Transactions and Concurrency Control

Problem Statement

- Goal: concurrent execution of independent transactions
  - utilization/throughput ("hide" waiting for I/Os)
  - response time
  - fairness
- Example:

	T1:	T2:
t0:	tmp1 := read(X)	
t1:		tmp2 := read(X)
t2:	tmp1 := tmp1 – 20	
t3:		tmp2 := tmp2 + 10
t4:	write tmp1 into X	
t5:		write tmp2 into X

Arbitrary interleaving can lead to inconsistencies

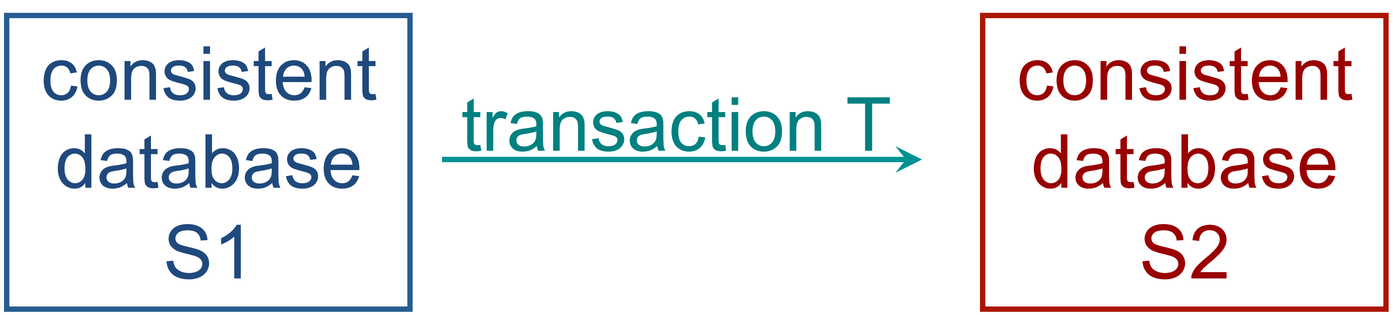
Correctness: The **ACID** properties

- A tomicity: All actions in the transaction happen, or none happen
- C onsistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent
- I solation: Execution of one transaction is isolated from that of other transactions
- D urability: If a transaction commits, its effects persist

C

Transaction Consistency

- "Consistency" - data in DBMS is accurate in modeling real world and follows integrity constraints
- User must ensure that transaction is consistent
- Key point:



Isolation of Transactions

- Users submit transactions, and
- Each transaction executes as if it was running by itself
  - Concurrency is achieved by DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.

- Techniques for achieving isolation:
  - Pessimistic – don't let problems arise in the first place
  - Optimistic – assume conflicts are rare, deal with them *after* they happen.

Example

- Consider two transactions:

```
T1:   BEGIN   A=A+100,   B=B-100   END
T2:   BEGIN   A=1.06*A,   B=1.06*B   END
```

- Legal outcomes: A=1166,B=954 or A=1160,B=960
- Consider a possible interleaved *schedule*:

```
T1:   A=A+100,                  B=B-100
T2:              A=1.06*A,                B=1.06*B
```

- This is OK (same as T1;T2). But what about:

```
T1:   A=A+100,                            B=B-100
T2:              A=1.06*A, B=1.06*B
```

Anomalies with Interleaved Execution

- Reading Uncommitted Data (WR Conflicts, “dirty reads”):

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

- Unrepeatable Reads (RW Conflicts):

```
T1:   R(A),                        R(A), W(A), C
T2:          R(A), W(A), C
```

