CS3223 Tutorial 8

Gary Lim

Chapter Review

- ACID Transactions
- Serializability
- Recoverable, Cascadeless, Strict

ACID Transactions

- **Atomicity**:
- **Consistency**:
- **Isolation**:
- **Durability:**

ACID Transactions

- **Atomicity**: All actions in a Xact happen, or none of them happen
  - Guaranteed by using COMMIT after all actions has finished, or ABORT if some actions are not allowed to be executed
- **Consistency**:
- **Isolation**:
- **Durability:**

ACID Transactions

- **Atomicity**: All actions in a Xact happen, or none of them happen
  - Guaranteed by using COMMIT after all actions has finished, or ABORT if some actions are not allowed to be executed
- **Consistency**: If each Xact is consistent, and the DB was in a consistent state initially, then it should end up being consistent
- **Isolation**:
- **Durability:**

ACID Transactions

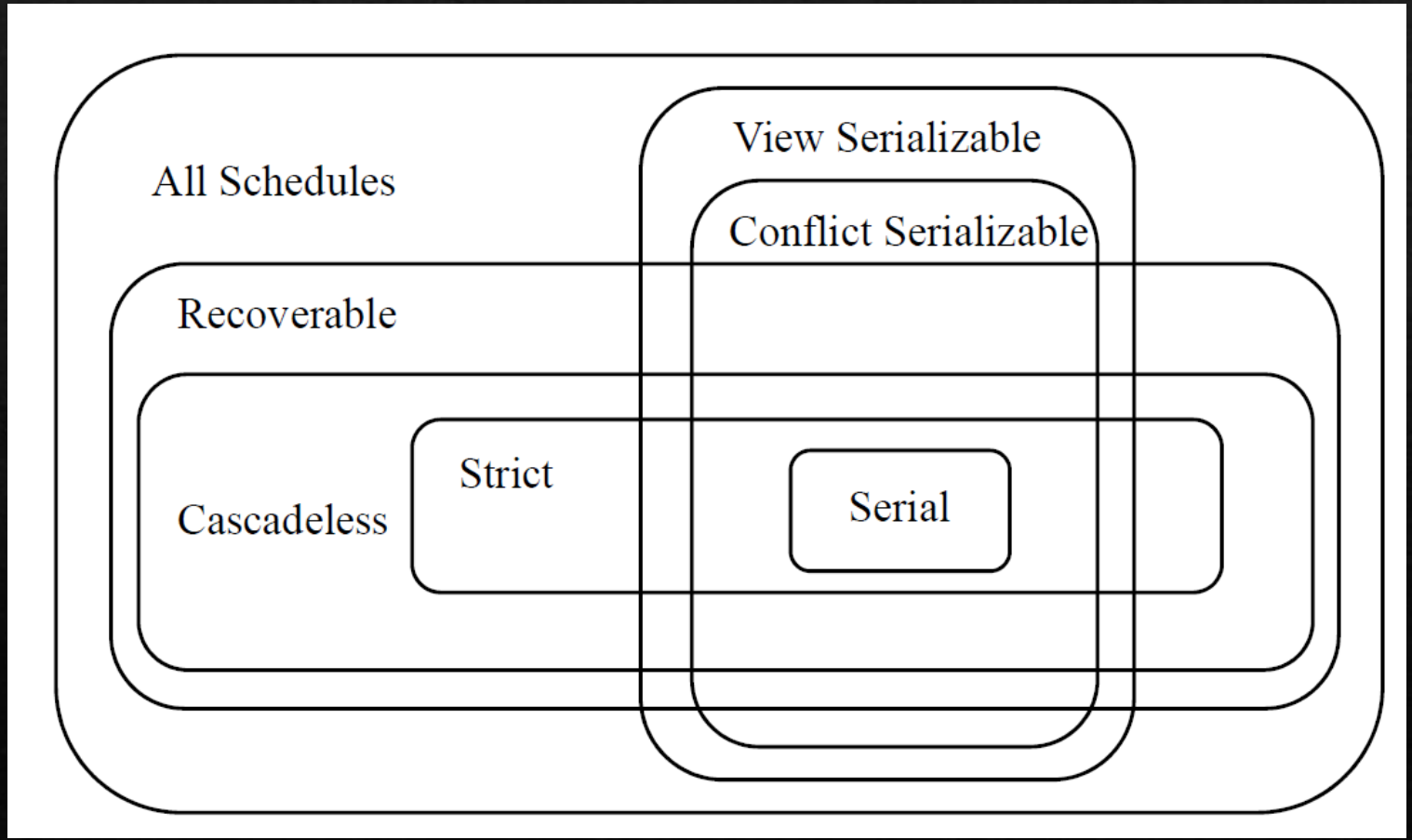
- **Atomicity**: All actions in a Xact happen, or none of them happen
  - Guaranteed by using COMMIT after all actions has finished, or ABORT if some actions are not allowed to be executed
- **Consistency**: If each Xact is consistent, and the DB was in a consistent state initially, then it should end up being consistent
- **Isolation**: Execution of one Xact is isolated from that of another Xacts
  - Net effect must be identical to executing all Xacts in some serial order
- **Durability:**

