Transaction Management

Concurrency Control (3)

Review: Enforcing Serializability by Locks

- We introduce two new actions:
  - li (X): *lock* database element X
  - ui (X): *unlock* database element X, i.e. release lock.
- A locking protocol must guarantee the consistency of transactions:

$$T_i: \quad \ldots\ l_i(A)\ \ldots\ p_i(A)\ \ldots\ u_i(A)\ \ldots$$

- A locking protocol must also guarantee the legality of schedules:

$$S = \text{........}\ l_i(A)\ \underset{\text{no } l_j(A)}{\underset{\longleftrightarrow}{\text{............}}}\ u_i(A)\ \text{.........}$$

Enforcing Serializability by Locks

- Schedule F

T1	T2	A	B
		25	25
$l_1(A)$;Read(A)			
A←A+100;Write(A);$u_1(A)$		125	
	$l_2(A)$;Read(A)		
	A←Ax2;Write(A);$u_2(A)$	250	
	$l_2(B)$;Read(B)		
	B←Bx2;Write(B);$u_2(B)$		50
$l_1(B)$;Read(B)			
B←B+100;Write(B);$u_1(B)$			150
		250	150

- Schedule F is legal, but not serializable.

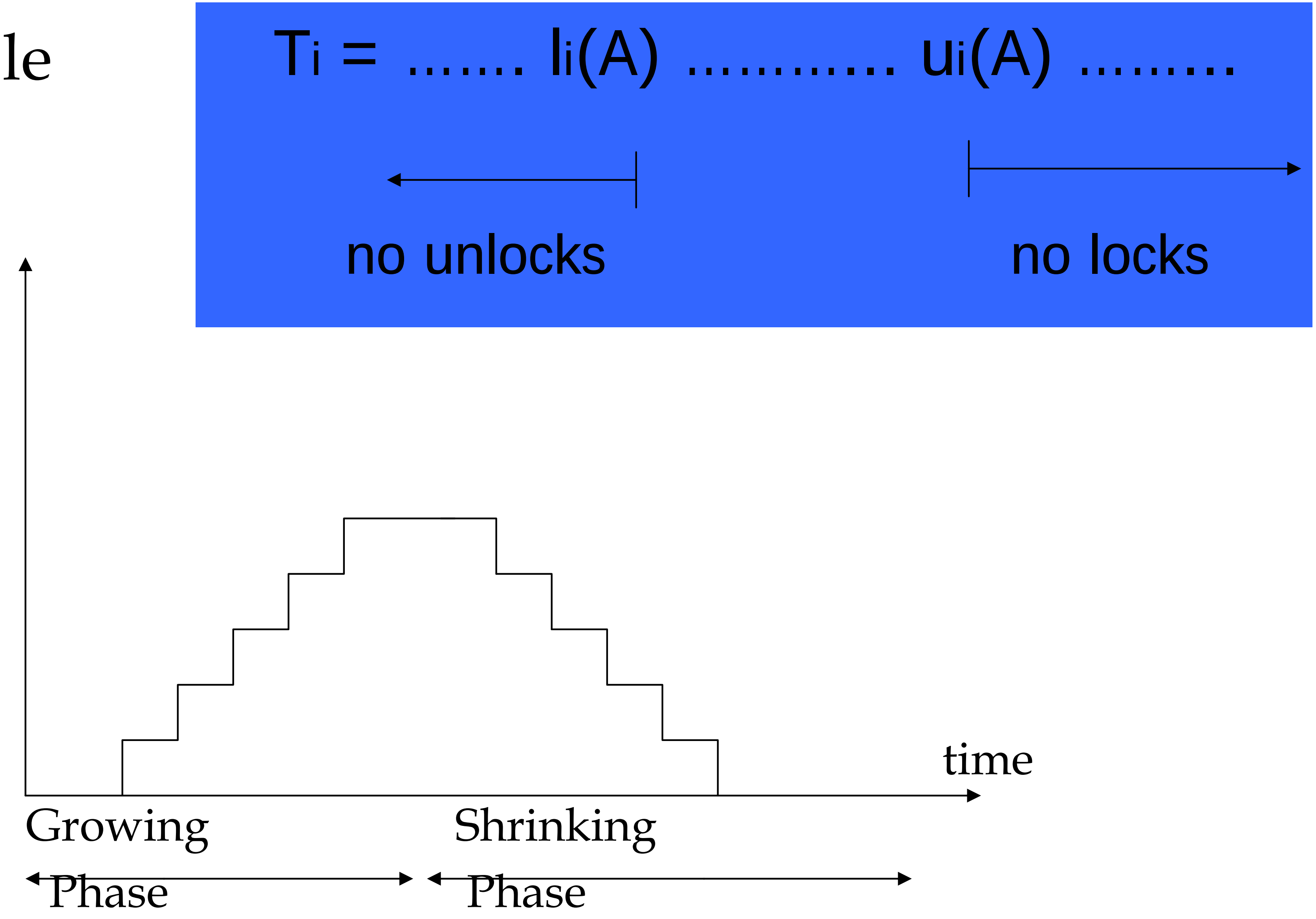
Two-Phase Locking

- A legal schedule of consistent transactions is not necessarily conflict-serializable.
- However, a legal schedule with the following locking protocol is conflict-serializable.
- *Two-phase locking (2PL)*

  In every transaction, all lock actions precede all unlock actions.
- *Growing phase:* acquire locks, no unlocks.
- *Shrink phase:* release locks, no locks.

Two-Phase Locking

- Example



T_i = ... $l_i(A)$ .... $u_i(A)$ ....

no unlocks

no locks

Two-Phase Locking

- Schedule G

T1	T2
$l_1(A)$;Read(A)	
A←A+100;Write(A)	
$l_1(B)$; $u_1(A)$	
changed order!	delayed
	$l_2(A)$;Read(A)
	A←Ax2;Write(A); $l_2(B)$
Read(B);B ← B+100	
