Granularity of Locks and **Degree of Consistency** in a Shared Data Base(PART 2)

Presented by: Jinglin Peng

Authors: J.N. Gray, R.A. Lorie, G.R. Putzolu, I.L. Traiger

You're designing a transaction system…

Requirement: account A wants to transfer $100 to account B.

Account A

- ✓ Read balance from account A.
- ✓ Subtract $100 from balance.
- ✓ Write new balance to account A.

Account B

- ✓ Read balance from account B.
- ✓ Add $100 to balance.
- ✓ Write new balance to account B.

Ideally, everything is alright. But how about in real world?

You’re designing a transaction system…

Scenario 1:

Account A

- ✓ Read balance from account A.
- ✓ Subtract $100 from balance.
- ✓ Write new balance to account A.

Power Failure

Account B

- × Read balance from account B.
- × Add $100 to balance.
- × Write new balance to account B.

TimeLine

Result: A lost $100, but B didn’t get the $100

You're designing a transaction system…

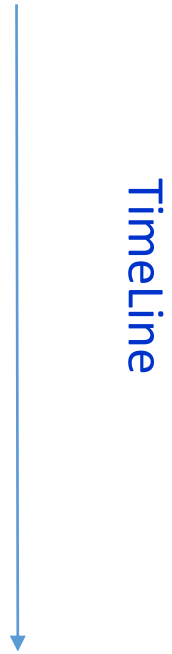
Scenario 2:

Account A

- ✓ Read balance from account A.
- ✓ Subtract $100 from balance.
- ✓ Write new balance to account A.

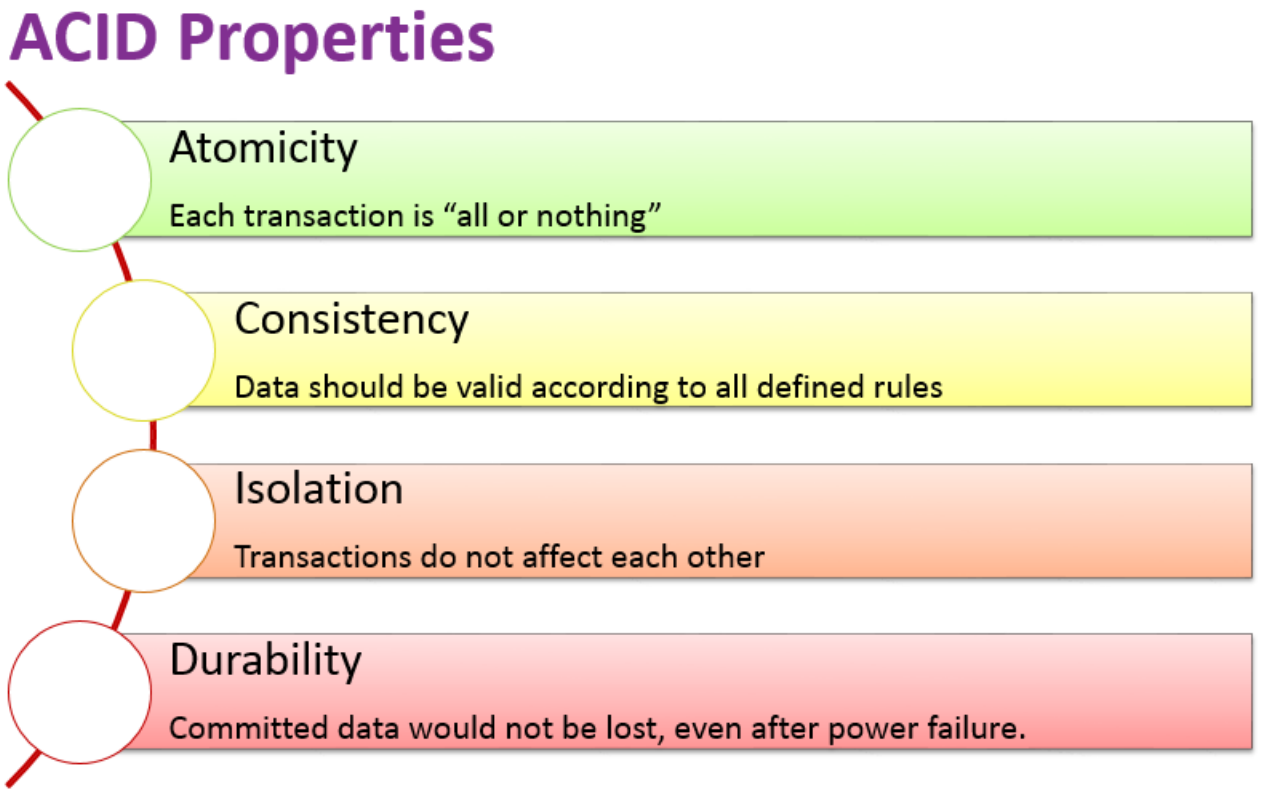
Account B

- ✓ Read balance from account B.
- × B deposits $50.
- ✓ Add $100 to balance.
- ✓ Write new balance to account B.



