PURDUE UNIVERSITY | Department of Computer Science

CS 44800: Introduction To Relational Database Systems

Locking

Prof. Chris Clifton

4 November 2021

Indiana Center for Database Systems ™

PURDUE UNIVERSITY
Department of Computer Science

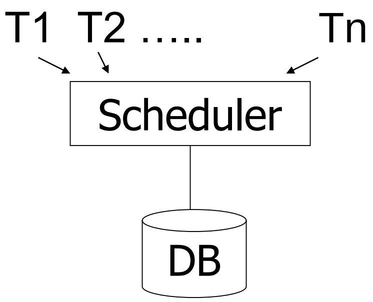
How to enforce serializable schedules?

- Option 1: run system, recording P(S);
  - at end of day, check for P(S)cycles and declare if execution was good

Department of Computer Science

How to enforce serializable schedules?

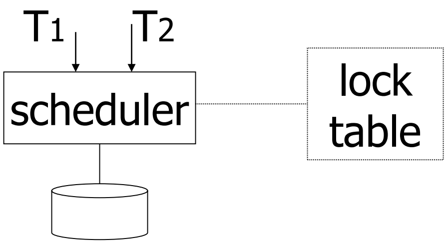
- Option 2: prevent P(S) cycles from occurring



Department of Computer Science

A locking protocol

- Two new actions:
  - lock (exclusive): l_i (A)
  - unlock: u_i (A)



Rules for Locking

- Rule #1: Well-formed transactions
  - T_i: … $l_i(A)$ … $p_i(A)$ … $u_i(A)$ ...
    (p_i is a read or write)
- Rule #2: Legal scheduler
  - S = …….. $l_i(A)$ $\underset{\longleftrightarrow}{\text{...........}}$ $u_i(A)$ ……..

no $l_j(A)$

Exercise:

- What schedules are legal?
  What transactions are well-formed?
  - S1 =
    $l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$
  - S2 = $l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$
  - S3 =
    $l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

PURDUE UNIVERSITY
Department of Computer Science

Schedule F

A	B
25	25

T1	T2
l_1(A);Read(A)	
A←A+100;Write(A);u_1(A)	
	l_2(A);Read(A)
	A←Ax2;Write(A);u_2(A)
	l_2(B);Read(B)
	B←Bx2;Write(B);u_2(B)
l_1(B);Read(B)	
B←B+100;Write(B);u_1(B)	

PURDUE UNIVERSITY
Department of Computer Science

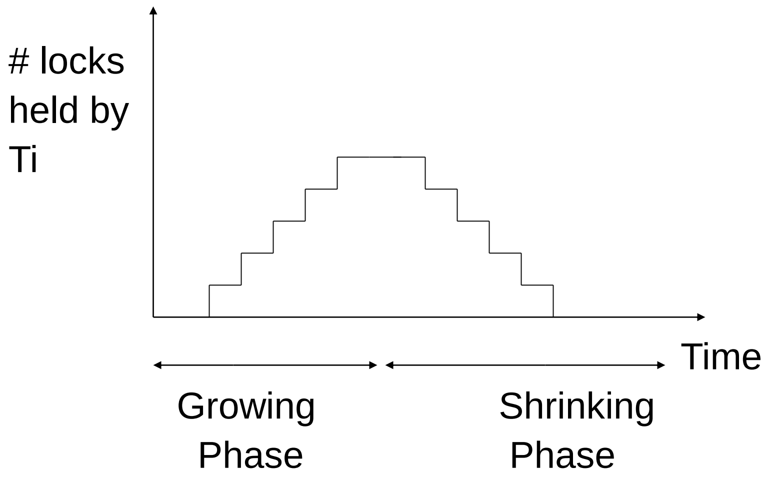
Rule #3: Two phase locking (2PL)

Ti = ……. li(A) ………... ui(A) ……...

