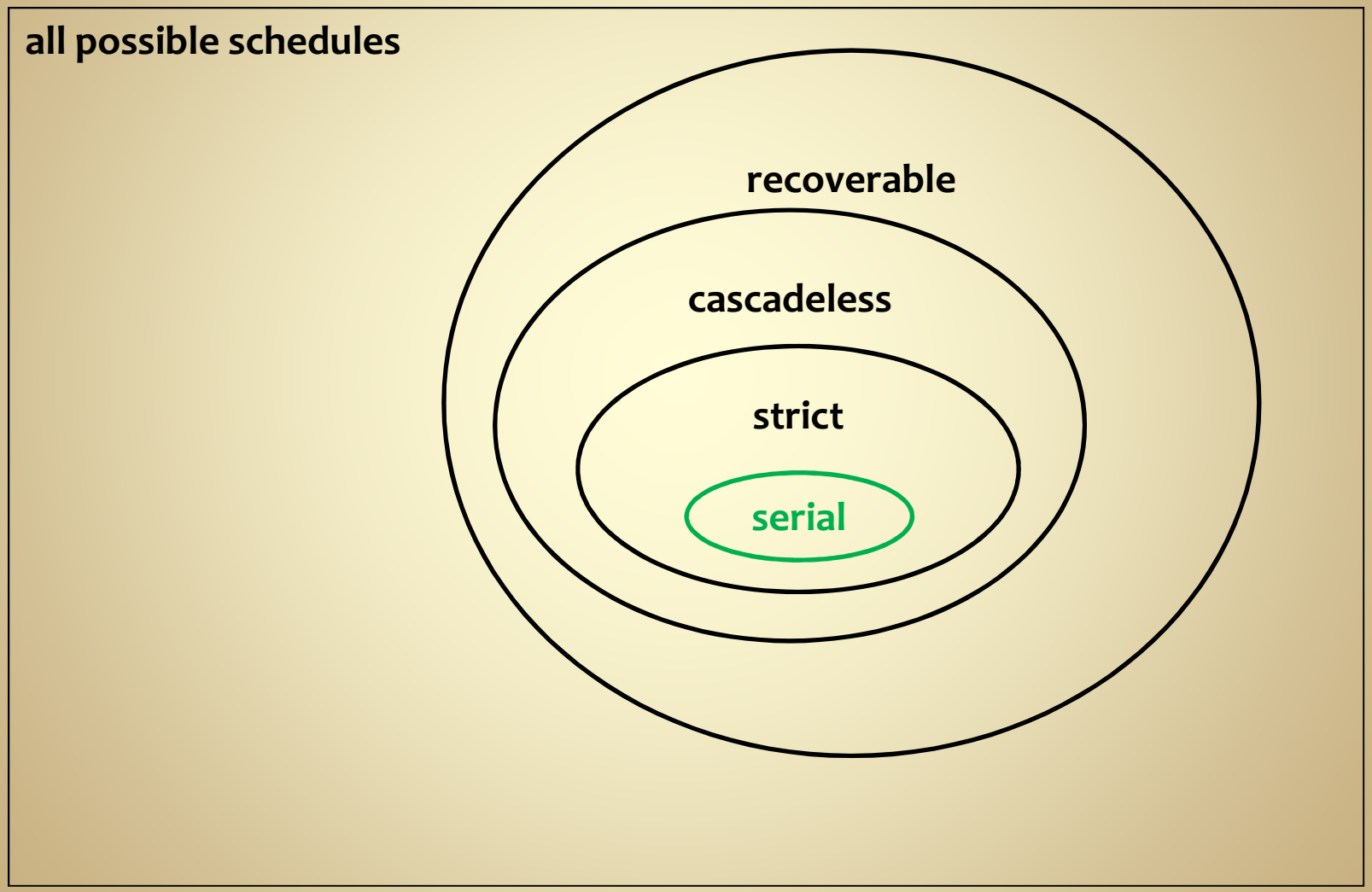
COMP163

Database Management Systems

Recoverability Classes - Chapter 21
Concurrency Control – Chapter 22

Recoverability Classes



Recoverability Classes

- Recoverability indicates whether a schedule will allow for recovery in the case of a transaction failure
- If a schedule is ***recoverable***, it will never be necessary to roll-back a committed transaction
  - any potential problems can be handled by aborting non-committed transactions
- Other recoverability classes indicate the ease of recovery for schedules in that class

- *Recoverability is not an indicator of correctness*

Recoverable Schedules

- In a *recoverable schedule*, a committed transaction never needs to be rolled back.
  - a transaction cannot be committed if it is potentially involved in an incorrect schedule
- Recoverable schedule test:
  - ***no transaction T commits until all transactions that wrote something that T reads have committed***
  - test prevents T from committing if it uses data that might later become invalid

Casecadeless Schedules

- Cascadeless = Avoids Cascading Rollbacks (ACR)