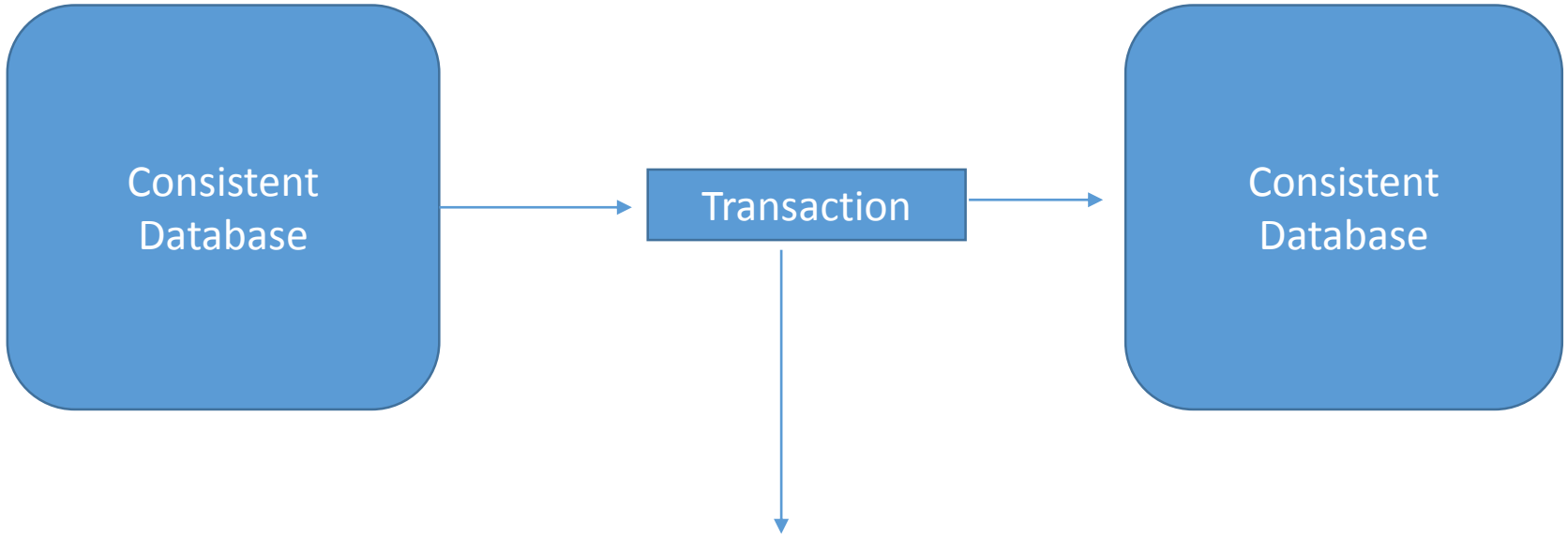
Result: B lost the $50 deposit

Requirement for Transaction System



Today we’ll talk about **consistency and isolation**.

