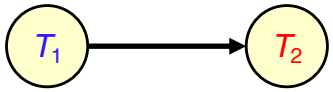
COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 20 EXERCISES
TRANSACTIONS

EXERCISE 1

Indicate which of the following schedules involving T_1 and T_2 is serial, serializable or not serializable. r_i denotes a read (of transaction T_i) and w_i is a write (of transaction T_i).

a) $r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)$



T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)

Serial

b) $r_1(A)\ r_2(A)\ w_1(A)\ w_2(B)$

T_1	T_2
read(A)	
	read(A)
write(A)	
	write(B)

Serialize

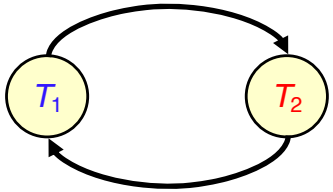
What is the equivalent serial schedule?

$T_2\,T_1$

EXERCISE 1 (CONT'D)

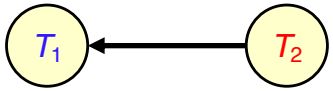
c) $r_1(A)\ r_2(A)\ w_1(A)\ w_2(A)$

Not Serializable



T_1	T_2
read(A)	
	read(A)
write(A)	
	write(A)

d) $r_2(A)\ r_1(A)\ w_2(B)\ w_1(A)$



T_1	T_2
	read(A)
read(A)	
	write(B)
write(A)	

Serializable

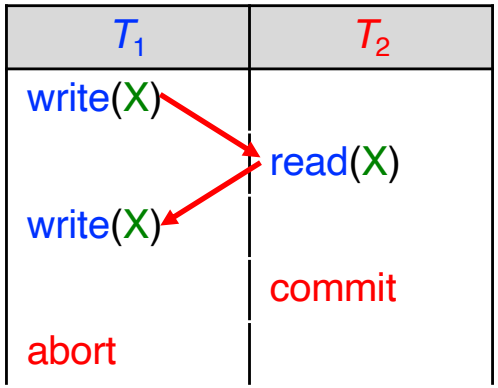
What is the equivalent serial schedule?

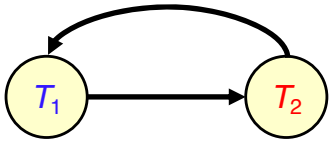
$T_2 T_1$

EXERCISE 2

For each of the following schedules, state whether it is serializable, recoverable and cascadeless. Justify your answers. r_i denotes a read (of transaction T_i) and w_i a write (of transaction T_i).

a) $w_1(X)\ r_2(X)\ w_1(X)\ c_2\ a_1$





Serializable?

No The precedence graph has a cycle.

Recoverable?

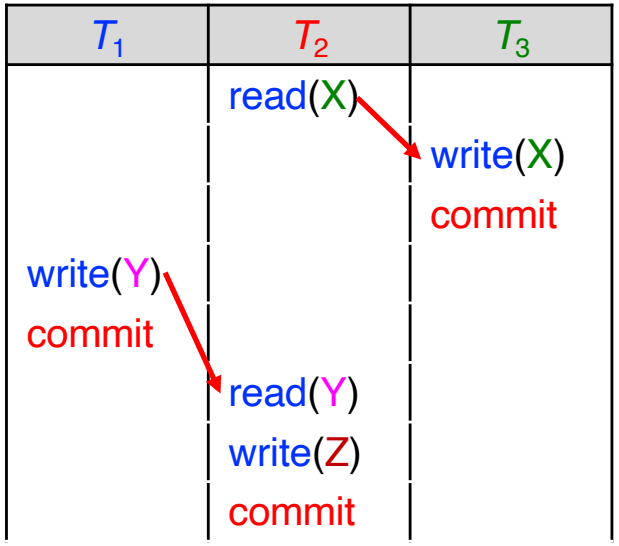
No T_2 reads data item X and commits, but then T_1 aborts.

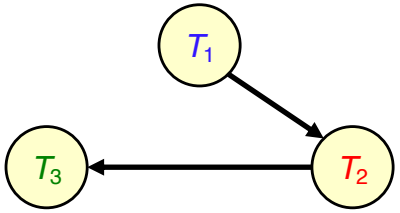
Cascadeless?

No If T_1 aborts, T_2 must also abort.

EXERCISE 2 (CONT'D)

b) $r_2(X)\ w_3(X)\ c_3\ w_1(Y)\ c_1\ r_2(Y)\ w_2(Z)\ c_2$





Serializable?

Yes The equivalent serial schedule is $T_1\,T_2\,T_3$

Recoverable?

