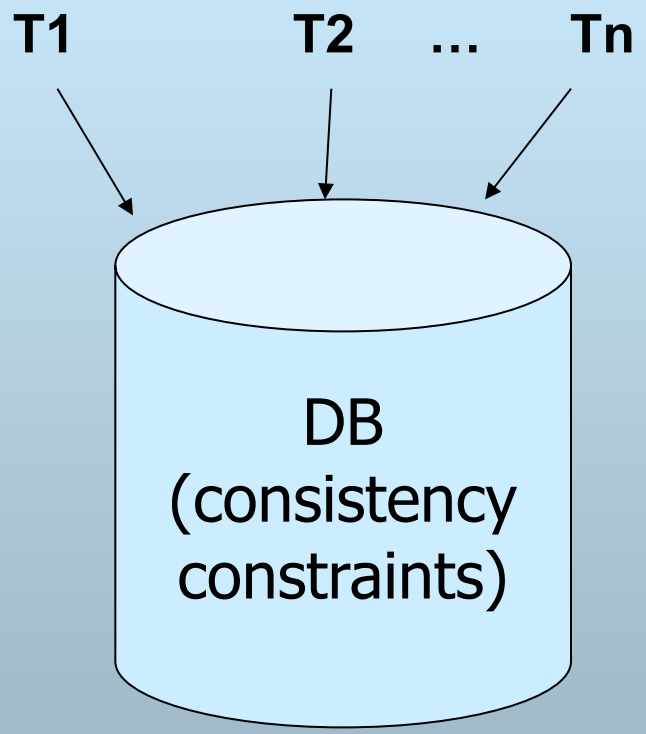
Concurrency Control

金培权（jpq@ustc.edu.cn）

Databases protection

- 数据库保护：排除和防止各种对数据库的干扰破坏，确保数据安全可靠，以及在数据库遭到破坏后尽快地恢复
- 数据库保护通过四个方面来实现
  - 数据库的恢复技术 [Chp. 17]
    - Deal with failure
  - 并发控制技术 [Chp. 18 & 19]
    - Deal with data sharing
  - 完整性控制技术
    - Enable constraints
  - 安全性控制技术
    - Authorization and authentication

Concurrency Control



多个事务同时存取共享的数据库时，如何保证数据库的一致性?

- 丢失更新 Lost update
- 脏读 Dirty read
- 不一致分析 Inconsistent analysis
  - 不可重复读 Nonrepeatable read
  - 幻像读 Phantom read

主要内容

- 并发操作与并发问题
- 并发事务调度与可串性 (Scheduling and Serializability)
- 锁与可串性实现 (Locks)
- 事务的隔离级别
- 死锁
- 乐观并发控制

一、并发操作和并发问题

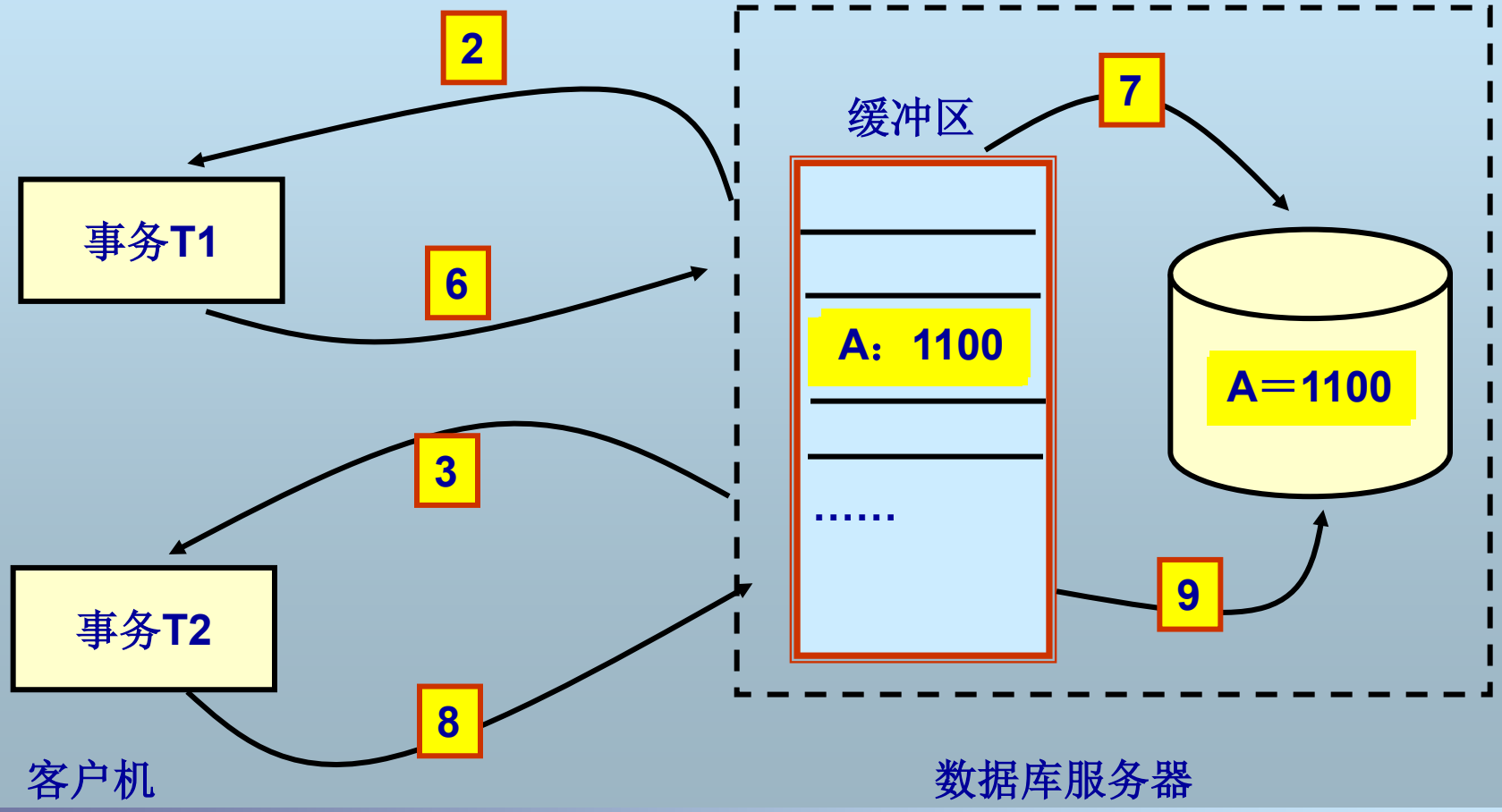
- 并发操作
  - 在多用户DBS中，如果多个用户同时对同一数据进行操作称为并发操作
  - 并发操作使多个事务之间可能产生相互干扰，破坏事务的隔离性（Isolation）
  - DBMS的并发控制子系统负责协调并发事务的执行，保证数据库的一致性，避免产生不正确的数据
- 并发操作通常会引起三类问题（三大并发问题）
  - 丢失更新（Lost update）
  - 脏读（Dirty read / Uncommitted update）
  - 不一致分析 （Inconsistent analysis）