Temporary Inconsistency

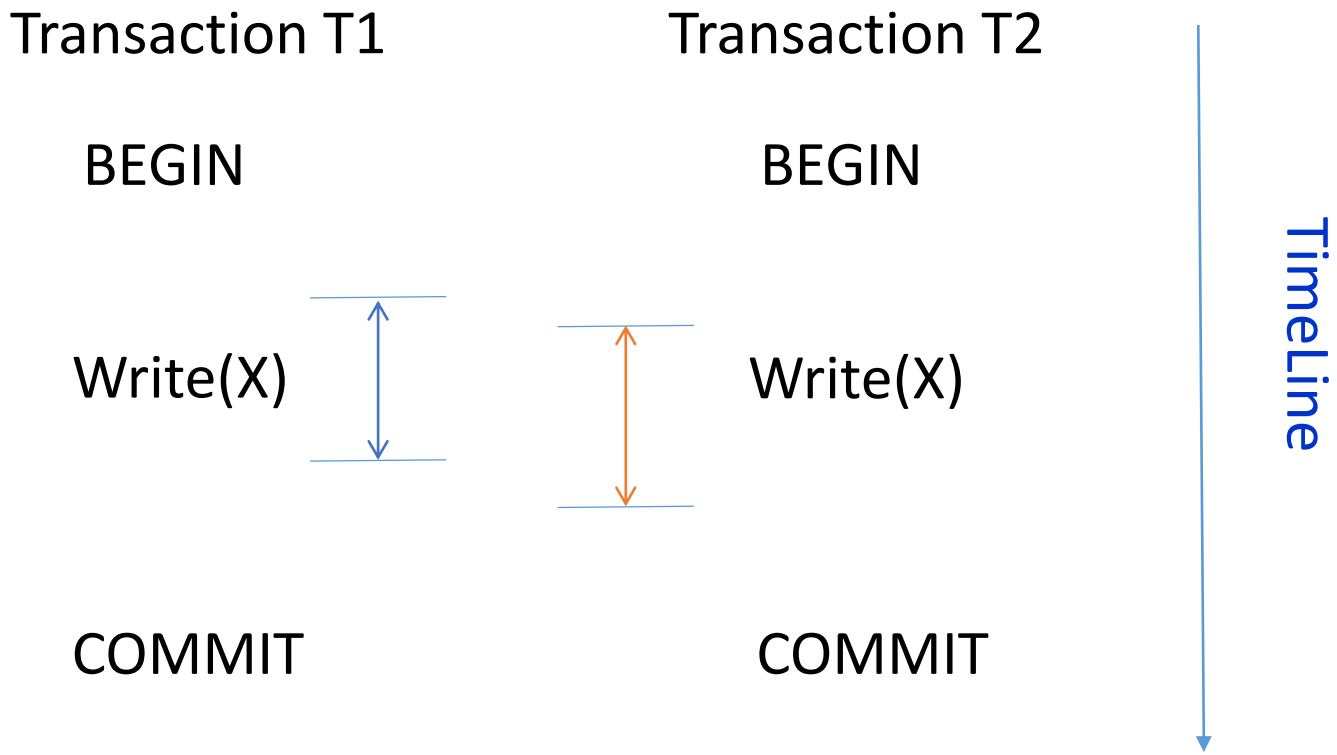


Among transaction, temporary inconsistency will occur.

Goal: keep consistency after transaction.

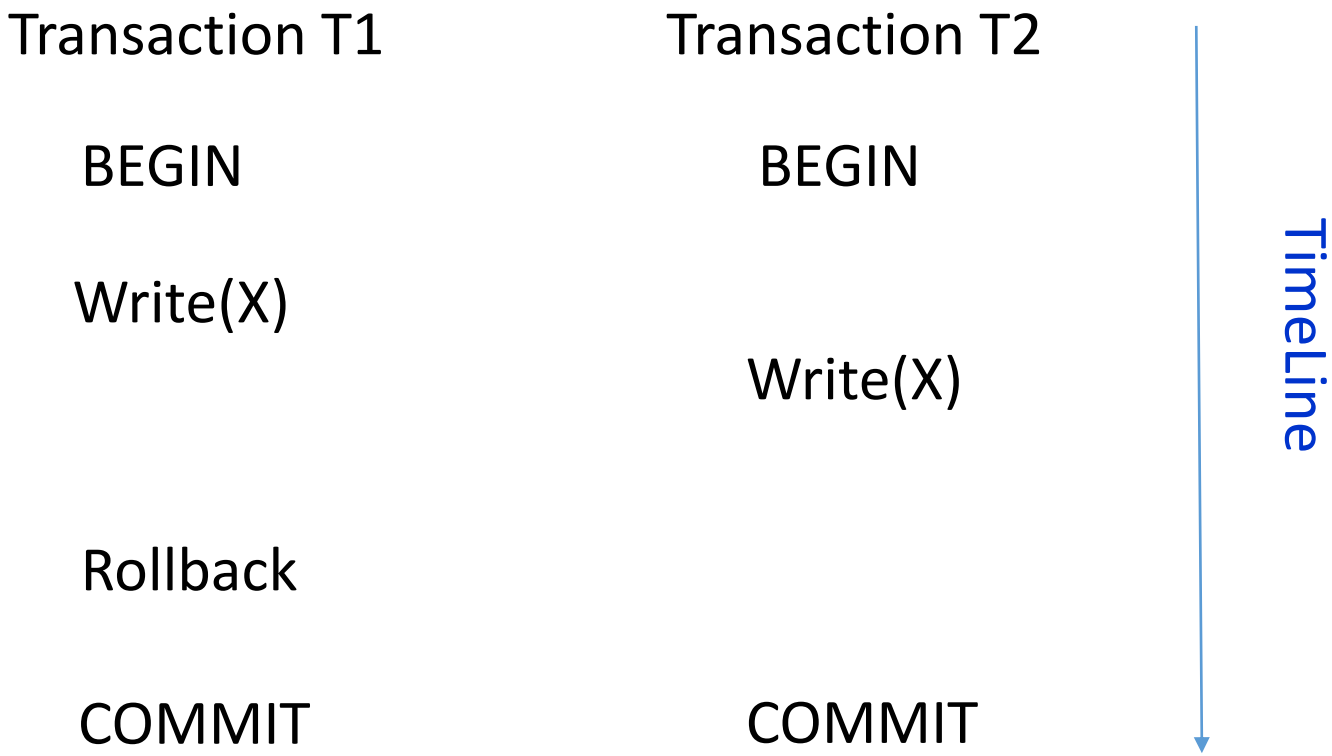
What's problem the temporary inconsistency may cause?

