



Concurrency Control



- Schedule
 - Conflict Serializable Schedule
 - Two phase locking
 - Warning Protocol
 - Validation

Database System - Nankai



Week15_Course

Database System_Concurrency Control part1

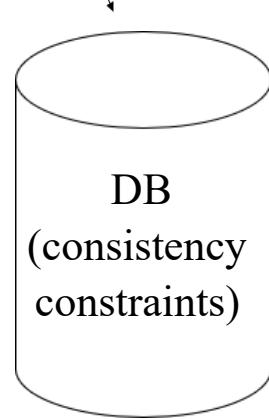
Schedule

Database System - Nankai



Concurrency Control

T1 T2 ... Tn



How to prevent harmful interference
between transactions?

=> **scheduling techniques** based on
- locks
- timestamps and validation

Database System - Nankai



Correctness depends on scheduling of transactions

A schedule

- Chronological (possibly interleaving) order in which actions of transactions are executed
- A correct schedule is equivalent to executing transactions one-at-a-time in some order

T1	T2	T3
Write (A)	Write (B)	Write (C)
Write (B)	Write (C)	Write (A)
Write (D)	Write (D)	

Database System - Nankai



Example:

Constraint: A=B

T1: Read(A)
 $A \leftarrow A + 100$
Write(A)
Read(B)
 $B \leftarrow B + 100$
Write(B)

T2: Read(A)
 $A \leftarrow A \times 2$
Write(A)
Read(B)
 $B \leftarrow B \times 2$
Write(B)

Database System - Nankai



Schedule A

T1

```
Read(A); A ← A+100;  
Write(A);  
Read(B); B ← B+100;  
Write(B);
```

T2

```
Read(A);A ← A×2;  
Write(A);  
Read(B);B ← B×2;  
Write(B);
```

$S_A = r1(A)w1(A) \text{ r1}(B)w1(B)r2(A)w2(A)r2(B)w2(B)$

OK

A	B
25	25
125	125
250	125
250	250
250	250

Database System - Nankai



Schedule B

T1

Read(A); A \leftarrow A+100;
Write(A);
Read(B); B \leftarrow B+100;
Write(B);

T2

Read(A); A \leftarrow A×2;
Write(A);
Read(B); B \leftarrow B×2;
Write(B);

OK

A	B
25	25
50	50
150	150
150	150

Database System - Nankai



Schedule C

T1
Read(A); $A \leftarrow A+100$
Write(A);

Read(B); $B \leftarrow B+100$;
Write(B);

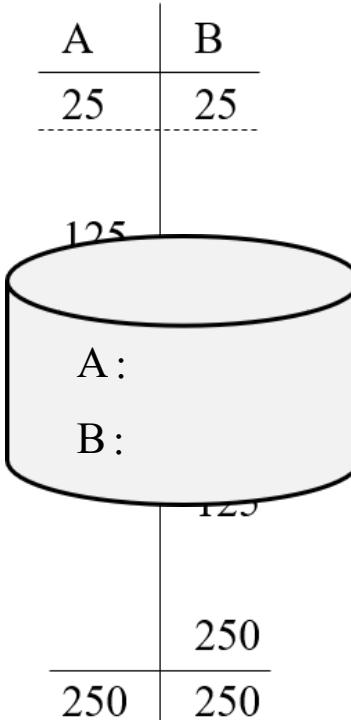
T2

Read(A); $A \leftarrow A \times 2$;
Write(A);

Read(B); $B \leftarrow B \times 2$;
Write(B);

OK

$Sc = r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)$



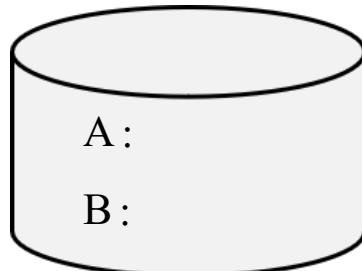
Database System - Nankai



Schedule D

T1

Read(A); $A \leftarrow A+100;$
Write(A);



Read(B); $B \leftarrow B+100;$
Write(B);

T2

Read(A); $A \leftarrow A \times 2;$
Write(A);
Read(B); $B \leftarrow B \times 2;$
Write(B);

Constraint violation!

