Database Systems

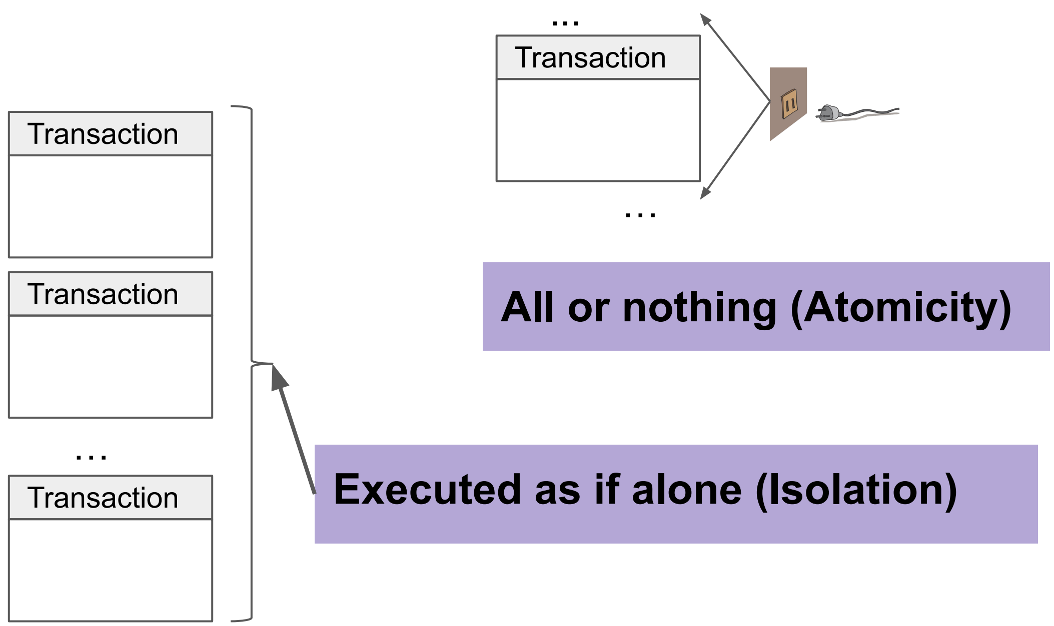
Lab 9 - Transactions

Today

- **Recap**
- **Atomicity**
- Isolation

Transaction

Transaction = A sequence of operations, executed together as **<u>one indivisible unit</u>**



Atomicity

- Crash before COMMIT
  - When recover
    - Nothing written to disk
- Crash during COMMIT
  - When recover:
    - Either nothing written to disk
    - Or, all updates written to disk
- Crash after COMMIT
  - When recover
    - All written updates written to disk

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

Atomicity

- All or nothing:
  - Crashes
  - Then recovers

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

A=1	**B=8**	**C=7**

Crash here

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

A=1	**B=8**	**C=7**

Atomicity

- All or nothing:
  - Crashes
  - Then recovers

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

A=1	B=8	C=7

Crash here →

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

A=1	B=8	C=7

OR

A=8	B=9	C=7

Atomicity

- All or nothing:
  - Crashes
  - Then recovers

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

A=1	**B=8**	**C=7**

Crash here

...
BEGIN
Write(A,8)
Write(B,9)
COMMIT
...