Problem 1: Garbage Read



Result: the value of X is unclear.

Problem 2: Lost Update



Result: update of transaction T2 is lost (rewritten by transaction T1)!

W-W conflict!

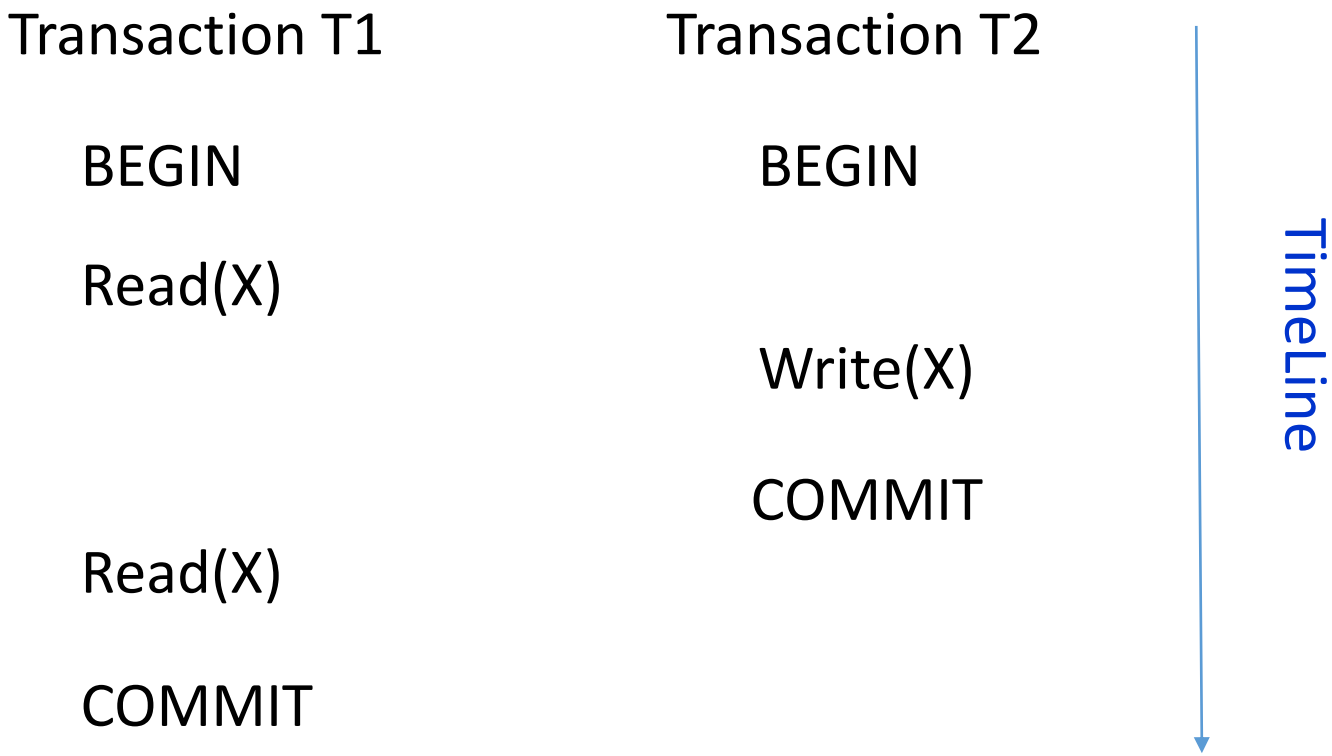
Problem 3: Dirty Read

Transaction T1	Transaction T2
BEGIN	BEGIN
Write(X)	
	Read(X)
Rollback	
COMMIT	COMMIT

Result: record X in transaction T2 is now dirty!

W-R conflict!

