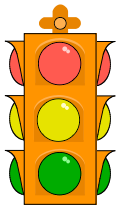
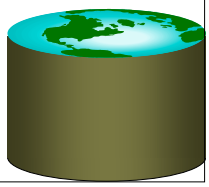
Concurrency Control

R&G - Chapter 17



Smile, it is the key that fits the lock of everybody's heart.

Anthony J. D'Angelo, The College Blue Book



Review

- ACID transaction semantics.
- Today: focus on Isolation property
  - Serial schedules safe but slow
  - Try to find schedules ***equivalent*** to serial …

Conflicting Operations

- Need a tool to decide if 2 schedules are equivalent
- Use notion of "conflicting" operations
- Definition: Two operations **conflict** if:
  - They are by different transactions,
  - they are on the same object,
  - and at least one of them is a write.

Conflict Serializable Schedules

- Definition: Two schedules are conflict equivalent iff:
  - They involve the same actions of the same transactions, and
  - every pair of conflicting actions is ordered the same way
- Definition: Schedule S is conflict serializable if:
  - S is conflict equivalent to some serial schedule.
- Note, some "serializable" schedules are NOT conflict serializable
  - A price we pay to achieve efficient enforcement.

Conflict Serializability – Intuition

- A schedule S is conflict serializable if:
  - You are able to transform S into a serial schedule by swapping **consecutive non-conflicting** operations of different transactions.
- *Example:*

R(A) W(A) R(B) W(B)
R(A) W(A) R(B) W(B)

≡

R(A) W(A) R(B) W(B)
R(A) W(A) R(B) W(B)

Conflict Serializability (Continued)

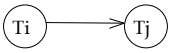
- Here's another example:

R(A) W(A)
R(A) W(A)

- Serializable or not????

NOT!

Dependency Graph

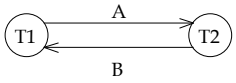


- *Dependency graph*:
  - One node per Xact
  - Edge from Ti to Tj if:
    - An operation Oi of Ti conflicts with an operation Oj of Tj and
    - Oi appears earlier in the schedule than Oj.
- Theorem: Schedule is conflict serializable *if and only if* its dependency graph is acyclic.

Example

- A schedule that is not conflict serializable:

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



Dependency graph

- The cycle in the graph reveals the problem. The output of T1 depends on T2, and vice-versa.

An Aside: View Serializability

- Alternative (weaker) notion of serializability.
- Schedules S1 and S2 are view equivalent if:
  1. If Ti reads initial value of A in S1, then Ti also reads initial value of A in S2
  2. If Ti reads value of A written by Tj in S1, then Ti also reads value of A written by Tj in S2
  3. If Ti writes final value of A in S1, then Ti also writes final value of A in S2
- Basically, allows all conflict serializable schedules + "blind writes"

T1: R(A)		W(A)	
T2:	W(A)		
T3:			W(A)

view ≡

T1: R(A),W(A)		
T2:		W(A)
T3:		W(A)

Notes on Serializability Definitions

- View Serializability allows (slightly) more schedules than Conflict Serializability does.
  - Problem is that it is difficult to enforce efficiently.
- Neither definition allows all schedules that you would consider "serializable".
  - This is because they don't understand the meanings of the operations or the data.
- In practice, Conflict Serializability is what gets used, because it can be enforced efficiently.
  - To allow more concurrency, some special cases do get handled separately, such as for travel reservations, etc.

Two-Phase Locking (2PL)

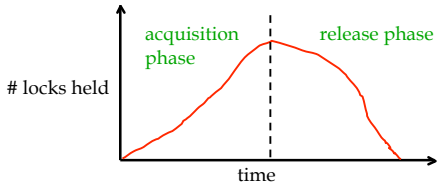
Lock Compatibility Matrix

	S	X
S	√	–
X	–	–

rules:

- Xact must obtain a **S** (*shared*) lock before reading, and an **X** (*exclusive*) lock before writing.
- Xact cannot get new locks after releasing any locks.

Two-Phase Locking (2PL), cont.



2PL guarantees conflict serializability

But, does *not* prevent **Cascading Aborts**.

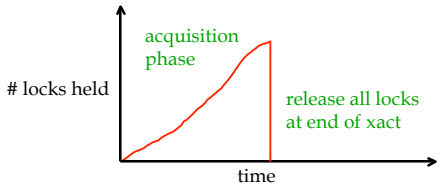
Strict 2PL

- *Problem:* Cascading Aborts
- *Example:* rollback of T1 requires rollback of T2!