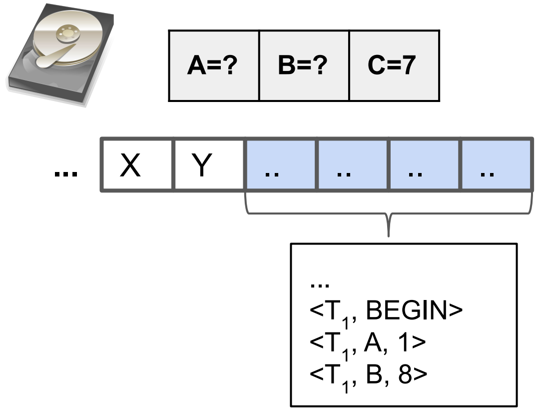
A=8	**B=9**	**C=7**

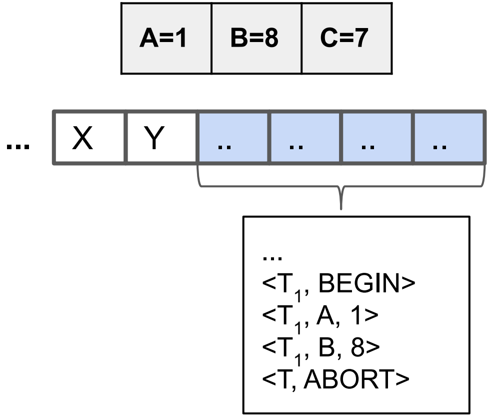
Undo Logging

- Executing a transaction
  - Start tx: write `<T,BEGIN>` to the log in memory
  - Update a value X: write `<T,X,oldVal>` to the log in memory
- **Rule 1:** before writing an update of X to disk:
  - Must write `<T,X,oldVal>` to the log on disk
- **Rule 2:** when commit:
  - Make sure all log entries are on disk
  - Make sure all updates are on disk
  - Then write `<T,COMMIT>` to the log on disk

Undo Logging

- Recovery
  - Find ***uncompleted*** transactions:
    - Ones without `<T,COMMIT> or <T,ABORT>` entries
  - Scanning the log backward
    - If see `<T,X,oldVal>` and T is uncompleted
      - Set X to `oldVal`
  - For each uncompleted T
    - Write `<T,ABORT>`





Checkpoints

- Undo logging

```
…
<T1, BEGIN>
…
<T2, BEGIN>
..
<START CHKPT (T1, T2)>
…
…
…
<END CHKPT (T1, T2)>
...
```

Other transactions may have started

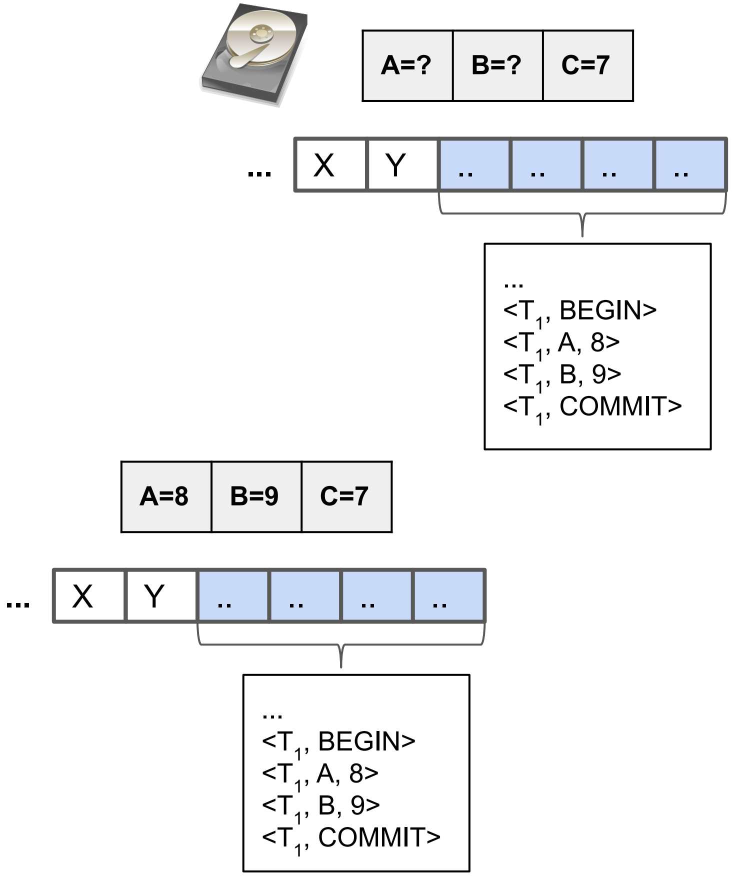
T1, T2 are committed at this point

Redo Logging

- Executing a transaction
  - Start tx: write `<T,BEGIN>` to the log in memory
  - Update a value X: write `<T,X,newVal>` to the log in memory
- **Rule 1:** before writing an update of X to disk:
  - `<T,X,newVal>` must be on disk
  - `<T,COMMIT>` must be on disk