Problem 4: Unrepeatable Read

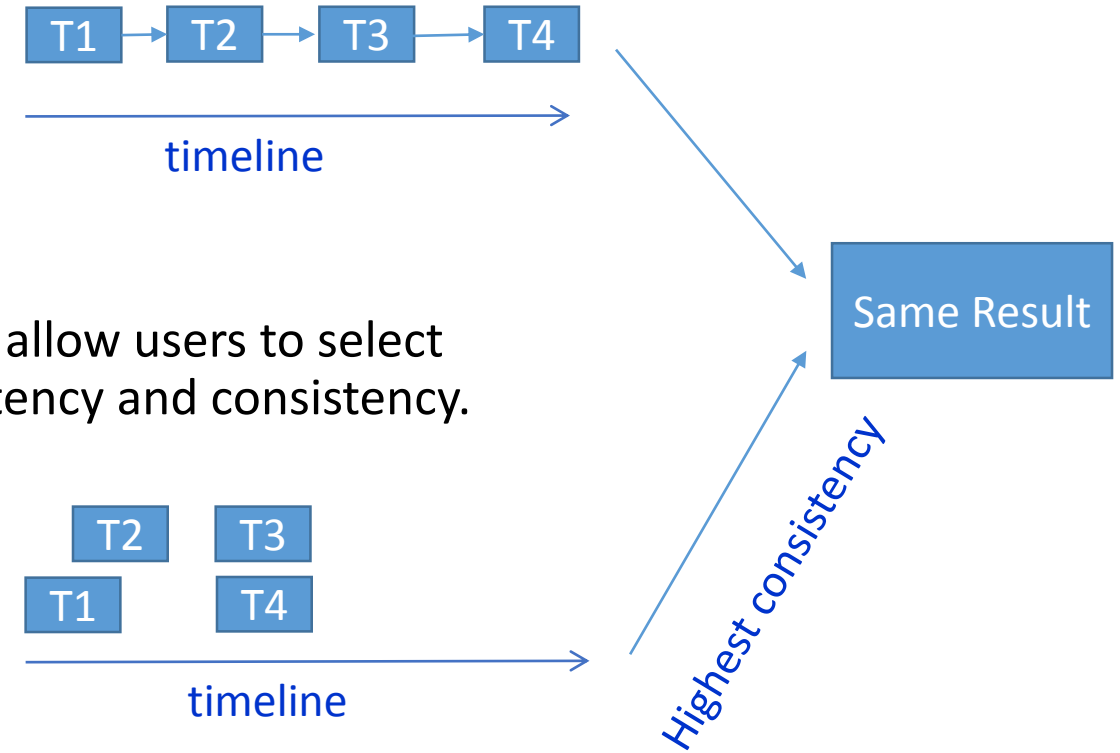


Result: transaction T1 might get a record with different values between reads!

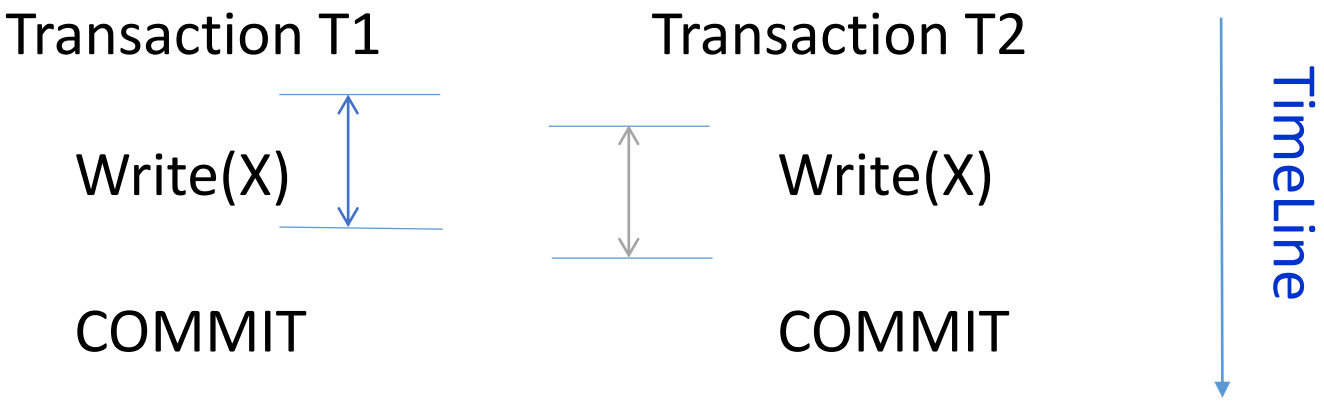
R-W conflict!

Prevention of Inconsistency

- Naïve Solution:
  - Execute transaction one at a time in a sequence.
- Drawback: latency!
- Smart Solution:
  - Classify consistency into different degrees, allow users to select degree to achieve the tradeoff between latency and consistency.