1、丢失更新问题

时间	事务T1	事务T2	数据库中A的值
1			1000
2	Read(A,t)		
3		Read(A,t)	
4	t=t-100		
5		t=t+100	
6	Write(A,t)		
7	Commit		900
8		Write(A,t)	
9		Commit	1100

并发执行造成丢失更新示例

时间	事务T1	事务T2	数据库中A的值
1			1000
2	Read(A,t)		
3		Read(A,t)	
4	t=t-100		
5		t=t+100	
6	Write(A,t)		
7	Commit		900
8		Write(A,t)	
9		Commit	1100



再论丢失更新问题

Step	T1	T2
1	read(A, t), t = t+1	
2		read(A, t), t = t +1
3	write(A, t)	
3		write(A, t)
4		Commit
6	Commit	

Lost update

两次提交写导致的写覆盖

Step	T1	T2
1	read(A, t), t = t+1	
2		read(A, t), t = t + 1
3	write(A, t)	
4		write(A, t)
5		read(B, u), u = u + 1
6		write(B, u)
7		Commit
8	Abort	

Dirty write

由于Rollback导致的提交事务的写失效
破坏了T2的原子性

DBMS中不允许出现Dirty write
在任何情况下都要求X锁保留到事务结束

Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, Patrick E. O'Neil:
A Critique of ANSI SQL Isolation Levels. SIGMOD 1995: 1-10

2、脏读问题

时间	事务T1	事务T2	数据库中A的值
1			1000
2	Read(A,t)		
3	t=t-100		
4	Write(A,t)		
5		Read(A,t)	
6	Rollback	t=t+100	900
7		Write(A,t)	
8		Commit	1000

脏数据：事务在内存中更新了但还未最终提交的数据

3、不一致分析问题

时间	事务T1	事务T2
1		
2	Read(A,t)	Read(B,t)
3	t=t-100	
4		Read(A,v)
5	Write(A,t)	
6	Commit	
7		Sum=t+v
8		Commit

不可重复读
Nonrepeatable read

事务内读的数据被其它事务update或者delete了

Read(A,v)=?

Sum不是数据库实际汇总值

不一致分析问题：事务读了过时的数据，不是数据库的当前状态

3、不一致分析问题

时间	事务T1	事务T2
1		
2		Read(B,t)
3		
4	t=100	Read(A,v)
5	Write(C,t)	
6	Commit	
7		Sum=t+v
8		Commit

Insert a new C

幻像读
Phantom read

事务内读到的数据内容被其它事务的insert操作改变了

Sum不是数据库实际汇总值

不一致分析问题：事务读了过时的数据，不是数据库的当前状态

4、并发控制的问题该如何解决？

- 一种方法
  - 让所有事务一个一个地串行执行
    - 一个事务在执行时其它事务只能等待
    - 不能充分利用系统资源，效率低下
- 另一种方法
  - 为了充分发挥DBMS共享数据的特点，应允许事务内部的读写操作并发执行
  - 挑战
    - 必须保证事务并发执行的正确性；必须用正确方法调度执行事务的并发操作

二、调度(Schedule)

Example

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

Constraint: A=B

二、调度(Schedule)

Schedule A

T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
Read(B, t); t ← t+100;			
Write(B, t);		125	125
	Read(A, s); s ← s×2;		
	Write(A, s);	250	125
	Read(B, s); s ← s×2;		
	Write(B, s);	250	250

二、调度(Schedule)

Schedule B

T1	T2	A	B
	Read(A, s); s ← s×2;	25	25
	Write(A, s);	50	25
	Read(B, s); s ← s×2;		
	Write(B, s);	50	50
Read(A, t); t ← t+100			
Write(A, t);		150	50
Read(B, t); t ← t+100;			
Write(B, t);		150	150

二、调度(Schedule)

Schedule C

T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
Read(B, t);			
t ← t+100;			
Write(B, t);		250	125
	Read(B, s); s ← s×2;		
	Write(B, s);	250	250

二、调度(Schedule)

Schedule D

T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
	Read(B, s); s ← s×2;		
	Write(B, s);	250	50
Read(B, t);			
t ← t+100;			
Write(B, t);		250	150

二、调度(Schedule)

Schedule D

T1	T2′	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×1;		
	Write(A, s);	125	25
	Read(B, s); s ← s×1;		
	Write(B, s);	125	25
Read(B, t);			
t ← t+100;			
Write(B, t);		125	125

1、调度的定义

- 调度
  - 多个事务的读写操作按时间排序的执行序列

T1: r1(A) w1(A) r1(B) w1(B)
T2: r2(A) w2(A) r2(B) w2(B)
Sc = r1(A) w1(A) r2(A) w2(A) r1(B) w1(B) r2(B) w2(B)

Note

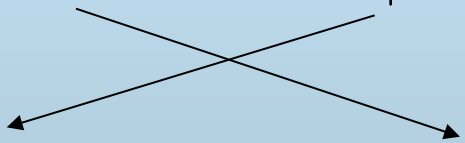
- 调度中每个事务的读写操作保持原来顺序
- 事务调度时不考虑
  - 数据库的初始状态 (Initial state)
  - 事务的语义 (Transaction semantics)

1、调度的定义

- 多个事务的并发执行存在多种调度方式

Example:

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



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

T1 T2

What is a correct schedule?