ACID Transactions

- **Atomicity**: All actions in a Xact happen, or none of them happen
  - Guaranteed by using COMMIT after all actions has finished, or ABORT if some actions are not allowed to be executed
- **Consistency**: If each Xact is consistent, and the DB was in a consistent state initially, then it should end up being consistent
- **Isolation**: Execution of one Xact is isolated from that of another Xacts
  - Net effect must be identical to executing all Xacts in some serial order
- **Durability:** If a Xact commits, its effects persist
  - Recovery schemes

Schedules

- Serial
- Conflict Serializable
- View Serializable
- Strict
- Cascadeless
- Recoverable



- **Serial**
- Conflict Serializable
- View Serializable
- Strict
- Cascadeless
- Recoverable

Serializability

Serial

- $S_1 = R_1(X), R_1(Y), W_1(Z), W_3(X), W_3(Z), R_2(Y), R_2(X), W_2(Z)$
- **No interleaving** of actions from different transactions
  - All transactions execute one after another (as though in an atomic manner)

T1	T2	T3
$R_1(X)$ $R_1(Y)$ $W_1(Z)$		
		$W_3(X)$ $W_3(Z)$
	$R_2(Y)$ $R_2(X)$ $W_2(Z)$	

- **Serial**
- Conflict Serializable
- View Serializable
- Strict
- Cascadeless
- Recoverable

Serializability

Serializable

- $S_1 = R_1(X), R_1(Y), W_3(X), W_3(Z), R_2(Y), W_1(Z), R_2(X), W_2(Z)$
- **effect** is **identical** to that of some complete **serial** schedule
  - $R_1(X) \rightarrow W_3(X) \rightarrow R_2(X)$ // preserved
  - $W_2(Z)$ performs last write on Z
  - Net effect is the same as serial schedule!

Equivalent serial schedule:

- $S_2 = R_1(X), R_1(Y), W_1(Z), W_3(X), W_3(Z), R_2(Y), R_2(X), W_2(Z)$

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
		$W_3(X)$
		$W_3(Z)$
	$R_2(Y)$	
$W_1(Z)$		
	$R_2(X)$	
	$W_2(Z)$	

- **Serial**
- Conflict Serializable
- View Serializable
- Strict
- Cascadeless
- Recoverable

Serializability

Serializable BUT NOT Conflict-serializable

- $S_1 = R_1(X), R_1(Y), W_3(X), W_3(Z), R_2(Y), W_1(Z), R_2(X), W_2(Z)$
- **effect** is **identical** to that of some complete **serial** schedule
  - $R_1(X) \rightarrow W_3(X) \rightarrow R_2(X)$ // preserved
  - $W_2(Z)$ performs last write on Z
  - Net effect is the same as serial schedule!
- Not conflict serializable
  - $R_1(X) \rightarrow W_3(X)$
  - $W_3(Z) \rightarrow W_1(Z)$

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
		$W_3(X)$
		$W_3(Z)$
	$R_2(Y)$	
$W_1(Z)$		
	$R_2(X)$	
	$W_2(Z)$	

Serializability

- Serial
- **Conflict Serializable**
- View Serializable
- Strict
- Cascadeless
- Recoverable

Conflict Serializable

- $S_1 = R_1(X), R_1(Y), W_3(X), W_1(Z), R_2(Y), W_3(Z), R_2(X), W_2(Z)$
- it is conflict equivalent to some serial schedule
  - $R_1(X) \rightarrow W_3(X) \rightarrow R_2(X)$ // preserved
  - $W_1(Z) \rightarrow W_3(Z) \rightarrow W_2(Z)$ // preserved

