第10章 并发控制

金培权（jpq@ustc.edu.cn）

主要内容

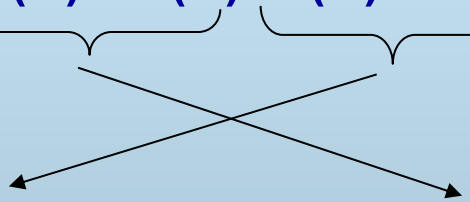
- 并发操作与并发问题
- 并发事务调度与可串性 (Scheduling and Serializability)
- 锁与可串性实现 (Locks)
- 事务的隔离级别
- 死锁

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**

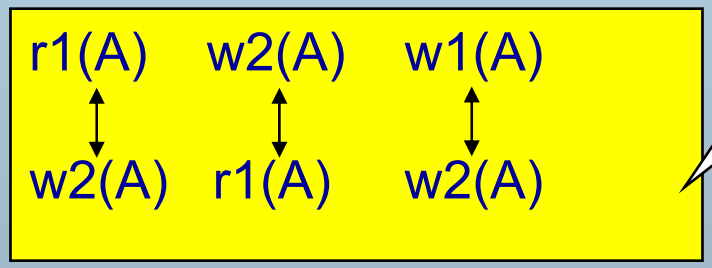
2、可串化调度

- **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**

Too difficult to realize ⇒ **Other approaches?** ⇓ **冲突可串性**

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**



涉及同一个数据库元素，并且至少有一个是写操作

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

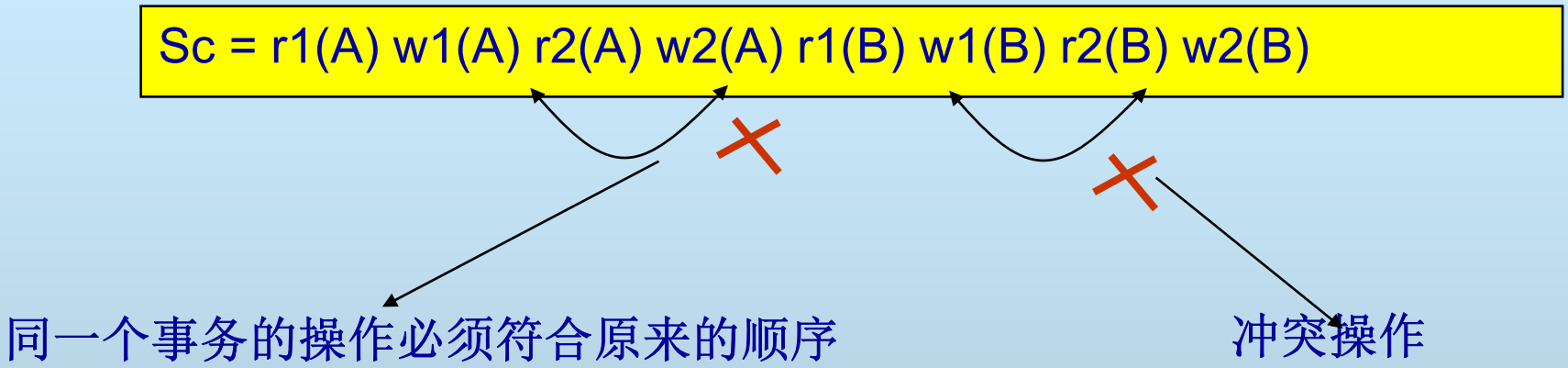
T1	T2	A	B
Read(A, t); t ← t+100		25	25
Write(A, t);		125	25
	Read(A, s); s ← s×2;		
	Write(A, s);	250	25
Read(B, t);			
t ← t+100;			
Write(B, t);		250	125
	Read(B, s); s ← s×2;		
	Write(B, s);	250	250