And how to get a correct schedule?

2、可串化调度 (Serializable Schedule)

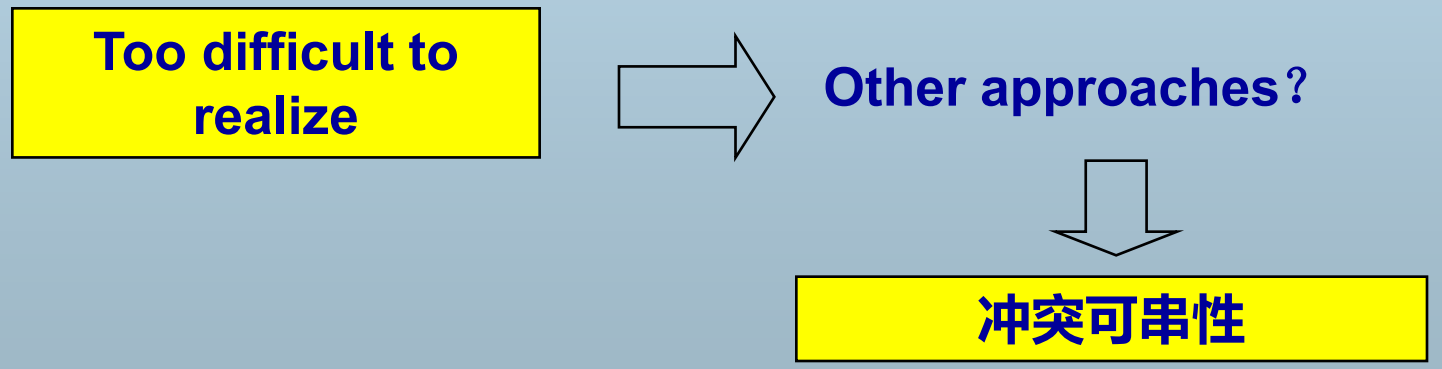
- What is a correct schedule?
  - Answer: a serializable schedule!
- 串行调度 (Serial schedule)
  - 各个事务之间没有任何操作交错执行，事务一个一个执行
  - S = T1 T2 T3 ... Tn
- Serialzable Schedule
  - 如果一个调度的结果与某一串行调度执行的结果等价，则称该调度是可串化调度，否则是不可串调度

2、可串化调度

- 可串化调度的正确性
  - Consistence of transaction: 单个事务的执行保证DB从一个一致状态变化到另一个一致状态
  - N个事务串行调度执行仍保证 Consistence of DB

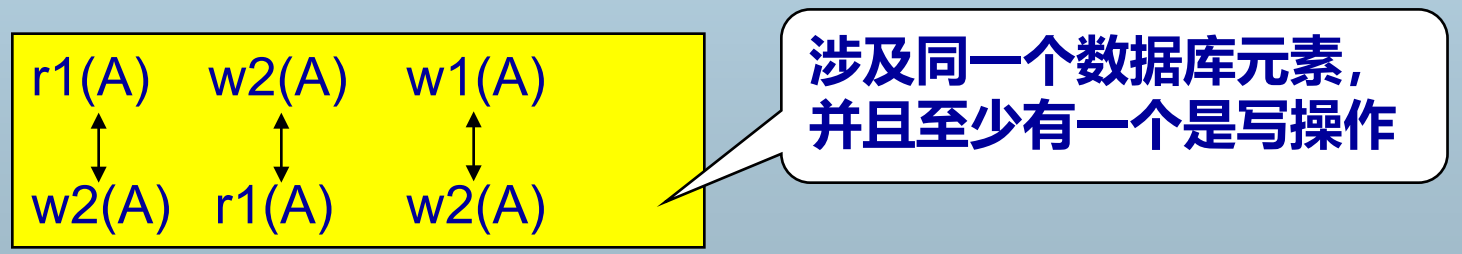
2、可串化调度

- **Is a schedule a serializable one?**
  - **We MUST**
    - **Get all results of serial schedules, n!**
    - **See if the schedule is equivalent to some serial schedule**



3、冲突可串性 (conflict serializable)

- **Conflicting actions**
  - **Say**
    - **$r_i(X)$: 事务T_i的读X操作 (Read(X, t))**
    - **$w_i(X)$:事务T_i的写X操作 (Write(X, t))**
  - **冲突操作**



3、冲突可串性 (conflict serializable)

- **Conflicting actions**
  - 如果调度中一对连续操作是冲突的，则意味着如果它们的执行顺序交换，则至少会改变其中一个事务的最终执行结果
  - 如果两个连续操作不冲突，则可以在调度中交换顺序