Redo Logging

- Recovery
  - Find committed transactions:
    - Ones with `<T,COMMIT>`
  - Scanning the log forward
    - If see `<T,X,newVal>` and T is committed
      - Set X to newVal



Checkpoint

- Redo Logging

```
…
<T1, BEGIN>
…
<T2, BEGIN>
..
<T3, COMMIT>
...
<START CHKPT (T1, T2)>
…
…
…
<END CHKPT (T1, T2)>
...
```

T3’s updates are on disk at this point

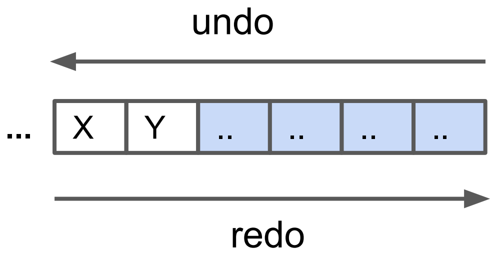
Undo/Redo Logging

- Executing a transaction
  - Start tx: write `<T,BEGIN>` to the log in memory
  - Update a value X: write `<T,X,oldVal,newVal>` to the log in memory
- **Rule 1:** before writing an update of X to disk:
  - `<T,X,oldVal,newVal>` must be on disk

	STEAL	NO STEAL
FORCE	Undo Logging	Not good
NO FORCE	Undo/Redo Logging	Redo Logging

Undo/Redo Logging

- Recovery: 2 passes
  - Backward pass:
    - Undo uncompleted transactions
    - Like undo logging
  - Forward pass:
    - Redo committed transactions
    - Like redo logging



Checkpoint

- Undo/Redo Logging

```
…
<T1, BEGIN>
…
<T2, BEGIN>
..
<T3, COMMIT>
...
<START CHKPT (T1, T2)>
…
…
…
<END CHKPT (T1, T2)>
...
```

All updates made before `<START CHKPT (T1,T2)>` are on disk

Exercise 1

T1:
A=10
B=20

T2:
A=40
C=30
A=50

T3:
B=75

1. <T1, BEGIN>
2. <T1, A, xxx>
3. <T1, B, xxx>
4. <T1, COMMIT>
5. <T2, BEGIN>
6. <T2, A, xxx>
7. <T2, C, xxx>
8. <T2, A, xxx>
9. <T2, COMMIT>
10. <T3, BEGIN>
11. <T3, B, xxx>

Given 3 transactions T1, T2, T3. Initially, A=B=C=0

They are executed using ***undo logging***, with the log content (on disk) above

[Q1] Fill in the xxx in the log

Exercise 1

T1:
A=10
B=20

T2:
A=40
C=30
A=50

T3:
B=75

1. <T1, BEGIN>
2. <T1, A, xxx>
3. <T1, B, xxx>
4. <T1, COMMIT>
5. <T2, BEGIN>
6. <T2, A, xxx>
7. <T2, C, xxx>
8. <T2, A, xxx>
9. <T2, COMMIT>
10. <T3, BEGIN>
11. <T3, B, xxx>

[Q2] The system crashes and recovers. What are values of A,B,C on disk after recovery:

- If the log when the crash happened contains line 1-10
- If the log when the crash happened contains line 1-7

Exercise 2

T1:
A=10
B=20

T2:
A=40
C=30
A=50

T3:
B=75

1. <T1, BEGIN>
2. <T1, A, xxx>
3. <T1, B, xxx>
4. <T1, COMMIT>
5. <T2, BEGIN>
6. <T2, A, xxx>
7. <T2, C, xxx>
8. <T2, A, xxx>
9. <T2, COMMIT>
10. <T3, BEGIN>
11. <T3, B, xxx>