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

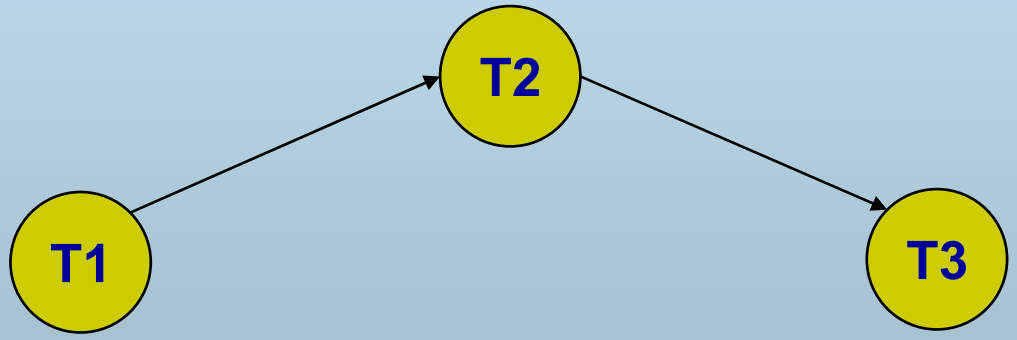
4、优先图 (Precedence Graph)

- 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

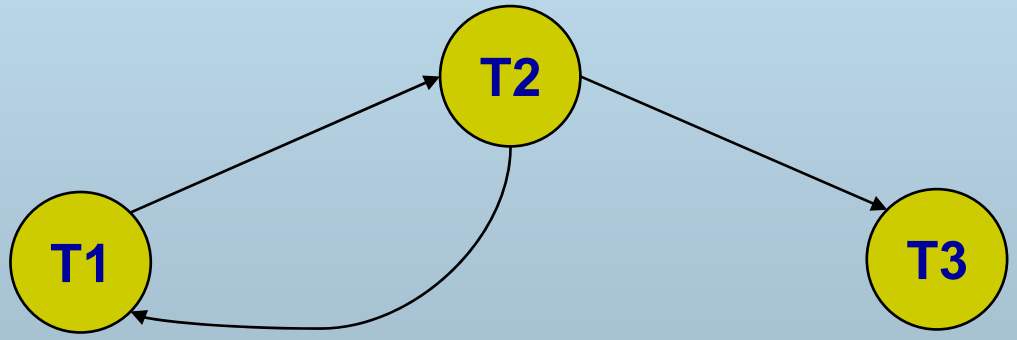
S = r2(A) r1(B) w2(A) r3(A) w1(B) w3(A) r2(B) w2(B)



4、优先图

Example

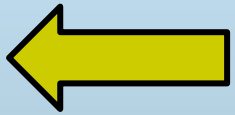
S = r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)



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*"

主要内容

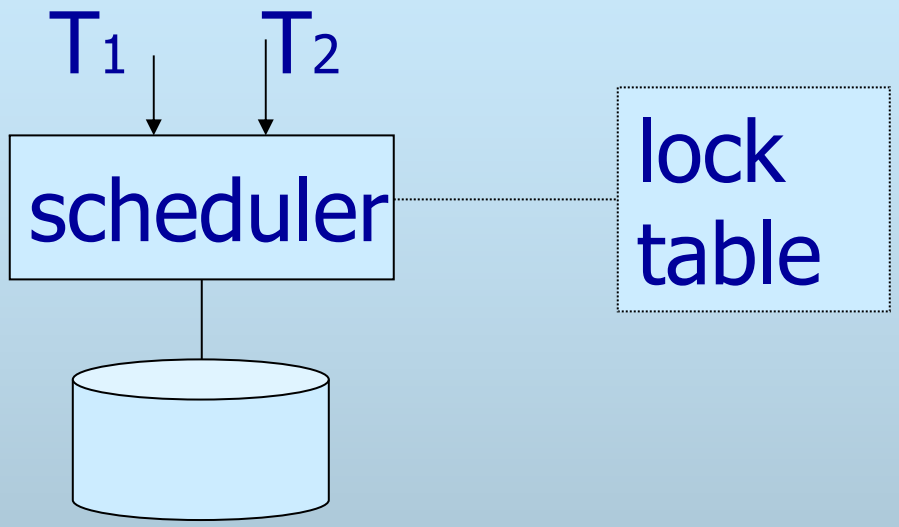
- 并发操作与并发问题
- 并发事务调度与可串性 (Scheduling and Serializability)
- 锁与可串性实现 (Locks) 
- 事务的隔离级别
- 死锁