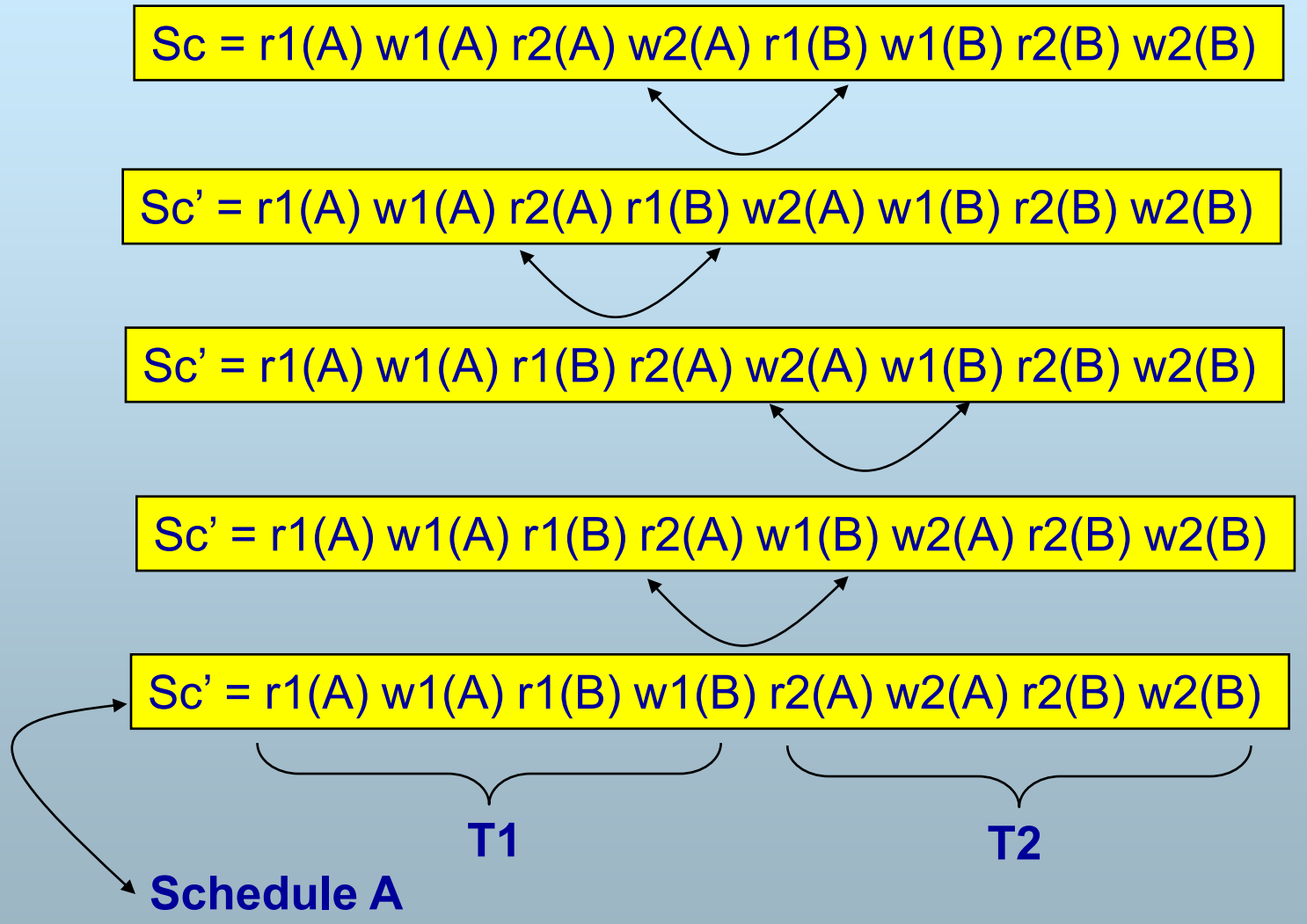
3、冲突可串性

Schedule C

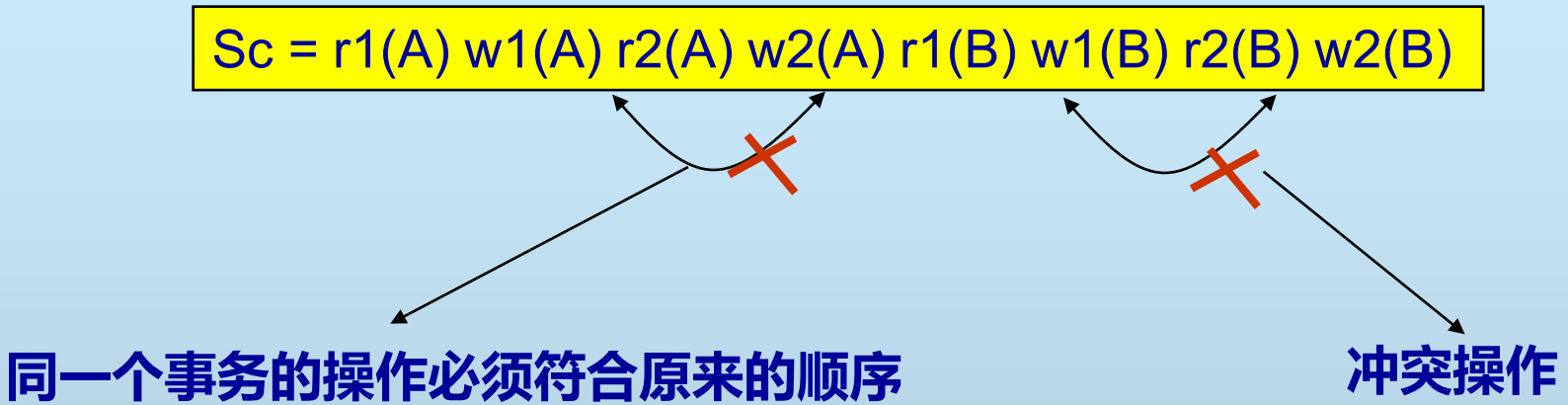
T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
Read(B, t);			
t ← t+100;			
Write(B, t);		250	125
	Read(B, s); s ← s×2;		
	Write(B, s);	250	250

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

3、冲突可串性


Sc = r1(A) w1(A) r2(A) w2(A) r1(B) w1(B) r2(B) w2(B)
Sc' = r1(A) w1(A) r2(A) r1(B) w2(A) w1(B) r2(B) w2(B)
Sc' = r1(A) w1(A) r1(B) r2(A) w2(A) w1(B) r2(B) w2(B)
Sc' = r1(A) w1(A) r1(B) r2(A) w1(B) w2(A) r2(B) w2(B)
Sc' = r1(A) w1(A) r1(B) w1(B) r2(A) w2(A) r2(B) w2(B)
T1
T2
Schedule A