A	B
25	25
125	
250	
50	
150	
250	150

Database System - Nankai



Schedule E

Same as Schedule D
but with new T2'

T1	T2'
Read(A); A \leftarrow A+100; Write(A);	Read(A); A \leftarrow A×1; Write(A); Read(B); B \leftarrow B×1; Write(B);
Read(B); B \leftarrow B+100; Write(B);	

A	B
25	25
125	
125	
25	
125	
125	125

Database System - Nankai



- Want schedules that are “good”, regardless of
 - initial state (\leftrightarrow “good” in any DB state) and
 - transaction semantics
- Only look at order of READs and WRITEs
 - Note: transactions see values in buffers, not on disk
=> this time ignore INPUT/OUTPUTs

Example: Sa (Schedule a) =

r1(A)w1(A)r1(B)w1(B) r2(A)w2(A) r2(B)w2(B)

T1

T2

Database System - Nankai



Scheduling Transactions: Definitions

- Serial schedule: no concurrency
 - Does not interleave the actions of different transactions.
- Equivalent schedules: same result on any DB state
 - For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- Serializable schedule: equivalent to a serial schedule
 - A schedule that is equivalent to *some* serial execution of the transactions.
(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

Database System - Nankai



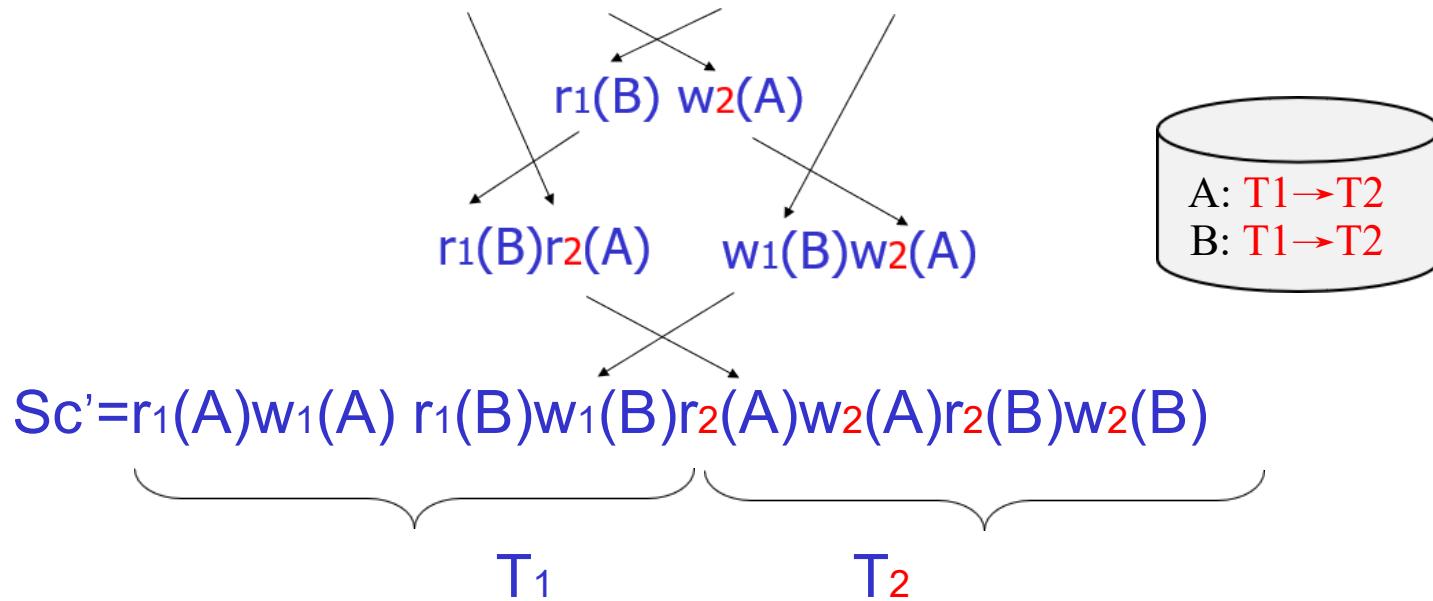
Serial Schedule

- A schedule is **serial**, if actions of transactions are not interleaved
 - e.g., (T1, T2) or (T2, T1)
 - A serial schedule obviously maintains consistency (assuming correctness of individual transactions)
- Could we reorder a schedule into an equivalent serial schedule?
 - Actions **conflict**, if swapping them may change the meaning of a schedule:
 - any two actions of a single transaction
 - two actions on a common DB element A, one of which is WRITE(A)