Write(B); $u_1(B)$	
	$l_2(B)$; $u_2(A)$;Read(B)
	B ← Bx2;Write(B);$u_2(B)$;

Schedule G is serializable.

Two-Phase Locking

- In 2PL, each transaction may be thought of as executing all of its actions when issuing the first unlock action.
- Thus, the order according to the first unlock action defines a conflict-equivalent serial schedule.
- Theorem 3

  (1) legality of schedule, and (2) consistency of transactions and (3) 2PL

  $\Rightarrow$ conflict-serializability.

Two-Phase Locking

- Lemma 4

  $Ti \rightarrow Tj$ in S $\Rightarrow$ $SH(Ti) <_S SH(Tj)$

  where Shrink(Ti) = SH(Ti) = first unlock action of Ti

- Proof

  $Ti \rightarrow Tj$ means that

  S = … pi(A) … qj(A) … and pi,qj conflict

  According to (1), (2):

  $S = \ldots \underleftarrow{pi(A) \ldots ui(A)} \ldots \underrightarrow{lj(A) \ldots qj(A)} \ldots$

  According to (3): SH(Ti) SH(Tj)

  Therefore, $SH(Ti) <_S SH(Tj)$.

Two-Phase Locking

- Proof of theorem 3

  Given a schedule S.

    Assume P(S) has cycle

    T1 → T2 →…. Tn → T1

  By lemma 4: SH(T1) < SH(T2) < ... < SH(T1).

  Contradiction, so P(S) acyclic.

  By theorem 2, S is conflict serializable.

- 2PL allows only serializable schedules.

Two-Phase Locking

- Not all serializable schedules are allowed by 2PL.
- Example S1: w1(x) w3(x) w2(y) w1(y)
- The lock by T1 for y must occur after w2(y), so the unlock by T1 for x must also occur after w2(y) (according to 2PL).
- Because of the schedule legality, w3(x) cannot occur where shown in S1 because T1 holds the x lock at that point.
- However, S1 serializable (equivalent to T2, T1, T3).