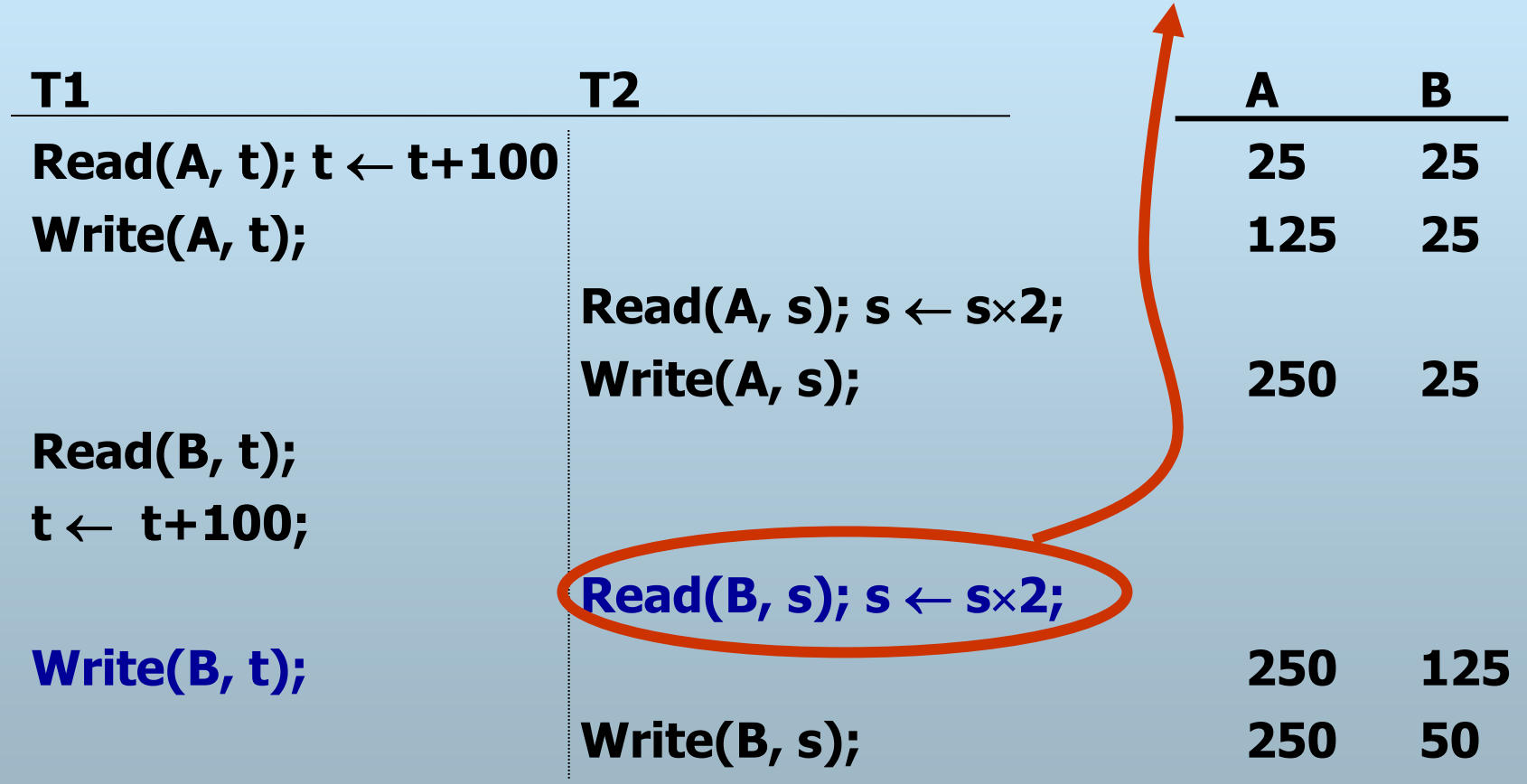
3、冲突可串性



3、冲突可串性

Schedule C

此步读入的B为25



T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
Read(B, t);			
t ← t+100;			
	Read(B, s); s ← s×2;		
Write(B, t);		250	125
	Write(B, s);	250	50

3、冲突可串性

- 冲突等价 (conflict equivalent)
  - S1, S2 are conflict equivalent schedules if S1 can be transformed into S2 by a series of swaps on non-conflicting actions.
- 冲突可串性 (conflict serializable)
  - A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

3、冲突可串性

- 定理
  - 如果一个调度满足冲突可串性，则该调度是可串化调度
- Note
  - 仅为充分条件

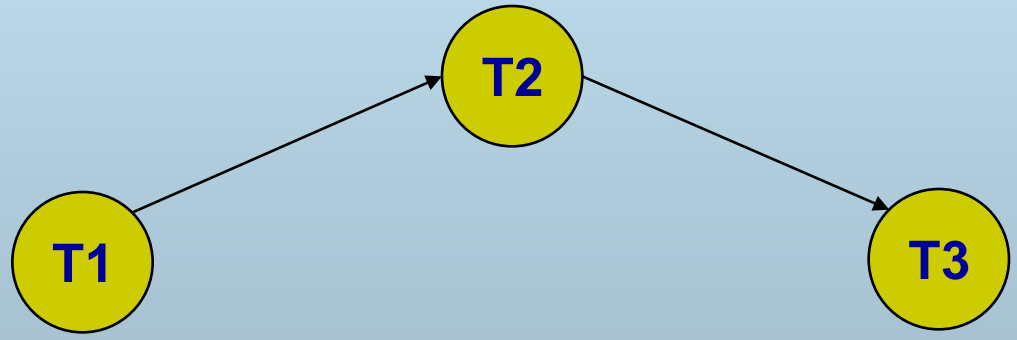
4、优先图 (Precedence Graph)

- 优先图用于冲突可串性的判断
- 优先图结构
  - 结点 (Node)：事务
  - 有向边 (Arc): Ti → Tj ，满足 Ti $<_s$ Tj
    - 存在Ti中的操作A1和Tj中的操作A2，满足
      - A1在A2前，并且
      - A1和A2是冲突操作

