Bases de données

Prof. Giovanna Di Marzo Serugendo
Assistant: Mohammad Parhizkar
mohammad.parhizkar@unige.ch

Semestre: printemps 2019

session 9- 13/05/2019

Q.1

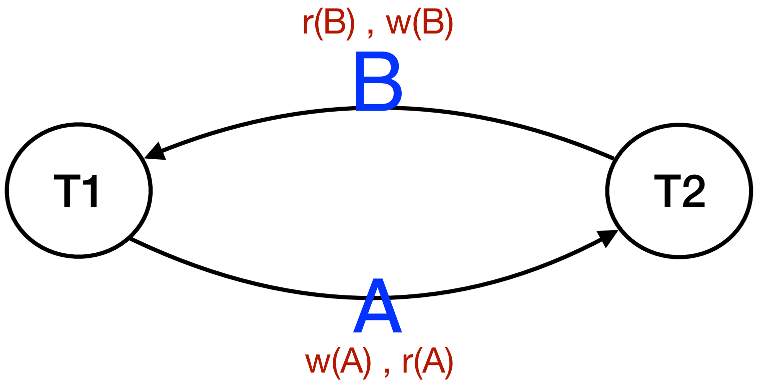
TP7-Solution

Different conflict scenarios:

Ti	Tj	
W(X)	R(X)	create arc Ti -> Tj
R(X)	W(X)	create arc Ti -> Tj
W(X)	W(X)	create arc Ti -> Tj

Schedule:

T1	T2
read (A)	
write (A)	
	read (A)
	read (B)
read (B)	
write (B)	



Serializability: Proof of correctness of the precedence graph test

www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/7-serializability/graph-conflict2.html

Q.1

TP7-Solution

b. No. T_1 might fail after T_2 already committed (and T_2 used A, which was produced by T_1).

Schedule:

T1	T2
read (A)	
write (A)	
	read (A)
	read (B)
	commit
read (B)	
write (B)	
commit	

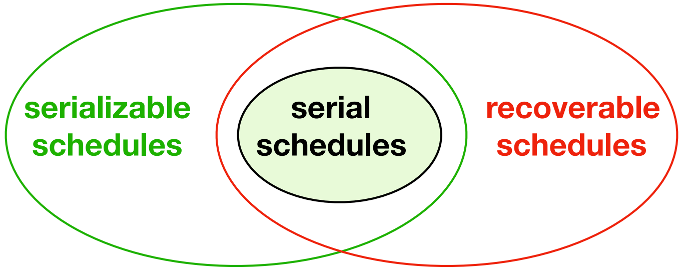
- A *serializable* **schedule** means that:
  When there is **no system failures**, the **(serializable) schedule** will **completes/results** in *consistent* **database state.**

- A *non* **recoverable schedule** means:
  When** there is a **system failures**, we **may** not **be able** to recover to a consistent **database state.

more in: http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/8-recv+serial/recoverable.html

Recoverability of Schedules

Example



A recoverable schedule is one where, for Ti and Tj, if Tj reads data item previously written by Ti , then the commit operation of Ti appears before the commit operation Tj.

serializability does not care about commit ordering

S1: *w2(A) w1(B) w1(A) r2(B) c1 c2*

recoverable, but not conflict-serializable

We will recover to the inconsistent state achieve by the execution of the non-serializable schedule S1.

S1': *w2(A) w1(B) w1(A) c1 r2(B) c2* ← **Cascadeless schedule**

S2: *w1(A) w1(B) w2(A) r2(B) c2 c1*

conflict-serializable, but not recoverable

S2': *w1(A) w1(B) w2(A) r2(B) c1 c2*

Recoverability of Schedules: *ACR*

Two different schedule types, based on recovery:

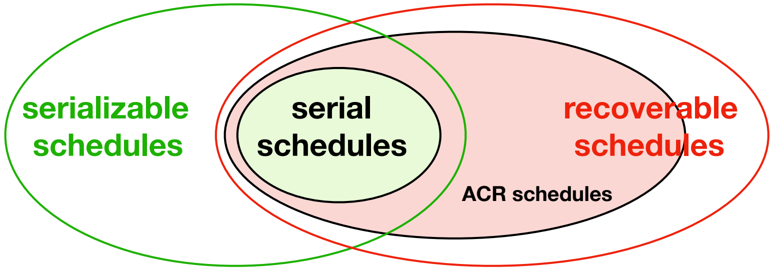
Let's add another strong condition

1. Recoverable schedule
2. Cascadeless schedule (*ACR: Avoid Cascading Rollback*)

When a single transaction failure leads to a series of **transaction rollbacks** is called *Cascading rollback.*

A transaction reads a data item after it is written by an uncommitted transaction.