no unlocks ← | … | → no locks

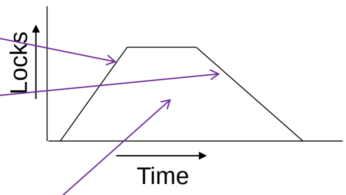
Department of Computer Science

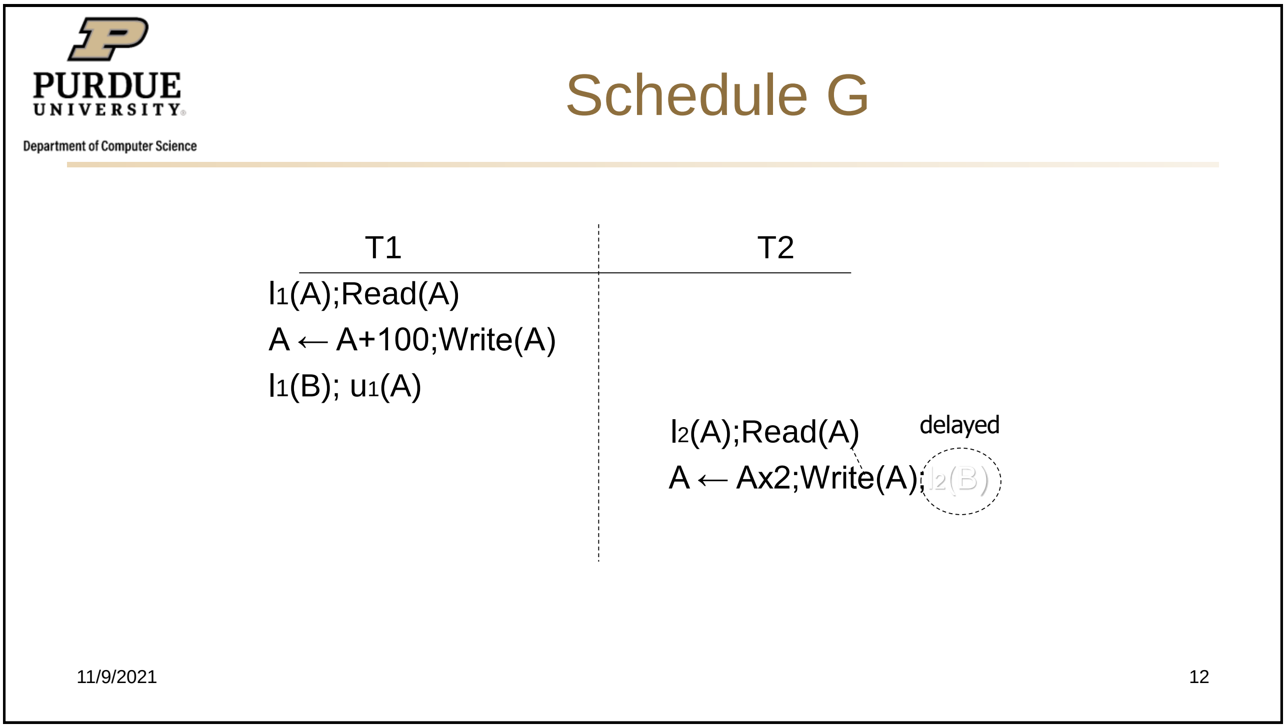
Two phase locking (2PL)



The Two-Phase Locking Protocol

- A protocol which ensures conflict-serializable schedules.
- Phase 1: **Growing Phase**
  - Transaction may obtain locks
  - Transaction may not release locks
- Phase 2: **Shrinking Phase**
  - Transaction may release locks
  - Transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e., the point where a transaction acquired its final lock).