Given 3 transactions T1, T2, T3. Initially, A=B=C=0

They are executed using ***redo logging***, with the log content (on disk) above

[Q1] Fill in the xxx in the log

Exercise 2

T1:	T2:	T3:
A=10	A=40	B=75
B=20	C=30	
	A=50	

1. <T1, BEGIN>
2. <T1, A, xxx>
3. <T1, B, xxx>
4. <T1, COMMIT>
5. <T2, BEGIN>
6. <T2, A, xxx>
7. <T2, C, xxx>
8. <T2, A, xxx>
9. <T2, COMMIT>
10. <T3, BEGIN>
11. <T3, B, xxx>

[Q2] The system crashes. What can you say about values of A,B,C on disk when the crash happened, and:

- The log on disk contains line 1-10.
- The log on disk contains line 1-3.

Exercise 2

T1:
A=10
B=20

T2:
A=40
C=30
A=50

T3:
B=75

1. <T1, BEGIN>
2. <T1, A, xxx>
3. <T1, B, xxx>
4. <T1, COMMIT>
5. <T2, BEGIN>
6. <T2, A, xxx>
7. <T2, C, xxx>
8. <T2, A, xxx>
9. <T2, COMMIT>
10. <T3, BEGIN>
11. <T3, B, xxx>

[Q3] The system crashes and recovers. What can you say about values of A,B,C on disk after recovery, if

- The log on disk contains line 1-11 when the crash happened
- The log on disk contains line 1-5 when the crash happened.

Exercise 3

T1:
A=10
B=20

T2:
A=40
C=30
A=50

T3:
B=75

1. <T1, BEGIN>
2. <T1, A, xxx>
3. <T1, B, xxx>
4. <T1, COMMIT>
5. <T2, BEGIN>
6. <START CHKPT (T2)>
7. <T2, A, xxx>
8. <T2, C, xxx>
9. <T2, A, xxx>
10. <END CHKP (T2)>
11. <T2, COMMIT>
12. <T3, BEGIN>
13. <T3, B, xxx>

Consider Exercise 2, but with non-quiescent checkpoints. What can you say about values of A,B,C on disk when the crash happened, and:

- The log on disk contains line 1-7.
- The log on disk contains line 1-10.

Exercise 4

1. <T1, BEGIN>
2. <T1, A, 5>
3. <T2, BEGIN>
4. <T1, COMMIT>
5. <T2, B, 10>
6. <START CHKPT (T2)>
7. <T2, C, 15>
8. <T3, BEGIN>
9. <T3, D, 20>
10. <END CHKP (T2)>
11. <T2, COMMIT>
12. <T3, COMMIT>

Consider the above ***redo log*** with non-quiescent checkpoints. The system crashed and the log on disk contains line 1-12

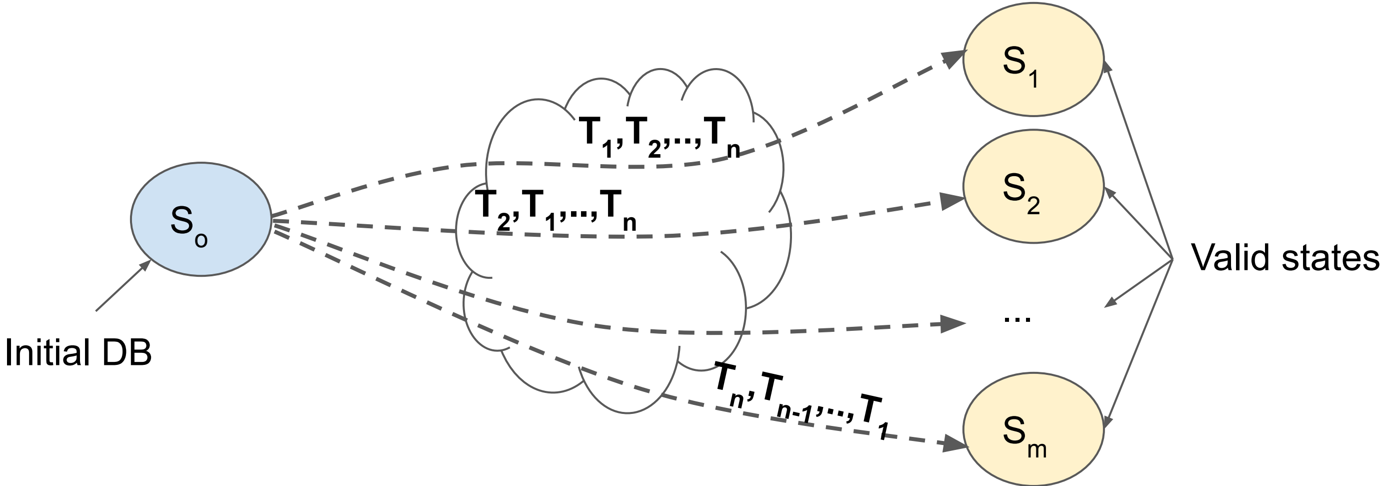
Describe the recovery steps.