S: *r1(A) w1(A) r2(B)* ***c1*** *r2(A)* ***c2***



Every Cascadeless schedule is also recoverable schedule.

Q.2

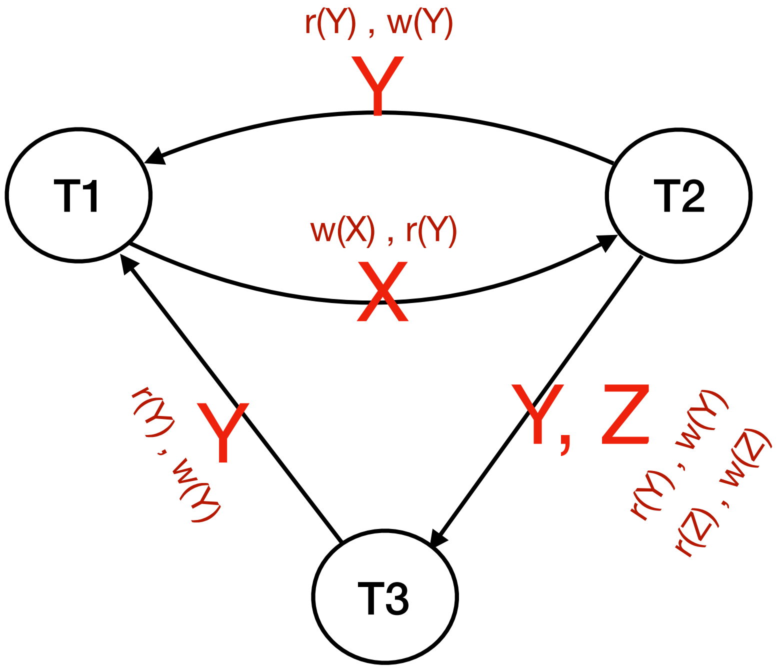
TP7-Solution

Different conflict scenarios:

Ti	Tj	
W(X)	R(X)	create arc Ti -> Tj
R(X)	W(X)	create arc Ti -> Tj
W(X)	W(X)	create arc Ti -> Tj

Schedule:

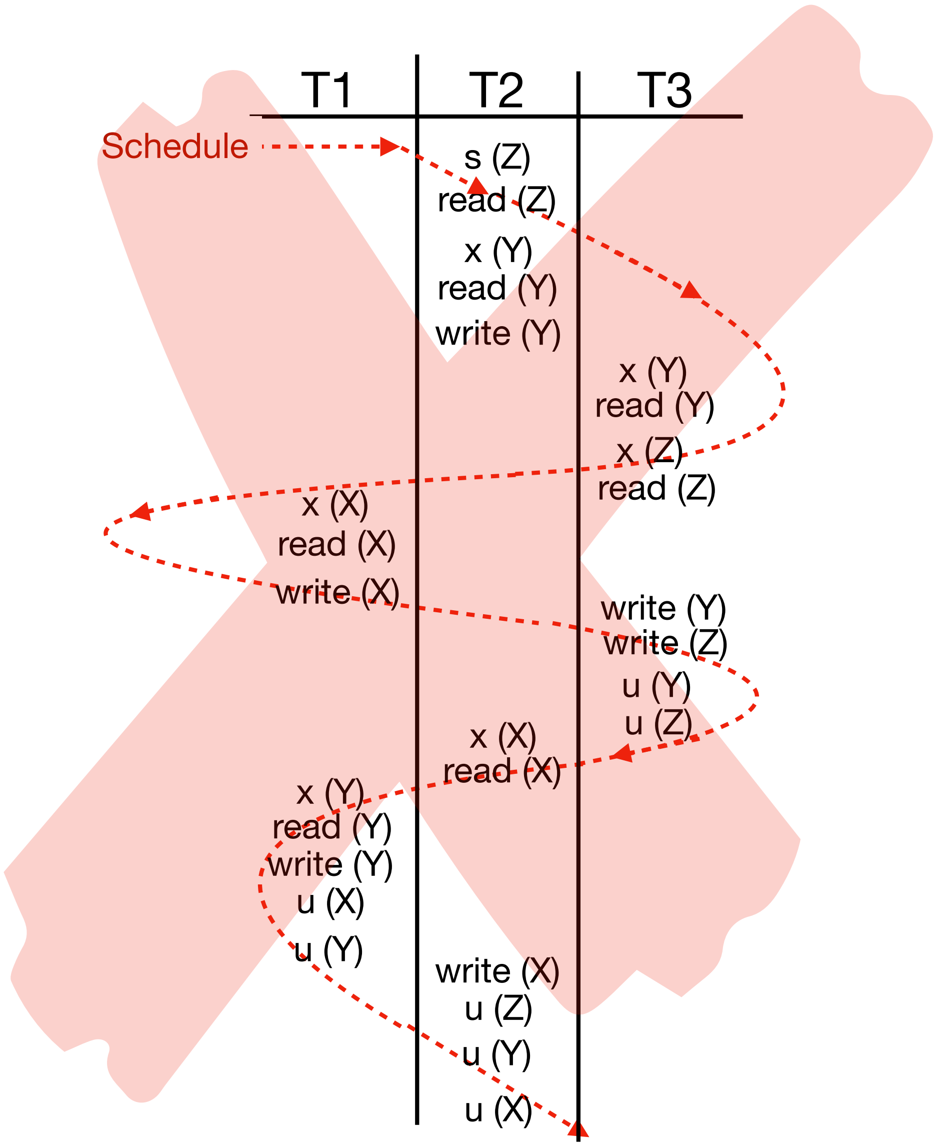
T1	T2	T3
	read (Z)	
	read (Y)	
	write (Y)	
		read (Y)
		read (Z)
read (X)		
write (X)		
		write (Y)
		write (Z)
	read (X)	
read (Y)		
write (Y)		
	write (X)	