- Cascading Roll-back:
  If an uncommitted transaction T1 reads data written by transaction T2, and T2 is rolled-back, then T1 also has to be rolled-back
  - the roll-back cascades from T2 to T1

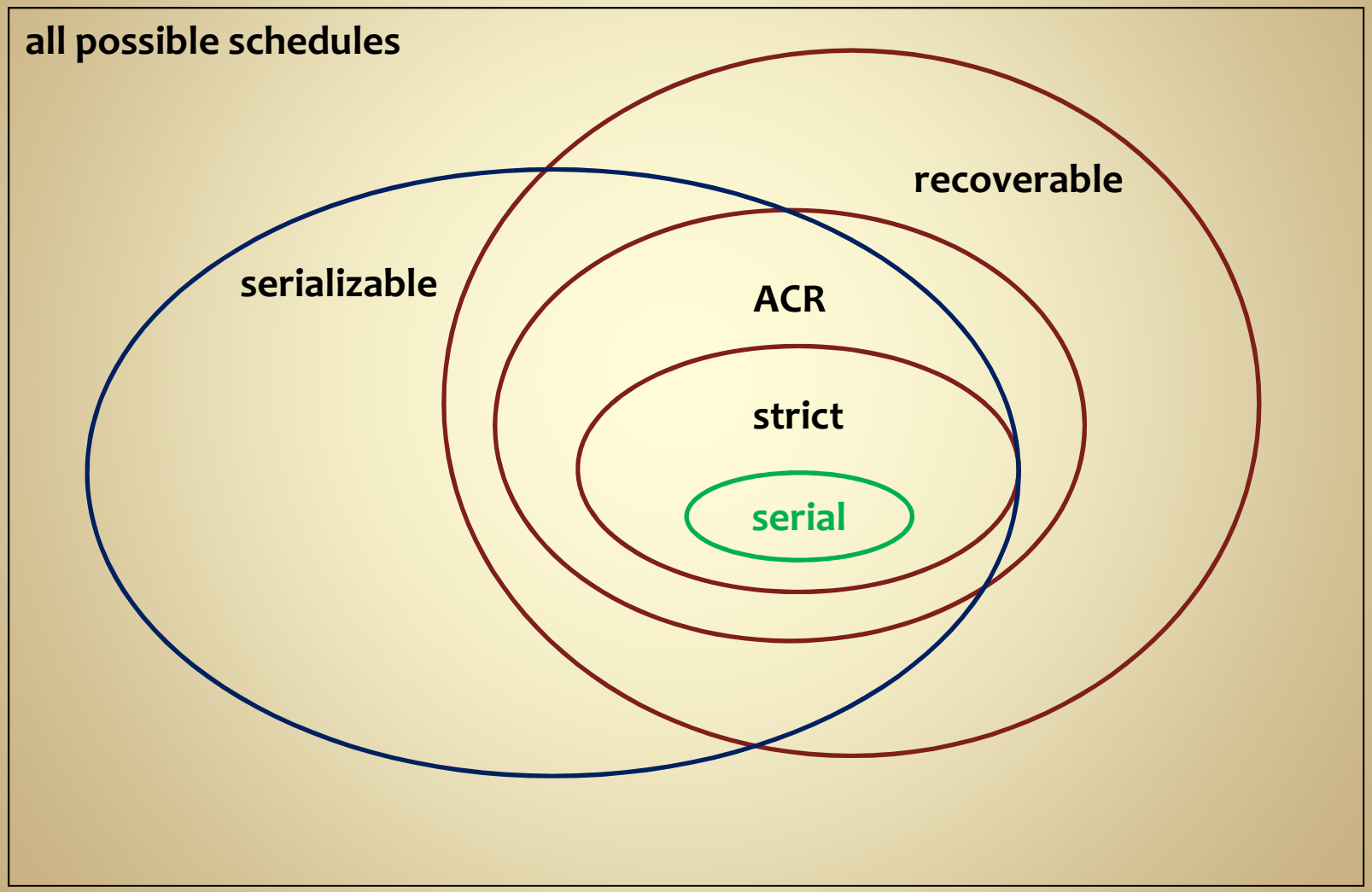
- Cascadeless test:
  - ***every transaction only reads things written by committed transactions***

Strict Schedules

- In a *strict schedule*, any transaction can be aborted by simply restoring the values of any object that it wrote

- Strict schedule test:
  - ***no transaction can read or write anything that was written by an uncompleted transaction***

Classes of Schedules



Concurrency Control

- Concurrency control is the enforcement of policies regarding allowed schedules
- Minimal policy:
  - never allow a schedule that is not in (serializable U recoverable)
- Other possible policies:
  - allow only serial schedules (no concurrency)
  - allow only serializable, casecadeless schedules
  - allow only strict schedules

SQL: CC and Transactions

- **`SET TRANSACTION`**

  sets the transaction *access mode*:
  - READ ONLY → only allows SELECT
  - READ WRITE → allows SELECT, UPDATE, INSERT, DELETE, CREATE