Conflict equivalent serial schedule:

- $S_2 = R_1(X), R_1(Y), W_1(Z), W_3(X), W_3(Z), R_2(Y), R_2(X), W_2(Z)$

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
		$W_3(X)$
$W_1(Z)$		
	$R_2(Y)$	
		$W_3(Z)$
	$R_2(X)$	
	$W_2(Z)$	

- Serial
- Conflict Serializable
- **View Serializable**
- Strict
- Cascadeless
- Recoverable

Serializability

View serializable

- $S_1 = R_1(X), R_1(Y), W_3(X), W_3(Z), R_2(Y), W_1(Z), R_2(X), W_2(Z)$
- **View equivalent** to some serial schedule
  - Ti reads the same initial value of A in both schedules
  - Ti reads the value of A written by Tj in both schedules
  - The same xact Ti performs the final write on A

View equivalent serial schedule:

- $S_2 = R_1(X), R_1(Y), W_1(Z), W_3(X), W_3(Z), R_2(Y), R_2(X), W_2(Z)$

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
		$W_3(X)$
		$W_3(Z)$
	$R_2(Y)$	
$W_1(Z)$		
	$R_2(X)$	
	$W_2(Z)$	

- Serial
- Conflict Serializable
- **View Serializable**
- Strict
- Cascadeless
- Recoverable

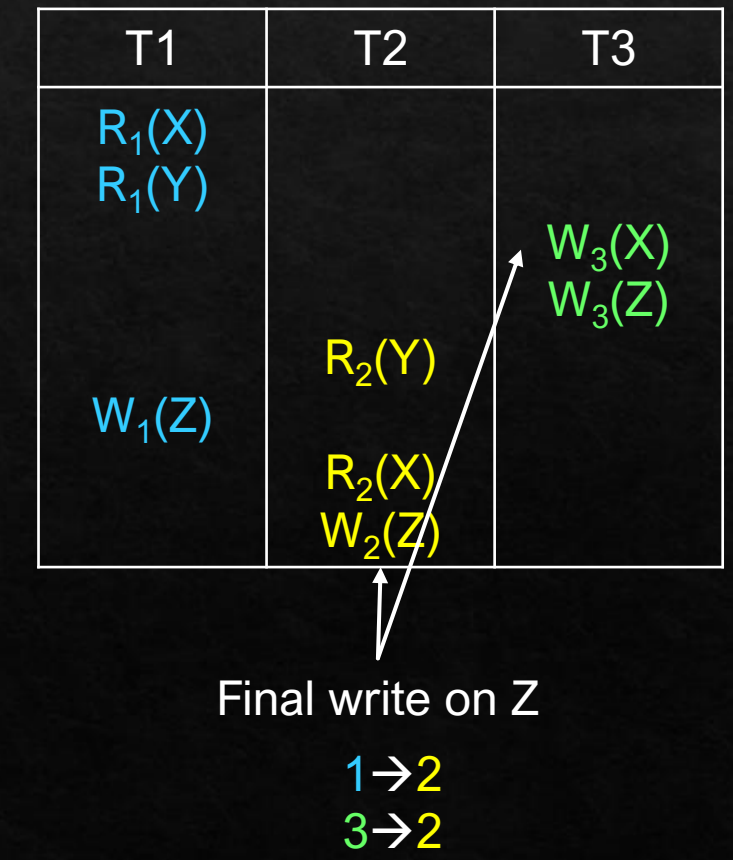
Serializability

View serializable

- S_1 = $R_1(X)$, $R_1(Y)$, $W_3(X)$, $W_3(Z)$, $R_2(Y)$, $W_1(Z)$, $R_2(X)$, $W_2(Z)$
- **View equivalent** to some serial schedule
  - Ti reads the same initial value of A in both schedules
  - Ti reads the value of A written by Tj in both schedules
  - The same xact Ti performs the final write on A

View equivalent serial schedule:

- S_2 = $R_1(X)$, $R_1(Y)$, $W_1(Z)$, $W_3(X)$, $W_3(Z)$, $R_2(Y)$, $R_2(X)$, $W_2(Z)$



- Serial
- Conflict Serializable
- **View Serializable**
- Strict
- Cascadeless
- Recoverable

