Exercises
Concurrency Control – part 1

Exercise 1

A.1 Search for anomalies

Indicate whether the following schedules can produce anomalies; the symbols c_i and a_i indicate the transactional decision (commit or abort).

1) $r_1(x)\ \ w_1(x)\ \ r_2(x)\ \ w_2(y)\ \ a_1\ c_2$
2) $r_1(x)\ \ w_1(x)\ \ r_2(y)\ \ w_2(y)\ \ a_1\ \ c_2$
3) $r_1(x)\ \ r_2(x)\ \ r_2(y)\ \ w_2(y)\ \ r_1(z)\ \ a_1\ \ c_2$
4) $r_1(x)\ \ r_2(x)\ \ w_2(x)\ \ w_1(x)\ \ c_1\ \ c_2$
5) $r_1(x)\ \ r_2(x)\ \ w_2(x)\ \ r_1(y)\ \ c_1\ \ c_2$
6) $r_1(x)\ \ w_1(x)\ \ r_2(x)\ \ w_2(x)\ \ c_1\ \ c_2$

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

Dirty read:

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

T_1 aborts, therefore r_2 should not read x as modified by w_1

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

No anomaly

(update patterns on different resources)

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

No anomaly

4) $r_1(x)\ r_2(x)\ w_2(x)\ w_1(x)\ c_1\ c_2$

Update loss

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

T_2 has no effect, as its write is overwritten by T_1. No transaction, however, has read inconsistent values.

5) $r_1(x)\ \ r_2(x)\ \ w_2(x)\ \ r_1(y)\ \ c_1\ \ c_2$

No anomaly

6) $r_1(x)\ \ w_1(x)\ \ r_2(x)\ \ w_2(x)\ \ c_1\ \ c_2$

No anomaly

The schedule is serial w.r.t. data modifications
(only commit of T1 is delayed)

Exercise 2

A.2 – Search for anomalies

$r_1(x)\ r_2(x)\ r_3(x)\ w_1(x)\ r_4(y)\ w_2(x)\ r_4(x)\ w_4(y)\ r_3(y)\ w_4(x)\ r_5(y)\ w_6(y)\ w_5(y)\ w_7(y)$

This schedule may produce 2 anomalies:

a **Lost Update** and a **Phantom Update**.

Identify them

A.2

$r_1(x)\ r_2(x)\ r_3(x)\ w_1(x)\ r_4(y)\ w_2(x)\ r_4(x)\ w_4(y)\ r_3(y)\ w_4(x)\ r_5(y)\ w_6(y)\ w_5(y)\ w_7(y)$

Lost Update : transactions 1 and 2

The notion of Phantom Update (reminder)

$A+B+C=100$, *A "global" constraint holds on some resources*

$A=50$, $B=30$, $C=20$ *Initially, the constraint is satisfied*

T_1: r(A,x), r(B,y)

T_2: r(B,s), r(C,t)

T_2: $s = s + 10$, $t = t - 10$

T_2: w(s,B), w(t,C) (now $B=40$, $C=10$, **$A+B+C=100$**)

T_1: r(C,z) (but, for T_1, $x+y+z = A+B+C =$ **90!**)

At the end, the constraints still hold, but T_1 has the impression that it is violated

Answer to one of the questions on the chat

Q: Does the phantom update occur only when there is a global constraint?

A: YES, this anomaly occurs only when an integrity constraint exists (over two or more variables).

In the past this anomaly was also called “constraint violation”

A.2

$r_1(x)\ r_2(x)\ \mathbf{r_3(x)}\ w_1(x)\ r_4(y)\ w_2(x)\ r_4(x)\ \mathbf{w_4(y)}\ \mathbf{r_3(y)}\ \mathbf{w_4(x)}\ r_5(y)\ w_6(y)\ w_5(y)\ w_7(y)$

Phantom update:

Transactions 3 and 4, with a constraint on x and y (for instance, the sum x+y has to be constant).

T3 may see the constraint as violated, because $\mathbf{r_3(x)}$ reads the unmodified version of x and $\mathbf{r_3(y)}$ reads the modified version of y.

Answer to one of the questions on the chat

$r_1(x)$ $r_2(x)$ r3(x) w1(x) r4(y) w2(x) r4(x) w4(y) r3(y) w4(x) r5(y) w6(y) w5(y) $w_7(y)$

Q: Is this a phantom update?
A: It looks like a phantom update, but let us analyze it in detail.

T4, after reading a value of y, reads an updated value of x (updated by T2). If there is a constraint on x+y it may be violated. Suppose that such a constraint exists. Then, also the serial execution of the two transactions should satisfy the constraint. T2 followed by T4 produces the same result of the above schedule → the serial execution has exactly the same problem. If T2 can be accepted and a constraint on x+y exists, then T2 can only insert a value that satisfies the constraint. If T2 can insert any value, then the constraint does not exist.

Compare this case with the previous slide: the behavior of any serial execution of T3 and T4 can produce a correct result. Instead, their interleaved execution may produce the anomaly.

Exercise 3

C.1 Classify the following schedule

as Non-VSR, VSR or CSR:

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

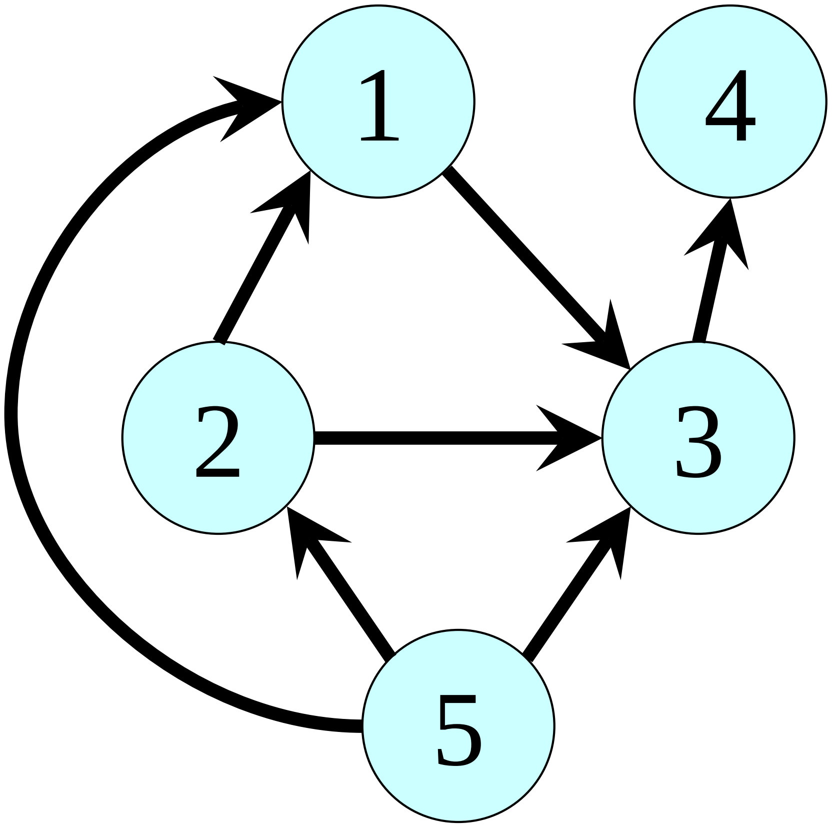
S: w_3

U: w_5 w_2 w_1

X: r_1 r_5 w_3

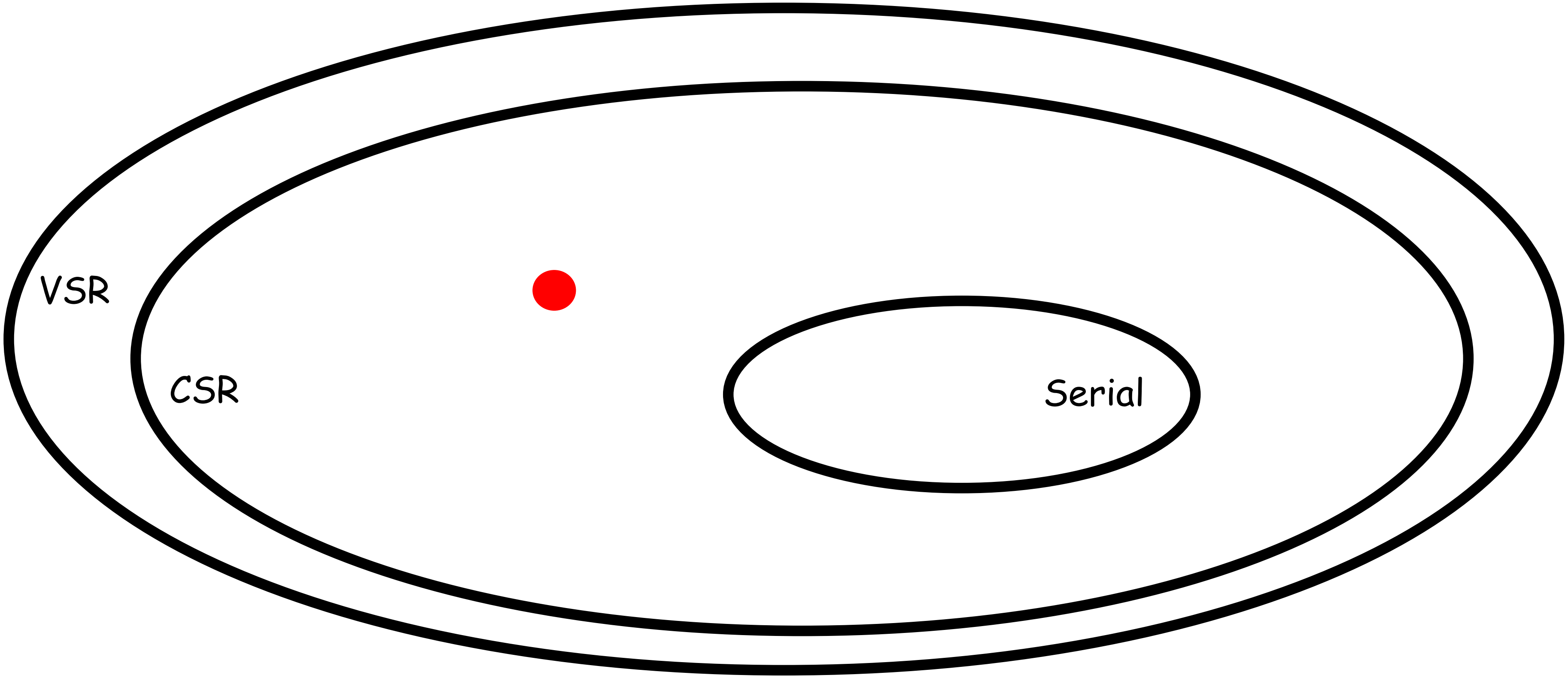
Y: r_2 w_3 r_4

Z: w_5 r_5



NO cycles in the graph:
the schedule is CSR
(and then also VSR)

Where is our schedule?



Exercise 4

C.2 Classify the following schedule

as Non-VSR, VSR or CSR:

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

(similar to C.1, with r2(u) w2(s) added at the beginning – we can exploit the previous graph and add the new arcs)