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: … $l_i(A)$ … $p_i(A)$ … $u_i(A)$ …

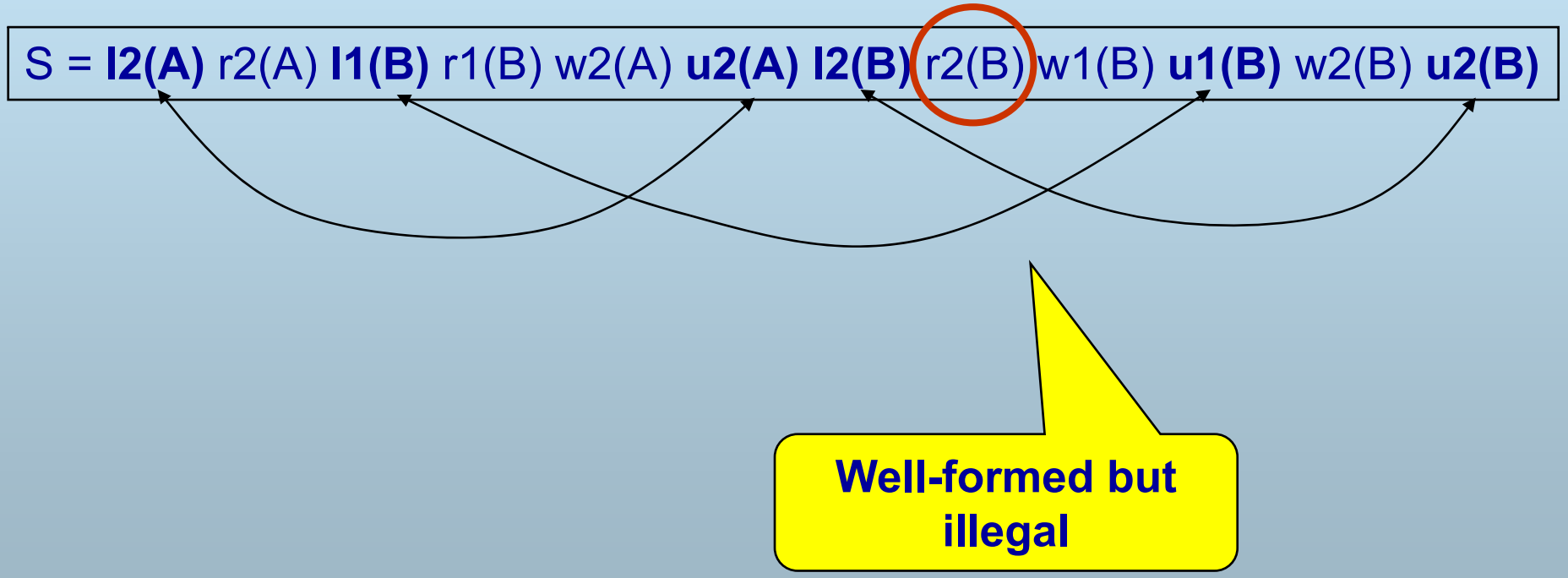
Rule #2 Legal scheduler

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)