a. The schedule is not conflict serializable, since the conflict graph contains cycles.

Q.2

TP7-Solution



T1	T2	T3
	s (Z)	
	read (Z)	
	x (Y)	
	read (Y)	
	write (Y)	
		x (Y)
		read (Y)
		x (Z)
		read (Z)
x (X)		
read (X)		
write (X)		
		write (Y)
		write (Z)
		u (Y)
		u (Z)
	x (X)	
	read (X)	
x (Y)		
read (Y)		
write (Y)		
u (X)		
u (Y)		
	write (X)	
	u (Z)	
	u (Y)	
	u (X)	

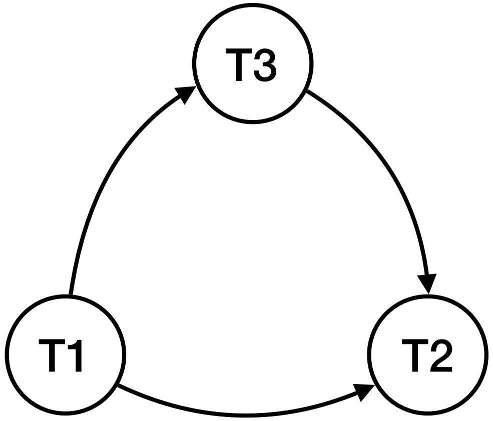
Schedule of Q2 after adding locks and unlocks in **2PL** protocol.

b. **No**. There are lots of conflicts between locks.

Serializability: Example

Schedule:

T1	T2	T3
read (X)		
read (Y)		
	read (X)	
	read (Z)	
write (Y)		
commit		
		read (Y)
		read (Z)
		write (Y)
		commit
	write (X)	
	write (Z)	
	commit	



conflict graph

The schedule is conflict serializable?

Yes. There is no cycle in this graph (**acyclic** graph).

Q.3

TP7-Solution

Serializability:

Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.

It assumes that all accesses to the database are done using read and write operations.

A schedule is called “**correct**” if we can find a serial schedule that is “equivalent” to it.

Q.3

TP7-Solution

- In this question, we want to know which schedule can be serializable.

- **As we can see, there are so many conflicts in all schedules.**

- We have two ways to check the serializability of the schedules:

1) Drawing the conflict graph and check if it contains any cycle or not.
2) By comparing the final result with the initial values and check the orders of transactions over the data sources.

Q.3

TP7-Solution

which one can have serializable behavior … ?