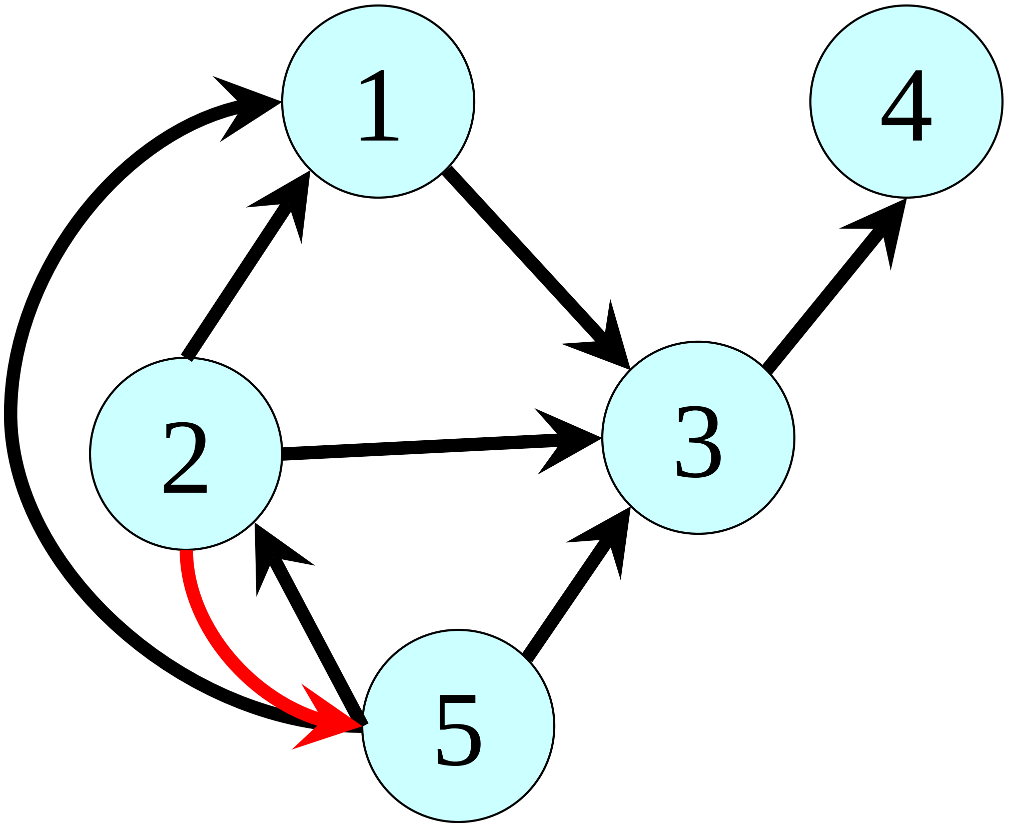
S: w_2 w_3

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



The graph is cyclic:
the schedule is NOT CSR.
Is it 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