Example: (of swapping non-conflicting actions)

$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

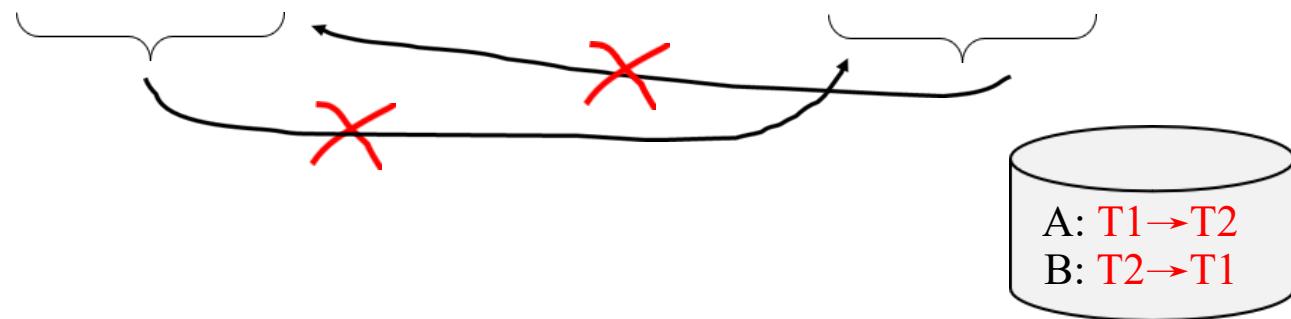


Database System - Nankai



However, for S_d :

$$S_d = r_1(A)w_1(A)r_2(A)w_2(A)r_2(B)w_2(B)r_1(B)w_1(B)$$



- S_d cannot be rearranged into a serial schedule
 - ➡ S_d is not “equivalent” to any serial schedule
 - ➡ S_d is “bad”



Summary

- Schedule
- Serial Schedule

Database System - Nankai

多选题 1分



互动交流——不定项选择题

以下哪一种写法表示的是事务的调度（Schedule）？

- A T1 = R1(C) R1(A) W1(C) R1(B) W1(D) W1(B) R1(A)
- B S1= W3(A) R1(B) R2(A) R4(D) W1(B) R4(A) W2(B) W1(D) R4(B)
- C T2 = R1(C) W1(D) W1(B) T3 = R2(A) W2(D) W2(B)
- D S2= W3(A) R1(B) W1(D) W1(B) R2(A) W2(B) R4(D) R4(A) R4(B)

提交

Database System - Nankai

多选题 1分



互动交流二

以下哪一个调度是串行调度（Serial Schedule）？

- A T1 = R1(C) R1(A) W1(C) R1(B) W1(D) W1(B) R1(A)
- B S1= W3(A) R1(B) R2(A) R4(D) W1(B) R4(A) W2(B) W1(D) R4(B)
- C T2 = R1(C) W1(D) W1(B) T23= R2(A) W2(D) W2(B)
- D S2= W3(A) R1(B) W1(D) W1(B) R2(A) W2(B) R4(D) R4(A) R4(B)

提交

Database System - Nankai



Week15_Course

Database System_Concurrency Control part2

Conflict Serializable Schedule

Database System - Nankai



Concurrency Control

- Schedule
- Conflict Serializable Schedule 
- Two phase locking
- Warning Protocol
- Validation

Database System - Nankai



Concepts

Transaction: sequence of $r_i(x)$, $w_i(x)$ actions

Conflicting actions: $r_h(A) \quad w_h(A) \quad w_h(A)$
 $w_k(A) < r_k(A) < w_k(A)$

If schedule S contains conflicting actions

$\dots, p_h(A), \dots, q_k(A), \dots$ [i.e., one of p, q is w],

transaction T_h must precede T_k in a corresponding serial schedule. Denote this by $T_h \rightarrow T_k$

Database System - Nankai



Returning to Sc

$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

$T_1 \rightarrow T_2$

$T_1 \rightarrow T_2$

No cycles \Rightarrow Sc is “equivalent” to a serial schedule
(in this case T_1, T_2)