Schedule A:

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
Commit	
	Commit

Schedule B:

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

Schedule C:

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

Schedule D:

T1	T2
	R(A)
	W(A)
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(B)
	W(B)
	Commit

Schedule E:

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	

Schedule F:

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)
W(B))	
	Commit
Commit	

solution:

A, B, E

Q.3

TP7-Solution

A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
Commit	
	Commit

B

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

C

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

D

T1	T2
	R(A)
	W(A)
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(B)
	W(B)
	Commit

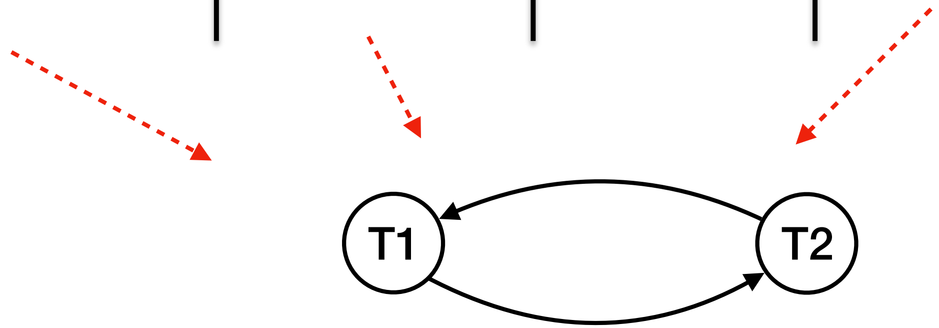
E

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	

F

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)
W(B))	
	Commit
Commit	

This two examples of reading uncommitted data (dirty read).