The schedule

T_2, T_5, T_1, T_3, T_4

has the same *reads-from* and *final writes*

S: w_2 w_3

U: r_2 w_2 w_5 w_1

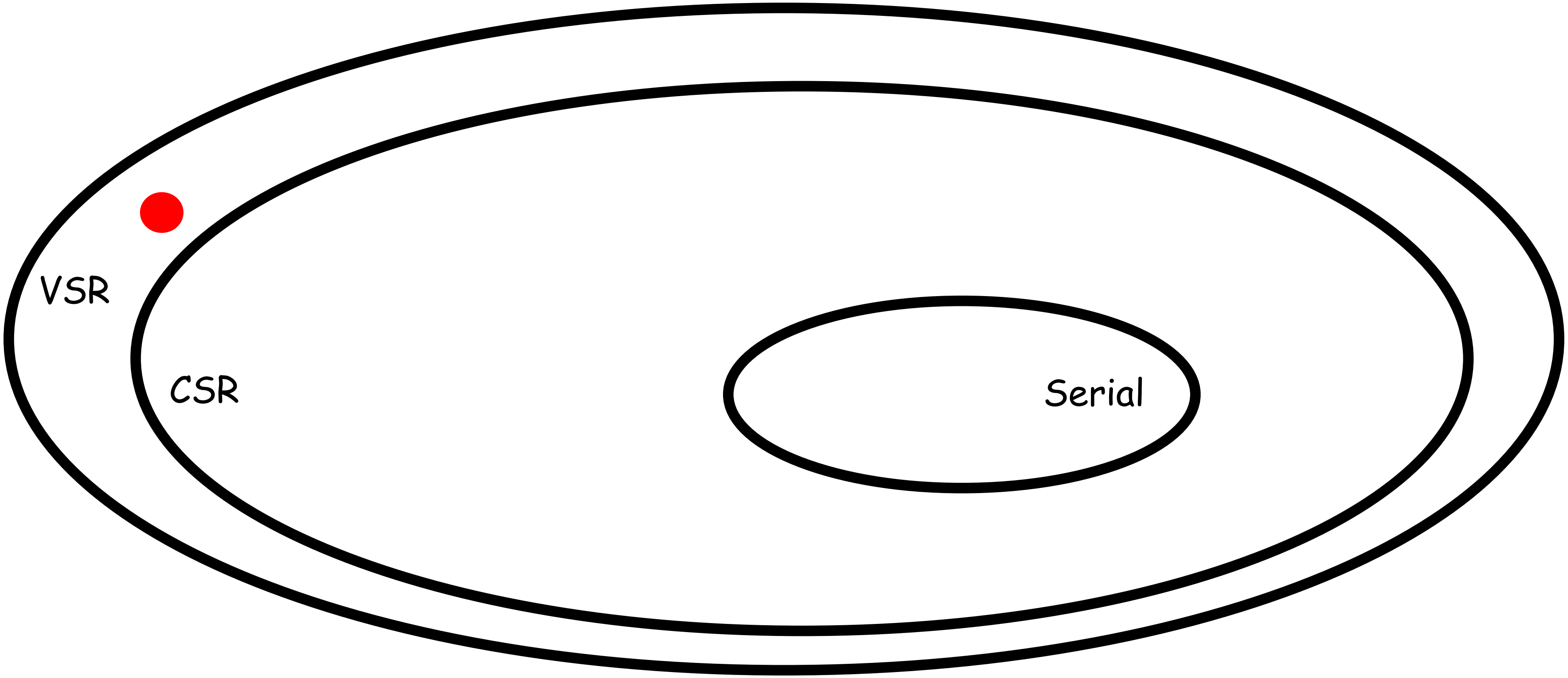
X: r_5 r_1 w_3

Y: r_2 w_3 r_4

Z: w_5 r_5

It is VSR

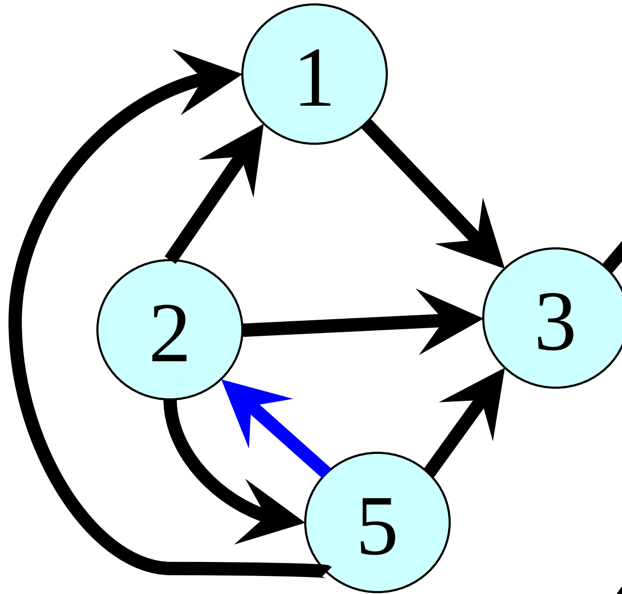
Where is our schedule?



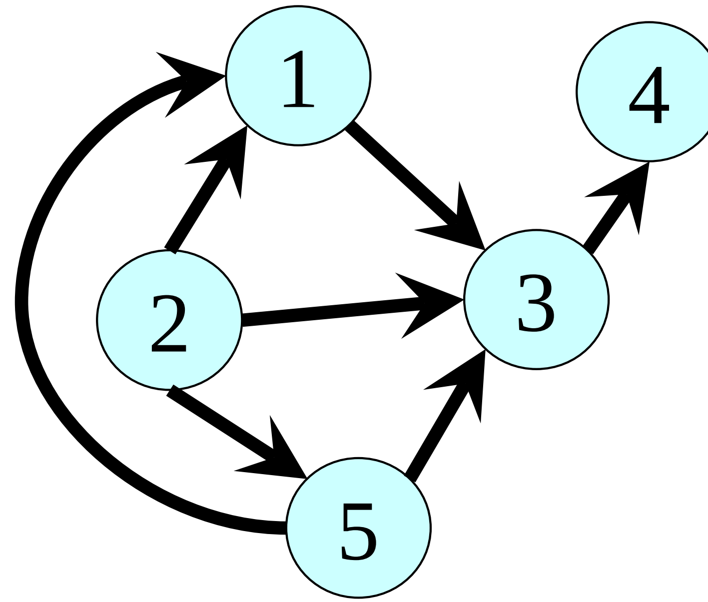
Definition: a write $w_i(X)$ is said to be ***blind*** if it is not the last action of resource X and the following action on X is a write $w_j(X)$

Property: each schedule $S \in VSR$ and $S \notin CSR$ has, in its conflict graph, cycles of arcs due to pairs of blind writes. These can be swapped without modifying the reads-from and final write relationships.
Once the graph is acyclic it is possible to find a serial schedule view-equivalent to the initial one.

S: w_2 w_3
U: r_2 $\mathbf{w_5}$ $\mathbf{w_2}$ w_1
X: r_1 r_5 w_3
Y: r_2 w_3 r_4
Z: w_5 r_5



S: w_2 w_3
U: r_2 $\mathbf{w_2}$ $\mathbf{w_5}$ w_1
X: r_1 r_5 w_3
Y: r_2 w_3 r_4
Z: w_5 r_5



The graph becomes acyclic by swapping the *blind writes* $\mathbf{w_5}$ $\mathbf{w_2}$

Exercise 5

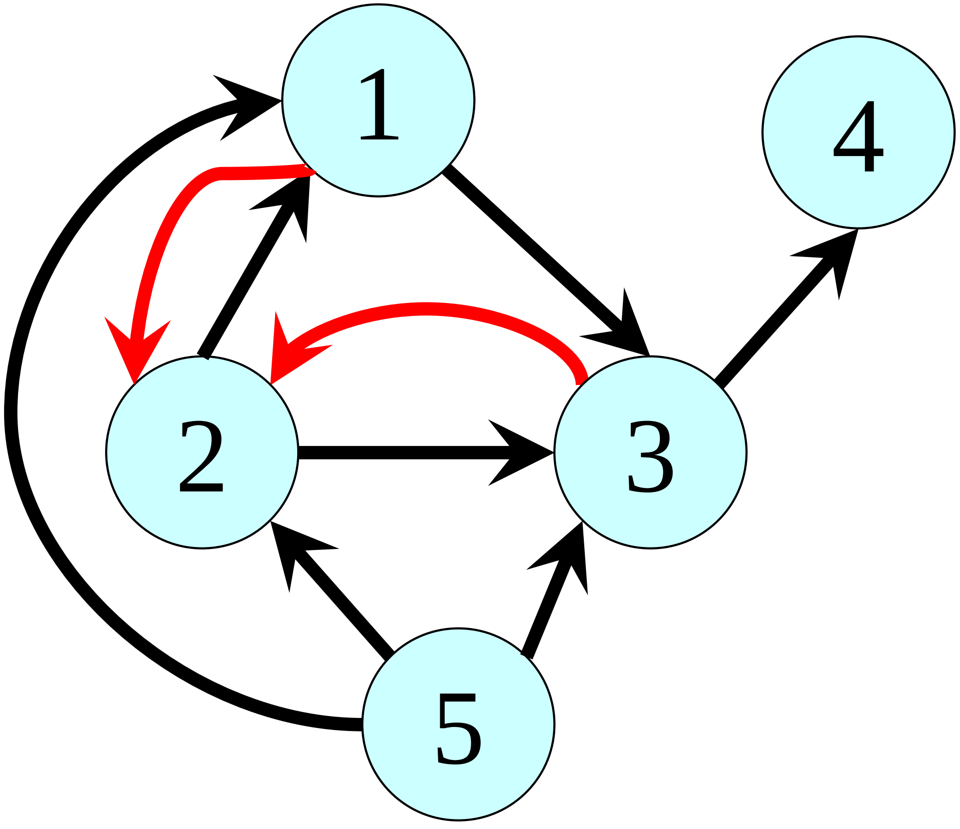
C.3 Classify the following schedule

as Non-VSR, VSR or CSR:

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

(similar to C.1, with r2(u) w2(s) added at the end)