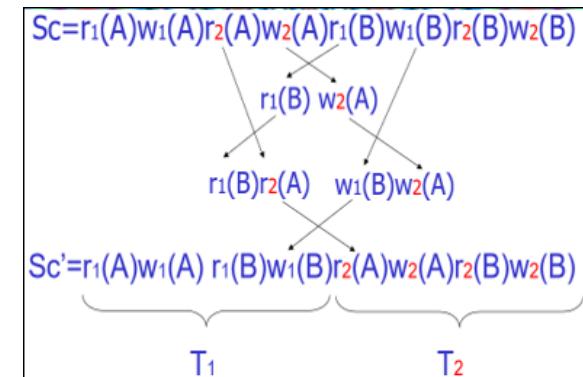
Definition

S1, S2 are **conflict equivalent** schedules

if S1 can be transformed into S2 by a series
of swaps on non-conflicting actions.

(=> effect of both S1 and S2 on the DB is the same)

A schedule is **conflict serializable** if it
is conflict equivalent to some serial
schedule.





Definition

A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule.

NB: Conflict serializability is a sufficient (but not a necessary) condition for serializability (equivalence to some serial schedule)

Easier to enforce than serializability, therefore generally assured by commercial systems



Precedence graph $P(S)$ (S is schedule)

Nodes: transactions T_1, T_2, \dots in S

Arcs: $T_i \rightarrow T_j$ for $i \neq j$ whenever

- $p_i(A), q_j(A)$ are conflicting actions in S ,
(same element A , at least one of actions is a write)
- action $p_i(A)$ precedes $q_j(A)$ in S

- What is $P(S)$ for
 $S = w3(A) w2(C) r1(A) w1(B) r1(C) w2(A) r4(A) w4(D)$
- Is S serializable?



Exercise:

- What is $P(S)$ for

$S = w_3(A) \ w_2(C) \ r_1(A) \ w_1(B) \ r_1(C) \ w_2(A) \ r_4(A) \ w_4(D)$

- Is S serializable?



Lemma Let S_1, S_2 be schedules for the same set of transactions

S_1, S_2 conflict equivalent $\Rightarrow P(S_1) = P(S_2)$

Proof:

Assume $P(S_1) \neq P(S_2)$

$\Rightarrow \exists T_i: T_i \rightarrow T_j$ in $P(S_1)$ and not in $P(S_2)$

$\Rightarrow S_1 = \dots p_i(A) \dots q_j(A) \dots$ $\left\{ \begin{array}{l} p_i, q_j \\ \text{conflict} \end{array} \right.$
 $S_2 = \dots q_j(A) \dots p_i(A) \dots$

$\Rightarrow S_1, S_2$ not conflict equivalent

Database System - Nankai



Note: $P(S_1) = P(S_2) \not\Rightarrow S_1, S_2$ conflict equivalent

Counter example:

$S_1 = w_1(A) \ r_2(A) \ w_2(B) \ r_1(B)$

$S_2 = r_2(A) \ w_1(A) \ r_1(B) \ w_2(B)$



Theorem I

$P(S_1)$ acyclic $\iff S_1$ conflict serializable

(\Leftarrow) Assume S_1 is conflict serializable

$\Rightarrow \exists$ serial S_s : S_s, S_1 conflict equivalent

$\Rightarrow P(S_s) = P(S_1)$ [\leftarrow Lemma]

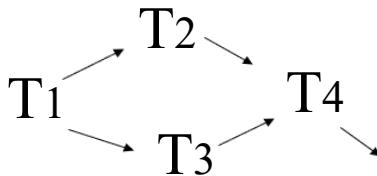
$\Rightarrow P(S_1)$ acyclic since $P(S_s)$ is acyclic



Theorem (cont.)

$P(S_1)$ acyclic $\iff S_1$ conflict serializable

(\Rightarrow) Assume $P(S_1)$ is acyclic



Transform S_1 as follows:

- (1) Take T_1 to be transaction with no incoming arcs
- (2) Move all T_1 actions to the front

$S_1 = \dots q_j(X) \dots p_1(A) \dots$



- (3) we now have $S_1 = < T_1 \text{ actions } > < \dots \text{ rest } \dots >$
- (4) repeat above steps to serialize rest!