Two-Phase Locking

- *Deadlocks* may happen under 2PL, when two or more transactions have got a lock and are waiting for another lock currently held by one of the other transactions.
- Example (T2 reversed)

```
T1: Read(A, t)              T2:   Read(B,s)
    t ← t+100                     s ← s×2
    Write(A,t)                    Write(B,s)
    Read(B,t)                     Read(A,s)
    t ← t+100                     s ←  s×2
    Write(B,t)                    Write(A,s)
```

Two-Phase Locking

- Possible schedule

T1	T2
l_1(A); Read(A)	l_2(B);Read(B)
A ← A+100;Write(A)	B ← Bx2;Write(B)
l_1(B)	l_2(A)
delayed, wait for T2	delayed, wait for T1

- Deadlock cannot be avoided, but can be detected (cycle in wait graph).
- At least one of the participating transactions needs to be aborted by the DBMS.

Two-Phase Locking

- So far, we have introduced the simplest possible 2PL protocol and showed that it works.
- There are many approaches for improving its performance, i.e. allowing a higher degree of concurrency:

  - shared locks,
  - increment locks,
  - multiple granularity locks,
  - tree-based locks.

Shared and Exclusive Locks

- In principle, several transactions can read database element A at the same time, as long as none is allowed to write A.
- In order to enable more concurrency, we distinguish two different types of locks:

- *shared (S) lock*: there can be multiple shared locks on X, permission only to read A.
- *exclusive (X) lock*: there can be only one exclusive lock on A, permission to read and write A.

Shared and Exclusive Locks

- We introduce the following lock actions for database element A and transaction i:
  sl-i(A): lock A in shared mode
  xl-i(A): lock A in exclusive mode
  u-i(A): unlock whatever modes Ti has locked A
- Modify *consistency* of transactions as follows:

- A read action ri(A) must be preceded by sl-i(A) or xl-i(A) with no intervening ui(A).
- A write action wi(A) must be preceded by xl-i(A) with no intervening ui(A).

Shared and Exclusive Locks

- Typically, a transaction does not know its needs for locks in advance.
- What if transaction Ti reads and writes the same database element A?
- Ti will request both shared and exclusive locks on A at different times.
- Example

  T1=... sl-1(A) … r1(A) ... xl-1(A) …w1(A) ...u1(A)…
- If Ti knows lock needs, request X lock right away.

Shared and Exclusive Locks

- Modify *legality of schedules* as follows:
  - If xl-i(A) appears in a schedule, then there cannot follow an xl-j(A) or sl-j(A), $i \neq j$, without an intervening ui(A).
  - If sl-i(A) appears in a schedule, then an xl-j(A) cannot follow without an intervening ui(A).
- All other consistency and legality as well as the 2PL requirements remain unchanged.
- The proof of Theorem 3 still works.

Shared and Exclusive Locks

- A *compatibility matrix* is a convenient way to specify a locking protocol.
- Rows correspond to lock already held by another transaction, columns correspond to a lock being requested by current transaction.

		Lock requested	
		Shared (S)	Exclusive (X)
Lock held in mode	Shared (S)	Yes	No
	Exclusive (X)	No	No

Shared and Exclusive Locks

- If a transaction first reads A and later writes A, it has to *upgrade* its S lock to an X lock.
- Upgrading is a frequent source of deadlocks.

T1	T2
sl-1(A)	
	sl-2(A)
r1(A)	
	r2(A)
xl-1(A)	
	xl-2(A)
w1(A)	

Update Locks

- In order to avoid such deadlocks (as far as possible), we introduce another type of lock.
- An *update lock (U)* ul-i(A) gives transaction i the privilege to
  - read database element A;
  - upgrade its lock on A to an X lock.
- An update lock is not shared.
- Read locks cannot be upgraded.

Update Locks

- Compatibility matrix

Lock requested

Lock held in mode

	S	X	U
S	Yes	No	Yes
X	No	No	No
U	No	No	No

U is not symmetric!

- Example

T1	T2
ul-1(A)	
	ul-2(A)
r1(A)	
xl-1(A)	
w1(A)	

Next to Discuss

- Locks With Multiple Granularity (Chapter 18.6)
- Concurrency Control by Validation (Chapter 18.9)