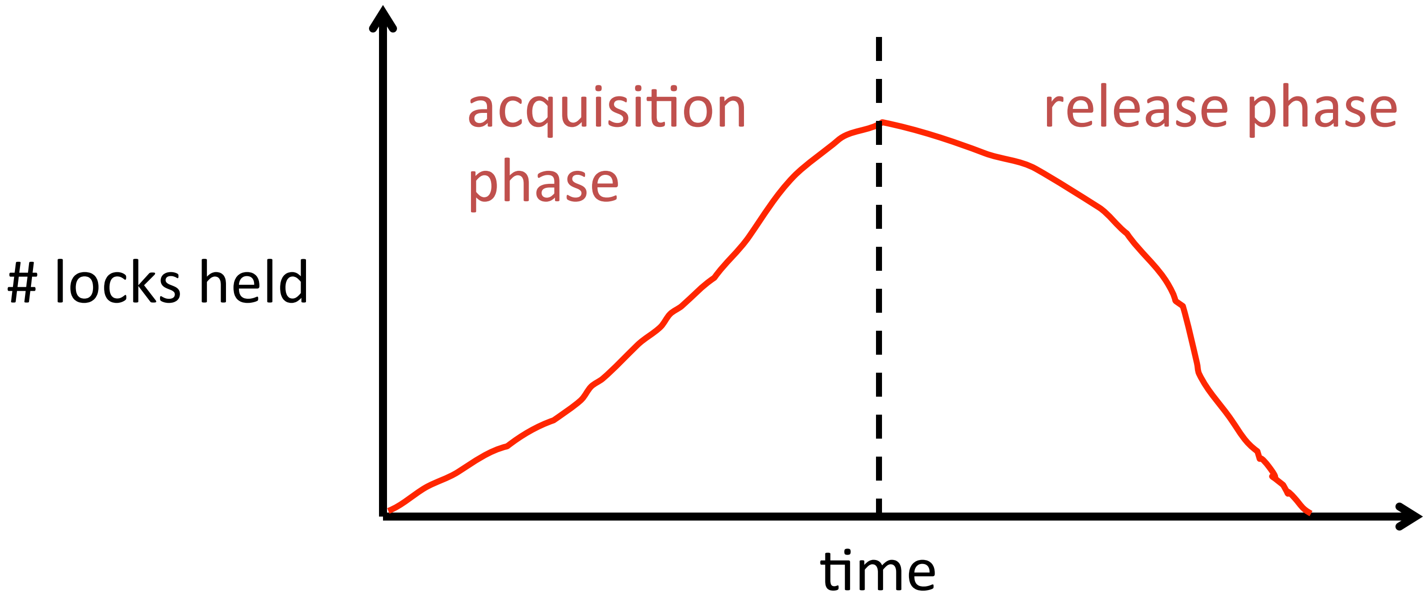
Two-Phase Locking (2PL)

Lock Compatibility Matrix

	S	X
S	√	–
X	–	–

- Each transaction must obtain an S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing
- A transaction can not request additional locks once it releases any locks
- Thus, there is a "growing phase" followed by a "shrinking phase"

Two-Phase Locking (2PL)



- 2PL on its own is sufficient to guarantee serializability, but, it is subject to **Cascading Aborts**