Example of Theorem Application

For each of the following schedules, answer the questions below:

$S_b = W3(A)R1(B)R2(A)R1(D)W1(B)R4(A)W2(B)R4(D)R3(D)R4(B)$

- What is the precedence graph for the schedule S_b ?
- Is the schedule conflict serializable? If so, show all equivalent serial transaction orders. If not, describe why not.



Summary

- Schedule, Serial Schedule
- Conflict Serializable Schedule
- Precedence Graph $P(S)$
- $P(S_1)$ acyclic $\iff S_1$ conflict serializable

Database System - Nankai

多选题 1分



互动交流——不定项选择题

以下哪一个调度可以保证数据库的一致性？

A

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

B

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

C

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

提交

Database System - Nankai

多选题 1分



互动交流二 — 不定项选择题

以下调度的哪个部分应该在优先图中出现?

S= W3(**A**) R1(**B**) R2(**A**) R1(**D**) R2(**B**) W4(**A**) R1(**A**) R3(**D**) W4(**B**)

A T3 → T2 → T4 → T1

B T1 → T2 → T4

C T1 → T3

提交

Database System - Nankai



Week15_Course

Database System Concurrency Control part3

Two phase locking

Database System - Nankai



Concurrency Control

- Schedule
- Conflict Serializable Schedule
- Two phase locking
- Warning Protocol
- Validation

Database System - Nankai



How to enforce serializable schedules?

Option 1: (Optimistic strategy)

Run system, recording P(S); At end of day, check P(S) for cycles, and declare if execution was good

Option 2: (Pessimistic strategy)

Prevent occurrence of cycles in P(S)

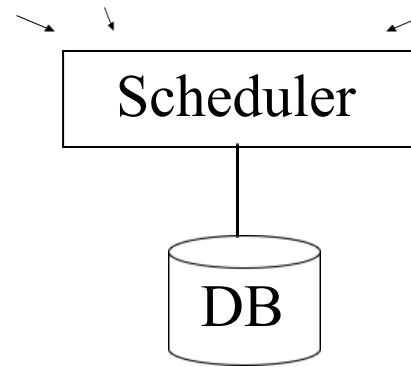


How to enforce serializable schedules?

Option 2: (Pessimistic strategy)

Prevent occurrence of cycles in $P(S)$

$T_1 \ T_2 \ \dots \ T_n$



Database System - Nankai

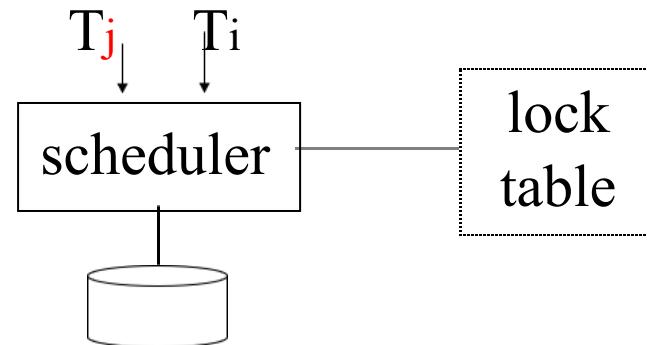


A locking protocol

Two new actions:

lock (exclusive): $l_j(A)$

unlock: $u_j(A)$



Database System - Nankai



Rule #1: Well-formed transactions

$T_i: \dots l_i(A) \dots p_i(A) \dots u_i(A) \dots$

- Lock elements (A) before accessing them
(p_i is a read or a write)
- Eventually, release the locks ($u_i(A)$)



Rule #2 Legal scheduler

$S = \dots \dots l_i(A) \dots \dots u_i(A) \dots \dots$

\longleftrightarrow
no $l_j(A)$ for $i \neq j$

- At most one transaction T_i can hold a lock on any element A



Exercise:

- What schedules are legal?
What transactions are well-formed?

$$S1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B) \\ r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$$
$$S2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B) \\ l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$$
$$S3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B) \\ l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$$

Database System - Nankai



Schedule F (with simple locking)

T1

T2

A B

$l_1(A); \text{Read}(A)$

$A := A + 100; \text{Write}(A); u_1(A)$

$l_2(A); \text{Read}(A)$

$A := Ax2; \text{Write}(A); u_2(A)$

$l_2(B); \text{Read}(B)$

$B := Bx2; \text{Write}(B); u_2(B)$

$l_1(B); \text{Read}(B)$

$B := B + 100; \text{Write}(B); u_1(B)$

Constraint violation!