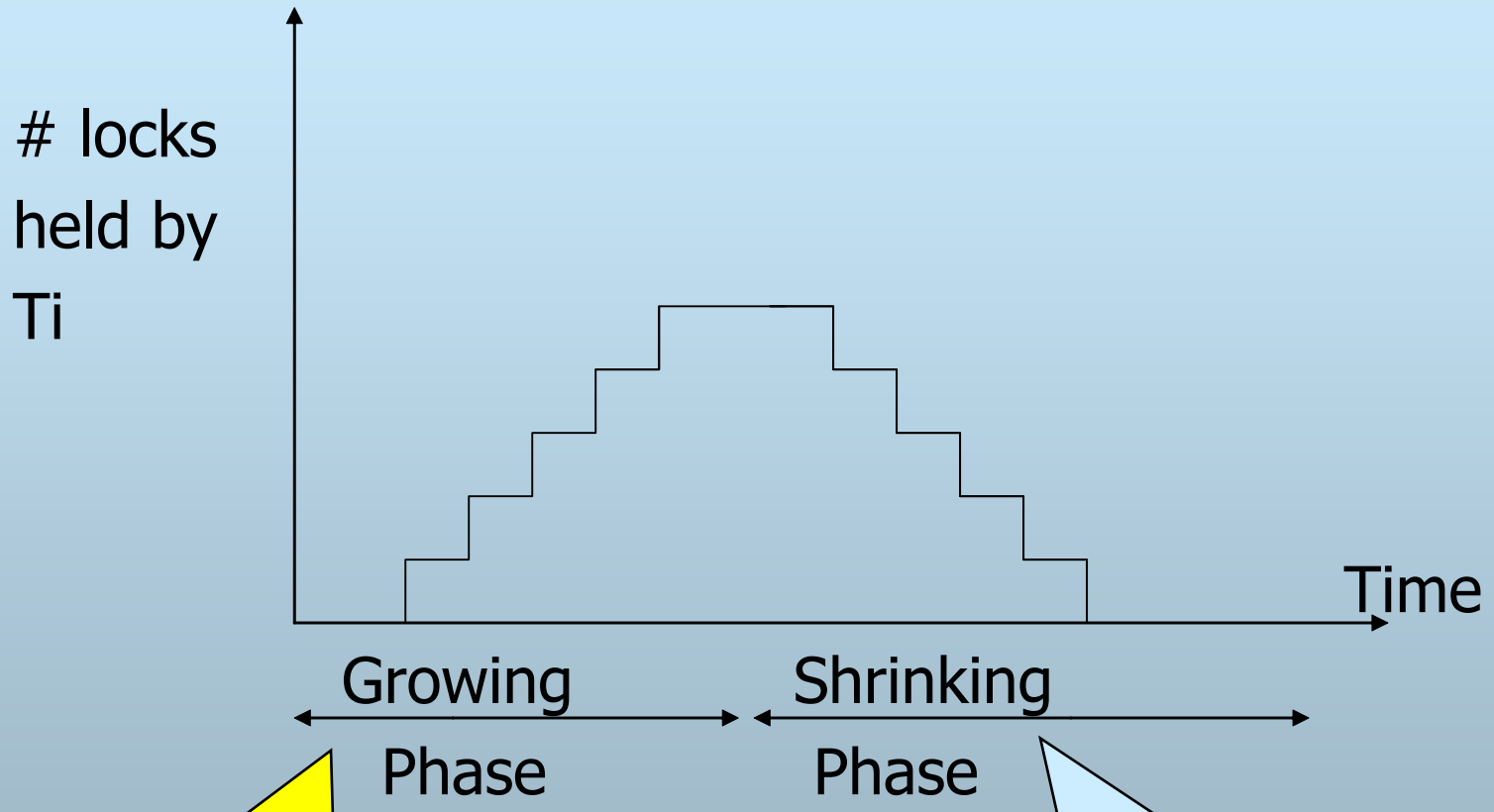
- **Two Phase Locking**

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

no unlocks　　no locks

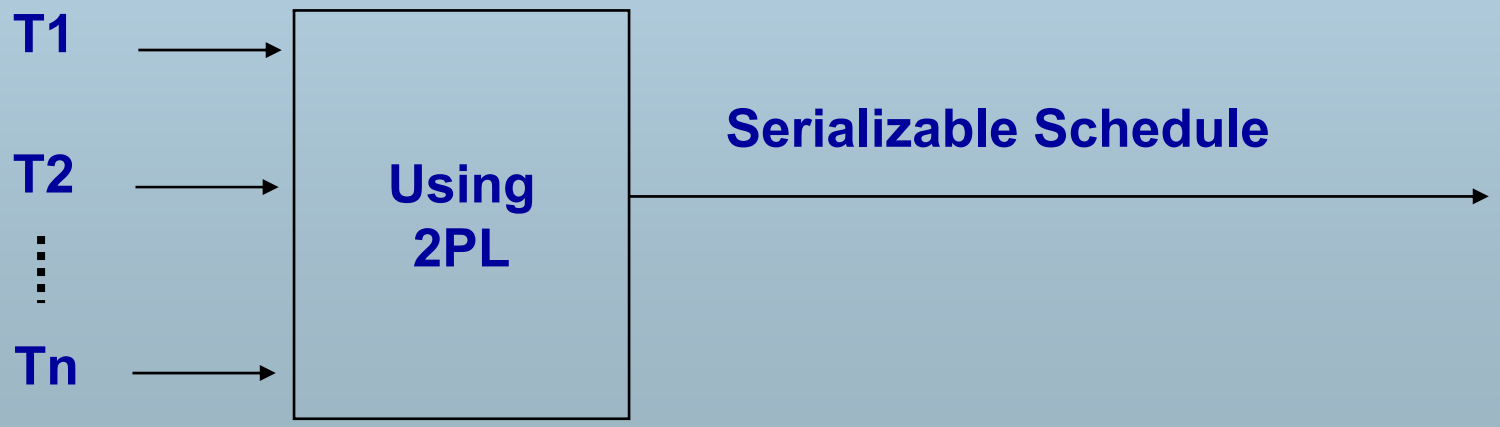
1. 事务在对任何数据进行读写之前，首先要获得该数据上的锁
2. 在释放一个锁之后，事务不再获得任何锁

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（冲突）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		……

7、再论2PL

- 2PL问题

t	T1	T2
1	xL1(A)	
2	A=A-100	
3	xL1(B)	
4	Write(A)	
5	xL1(B)	
6	Unlock(A)	
7		sL2(A)
8	B=B+100	Read(A)
9	Write(B)	
10	Unlock(B)	
11	Rollback	

脏读

t	T1	T2
1	sL1(A)	
2	sL1(B)	
3	Read(A)	
4	Unlock(A)	
5		xL2(A)
6		Read(A)
7	Read(B)	A=A+100
8		Write(A)
9		Commit
10	Sum=A+B	Unlock(A)
11	Read(A)	

不一致分析

7、再论2PL

- **Strict 2PL（严格2PL）：** 要求X锁必须保持到事务结束(commit/rollback)
- **Rigorous 2PL（强2PL）：** 要求所有锁都保持到事务结束