Serializability

View serializable

- S_1 = $R_1(X)$, $R_1(Y)$, $W_3(X)$, $W_3(Z)$, $R_2(Y)$, $W_1(Z)$, $R_2(X)$, $W_2(Z)$
- **View equivalent** to some serial schedule
  - Ti reads the same initial value of A in both schedules
  - Ti reads the value of A written by Tj in both schedules
  - The same xact Ti performs the final write on A

View equivalent serial schedule:

- S_2 = $R_1(X)$, $R_1(Y)$, $W_1(Z)$, $W_3(X)$, $W_3(Z)$, $R_2(Y)$, $R_2(X)$, $W_2(Z)$

T1 reads initial value (before T3 writes) 1→3

T1	T2	T3
$R_1(X)$ $R_1(Y)$		
		$W_3(X)$ $W_3(Z)$
	$R_2(Y)$	
$W_1(Z)$		T2 reads from T3 3→2
	$R_2(X)$ $W_2(Z)$	

- Serial
- Conflict Serializable
- View Serializable
- **Strict**
- Cascadeless
- Recoverable

Other Schedule Types

strict is like a cascadeless but for both read and write

Strict

- $R_1(X), R_1(Y), R_2(Y), W_3(X), W_3(Z), c3, R_2(X), W_1(Z), c1, W_2(Z), c2$
- A schedule S is a **strict** schedule if for every $W_i(O)$ in S, O is **not read** or **written** by another Xact until T_i either **aborts** or **commits**
  - If $W_1(Z) \rightarrow W_2(Z)$, then $W_1(Z), c1 \ldots W_2(Z)$
    - T2 only writes to Z after T1 commits $W_1(Z)$
  - If $W_3(X) \rightarrow R_2(X)$, then $W_3(X), c3 \ldots R_2(X)$
    - T2 only reads X after T3 commits $W_3(X)$

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
	$R_2(Y)$	
		$W_3(X)$
		$W_3(Z)$
		commit
	$R_2(X)$	
$W_1(Z)$		
commit		
	$W_2(Z)$	
	commit	

Other Schedule Types

- Serial
- Conflict Serializable
- View Serializable
- Strict
- **Cascadeless**
- Recoverable

Cascadeless (but not strict)

- $R_1(X), R_1(Y), R_2(Y), W_3(X), W_3(Z), c3, R_2(X), W_1(Z), W_2(Z), c2 , c1$
- A schedule S is a cascadeless schedule if whenever T_i reads from T_j in S, $Commit_j$ must precede this read action
  - ~~If $W_1(Z) \rightarrow W_2(Z)$, then $W_1(Z), c1 \ldots W_2(Z)$~~
    - ~~T2 only writes to Z after T1 commits $W_1(Z)$~~
  - If $W_3(X) \rightarrow R_2(X)$, then $W_3(X), c3 \ldots R_2(X)$
    - T2 only reads X after T3 commits $W_3(X)$

Drop the requirement on write after write

T1	T2	T3
$R_1(X)$ $R_1(Y)$		
	$R_2(Y)$	
		$W_3(X)$ $W_3(Z)$ commit
	$R_2(X)$	
$W_1(Z)$		
	$W_2(Z)$ commit	
commit		

Other Schedule Types

- Serial
- Conflict Serializable
- View Serializable
- Strict
- Cascadeless
- **Recoverable**

Recoverable (but not cascadeless)