25	25
125	
250	
	50
150	
250	150

Database System - Nankai



Simple-minded locking not sufficient to ensure serializability (i.e., correctness)!

→ More advanced protocol known as "two phase locking"

Database System - Nankai



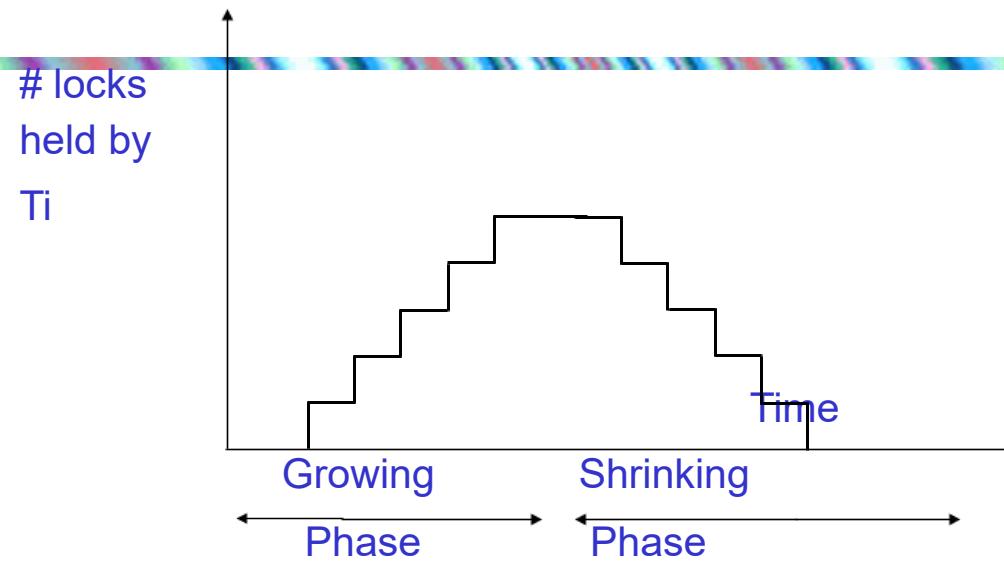
Rule #3 Two phase locking (2PL)

for transactions

$T_i = \dots \cdot l_i(A) \dots \dots \dots u_i(A) \dots \dots \dots$



- All lock requests of a transaction have to precede its unlock requests



Database System - Nankai



Schedule F (with simple locking)

T1

T2

A B

$l_1(A); Read(A)$

$A := A + 100; Write(A); u_1(A)$

$l_2(A); Read(A)$

$A := Ax2; Write(A); u_2(A)$

$l_2(B); Read(B)$

$B := Bx2; Write(B); u_2(B)$

$l_1(B); Read(B)$

$B := B + 100; Write(B); u_1(B)$

Constraint violation!

25	25
125	
250	
50	
150	
250	150

Database System - Nankai



Schedule G (with 2PL)

T1

l₁(A); Read(A)

A:=A+100;Write(A)

l₁(B); u₁(A);

Read(B);B:=B+100

Write(B); u₁(B);

T2

l₂(A); Read(A) delayed

A:=Ax2;Write(A);

l₂(B); u₂(A);Read(B)

B:=Bx2;Write(B); u₂(B);

Database System - Nankai



Next step:

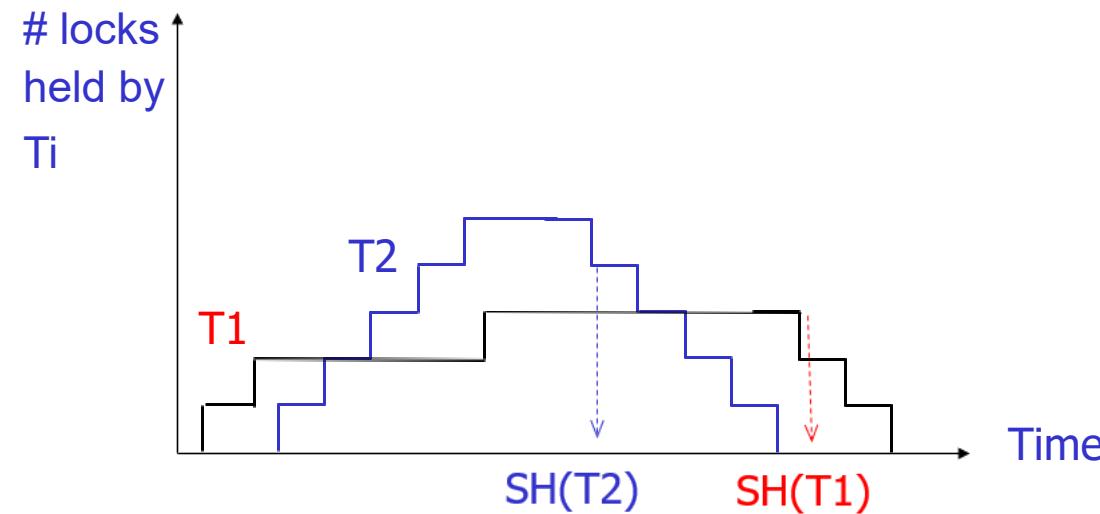
Show that Rules #1,2,3 \Rightarrow conflict
(2PL) serializable schedule

Database System - Nankai



To help in proof:

Definition $\text{Shrink}(T_i) = \text{SH}(T_i)$ =first unlock action of T_i



Database System - Nankai



Lemma Let S be a 2PL schedule.

$$T_i \rightarrow T_j \text{ in } P(S) \Rightarrow SH(T_i) <_S SH(T_j)$$

Proof of lemma:

$T_i \rightarrow T_j$ means that

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

By rules 1,2:

$S = \dots p_i(A) \dots u_i(A) \dots l_j(A) \dots q_j(A) \dots$



By rule 3: $SH(T_i) \qquad SH(T_j)$

So, $SH(T_i) <_S SH(T_j)$

Database System - Nankai



Theorem II Rules #1,2,3
(that is, 2PL) \Rightarrow { conflict
 serializable
 schedule }

Proof: Let S be a 2PL schedule.

Assume $P(S)$ has cycle

$$T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$$

By Lemma: $SH(T_1) < SH(T_2) < \dots < SH(T_1)$

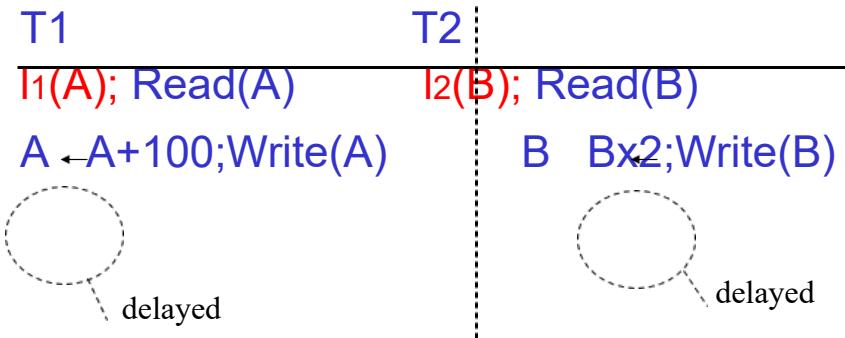
Impossible, so $P(S)$ acyclic

$\Rightarrow S$ is conflict serializable (by Th. I)

Database System - Nankai



Schedule H (T₂ reversed)



- Neither proceeds: a **deadlock**
 - System must rollback (= abort & restart) at least one of T1, T2

Database System - Nankai



死锁的预防

- 一次封锁法
 - 要求每个事务必须一次将所有要使用的数据全部加锁，否则就不能继续执行
 - 存在的问题：降低系统并发度、难于事先精确确定封锁对象
- 顺序封锁法
 - 顺序封锁法是预先对数据对象规定一个封锁顺序，所有事务都按这个顺序实行封锁。
 - 顺序封锁法存在的问题：维护成本高、难以实现

Database System - Nankai



Summary-Two phase locking

- 能够判断一个事务是否是Well-formed transactions
- 能够判断一个调度是否是Legal scheduler
- 能够判断一个事务是否满足two phase locking
- 可以在一个事务中添加加锁和解锁动作，使其满足规则1和规则3
- 对满足规则1和规则3的事务可以生成可串行化的调度

Database System - Nankai