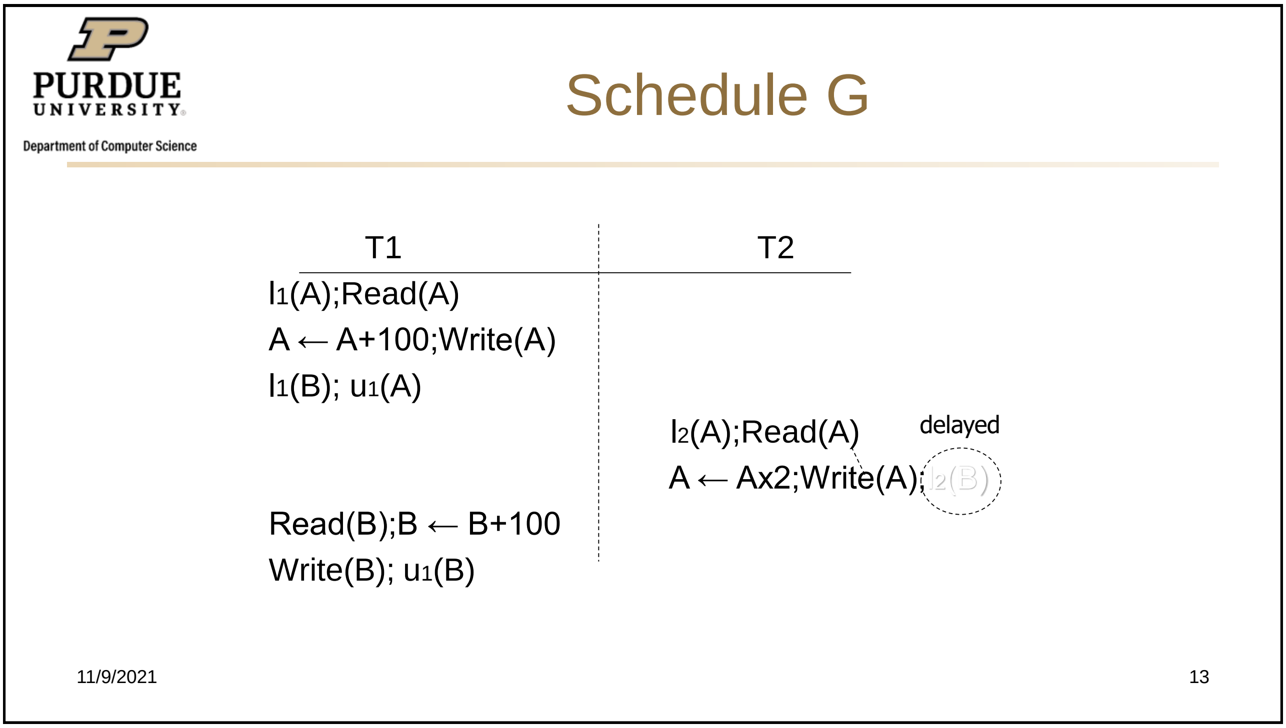




PURDUE UNIVERSITY
Department of Computer Science

Schedule G

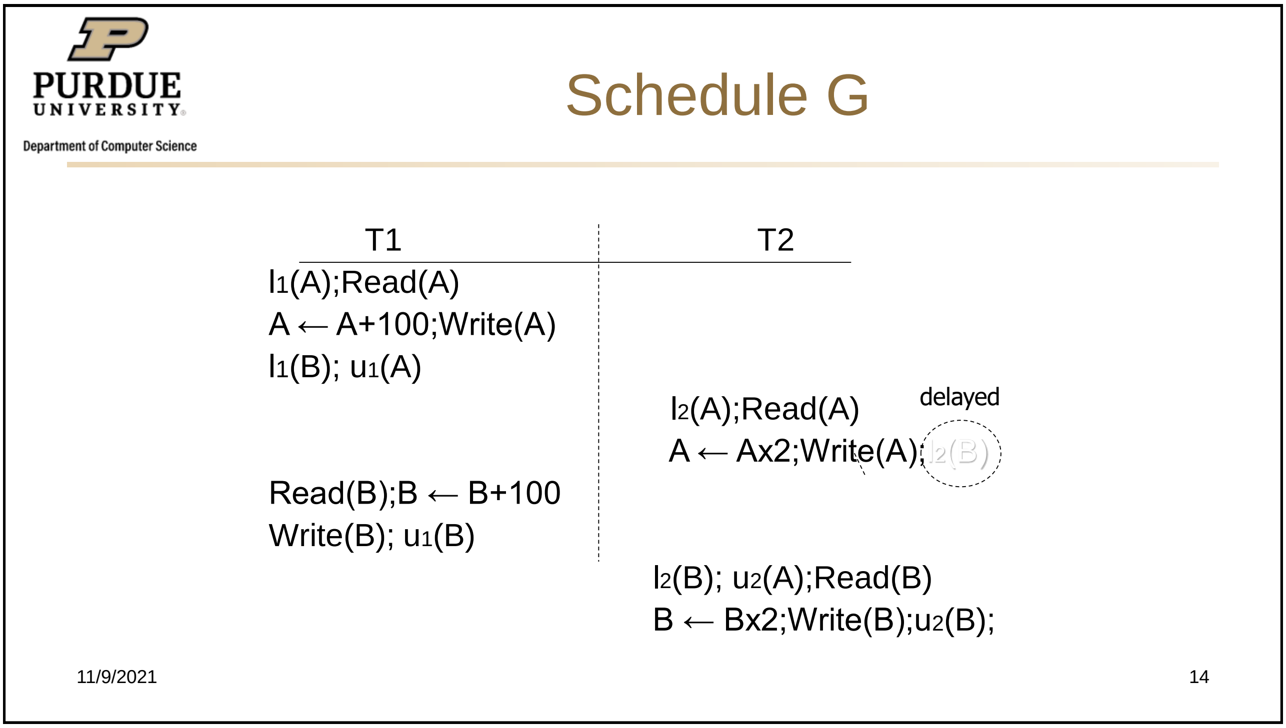
T1	T2
$l_1(A)$;Read(A)	
A ← A+100;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	A ← Ax2;Write(A); $l_2(B)$ delayed