4、优先图

Example

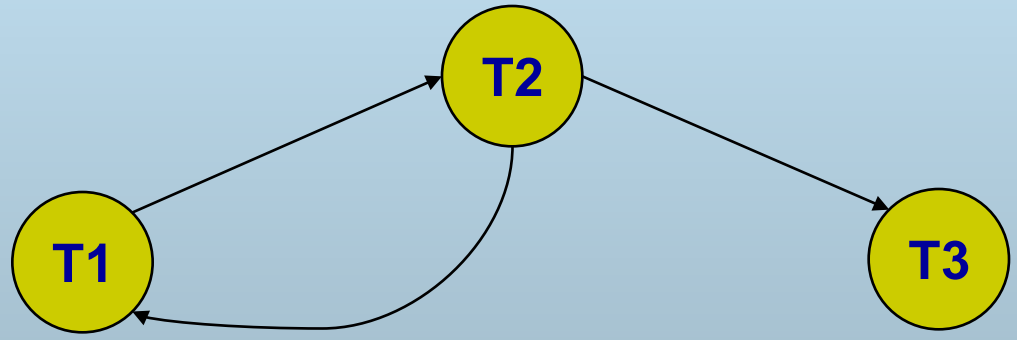
S = r2(A) r1(B) w2(A) r3(A) w1(B) w3(A) r2(B) w2(B)



4、优先图

Example

S = r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)



4、优先图

- 优先图与冲突可串性
  - 给定一个调度S，构造S的优先图P(S)，若P(S)中无环，则S满足冲突可串性
  - 证明：归纳法
    - see "H. Molina et al. *Database System Implementation*"

Next

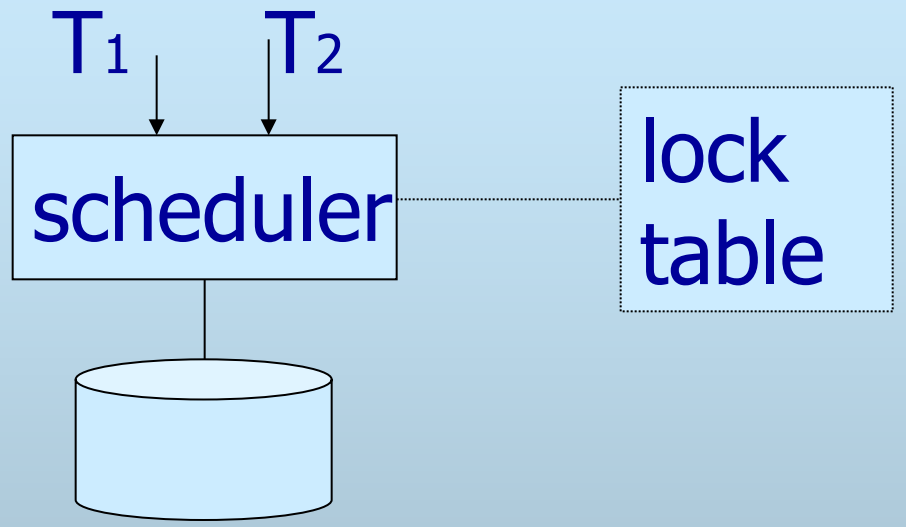
- 并发操作与并发问题
- 并发事务调度与可串性 (Scheduling and Serializability)
- 锁与可串性实现 (Locks) 
- 事务的隔离级别
- 死锁
- 乐观并发控制

三、锁与可串性实现

- What is a correct schedule?
  - a serializable schedule!
- How to get a serializable schedule?
  - Using locks

给定n个并发事务，确定一个可串化调度

1、锁简介



Two new actions:

lock (exclusive): l_i (A)

unlock: u_i (A)

1、锁简介

- 锁协议(protocol): 使用锁的规则

Rule #1: Well-formed transactions

Ti: … $l_i(A)$ … $p_i(A)$ … $u_i(A)$ …

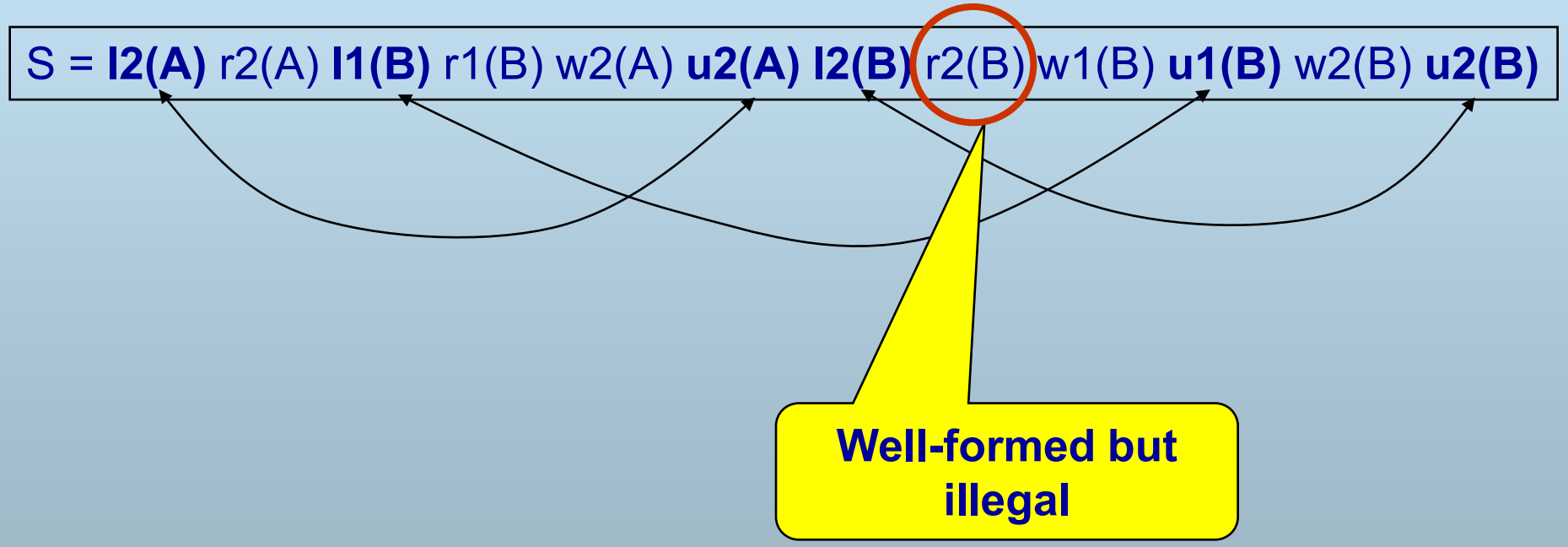
Rule #2 Legal scheduler

S = …… $l_i(A)$ …… $u_i(A)$ ……

no $l_j(A)$

1、锁简介

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



2、两阶段锁(2PL)