Prevention of Inconsistency: Degree 0

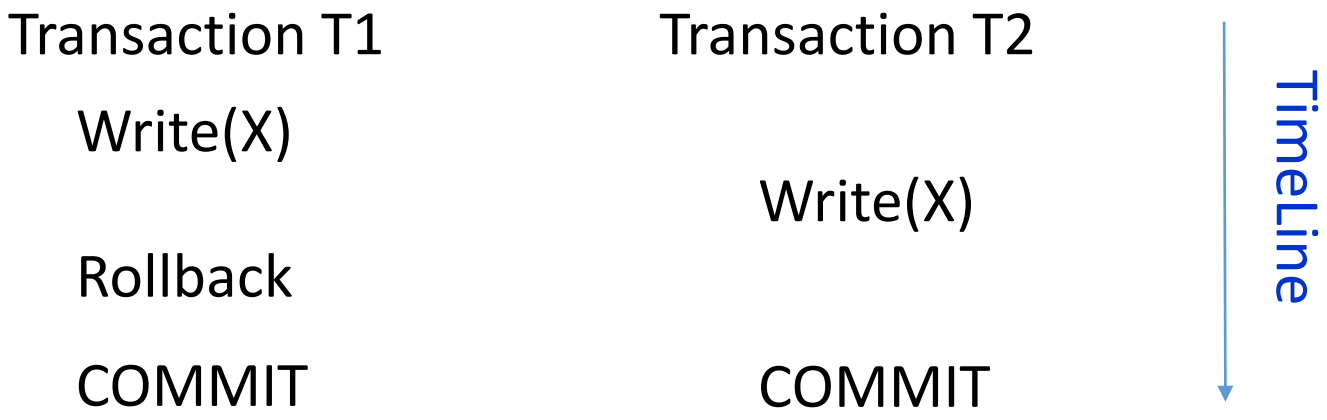


Problem: Who knows what value X will wind up holding? (**garbage read**)

Solution: set short write locks (short=until action finished)

- Before T1 write X, set a write lock to X
- After T1 write X, release the lock.

Prevention of Inconsistency: Degree 1

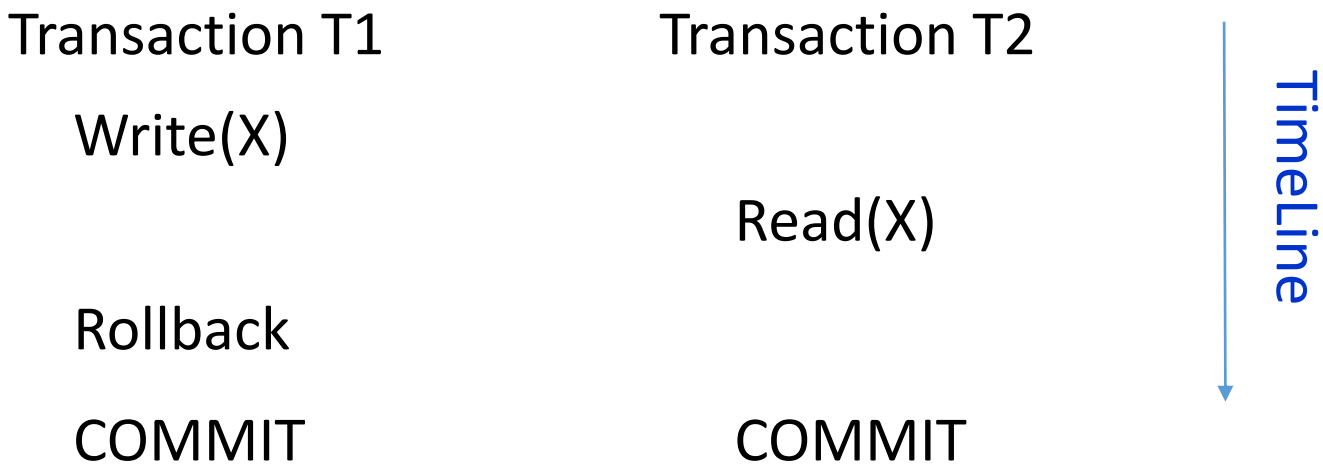


Problem: Update due to T2 is lost. (**lost update**)

Solution: set long write locks (long=until transaction committed)

- Before T1 write X, set a write lock to X
- After T1 is committed, release the write lock.