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

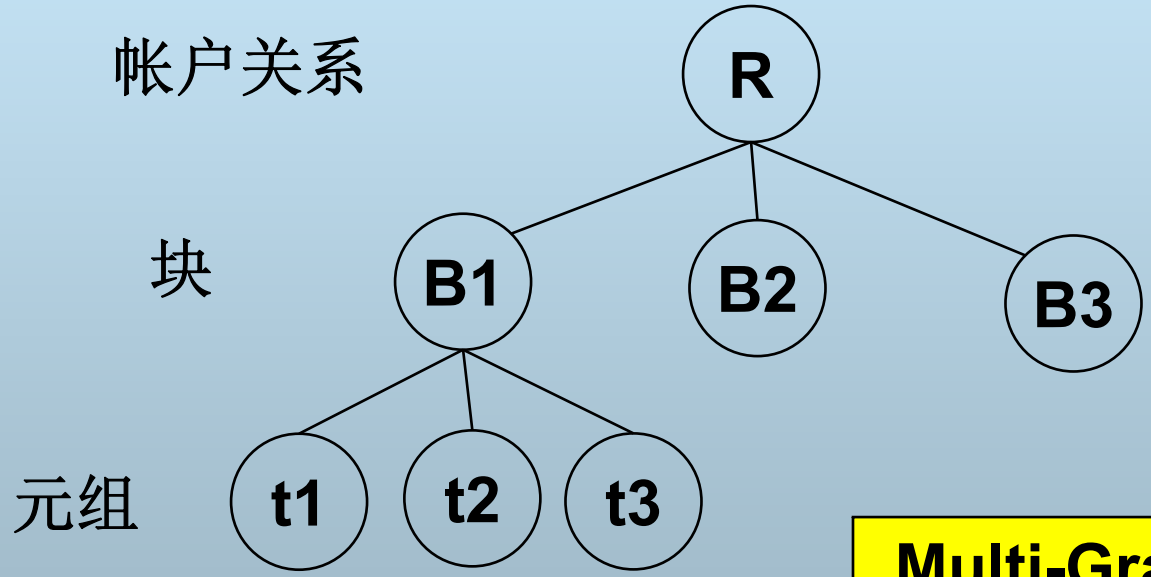
Next

8、Multi-Granularity Lock

- **Lock Granularity**
  - 指加锁的数据对象的大小
    - 可以是整个关系、块、元组、整个索引、索引项
- 锁粒度越细，并发度越高；锁粒度越粗，并发度越低

8、Multi-Granularity Lock

- 多粒度锁：同时支持多种不同的锁粒度



Multi-Granularity Tree

8、Multi-Granularity Lock

- 多粒度锁协议
  - 允许多粒度树中的每个结点被独立地加S锁或X锁，对某个结点加锁意味着其下层结点也被加了同类型的锁

8、Multi-Granularity Lock

- **Why we need MGL?**

8、Multi-Granularity Lock

T1: 求当前数据库中所有帐户的余额之和
T2: 增加一个新帐户(余额为1000)
Use tuple locks，suppose total two tuples in R

T1	T2
sL1(o1)	
sL1(o2)	
Read(o1,t1)	
Read(o2,t2)	
	Write(o3)
Sum=t1+t2	
U1(o1)	
U1(o2)	

并不是当前数据库的实际状态

8、Multi-Granularity Lock

- 原因
  - Lock只能针对已存在的元组，对于开始时不存在后来被插入的元组无法Lock
  - o3：Phantom tuple 幻像元组
    - 存在，却看不到物理实体