T1:	R(A), W(A),	R(B), W(B), Abort
T2:	R(A), W(A)	

- Strict Two-phase Locking (Strict 2PL) protocol:
  Same as 2PL, except:
  Locks released only when transaction completes
  i.e., either:
  (a) transaction has committed (commit record on disk),
  or
  (b) transaction has aborted and rollback is complete.

Strict 2PL (continued)



Next ...

- A few examples

Non-2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Unlock(A)	
	Read(A)
	Unlock(A)
	Lock_S(B)
Lock_X(B)	
	Read(B)
	Unlock(B)
	PRINT(A+B)
Read(B)	
B := B +50	
Write(B)	
Unlock(B)	

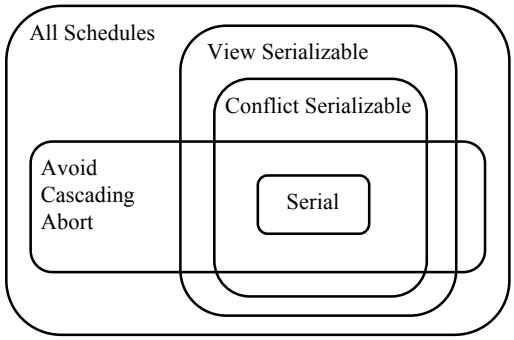
2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Lock_X(B)	
Unlock(A)	
	Read(A)
	Lock_S(B)
Read(B)	
B := B +50	
Write(B)	
Unlock(B)	Unlock(A)
	Read(B)
	Unlock(B)
	PRINT(A+B)

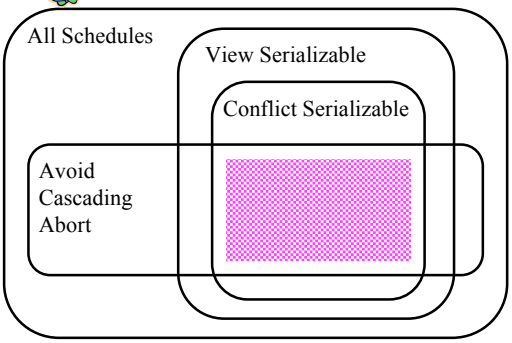
Strict 2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Lock_X(B)	
Read(B)	
B := B +50	
Write(B)	
Unlock(A)	
Unlock(B)	
	Read(A)
	Lock_S(B)
	Read(B)
	PRINT(A+B)
	Unlock(A)
	Unlock(B)

Venn Diagram for Schedules



Which schedules does Strict 2PL allow?



Lock Management

- Lock and unlock requests handled by **Lock Manager**
- LM keeps an entry for each currently held lock.
- Entry contains:
  - List of xacts currently holding lock
  - Type of lock held (shared or exclusive)
  - Queue of lock requests

Lock Management, cont.

- When lock request arrives:
  - Does any other xact hold a conflicting lock?
    - If no, grant the lock.
    - If yes, put requestor into wait queue.
- Lock upgrade:
  - xact with shared lock can request to upgrade to exclusive

Example

Lock_X(A)	
	Lock_S(B)
	Read(B)
	Lock_S(A)
Read(A)	
A: = A-50	
Write(A)	
Lock_X(B)	

Deadlocks

- **Deadlock**: Cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
  - prevention
  - detection
- Many systems just punt and use Timeouts
  - What are the dangers with this approach?

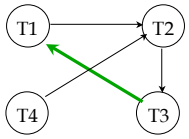
Deadlock Detection

- Create and maintain a "waits-for" graph
- Periodically check for cycles in graph

Deadlock Detection (Continued)

Example:

T1: S(A), S(D), S(B)
T2: X(B) X(C)
T3: S(D), S(C), X(A)
T4: X(B)



Deadlock Prevention

- Assign priorities based on timestamps.
- Say Ti wants a lock that Tj holds
  Two policies are possible:
  Wait-Die: If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits
- Why do these schemes guarantee no deadlocks?
- Important detail: If a transaction re-starts, make sure it gets its original timestamp. -- Why?

Summary

- Correctness criterion for isolation is "serializability".
  - In practice, we use "conflict serializability," which is somewhat more restrictive but easy to enforce.
- Two Phase Locking and Strict 2PL: Locks implement the notions of conflict directly.
  - The lock manager keeps track of the locks issued.
  - **Deadlocks** may arise; can either be prevented or detected.