- **Two Phase Locking**

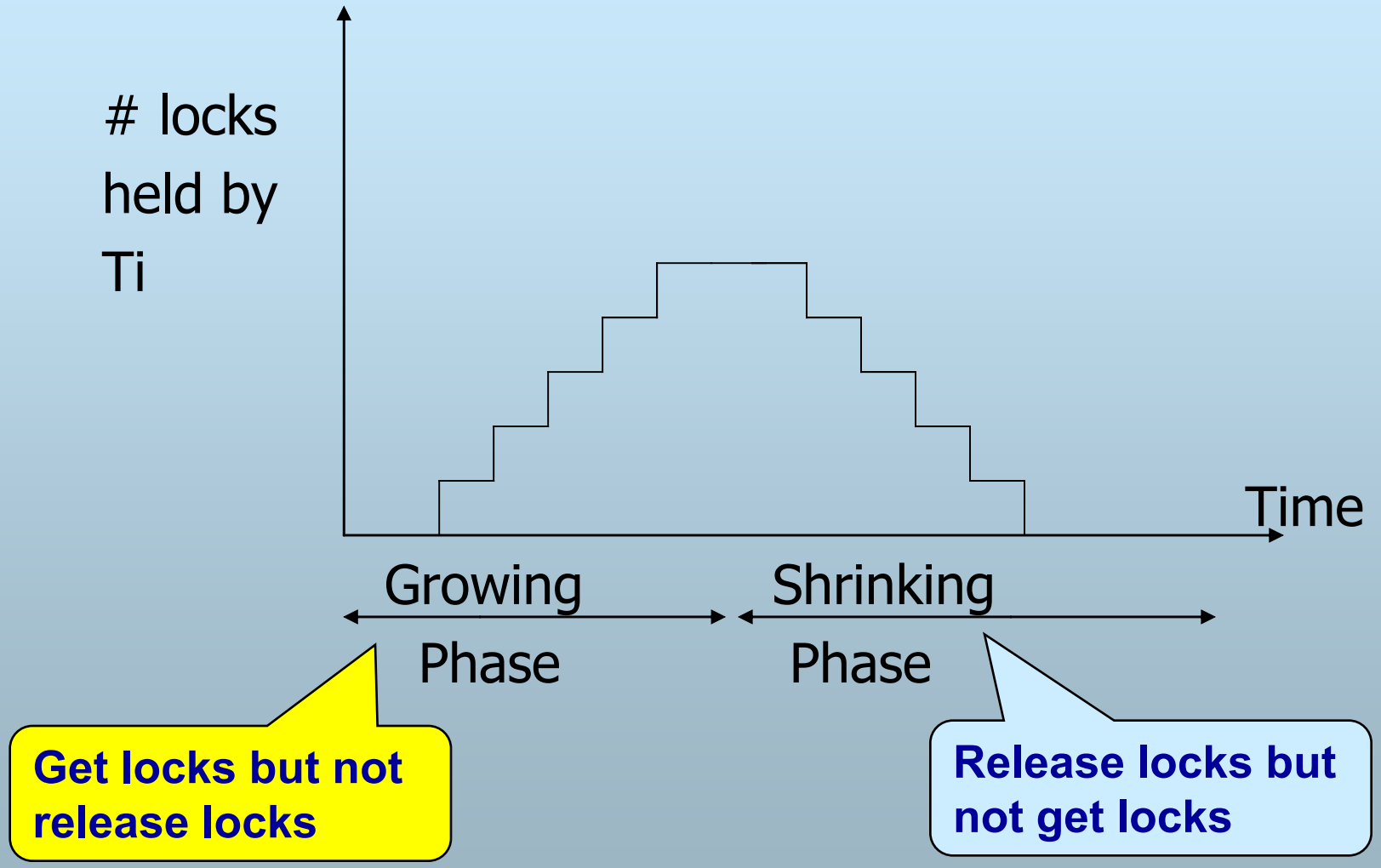
T_i = ... $l_i(A)$ ..... $u_i(A)$ .....