conflict graph

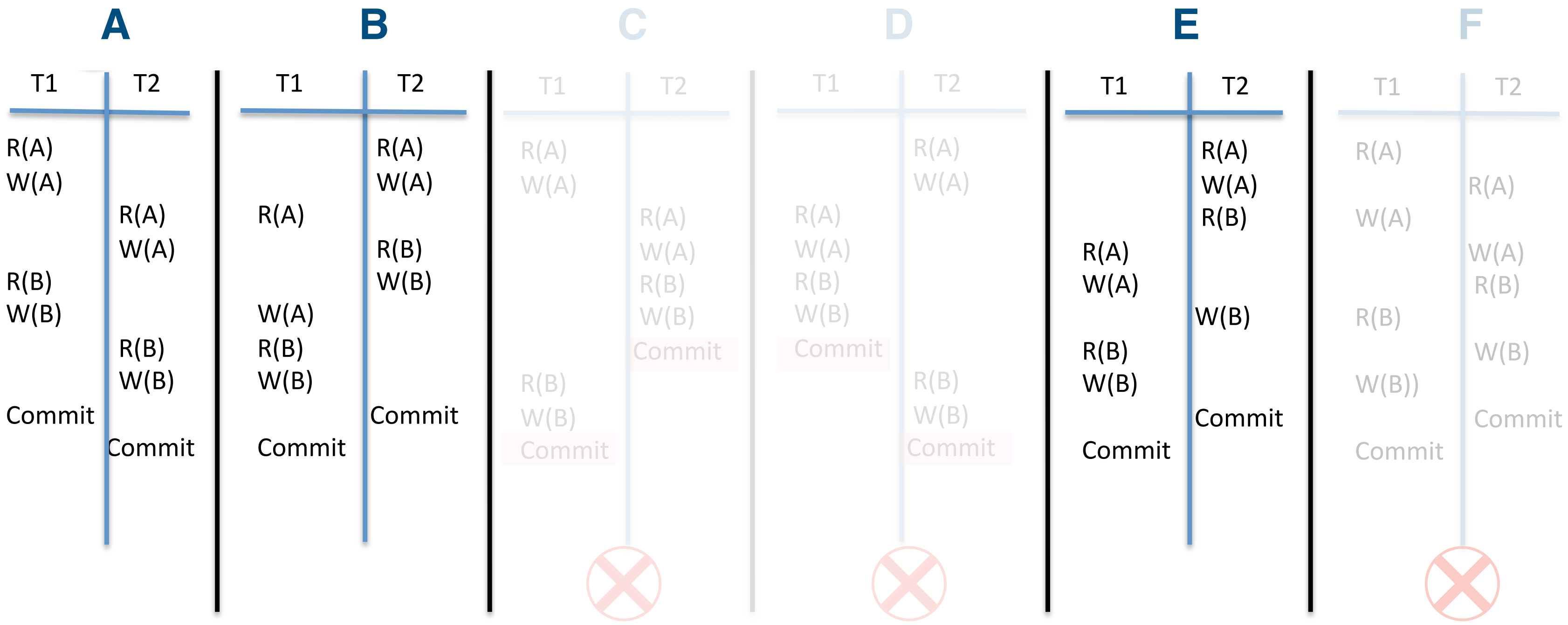
Q.3

another approach

TP7-Solution

T1: Read(A), Op$_{11}$(A), Write(A), Read(B), Op$_{12}$(B), Write(B), Commit

T2: Read(A), Op$_{21}$(A), Write(A), Read(B), Op$_{22}$(B), Write(B), Commit



A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
Commit	
	Commit

B

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

C

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

D

T1	T2
	R(A)
	W(A)
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(B)
	W(B)
	Commit

E

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	

F

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)
W(B))	
	Commit
Commit	

1. Suppose that A = 5 and B = 2 before T1 and T2

2. Suppose Op11(A) = A+1 ; Op12(B)=B*2 ; Op21(A)=2*A ; Op22(B)=1+B

T1: Read(A), A+1, Write(A), Read(B), B*2, Write(B), Commit

T2: Read(A), 2*A, Write(A), Read(B), 1+B, Write(B), Commit

Q.3

TP7-Solution

Two possible serial executions:

(A = 5 and B = 2 before T1 and T2)

1 All of T1 followed by all of T2:

A=5, B=2 ➔ T1 ➔ A=6, B=4 ➔ T2 ➔ **A=12, B=5**

2 All of T2 followed by all of T1:

A=5, B=2 ➔ T2 ➔ A=10, B=3 ➔ T1 ➔ **A=11, B=6**

Q.3

TP7-Solution

Generally, consider T1 and T2 simplified into two transactions, T(A) and T(B), based on each shared object, A and B. If simplified, but still dependent, transactions follow same serial order, such as T1(A),T2(A) and T1(B),T2(B)…

↓

Then the schedule is serializable.

Q.3

TP7-Solution

A=5, B=2

➔ T1(A) ➔ Read(A), A+1, Write(A) ➔ A=6

➔ T2(A) ➔ Read(A), 2*A, Write(A) ➔ **A=12**

➔ T1(B) ➔ Read(B), B*2, Write(B) ➔ B=4

➔ T2(B) ➔ Read(B), 1+B, Write(B) ➔ **B=5**

same serial order:

T1(A), T2(A) ✓

T1(B), T2(B)

same result with one of simple serial orders:

A=12, B=5 ✓

A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
Commit	
	Commit

B

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

C

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

D

T1	T2
	R(A)
	W(A)
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(B)
	W(B)
	Commit