Today

- **Recap**
- Atomicity
- **Isolation**

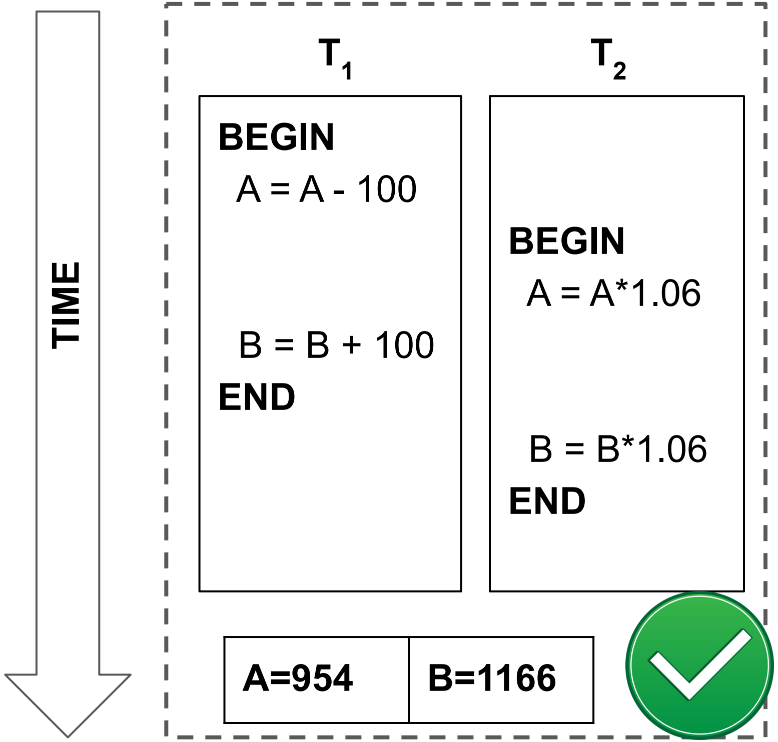
Serializability

- The gold standard for correctness
- Given $T_1, T_2, \ldots, T_n$ executed concurrently from initial state S_0
    - Execute $T_1, T_2, \ldots, T_n$ serially in random order (**n!** choices)
    - All final states reached by these executions are **valid**
- Serializable execution: reach one of the valid state



Serializability

- Interleaving:
  - vs. serial execution
  - To maximize concurrency (like threads)
    - Some operations are slow
    - Some waits for input, etc.



Serializability

- Given an interleaving sequence
  - Can DBMS check if it is serializable without executing?
    - VERY HARD!!
- In practice:
  - Check if the sequence is **conflict serializable**

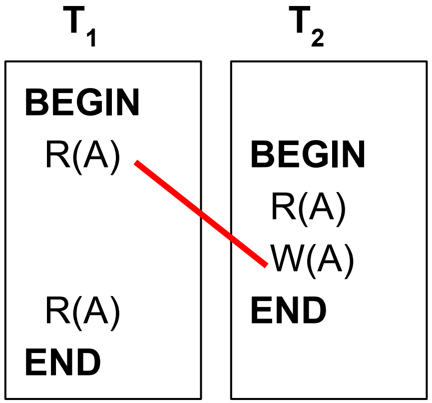
ConflictSerializable(X) → Serializable(X)