Prevention of Inconsistency: Degree 2

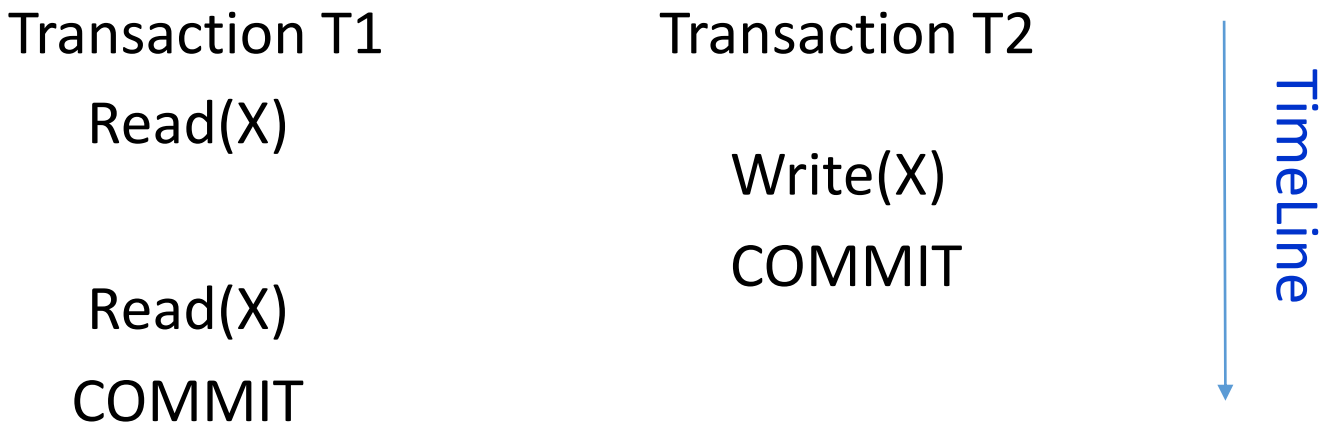


Problem: Now T2's read is bogus. (**dirty read**)

Solution: long write locks + short read locks

- Before T1 write X, set a write lock to X.
- Before T2 read, set a read lock to X.
- After finish T2's read, release the read lock.
- After T1 is committed, release the write lock.

Prevention of Inconsistency: Degree 3



Problem: Now T1 has two different values for X. (**unrepeatable read**)

Solution: long write locks + long read locks → 'two-phase locking'

- Before T1 read X, set a read lock to X.
- After T1 commit, release the read lock.
- Before T2 write X, set a write lock to X.
- After finish T2's write, release the write lock.

Conclusion of Consistency(Isolation) Degree

- Degree 0: short write locks -> garbage read
- Degree 1: long write locks -> lost update (W-W conflict)
- Degree 2: long write locks + short read locks -> dirty read (W-R conclict)
- Degree 3: long write locks + long read locks -> unrepeatable read (R-W conflict)

long = end of transaction
Short=end of action (write/read)

Definitions of Schedule

Problem: how to determine the consistency of a set transactions?

- Schedule:
  - A sequence of actions of a set of transactions.

- Schedule Consistency:
  - If **all** transactions run at degree 0 (1, 2 or 3) consistency in schedule S then S is said to be a degree 0 (1,2 or 3) consistent schedule.

Figure out how the transaction rely on each other.

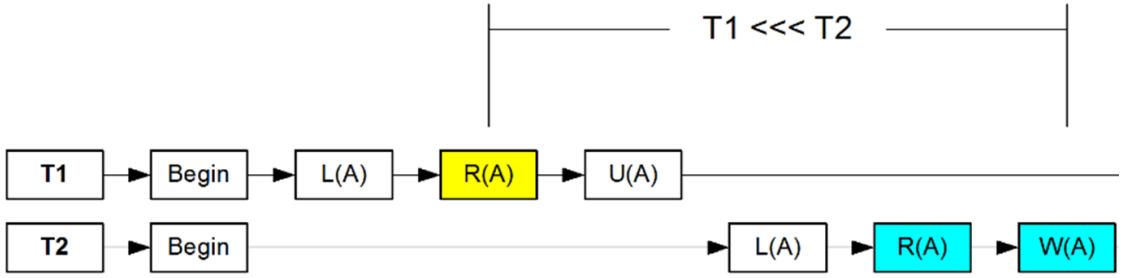
Transaction Interdependency

Suppose T1 & T2 perform action on same entity. T1 performs before T2.

Definition of Dependency

Dependency	Action	Degree
T1<T2	W->W	1
T1<<T2	W->W | W->R	2
T1<<<T2	W->W | W->R |R->W	3

Example:



Assertion w.r.t Dependency

A schedule is degree 1 (2 or 3) consistent if and only if the closure of relation < (<< or <<<) is a partial order.

Example 1

T1	Lock	A
T1	Read	A
T1	Unlock	A
T2	Lock	A
T2	Write	A
T2	Lock	B
T2	Write	B
T2	Unlock	A
T2	Unlock	B
T1	Lock	B
T1	Write	B
T1	Unlock	B

Definition of Dependency

Dependency	Action	Degree
T1<T2	W->W	1
T1<<T2	W->W | W->R	2
T1<<<T2	W->W | W->R |R->W	3

T2<T1, T2<<T1, T2<<<T1

T1<<<T2

<<< is not partial order

- The schedule is degree 2 consistent but not degree 3 consistent.
  - T1 runs at degree 2 consistency, T2 runs at degree 3 consistency.