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



The graph is **cyclic**:
the schedule is NOT CSR.
Is it 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

It is not possible to decide whether T2 is to be placed before or after T3, it is not VSR

S: w_3 w_2

U: w_5 w_2 w_1 r_2

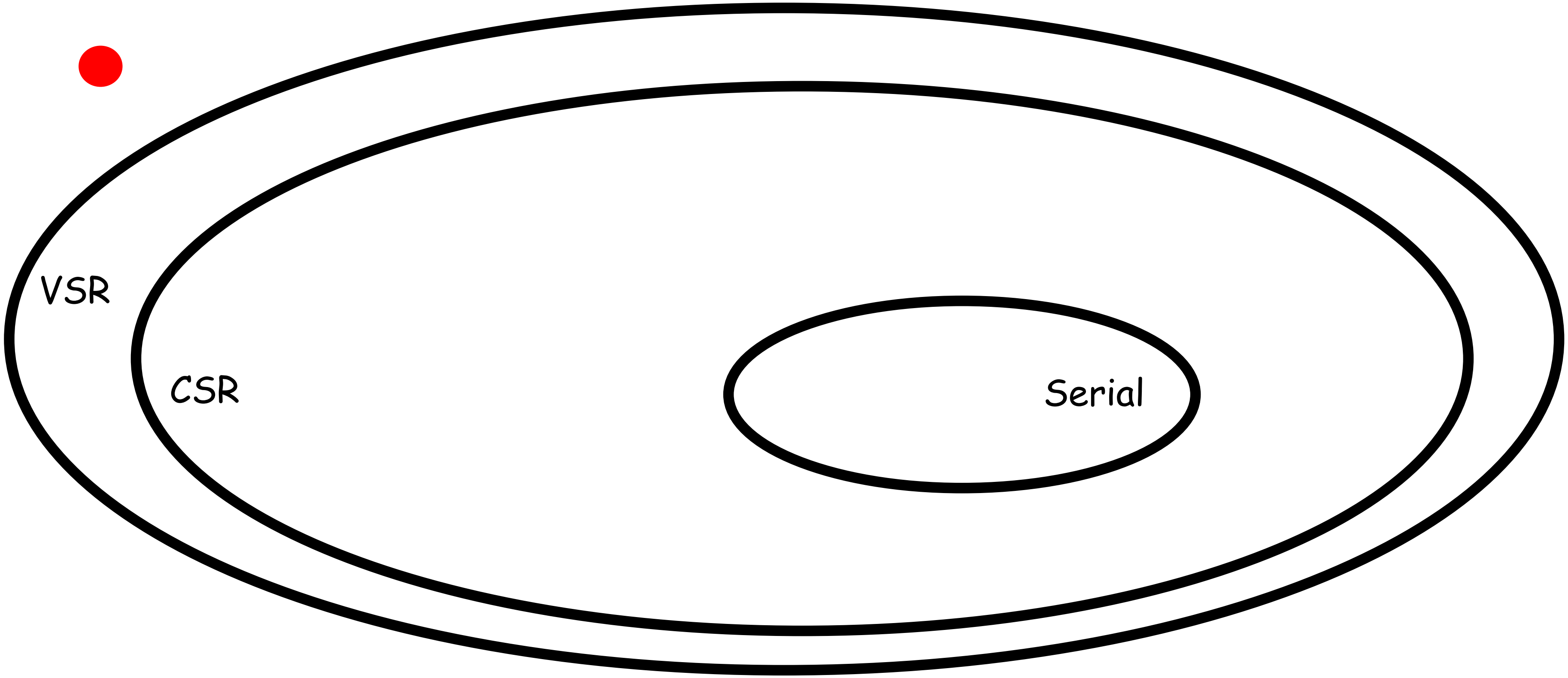
X: r_1 r_5 w_3

Y: r_2 w_3 r_4

Z: w_5 r_5

Alternatively, just consider the three highlighted operations on resource U. No serial schedule can preserve the relationship "r_2 reads from w_1" indicated by the red arrow (as w_2 and r_2 are contiguous in all serial schedule).

Where is our schedule?



Exercise 6

C.4 Classify the following schedule.

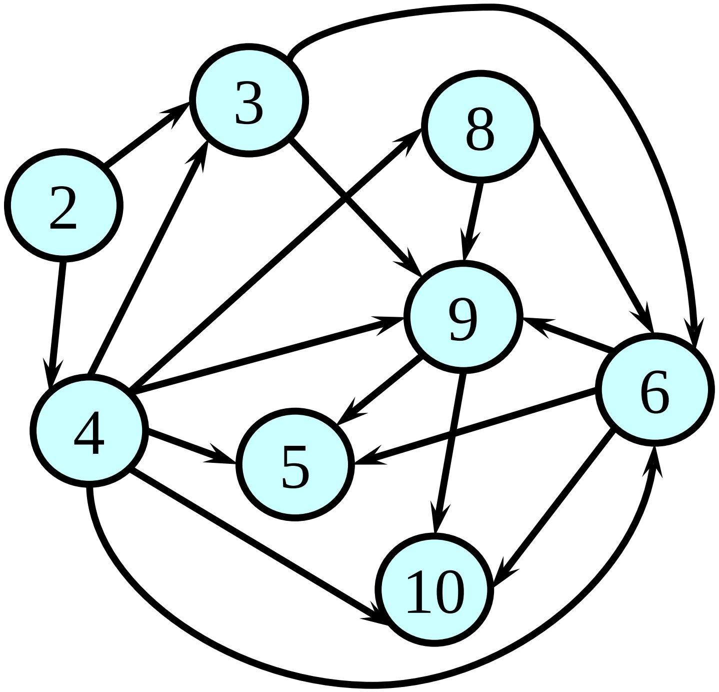
$r_4(x)$ $r_2(x)$ $w_4(x)$ $w_2(y)$ $w_4(y)$ $r_3(y)$ $w_3(x)$
$w_4(z)$ $r_3(z)$ $r_6(z)$ $r_8(z)$ $w_6(z)$ $w_9(z)$ $r_5(z)$ ${}_{10}(z)$

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}

We project the schedule on each resource and build the conflict graph



Is there any cycle? No.