The other direction is not true

Conflict

- Two operators **conflict** iff
  - Belong to two transactions
  - On the same object
  - One of them is write

Read-Write (R-W)
Write-Read (W-R)
Write-Write (W-W)



R-W conflict
(Unrepeatable Read)

Conflict Equivalence

- Two sequence X_1, X_2 are **conflict equivalent**:
  - From the same transactions
  - Every pair of conflict is ordered the same way.

T_1	T_2
BEGIN	**BEGIN**
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
END	**END**

=

T_1	T_2
BEGIN	**BEGIN**
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)
END	**END**

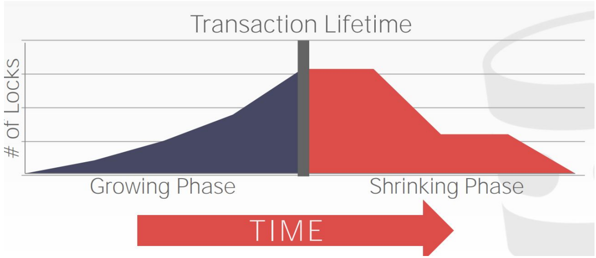
Conflict Serializable

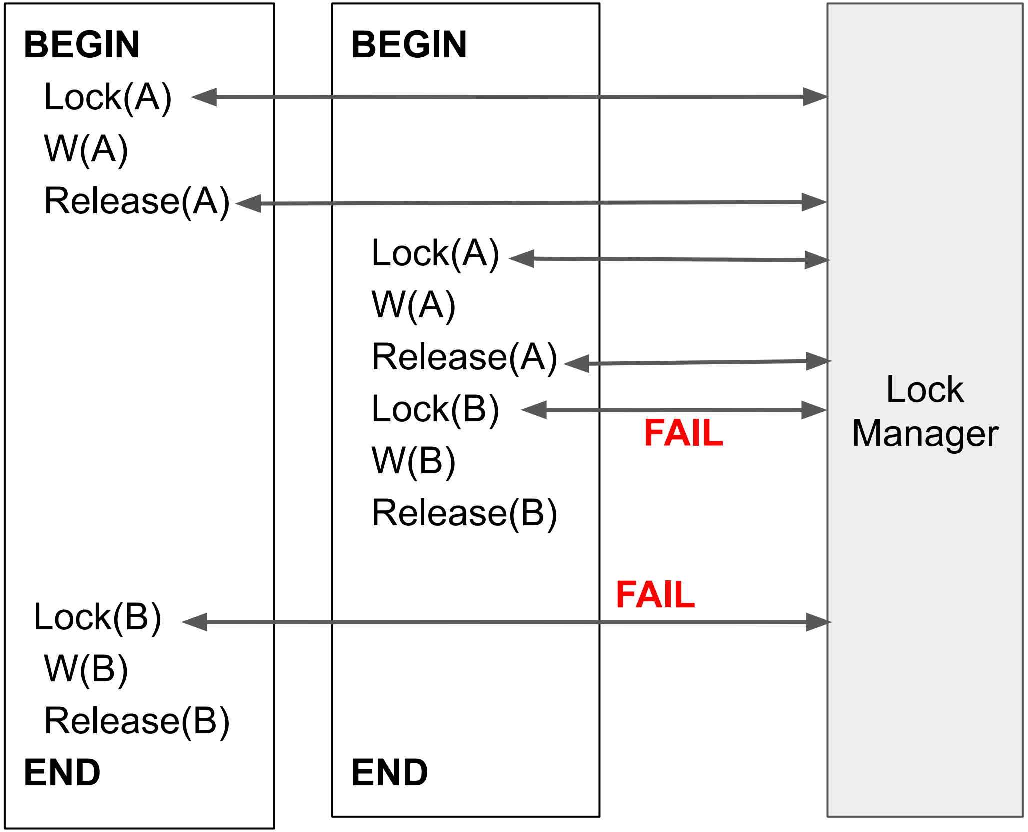
- A sequence X is conflict serializable
  - If it is conflict equivalent to a serial execution
- Intuition:
  - If X can be **transformed** to a serial execution
    - By swapping order of non-conflicting operations