E

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	

F

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)
W(B))	
	Commit
Commit	

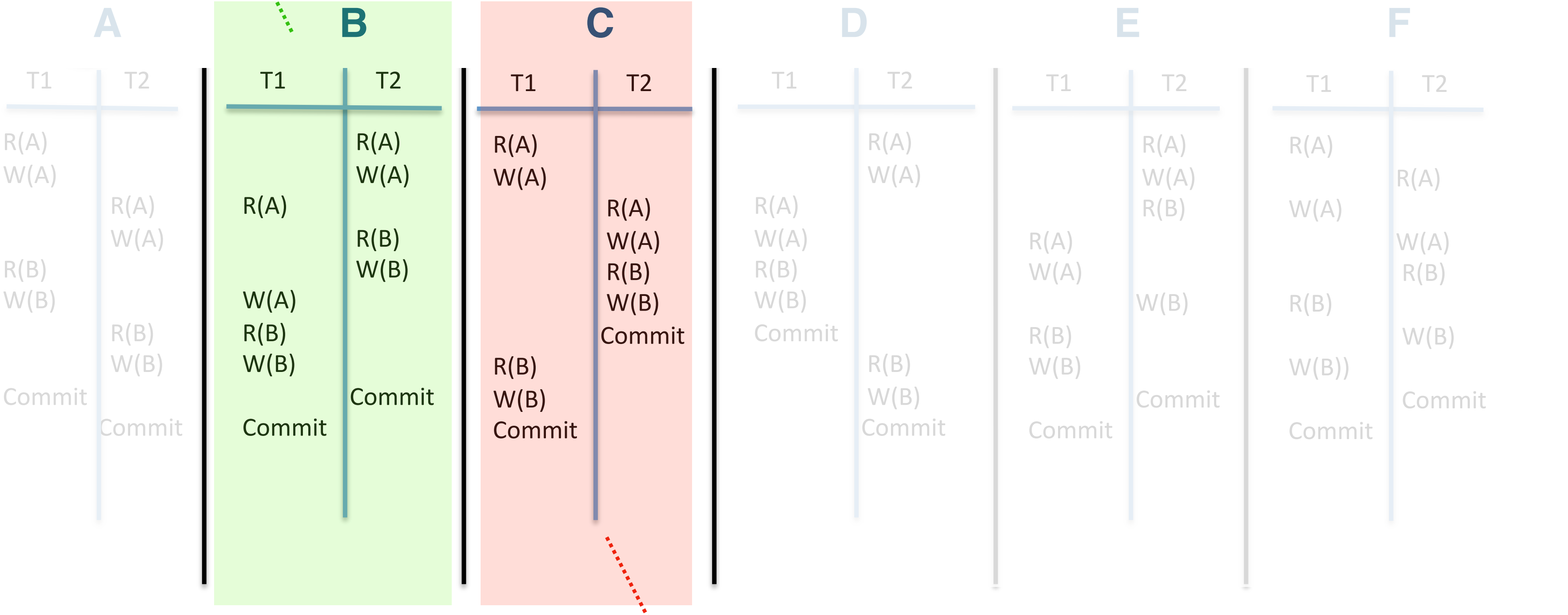
Q.3

TP7-Solution

A=5, B=2
→ T2(A) → Read(A), 2*A, Write(A) → A=10
→ T1(A) → Read(A)
→ T2(B) → Read(B), 1+B, Write(B) → B=3
→ T1(A) → A+1, Write(A)
→ T1(B) → Read(B), B*2 , Write (B) → **A=11, B=6**

same serial order:
T2(A), T1(A)
T2(B), T1(B) ✓

same result with one of simple serial orders:
A=11, B=6 ✓



A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
Commit	
	Commit

B

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

C

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

D

T1	T2
	R(A)
	W(A)
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(B)
	W(B)
	Commit

E

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	

F

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)
W(B))	
	Commit
Commit	

A=5, B=2
→ T1(A) → A=6, B=2 → T2(A) → **A=12**
→ T2(B) → B=3 → T1(B) → **B=6**

NOT same serial order:
T1(A), T2(A)
T2(B), T1(B) ✗

Q.3

TP7-Solution

A=5, B=2

➔ T2(A) ➔ Read(A), 2*A, Write(A)➔ A=10

➔ T2(B) ➔ Read(B)

➔ T1(A) ➔ Read(A), A+1, Write(A) ➔ **A=11**

➔ T2(B) ➔ 1+B, Write(B) ➔ B=3

➔ T1(B) ➔, Read(B), B*2 , Write (B) ➔ **B=6**

same serial order:

T2(A), T1(A) ✓

T2(B), T1(B)

same result with one of simple serial orders:

A=11, B=6 ✓

A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
Commit	
	Commit

B

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	Commit
Commit	

C

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

D

T1	T2
	R(A)
	W(A)
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(B)
	W(B)
	Commit