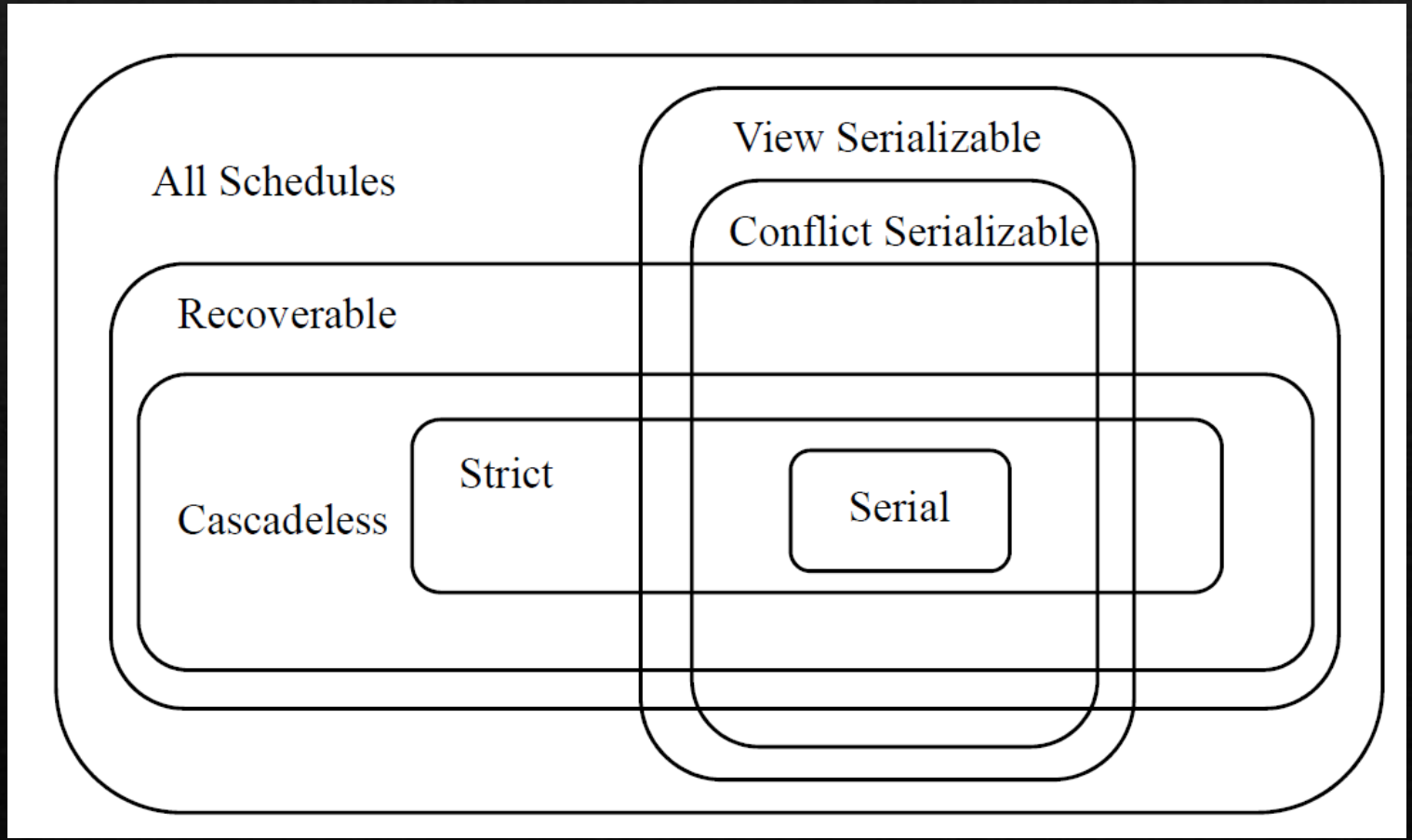
- $R_1(X)$, $R_1(Y)$, $R_2(Y)$, $W_3(X)$, $W_3(Z)$, $R_2(X)$, c3, $W_1(Z)$, $W_2(Z)$, c2 , c1
- A schedule S is said to be a recoverable schedule if for every Xact T that commits in S, T must commit after T′ if T reads from T′
  - If $W_3(X)$ → $R_2(X)$, then c3 … c2
    - T2 only commits after T3 commits

overable only checks commits

T1	T2	T3
$R_1(X)$		
$R_1(Y)$		
	$R_2(Y)$	
		$W_3(X)$
		$W_3(Z)$
	$R_2(X)$	
		commit
$W_1(Z)$		
	$W_2(Z)$	
	commit	
commit		

Schedules

- Serial
- Conflict Serializable
- View Serializable
- Strict
- Cascadeless
- Recoverable