ConflictSerializable(X) → Serializable(X)

2PL

- Two Phase Locking (2PL):
  - Once release a lock, cannot acquire new ones
  - Growing phase: lock acquired
  - Shrinking phase: lock release
    - May not be all at once
    - Cannot acquire new locks

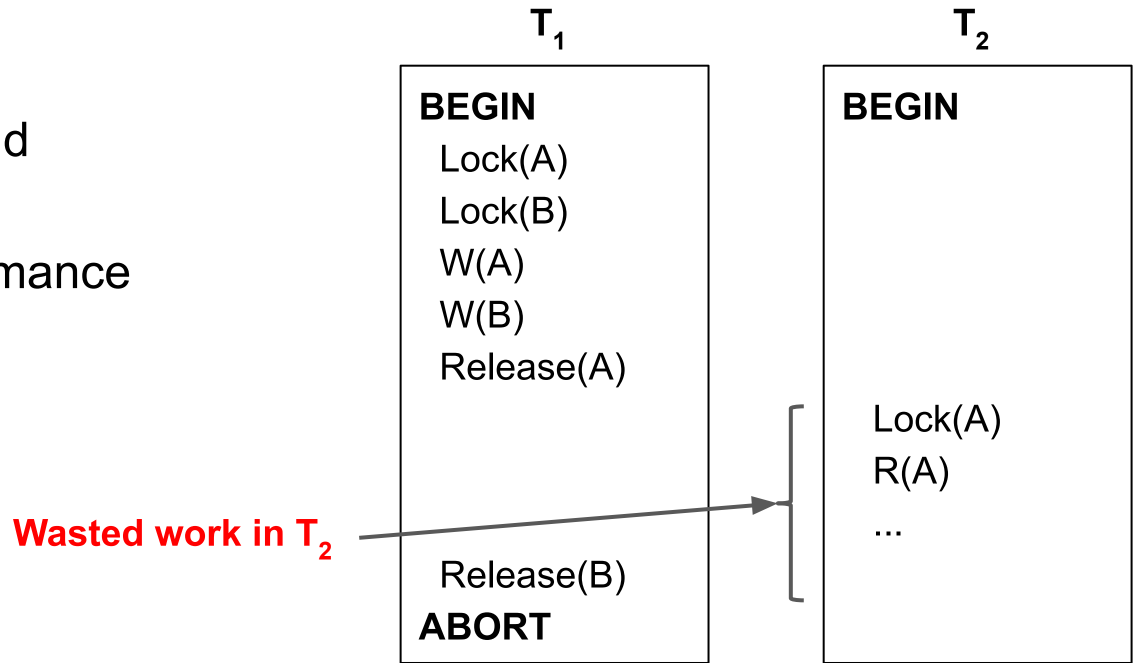




2PL violation

Strict 2PL

- Two Phase Locking (2PL):
  - Cascading Abort
- Strict 2PL (S2PL):
  - Only released at the end (COMMIT/ABORT)
  - Scarifying some performance



Exercise 5

T1		R(A)	W(A)	R(B)					
T2					W(B)	R(C)	W(C)	W(A)	
T3	R(C)								W(D)

Given the schedule for T1,T2,T3 above (time goes from left to right).

Is this execution conflict serializable?

Exercise 6

T1	R(A)		R(B)				W(A)	
T2		R(A)		R(B)				W(B)
T3					R(A)			
T4						R(B)		

Given the schedule for T1,T2,T3, T4 above (time goes from left to right).

Is this execution conflict serializable?

Exercise 7

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

Given the schedule for T1,T2 above. Each transaction commits immediately after the last operation.

- Is this schedule possible under 2PL? If yes, show where the locks are acquired and released