The schedule is in CSR (and then also in VSR). Can you write an equivalent serial schedule?

The graph suggests, e.g., the serial schedule 2,4,3,8,6,9,5,10 as view equivalent (built choosing and deleting nodes without incoming arcs)

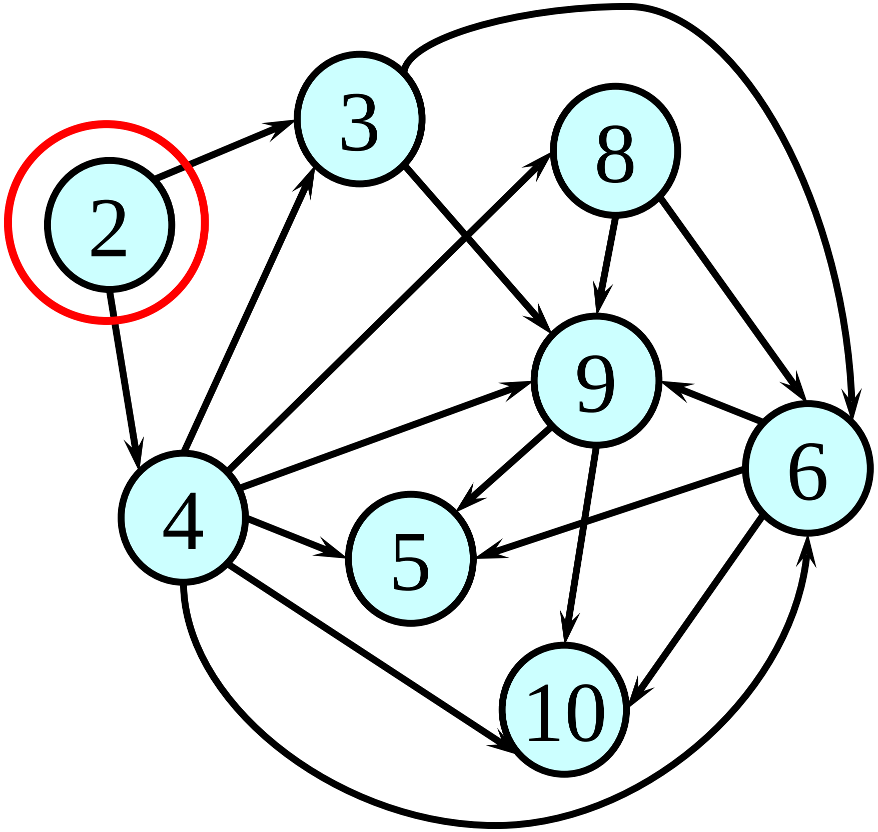
How do we check acyclicity in practice?

A node can be part of a cycle iff it has incoming **and** outgoing arcs.

Nodes with only incoming or only outgoing arcs cannot, and can be (***recursively***) ignored.

The same holds for arcs adjacent to such nodes.

In this way, we not only check for acyclicity, but also identify a serial schedule that is view-equivalent to the given one (if transactions are considered in the order in which the corresponding nodes are deleted)



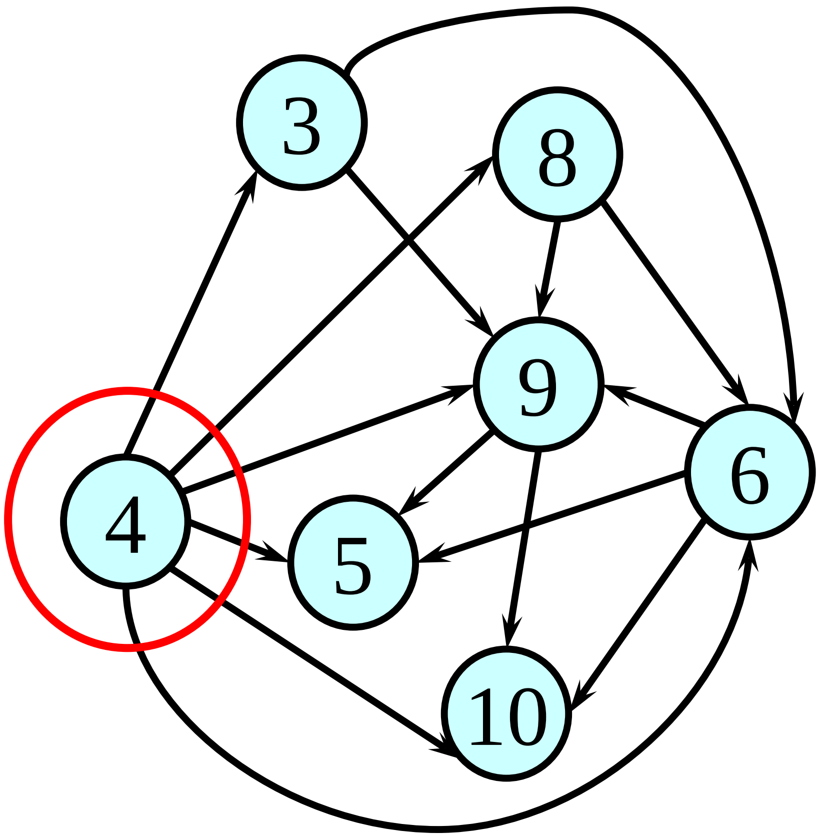
How do we check acyclicity in practice?

A node can be part of a cycle iff it has incoming **and** outgoing arcs.

Nodes with only incoming or only outgoing arcs cannot, and can be (***recursively***) ignored.

The same holds for arcs adjacent to such nodes.