PURDUE UNIVERSITY
Department of Computer Science

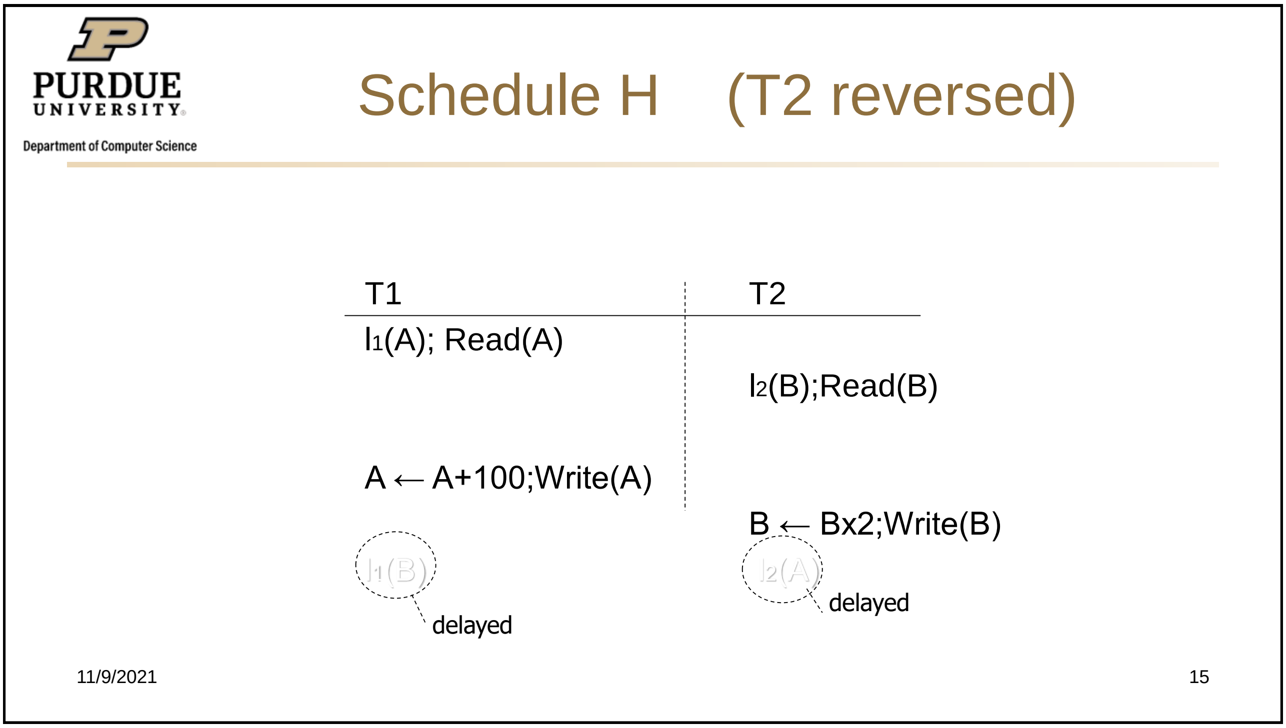
Schedule G

T1	T2
$l_1(A)$;Read(A)	
A ← A+100;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	A ← Ax2;Write(A); $l_2(B)$ delayed
Read(B);B ← B+100	
Write(B); $u_1(B)$	



Schedule G

T1	T2
$l_1(A)$;Read(A)	
$A \leftarrow A+100$;Write(A)	
$l_1(B)$; $u_1(A)$	
	$l_2(A)$;Read(A)
	$A \leftarrow Ax2$;Write(A);$l_2(B)$ delayed
Read(B);$B \leftarrow B+100$	
Write(B); $u_1(B)$	
	$l_2(B)$; $u_2(A)$;Read(B)
	$B \leftarrow Bx2$;Write(B);$u_2(B)$;

