such requests are in wait status.

	1	2	3	4	5	6	7	8	9	10	11
A	$r_1$			$r_2$						$w_1$	$w_3$
B		$r_2$			$r_1$			$w_2$	$r_3$		
C			$w_1$			$w_2$	$r_3$				

The locking rules imply, for  $t_1$  and  $t_2$

$$4 < U_2^r(A) < L_1^w(A)$$

$$U_1^r(B) < L_2^w(B) < 8$$

	1	2	3	4	5	6	7	8	9	10	11
A	$r_1$			$r_2$						$w_1$	$w_3$
B		$r_2$			$r_1$			$w_2$	$r_3$		
C			$w_1$			$w_2$	$r_3$				

The 2PL rule imposes not to acquire a new lock after an unlock

$$L_1^w(A) < U_1^r(B)$$

$$L_2^w(B) < U_2^r(A)$$

	1	2	3	4	5	6	7	8	9	10	11
A	$r_1$			$r_2$						$w_1$	$w_3$
B		$r_2$			$r_1$			$w_2$	$r_3$		
C			$w_1$			$w_2$	$r_3$				

Combining the two we get an impossible condition

$$U_2^r(A) < L_1^w(A) < U_1^r(B) < L_2^w(B) < U_2^r(A)$$

and the schedule is not 2PL.

## Exercise S.7

Verify whether the following sequence of operations is consistent with 2PL

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

(Note it's not the same as the previous exercise).

	1	2	3	4	5	6	7	8	9	10	11
A	$r_1$			$r_2$						$w_2$	$w_3$
B		$r_2$			$r_1$			$w_2$	$r_3$		
C			$w_1$			$w_2$	$r_3$				

- $t_1$  can acquire all locks at 1 and release after each use
- $t_3$  can acquire all locks just before use and release at the end
- $t_2$  must release C between 6 and 7, and cannot acquire B before 6, but all this is possible

## Exercise C.1

The isolation class *read-committed* requires that transactions comply with 2PL-strict for the write lock, and can release read lock for reading, even before the end of the transaction.

Show an example of a schedule that is *read-committed* but *not VSR*.

**Show an example of a schedule that is *read-committed* but *not VSR*.**

In order to find such a schedule, we must violate the rules of 2PL with some read operations (as writes follow the 2PL-strict rules), otherwise our schedule would still be 2PL, and therefore also VSR.

We need to build a schedule where releasing a read lock before some other lock is acquired by the same transaction is necessary for the schedule to complete.

We can simply interleave non-view-serializable operations on one resource, such as:

$$S_1 = r_1(X)w_2(X)w_1(X)$$

## **Exercise C.2**

Continued from the previous exercise.

Now, extend the known classification graph of classes VSR, CSR, and 2PL with the representation of the read-committed isolation class.

Read-committed is, by definition, a relaxation of class 2PL-strict, and therefore contains this class.

As for 2PL, CSR, and VSR, we can easily find a schedule that is in 2PL but not in read-committed, such as

$$S_2 = w_1(X)w_2(X)r_1(Y)$$

hence it does not contain the entire 2PL.

Graphically, this is the new diagram