Solution

- T2插入o3的操作看成是整个关系的写操作，对整个关系加锁
  - Need MGL！

8、Multi-Granularity Lock

Solution: Using MGL

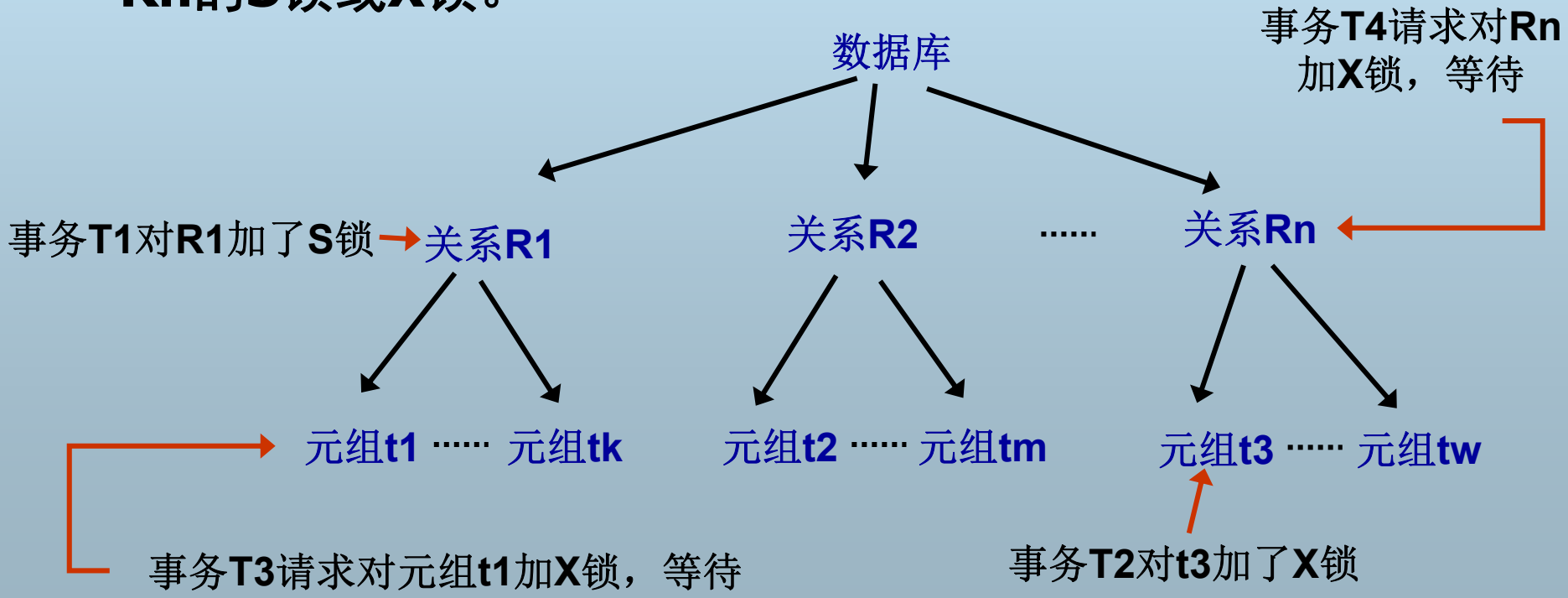
T1	T2
sL1(o1)	
sL1(o2)	
Read(o1,t1)	
Read(o2,t2)	
	xL2(R)
Sum=t1+t2	wait
U1(o1)	wait
U1(o2)	wait
	write(o3)

8、Multi-Granularity Lock

- 多粒度锁上的两种不同加锁方式
  - 显式加锁：应事务的请求直接加到数据对象上的锁
  - 隐式加锁：本身没有被显式加锁，但因为其上层结点加了锁而使数据对象被加锁
  - 给一个结点显式加锁时必须考虑
    - 该结点是否已有不相容锁存在
    - 上层结点是否已有不相容的的锁（上层结点导致的隐式锁冲突）
    - 所有下层结点中是否存在不相容的显式锁

8、Multi-Granularity Lock

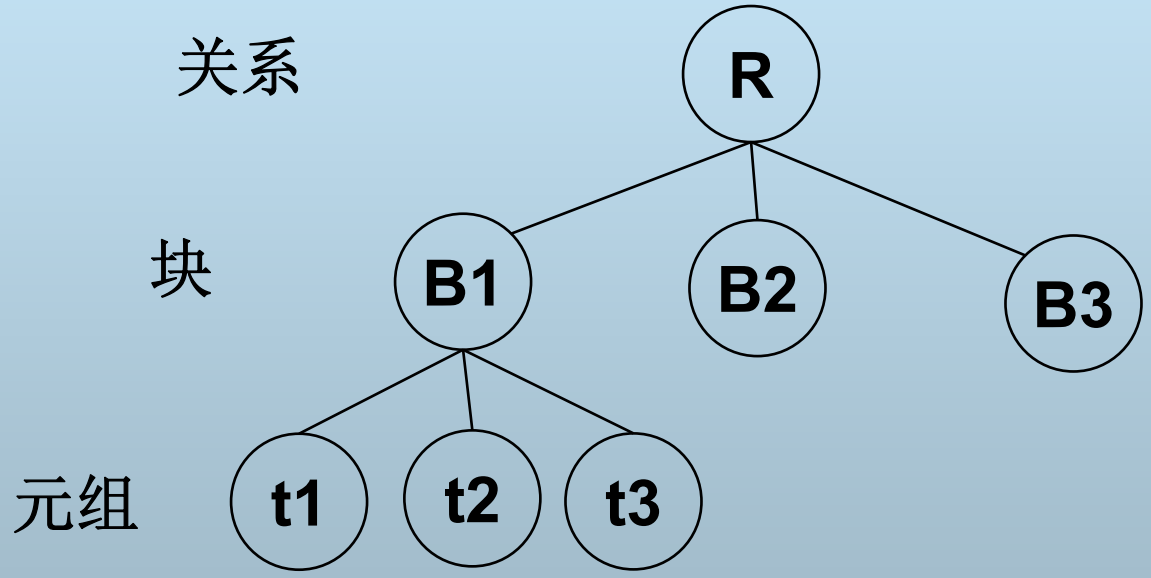
- 事务T1对关系R1显式加了S锁，意味着R1的所有元组也被隐式加了S锁。其它事务可以在R1的元组上加S锁，但不能加X锁
- 事务T2对元组t3加了X锁，其它事务不能请求对其上层结点Rn的S锁或X锁。