no unlocks　　no locks

1. 事务在对任何数据进行读写之前，首先要获得该数据上的锁
2. 在释放一个锁之后，事务不再获得任何锁

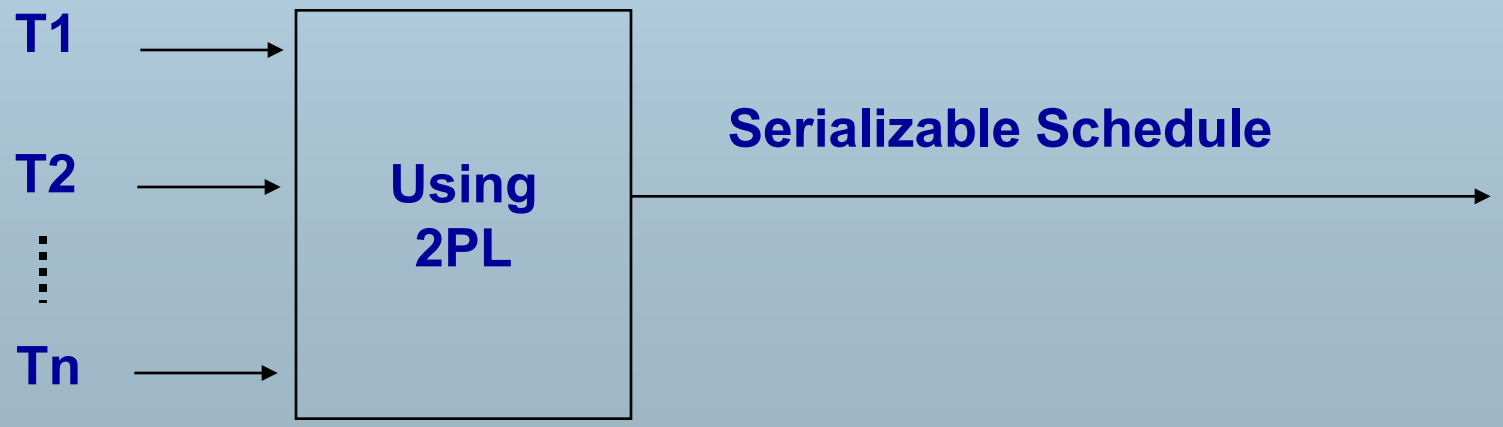
Kapali P. Eswaran, Jim Gray, Raymond A. Lorie, Irving L. Traiger:
The Notions of Consistency and Predicate Locks in a Database System. Commun. ACM 19(11): 624-633 (1976)

2、两阶段锁(2PL)


locks
held by
Ti
Time
Growing
Phase
Shrinking
Phase
Get locks but not release locks
Release locks but not get locks

2、两阶段锁(2PL)

- 两段式事务：遵守2PL协议的事务
- 定理
  - 如果一个调度S中的所有事务都是两段式事务，则该调度是可串化调度



2、两阶段锁(2PL)

- 如果事务T只是读取X，也必须加锁，而且释放锁之前其它事务无法对X操作，影响数据库的并发性
- 解决方法
  - 引入不同的锁，满足不同的要求
    - S Lock
    - X Lock
    - Update Lock
    - ……

3、X Lock

- **Exclusive Locks（X锁，也称写锁）**
- **X锁：若事务T对数据R加X锁，那么其它事务要等T释放X锁以后，才能获准对数据R进行封锁。只有获得R上的X锁的事务，才能对所封锁的数据进行 修改。**

3、X Lock

Example

Using X lock for schedules

T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
Read(B, t);			
t ← t+100;			
	Read(B, s); s ← s×2;		
Write(B, t);		250	125
	Write(B, s);	250	50

An incorrect schedule

3、X Lock

T1	T2	A	B
xL1(A)		25	25
Read(A, t); t ← t+100			
Write(A, t);		125	25
xL1(B)	xL2(A)		
Read(B, t); t ← t+100;	wait		
Write(B, t);	wait		
U1(A)	wait	125	125
U1(B)	Read(A, s); s ← s×2;		
	Write(A, t)	250	125
	xL2(B)		
	Read(B, s); s ← s×2;		
	Write(B, s);	250	250
	U2(A)		
	U2(B)		

X-lock-based 2PL

3、X Lock

- X锁提供了对事务的写操作的正确控制策略
- 但如果事务是只读事务，则没必要加X锁
  - 写——独占
  - 读——共享

4、S Lock

- Share Locks（S锁，也称读锁）
- S锁：如果事务T对数据R加了S锁，则其它事务对R的X锁请求不能成功，但对R的S锁请求可以成功。这就保证了其它事务可以读取R但不能修改R，直到事务T释放S锁。当事务获得S锁后，如果要对数据R进行修改，则必须在修改前执行Upgrade(R)操作，将S锁升级为X锁。

4、S Lock

S/X-lock-based 2PL

1. 事务在读取数据R前必须先获得S锁
2. 事务在更新数据R前必须要获得X锁。如果该事务已具有R上的S锁，则必须将S锁升级为X锁
3. 如果事务对锁的请求因为与其它事务已具有的锁不相容而被拒绝，则事务进入等待状态，直到其它事务释放锁。
4. 一旦释放一个锁，就不再请求任何锁

5、Compatibility of locks

Requests

Holds

T1 \ T2	X锁	S锁	无
X锁	N	N	Y
S锁	N	Y	Y
无	Y	Y	Y

- N：No，不相容的请求
- Y：Yes，相容的请求
- 如果两个锁不相容，则后提出锁请求的事务必须等待

6、Update Lock

Example

t	T1	T2
1	sL1(A)	
2		sL2(A)
3	Read(A)	Read(A)
4		A=A+100
5		Upgrade(A)
6	A=A+100	Wait
7	Upgrade(A)	Wait
8	Wait	Wait
9	Wait	Wait
10	……	……

6、Update Lock

- **Update Lock**
  - 如果事务取得了数据R上的更新锁，则可以读R，并且可以在以后升级为X锁
  - 单纯的S锁不能升级为X锁
  - 如果事务持有了R上的Update Lock，则其它事务不能得到R上的S锁、X锁以及Update锁
  - 如果事务持有了R上的S Lock，则其它事务可以获取R上的Update Lock

6、Update Lock

- 相容性矩阵

	S	X	U
S	Y	N	Y
X	N	N	N
U	N	N	N

NOTE

<S, U>是相容的：如果其它事务已经持有了S锁，则当前事务可以请求U锁，以获得较好的并发性

<U, S>不相容：如果某个事务已持有U锁，则其它事务不能再获得S锁，因为持有U锁的事务可能会由于新的S锁而导致永远没有机会升级到X锁

6、Update Lock

Example

t	T1	T2
1	uL1(A)	
2		uL2(A)
3	Read(A)	Wait
4		Wait
5		wait
6	A=A+100	Wait
7	Upgrade(A)	Wait
8	Write(A)	Wait
9	U1(A)	Wait
10		Read(A)
11		……

Where are we ?

- 并发操作与并发问题
- 并发调度与可串性
- 锁与可串性实现
  - 2PL
    - S Lock
    - X Lock
    - U Lock
  - Multi-granularity Lock 多粒度锁
  - Intension Lock 意向锁