In this way, we not only check for acyclicity, but also identify a serial schedule that is view-equivalent to the given one (if transactions are considered in the order in which the corresponding nodes are deleted)



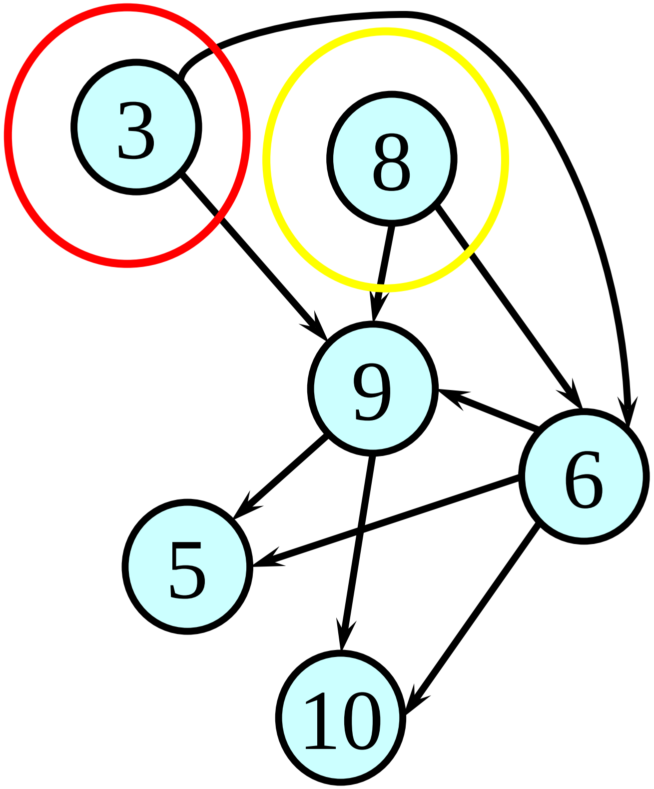
How do we check acyclicity in practice?

A node can be part of a cycle iff it has incoming **and** outgoing arcs.

Nodes with only incoming or only outgoing arcs cannot, and can be (***recursively***) ignored.

The same holds for arcs adjacent to such nodes.

In this way, we not only check for acyclicity, but also identify a serial schedule that is view-equivalent to the given one (if transactions are considered in the order in which the corresponding nodes are deleted)



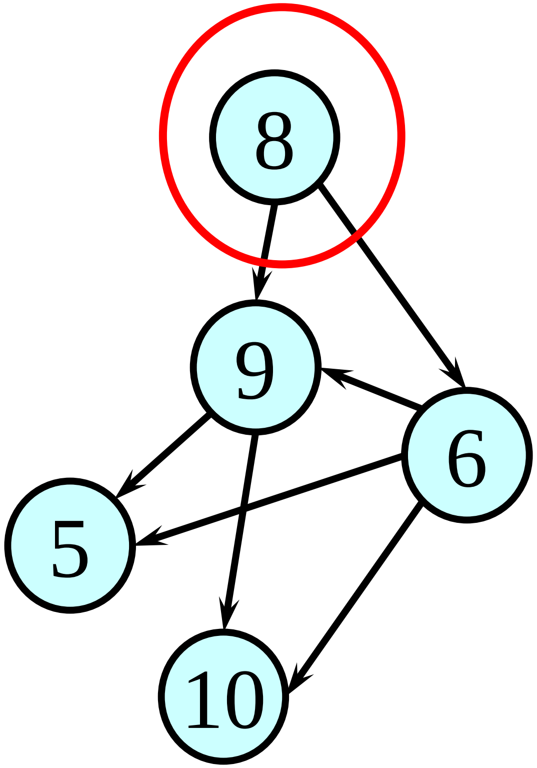
How do we check acyclicity in practice?

A node can be part of a cycle iff it has incoming **and** outgoing arcs.

Nodes with only incoming or only outgoing arcs cannot, and can be (***recursively***) ignored.

The same holds for arcs adjacent to such nodes.

In this way, we not only check for acyclicity, but also identify a serial schedule that is view-equivalent to the given one (if transactions are considered in the order in which the corresponding nodes are deleted)



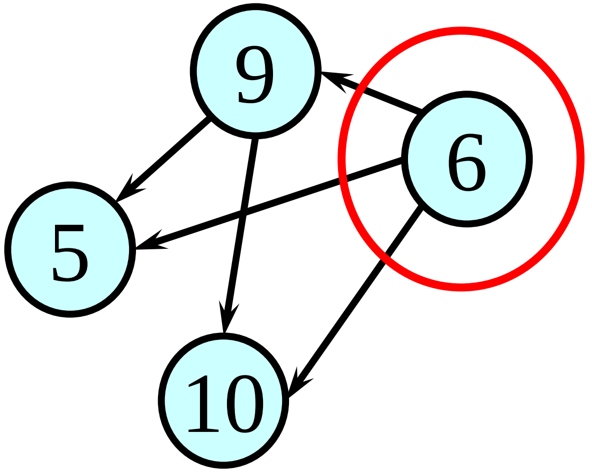
How do we check acyclicity in practice?

A node can be part of a cycle iff it has incoming **and** outgoing arcs.

Nodes with only incoming or only outgoing arcs cannot, and can be (***recursively***) ignored.

The same holds for arcs adjacent to such nodes.

In this way, we not only check for acyclicity, but also identify a serial schedule that is view-equivalent to the given one (if transactions are considered in the order in which the corresponding nodes are deleted)



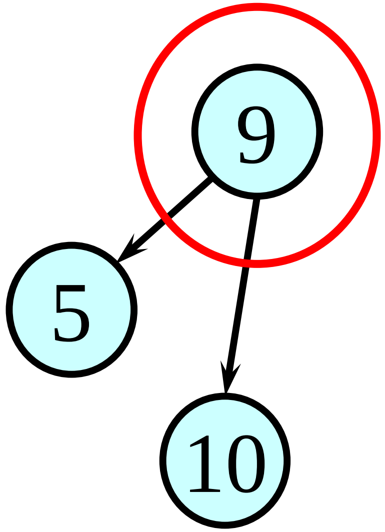
How do we check acyclicity in practice?

A node can be part of a cycle iff it has incoming **and** outgoing arcs.

Nodes with only incoming or only outgoing arcs cannot, and can be (***recursively***) ignored.

The same holds for arcs adjacent to such nodes.