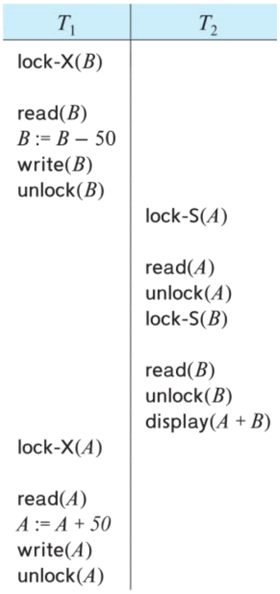


Schedule H (T2 reversed)

T1	T2
$l_1(A)$; Read(A)	
	$l_2(B)$;Read(B)
$A \leftarrow A+100$;Write(A)	
	$B \leftarrow Bx2$;Write(B)
$l_1(B)$ delayed	$l_2(A)$ delayed

The Two-Phase Locking Protocol (Cont.)

- Two-phase locking is not a necessary condition for serializability
  - There are conflict serializable schedules that cannot be obtained if the two-phase locking protocol is used.
- In the absence of extra information (e.g., ordering of access to data), two-phase locking is necessary for conflict serializability *in the following sense*:
  - *Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable.*



T_1	T_2
lock-X(B)	
read(B) $B := B - 50$ write(B) unlock(B)	
	lock-S(A)
	read(A) unlock(A) lock-S(B)
	read(B) unlock(B) display($A + B$)
lock-X(A)	
read(A) $A := A + 50$ write(A) unlock(A)	

Schedule H: Deadlock

- Assume deadlocked transactions are rolled back
  - They have no effect
  - They do not appear in schedule

E.g., Schedule H = $\underbrace{\qquad\qquad\qquad}$

This space intentionally left blank!

Deadlock: Another Example

- Consider the partial schedule

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Neither T_3 nor T_4 can make progress — executing **lock-S**(*B*) causes T_4 to wait for T_3 to release its lock on *B*, while executing **lock-X**(*A*) causes T_3 to wait for T_4 to release its lock on *A*.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

Deadlock (Cont.)

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- **Starvation** is also possible if concurrency control manager is badly designed. For example:
  - A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
  - The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.

The Two-Phase Locking Protocol (Cont.)

- Two-phase locking *does not* ensure freedom from deadlocks
- Extensions to basic two-phase locking needed to ensure recoverability of freedom from cascading roll-back
  - **Strict two-phase locking:** a transaction must hold all its exclusive locks till it commits/aborts.
    - Ensures recoverability and avoids cascading roll-backs
  - **Rigorous two-phase locking**: a transaction must hold *all* locks till commit/abort.
    - Transactions can be serialized in the order in which they commit.
- Most databases implement rigorous two-phase locking, *but refer to it as simply two-phase locking*

PURDUE UNIVERSITY
Department of Computer Science

Next step:

- Show that rules #1,2,3 $\Rightarrow$ conflict-serializable schedules

Conflict rules for $l_i(A)$, $u_i(A)$:

- $l_i(A)$, $l_j(A)$ conflict
- $l_i(A)$, $u_j(A)$ conflict

Note: no conflict < $u_i(A)$, $u_j(A)$>, < $l_i(A)$, $r_j(A)$>,...

Theorem Rules #1,2,3 $\Rightarrow$ conflict
(2PL) serializable
schedule

To help in proof:

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

Lemma

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

Proof of lemma:

Ti $\rightarrow$ Tj means that

S = … $p_i(A)$ … $q_j(A)$ …; p,q conflict

By rules 1,2:

S = … $p_i(A)$ … $u_i(A)$ … $l_j(A)$ ... $q_j(A)$ …

By rule 3: SH(Ti) SH(Tj)

So, SH(Ti) $<_S$ SH(Tj)

Theorem Rules #1,2,3 ⇒ conflict
(2PL) serializable
schedule

Proof:

(1) Assume P(S) has cycle

$$T_1 \rightarrow T_2 \rightarrow .... T_n \rightarrow T_1$$

(2) By lemma: $SH(T_1) < SH(T_2) < ... < SH(T_1)$

(3) Impossible, so P(S) acyclic

(4) ⇒ S is conflict serializable

PURDUE UNIVERSITY
Department of Computer Science

- Beyond this simple 2PL protocol, it is all a matter of improving performance and allowing more concurrency….
  - – Shared locks
  - – Multiple granularity
  - – Inserts, deletes and phantoms
  - – Other types of C.C. mechanisms