E

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	

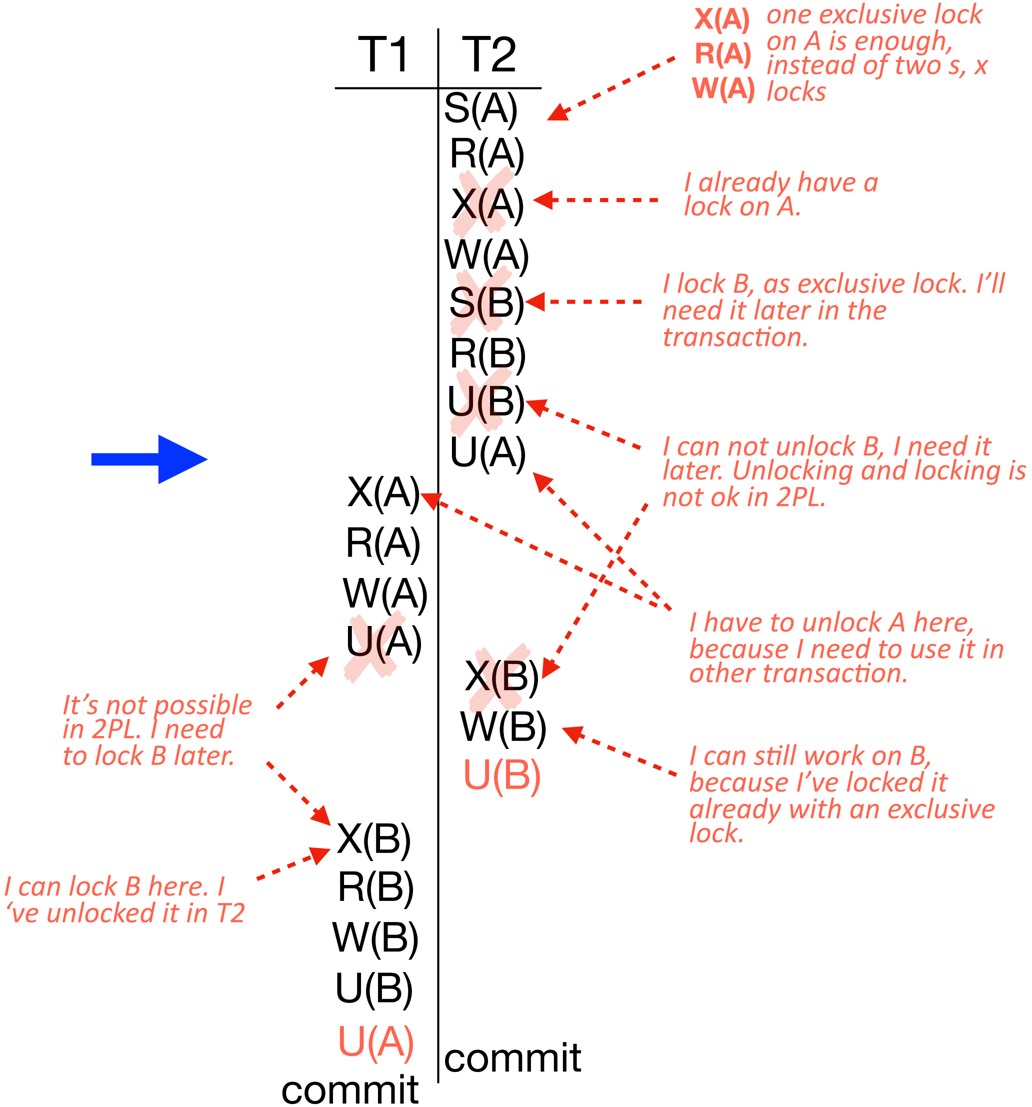
F

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)
	R(B)
R(B)	
	W(B)
W(B))	
	Commit
Commit	

2PL: Example

E

T1	T2
	R(A)
	W(A)
	R(B)
R(A)	
W(A)	
	W(B)
R(B)	
W(B)	
	Commit
Commit	



T1	T2
	S(A)
	R(A)
	~~X(A)~~
	W(A)
	~~S(B)~~
	R(B)
	~~U(B)~~
	U(A)
X(A)	
R(A)	
W(A)	
~~U(A)~~	
	~~X(B)~~
	W(B)
	U(B)
X(B)	
R(B)	
W(B)	
U(B)	
U(A)	
	commit
commit	

X(A) **R(A)** **W(A)** *one exclusive lock on A is enough, instead of two s, x locks*

I already have a lock on A.

I lock B, as exclusive lock. I'll need it later in the transaction.

I can not unlock B, I need it later. Unlocking and locking is not ok in 2PL.

I have to unlock A here, because I need to use it in other transaction.

I can still work on B, because I've locked it already with an exclusive lock.

It's not possible in 2PL. I need to lock B later.

I can lock B here. I 've unlocked it in T2

Each object has associated with a lock.

T1	T2
	X(A) +
	R(A)
	W(A)
	X(B) +
	R(B)
	U(A) -
+ X(A)	
R(A)	
W(A)	
	W(B)
	U(B) -
+ X(B)	
R(B)	
W(B)	
- U(B)	
- U(A)	
	commit
commit	

You can try on schedules A and B…

Thank you

&

Good luck in your exams!

For any question:

Email: *mohammad.parhizkar@unige.ch*

In person: *office 207*