In this way, we not only check for acyclicity, but also identify a serial schedule that is view-equivalent to the given one (if transactions are considered in the order in which the corresponding nodes are deleted)



Exercise 7

C.5 - A lesson about blind writes

Classify the following schedule:

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

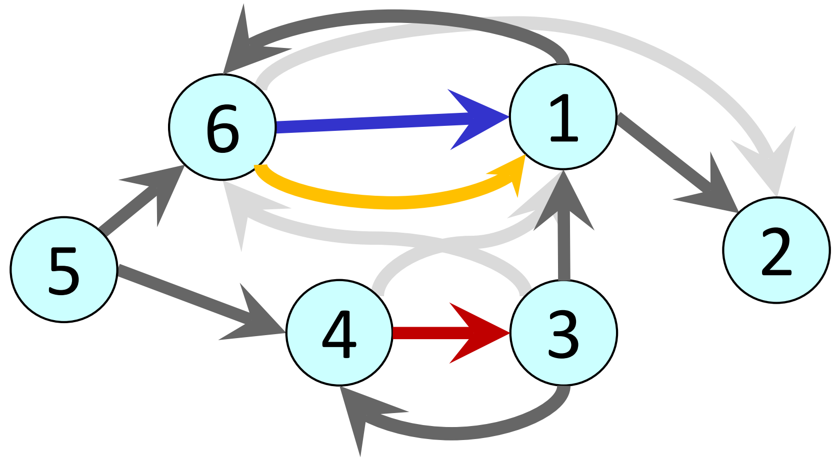
Conflict graph

$t : r_6\ r_5\ \boldsymbol{w_6}\ \boldsymbol{w_1}\ w_2$

$x : r_5\ \boldsymbol{w_4}\ \boldsymbol{w_3}\ w_1$

$y : r_3\ w_3\ w_1\ r_6$

$z : w_5\ r_3\ w_4\ r_1\ w_2$



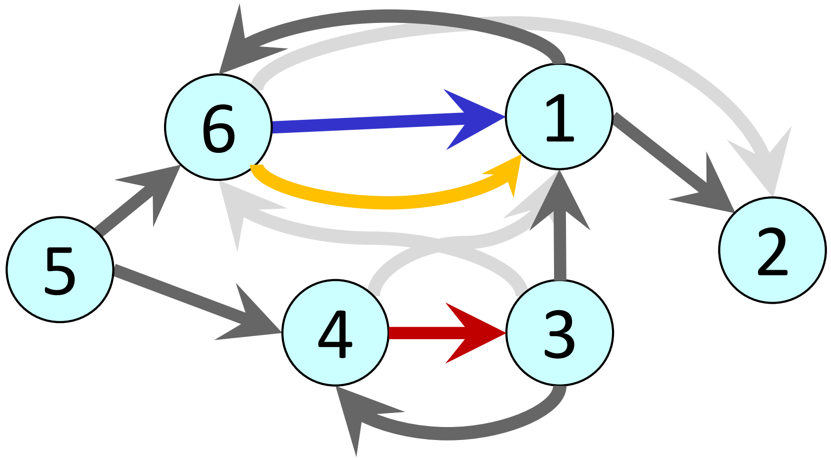
- *Not in CSR (cyclic graph: **two cycles**: transactions 6-1 e 4-3)*
- *Cycle **3-4** can be broken by considering that the schedule is VSR-equivalent to the one obtained by **swapping $w_4(x)$ and $w_3(x)$**. We also have a cycle 6-1 and **$w_6(t)$-$w_1(t)$** as blind writes.*
- *HOWEVER, the 6→1 conflict CANNOT BE "RESOLVED", because **swapping $w_6(t)$ and $w_1(t)$, the only other pair of blind writes, would not affect the conflict due to $r_6(t)$,** that would still precede $w_1(t)$.*

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: $w_5\ r_3\ w_4\ r_1\ w_2$



- *We therefore conclude that the schedule is* ***not even in VSR***.
- **BE CAREFUL WHEN YOU DECIDE NOT TO DRAW**
- **ONE ARC PER EACH CONFLICT!**

Comment on a discussion on the chat:

This example shows that blind writes are *not sufficient* to make the non-CSR schedule VSR.

However, if a non-CSR schedule is VSR, they are a *necessary* condition. Without blind writes, a non-CSR schedule cannot be VSR.

Exercise 8

C.6 Classify the following schedule:

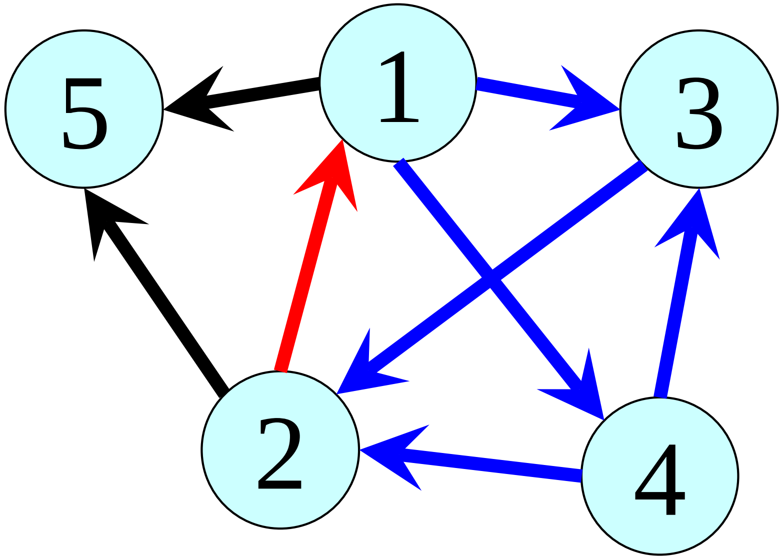
$r_1(x)$ $r_4(x)$ $w_4(x)$ $r_1(y)$ $r_4(z)$ $w_4(z)$ $w_3(y)$ $w_3(z)$ $w_2(t)$ $w_2(z)$ $w_1(t)$ $w_5(t)$

T: w_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



The graph is cyclic (1,4,2 – 1,3,2 – 1,4,3,2).

2,1 is the only arc common to all the three cycles

The schedule is not CSR – Is it 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

The schedule T1, T4, T3, T2, T5 is view-equivalent

It is VSR