Q1

- a. $W_1(X)$, $R_2(X)$, $W_1(X)$, c_2, c_1
  - S1. Not view serializable cos 1→2, and 2→1. Not recoverable because to be recoverable, T2 must commit after T1 (since T2 read from T1), but this is not true
- b. $R_1(X)$, $R_1(Y)$, $W_1(X)$, $R_2(Y)$, $W_3(Y)$, $W_1(X)$, $R_2(Y)$, c_3, c_2, c_1
  - S4. Not view serializable. Based on object Y, 2→3 (T2 reads initial value of Y), 3→2 (T2 reads Y from T3). Recoverable because T2 read from T3, and T2 committed after T3 commits. Not cascadeless: T2 did not read Y after T3 commits $W_3(Y)$. i.e. $W_3(Y)$, c_3, … $R_2(Y)$
- c. $W_1(X)$, $R_2(Y)$, $R_1(Y)$, $R_2(X)$, c_1, c_2
  - S6. Conflict serializable because we can have W1(X), R1(Y), R2(Y), R2(X). Recoverable because T2 read from T1, and T2 committed after T1 commits. Not cascadeless, same reason as part b
- d. $R_1(X)$, $R_2(X)$, $W_1(X)$, c_1, $W_2(X)$, c_2
  - S10. Not view serializable because 1→2 (T1 performs initial read, before T2 writes X, and because T2 must perform final write on X), and 2→1 (because T2 performs initial read, before T1 writes X). Strict because whenever T2 writes to X, it happens only after T1 has committed.

Q2

Give an example of a schedule with two or more transaction with the following three properties:

- T1 commits before T2 starts.
- The schedule is conflict serializable.
- In any equivalent serial schedule, T2 **must come before** T1 (there may be other transactions between T2 and T1).

Q2

Enforce this

- T2 → T3 (on some object X)
- T3 → T1 (on some object Y)
- S_1: $W_3(X)$, $R_1(X)$, c1, $R_2(Y)$, $W_3(Y)$
- S_2: $R_2(Y)$, $W_3(X)$, $W_3(Y)$, $R_1(X)$ // conflict equivalent serial schedule

Other alternatives

- $W_3(A)$, $W_1(A)$, c1, $W_2(B)$, c2, $W_3(B)$, c3

Q3a

- Consider the following transactions

T0	read(A) read(B) if A = 0 then B ← B+1 write(B)	**T1:**	read(B) read(A) if B = 0 then A ← A+1 write(A)

  - Show that every serial execution involving these two transactions preserves the consistency of the database.

Q3a

- Consider the following transactions

T0	read(A) read(B) if A = 0 then B ← B+1 write(B)	**T1:**	read(B) read(A) if B = 0 then A ← A+1 write(A)

  - Show that every serial execution involving these two transactions preserves the consistency of the database.
    - Show that T0→T1 and T1→T0 (symmetrical case) preserves the consistency requirement

Q3a

- If T0 → T1

T0	read(A) read(B) if A = 0 then B ← B+1 write(B)	A = 0 B = 0 B = 1 A = 0, B = 1
T1	read(B) read(A) if B = 0 then A ← A+1 write(A)	B = 1 A = 0 - A = 0, B = 1

- If T1 → T0

T1	read(B) read(A) if B = 0 then A ← A+1 write(A)	B = 0 A = 0 A = 1 A = 1, B = 0
T0	read(A) read(B) if A = 0 then B ← B+1 write(B)	A = 1 B = 0 - A = 1, B = 0

consistency requirement
- **A = 0** or **B = 0**
- A = B = 0 be the initial values

Q3b

Show a concurrent execution of T0 and T1 which produces a non-serializable schedule.

- S_1:

Q3b

Show a concurrent execution of T0 and T1 which produces a non-serializable schedule.

- S_1: **$R_0(A)$**, $R_0(B)$, **$R_1(B)$**, $R_1(A)$, **$W_0(B)$**, **$W_1(A)$**

Q3c

Is there a concurrent execution of T0 and T1 which produces a serializable schedule?

Q3c

- $R_0(A)$, $R_0(B)$, $R_1(B)$, $R_1(A)$, $W_0(B)$, $W_1(A)$
- To make it serializable, we must somehow have 0→1 (or 1→0)
- Let's try 0→1, by moving $W_0(B)$
- $R_0(A)$, $R_0(B)$, $W_0(B)$, $R_1(B)$, $R_1(A)$, $W_1(A)$
- But this is not a concurrent execution that we are looking for(no interleaving executions)
- It's the same outcome for 1→0
- No, we cannot achieve a concurrent execution that produces a serializable schedule

Q3c

- There is no parallel execution resulting in a serializable schedule. From part(a), we know that a serializable schedule results in A=0 OR B=0. Suppose we start with T0 read(A). Then when the schedule ends, no matter when we run the steps of T1, B=1. Now suppose we start executing T1 prior to completion of T0. Then T1 read(B) will give B a value of 0. So, when T1 completes, A=1. Thus, B=1 AND A=1. We get similar logic for starting with T1 read(B).