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

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

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

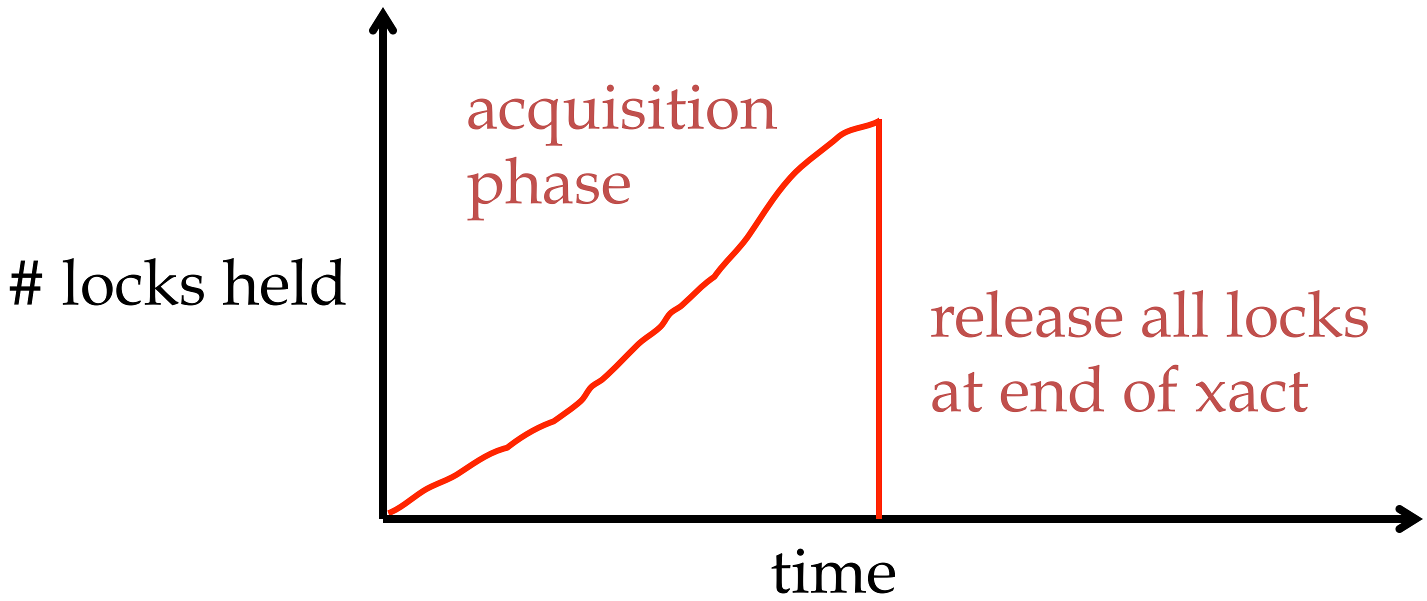
R(A) W(A) R(B) W(B)

First transaction aborts

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

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

Strict 2PL (continued)



- In effect, "shrinking phase" is delayed until
  a) Transaction has committed (commit log record on disk), or
  b) Decision has been made to abort the transaction (locks can be released after rollback)

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 =?

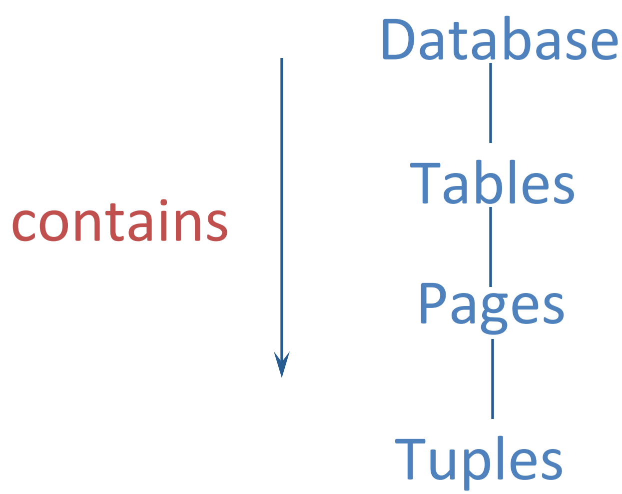
Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Lock_X(B)	
Read(B)	
B := B +50	
Write(B)	
Unlock(A)	
Unlock(B)	
	Read(A)
	Lock_S(B)
	Read(B)
	PRINT(A+B)
	Unlock(A)
	Unlock(B)

Locking vs. Latching

- Weird database terminology
- Lock – a logical concept that controls access to an entity
- Latch – a mutex, also called a lock elsewhere in computer science.
  - This is a physical concept often supported by hardware instructions

Multiple-Granularity Locks

- Hard to decide what granularity to lock (tuples vs. pages vs. tables)
- Shouldn't have to make same decision for all transactions!
- Data "containers" are nested:



Lock Manager Implementation

- In R & G, chapter 17, you can read the short implementation section which describes the lock manager as a hash map
- How would you implement a lock manager?
  - Consider locks on tuples vs locks on pages vs locks on tables
  - How should the implementation change if we are operating in memory vs. on disk?

Further Concepts

- MVCC
  - http://db.in.tum.de/~muehlbau/papers/mvcc.pdf
- Isolation Levels
- Tree Locking
- Predicate Locking