- **`SET TRANSACTION ISOLATION LEVEL`**

  sets the transaction *isolation level*:
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE

SQL Transaction Violations

- **Dirty reads:** occur when a transaction reads a record altered by another transaction that has not yet completed
- **Non-repeatable reads:** occur when one transaction reads a record while another transaction modifies it
- **Phantom records:** occur when a transaction reads a group of records, then another transaction changes the set of records that would have been read

SQL Isolation Levels

isolation level	prevents
READ UNCOMMITTED	
READ COMMITTED	dirty reads
REPEATABLE READ	dirty reads non-repeatable reads
SERIALIZABLE	dirty reads non-repeatable reads phantom records

Dirty Read

Schedule: b1, r1(X), w1(X), b2, r2(X) …

T2 has read the value written by T1.
What if T1 aborts?

Dirty Read: Accessing data that is not yet committed.
Dirty reads can cause cascading roll-backs.

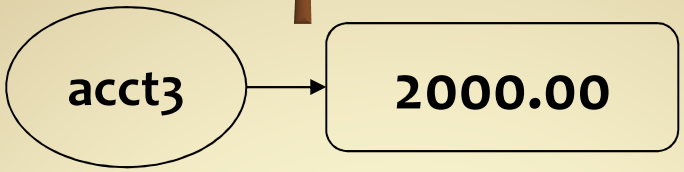
Non-repeatable Read

Schedule: b1, r1(X), b2, r2(X), w1(X), …

T1 has changed the value read by T2.
The value held by T2 is no longer valid
(or valid only if T1 aborts).

Non-repeatable Read: Data was changed since it was read.
If data is read again, a different value will be seen.

The Lost Update Problem



XACT 1

reads 2000

computes 1700

writes 1700

XACT 2

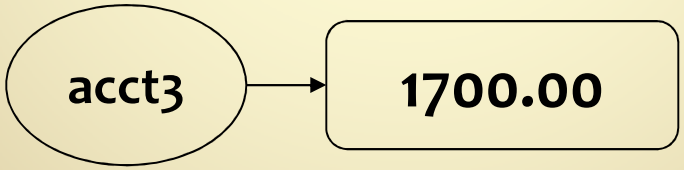
reads 2000

computes 2900

writes 2900

this update was lost

r2(acct3)
r1(acct3)
XACT1: acct3 := acct3 – 300
XACT2: acct3 := acct3 + 900
w2(acct3)
w1(acct3)



This is actually caused by the non-repeatable read r2(acct3)

Phantom Records

- T1: select accountNum from Account
  where balance > '$1000.00‘
- T2: update Account
  set balance = balance –'$500.00'
  where accountNum = 387
- If T1 reruns its query, the record for account 387 might no longer be included.

Phantom Records: The set of records read by a transaction is changed by another transaction.

This can also cause problems with aggregate queries.

Concurrency Control

- Transaction theory classifies possible schedules in terms of recoverability and correctness.
  - In theory, we can talk about modifying the schedules to gain desired characteristics
  - In practice, the schedule is determined by the real-time order in which the operations arrive and the only "rescheduling" that is possible is delaying certain operations
- Concurrency control implements mechanisms to achieve specific policies
- general protocol classifications:
  - pessimistic: prevent unwanted schedules from occurring even if it reduces concurrency and stalls transactions
  - optimistic: allow any schedule, later abort transaction(s) contributing to unwanted schedules

Pessimistic CC Protocols

- ***Pessimistic***: prevent unwanted schedules from occurring even if it reduces concurrency and stalls transactions
- ***Locking protocols***:
  Delay conflicting operations by *locking* data items
  - locking is most common CC protocol
  - reduces concurrency
  - may result in deadlock, livelock or starvation
- ***Timestamping protocols***:
  Abort transactions that request operations that violate serializability
  - timestamps used to order conflicting operations

Pessimistic CC Protocols

- Locking
  - locking is most common CC protocol
  - reduces concurrency
  - may result in deadlock, livelock or starvation
  - 2PL allows only serializable schedules
  - 2PL never requires xact roll-backs due to conflict
- Time-stamping
  - allows only serializable schedules
  - cannot result in deadlock
  - may cause cascading aborts due to conflict
  - may cause starvation

2PL = two phase locking
(next lecture)

Optimistic Protocols

- no checking is done while a transaction is executing
  - optimistically assume everything will be fine
- all operations are performed on local copies of data items
- validity is checked when transaction commits
  - invalid transactions determined at latest possible time
- maximum concurrency
- may cause aborts due to conflict
- no possibility of deadlock

SQL Isolation Levels

isolation level	prevents	locking
READ UNCOMMITTED		all locks released immediately following SQL statement execution
READ COMMITTED	dirty reads	read locks released immediately write locks held until end of transaction
REPEATABLE READ	dirty reads non-repeatable reads	all locks held until end of transaction (strict 2PL)
SERIALIZABLE	dirty reads non-repeatable reads phantom records	requires index or table locks to prevent phantom reads