T0	T1

read(A)
read(B)
if A = 0 then B ← B+1
 	
write(B) // B=1
	
	read(B) // B=0
	
	read(A)
	if B = 0 then A ← A+1
	write(A) // A=1

As long as the yellow line happens before the blue line, the data will not be consistent

Q4

Consider the schedule S:

- $R_2(B), W_2(A), R_1(A), R_3(A), W_1(B), W_2(B), W_3(B)$

Is S view-serializable?

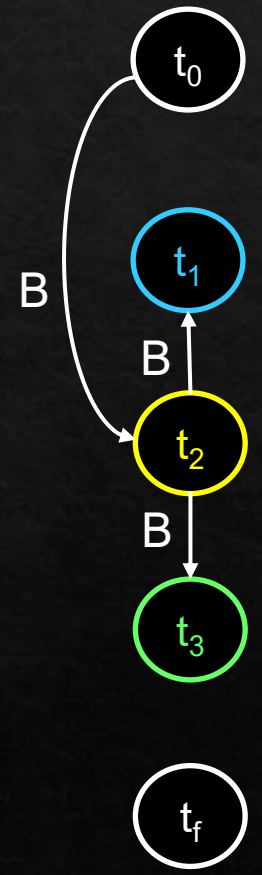
Q4

Consider the schedule S:

- $R_2(B)$, $W_2(A)$, $R_1(A)$, $R_3(A)$, $W_1(B)$, $W_2(B)$, $W_3(B)$

Is S view-serializable?

- T2 reads initial value of B (before anyone writes): 2→1, 2→3



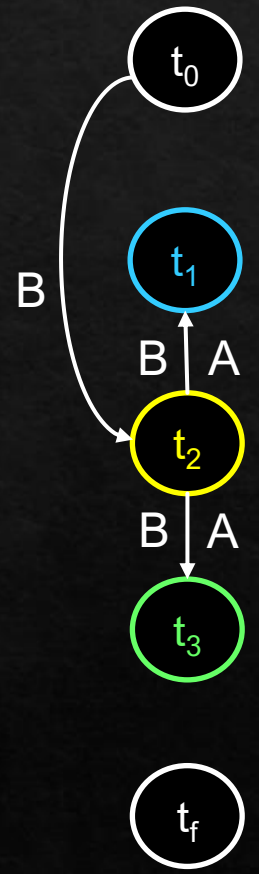
Q4

Consider the schedule S:

- $R_2(B)$, $W_2(A)$, $R_1(A)$, $R_3(A)$, $W_1(B)$, $W_2(B)$, $W_3(B)$

Is S view-serializable?

- T2 reads initial value of B (before anyone writes): 2→1, 2→3
- T1 reads value of A from T2: 2→1
- T3 reads value of A from T2: 2→3



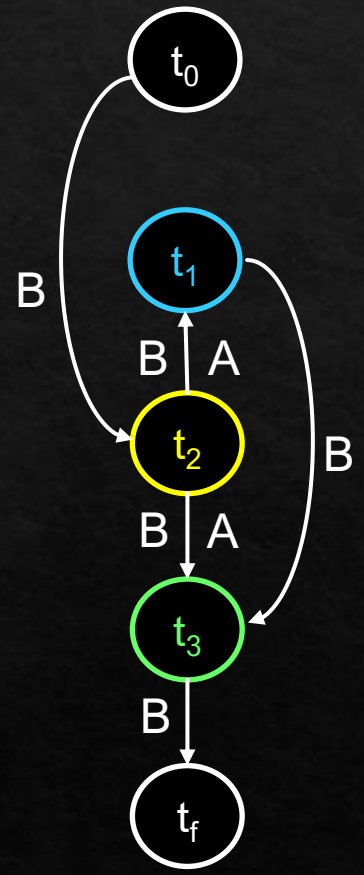
Q4

Consider the schedule S:

- $R_2(B)$, $W_2(A)$, $R_1(A)$, $R_3(A)$, $W_1(B)$, $W_2(B)$, $W_3(B)$

Is S view-serializable?