Example 2

T1	Lock	A
T1	write	A
T2	read	A
T2	Lock	B
T2	Write	B
T2	Unlock	B
T1	Lock	B
T1	Write	B
T1	Unlock	B
T1	Unlock	A

Definition of Dependency

Dependency	Action	Degree
T1<T2	W->W	1
T1<<T2	W->W | W->R	2
T1<<<T2	W->W | W->R |R->W	3

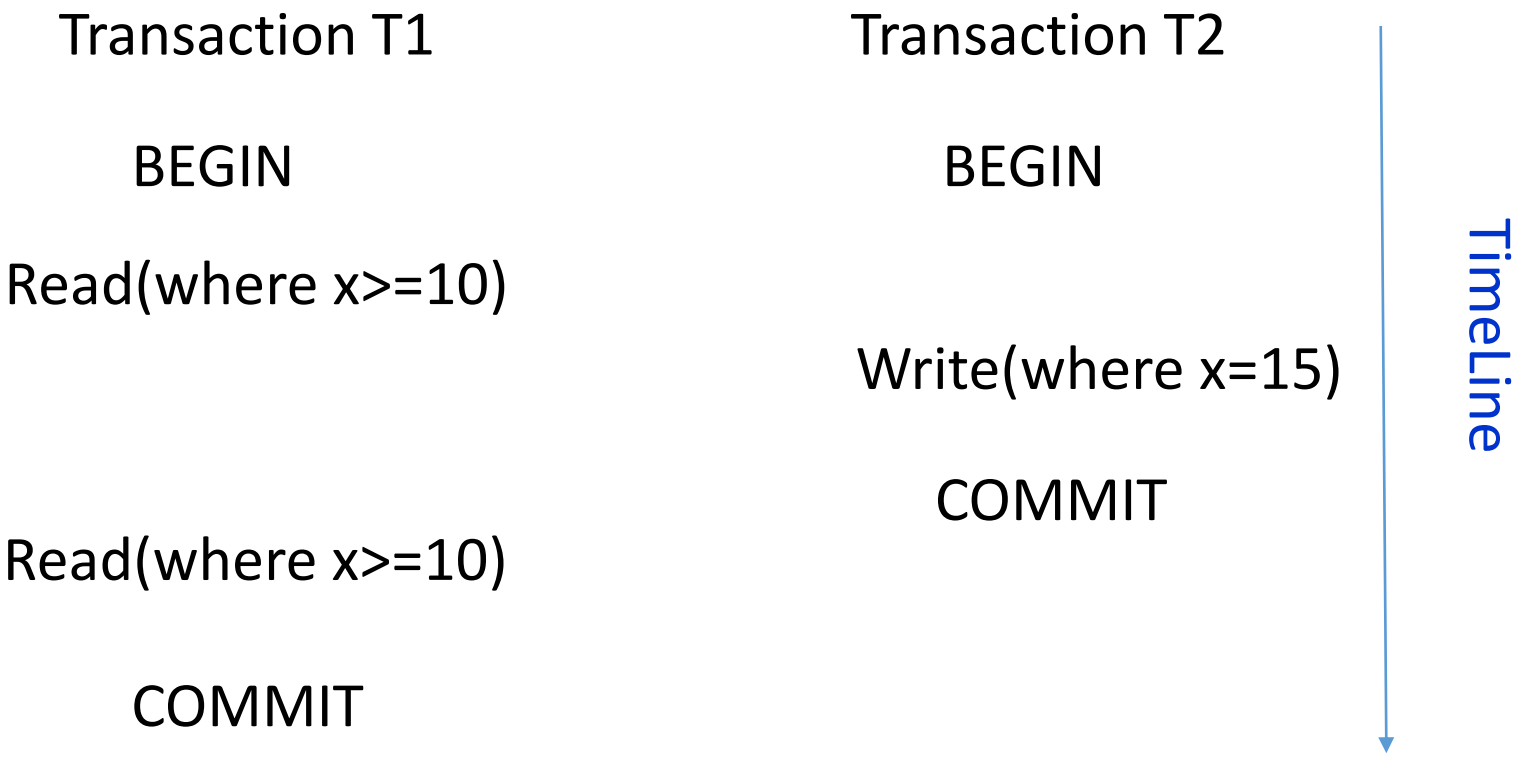
T2<T1, T2<<T1, T2<<<T1

T1<<T2, T1<<<T2

<<< and << are not partial order

- The schedule is degree 1 consistent.
  - T1 runs at degree 3 consistent, T2 runs at degree 1 consistent.

Problem out of the paper: Phantom Read



Results: results fetched by transaction T1 may be different in both reads.

Solution: range lock!

Summary

- 4 degrees of consistency: allow user tradeoff latency and consistency.

- Dependency among transactions: determine the consistency of schedule.

Thank you!

Q & A