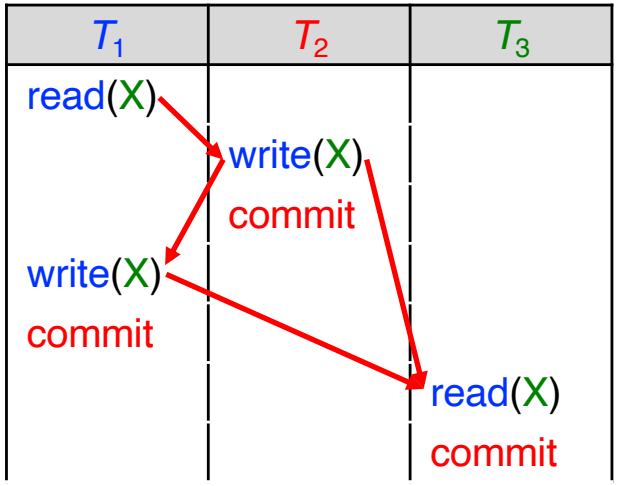
Yes T_2 reads a data item written by T_1 and commits after T_1.

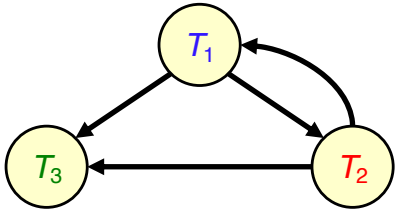
Cascadeless?

Yes The commit of T_1 appears before the commit of T_2.

EXERCISE 2 (CONT'D)

c) $r_1(X)\ w_2(X)\ c_2\ w_1(X)\ c_1\ r_3(X)\ c_3$





Serializable?

No The precedence graph has a cycle.

Recoverable?

Yes T_3 reads a data item written by T_1 and commits after T_1.

Cascadeless?

Yes The commit of T_1 appears before the commit of T_3.

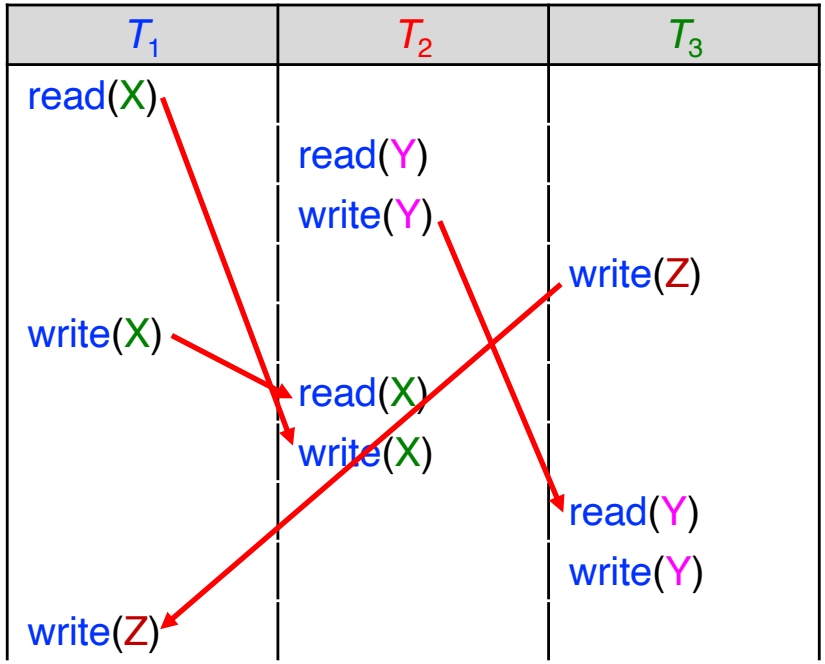
Why recoverable and cascadeless?

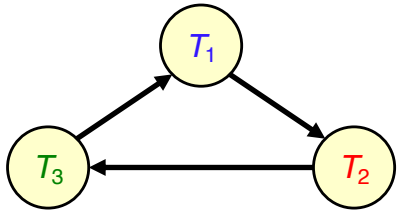
The T_2 write is useless (called a blind write). It is not preceded by a read and it is immediately overwritten by T_1. So, its value is never seen by anyone.

EXERCISE 3

For each of the following schedules, answer the questions.

a) Is the following schedule serializable?





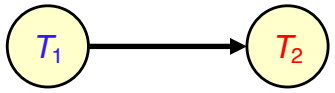
The schedule is not serializable since there is a cycle T_1, T_2, T_3, T_1 in the precedence graph.

There is no equivalent serial schedule because of the cycle.

EXERCISE 3 (CONT'D)

b) Is the following schedule serializable, recoverable and cascadeless?

T_1	T_2
	write(B)
read(A)	
write(A)	
	read(A)
commit	
	commit



Serializable?

Yes The equivalent serial schedule is $T_1 T_2$.

Recoverable?

Yes T_2 reads A, written by T_1 and commits after T_1.

Cascadeless?

No If T_1 aborts, T_2 must also abort.

How would you place the commit statements in order to make the schedule cascadeless?

T_1	T_2
	write(B)
read(A)	
write(A)	
commit	
	read(A)
	commit

EXERCISE 3 (CONT'D)

c) Is the following schedule recoverable and cascadeless?

T_1	T_2
	read(A)
read(A)	
write(A)	
	write(B)
	commit
commit	

Recoverable?

Yes No transaction reads data items after they have been written by the other.

Cascadeless?

Yes No transaction reads data items after they have been written by the other.

EXERCISE 4

Consider the following schedule consisting of three transactions T_1, T_2 and T_3. r_i denotes a read (of transaction T_i) and w_i is a write (of transaction T_i).