- T2 reads initial value of B (before anyone writes): 2→1, 2→3
- T1 reads value of A from T2: 2→1
- T3 reads value of A from T2: 2→3
- T3 performs final write on B (and no writes after that): 1→3, 2→3



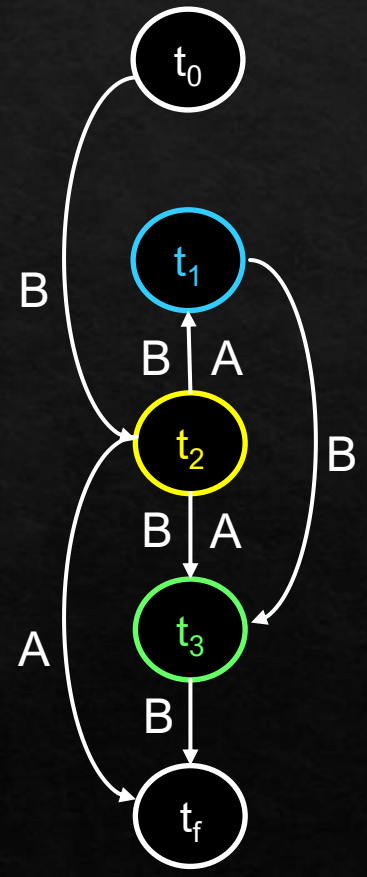
Q4

Consider the schedule S:

- $R_2(B)$, $W_2(A)$, $R_1(A)$, $R_3(A)$, $W_1(B)$, $W_2(B)$, $W_3(B)$

Is S view-serializable?

- T2 reads initial value of B (before anyone writes): 2→1, 2→3
- T1 reads value of A from T2: 2→1
- T3 reads value of A from T2: 2→3
- T3 performs final write on B (and no writes after that): 1→3, 2→3
- T2 performs final write on A



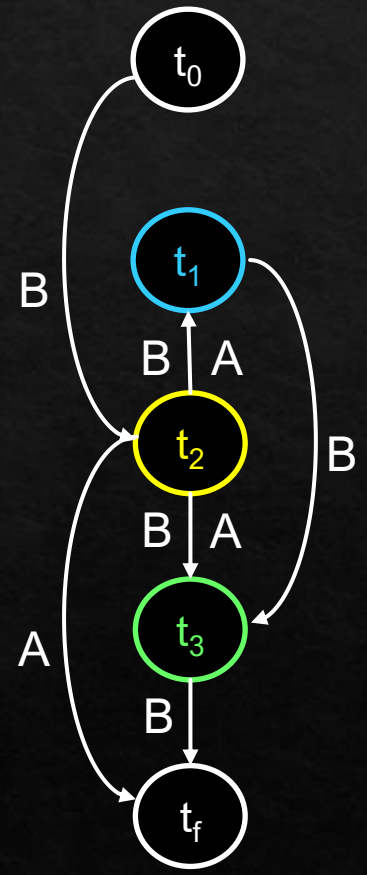
Q4

Consider the schedule S:

- $R_2(B)$, $W_2(A)$, $R_1(A)$, $R_3(A)$, $W_1(B)$, $W_2(B)$, $W_3(B)$

Is S view-serializable?

- T2 reads initial value of B (before anyone writes): 2→1, 2→3
- T1 reads value of A from T2: 2→1
- T3 reads value of A from T2: 2→3
- T3 performs final write on B (and no writes after that): 1→3, 2→3
- View equivalent serial schedule: 2→1→3
  - $R_2(B)$, $W_2(A)$, $W_2(B)$, $R_1(A)$, $W_1(B)$, $R_3(A)$, $W_3(B)$



Q4

Is S conflict serializable?

Blind writes

- $R_2(B)$, $W_2(A)$, $R_1(A)$, $R_3(A)$, $W_1(B)$, $W_2(B)$, $W_3(B)$
- S is not conflict serializable.
  - $W_1(B)$, $W_2(B)$ implies $1 \rightarrow 2$ but
  - $W_2(A)$, $R_1(A)$ suggests $2 \rightarrow 1$)
- But it is still view serializable.
  - View serializable schedule accepts blind writes (writes that are not read afterwards)

End