Schedule: $r_3(Z)$ $w_3(Z)$ $r_1(X)$ $r_2(Y)$ $w_2(Y)$ $w_1(X)$ $r_1(Y)$ $r_3(X)$

a) Show that the schedule is serializable by constructing the precedence graph.

b) Which is the equivalent serial schedule?

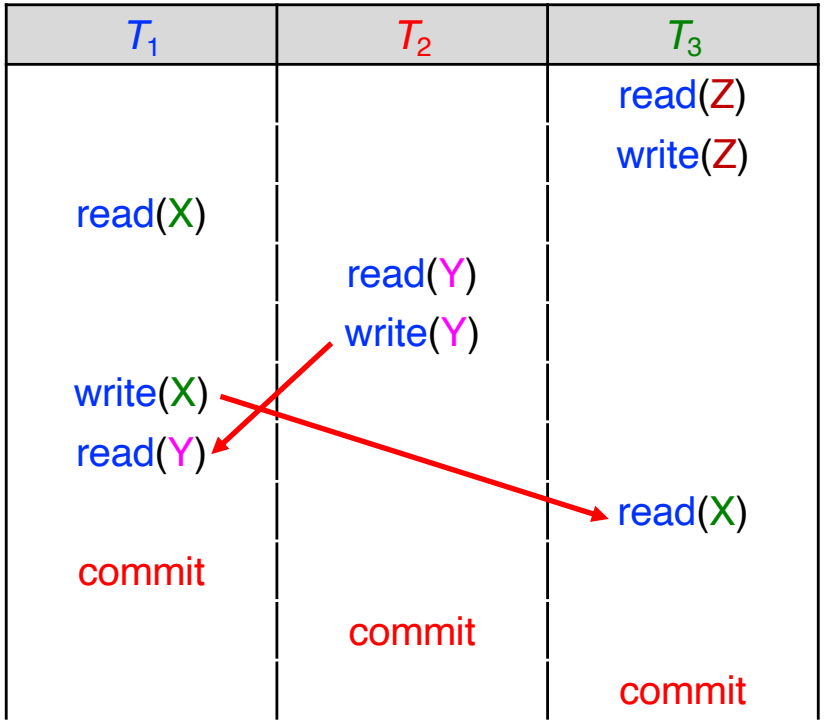
c) Modify the original schedule so it becomes recoverable, but not cascadeless by adding commit operations in the appropriate locations in the schedule.

d) Modify the schedule so it becomes both recoverable and cascadeless by adding commit operations in the appropriate locations in the schedule.

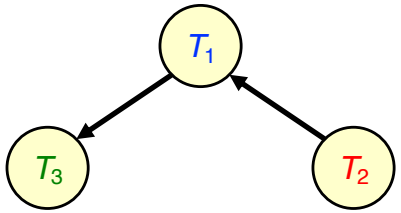
EXERCISE 4 (CONT'D)

Schedule: $r_3(Z)$, $w_3(Z)$, $r_1(X)$, $r_2(Y)$, $w_2(Y)$, $w_1(X)$, $r_1(Y)$, $r_3(X)$, c_1, c_2, c_3;

a) Show that the schedule is serializable by constructing the precedence graph.



Precedence Graph



b) What is the equivalent serial schedule?

T_2, T_1, T_3

EXERCISE 4 (CONT'D)

c) Modify the original schedule so it becomes recoverable, but not cascadeless by adding commit operations in the appropriate locations in the schedule

Recall: A schedule is recoverable if the commit of a transaction T_j that reads data items *previously written* by a transaction T_i appears after the commit operation of T_i.

T_1 reads Y written by T_2
$\Longrightarrow$ T_1 must commit after T_2.
T_2 does not read any data items written by other transactions.
T_3 reads X written by T_1
$\Longrightarrow$ T_3 must commit after T_1.

Schedule:

$r_3(Z)$, $w_3(Z)$, $r_1(X)$, $r_2(Y)$, $w_2(Y)$, $w_1(X)$, $r_1(Y)$, $r_3(X)$, c_2, c_1, c_3

T_1	T_2	T_3
		read(Z)
		write(Z)
read(X)		
	read(Y)	
	write(Y)	
write(X)		
read(Y)		
		read(X)
	commit	
commit		
		commit

EXERCISE 4 (CONT'D)

d) Rewrite the schedule so it becomes both recoverable and cascadeless by adding commit operations in the appropriate locations in the schedule.

Recall: A schedule is cascadeless if, for each pair of transactions T_i, T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears *before* the read operation of T_j.

- The commit of T_1 must appear *before* the read(X) of T_3.
- The commit of T_2 must appear *before* the read(Y) of T_1.
- The commit of T_3 must appear at the end of T_3.

Schedule:

$r_3(Z), w_3(Z), r_1(X), r_2(Y), w_2(Y), c_2, w_1(X), r_1(Y), c_1, r_3(X), c_3$

T_1	T_2	T_3
		read(Z)
		write(Z)
read(X)		
	read(Y)	
	write(Y)	
	commit	
write(X)		
read(Y)		
commit		
		read(X)
		commit