8、Multi-Granularity Lock

- 在对一个结点P请求锁时，必须判断该结点上是否存在不相容的锁
  - 有可能是P上的显式锁
  - 也有可能是P的上层结点导致的隐式锁
  - 还有可能是P的下层结点中已存在的某个显式锁
- 理论上要搜索上面全部的可能情况，才能确定P上的锁请求能否成功
  - 显然是低效的
  - 引入意向锁 (Intension Lock) 解决这一问题

9、Intension Lock



9、Intension Lock

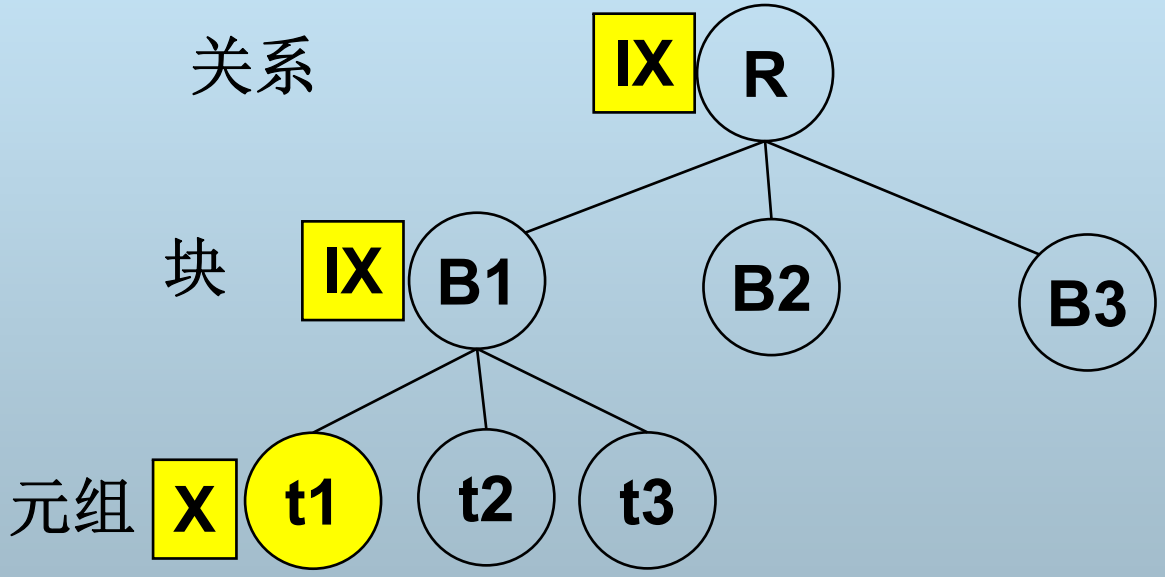
- IS锁（Intent Share Lock，意向共享锁，意向读锁）
- IX锁（Intent Exlusive Lock，意向排它锁，意向写锁）

9、Intension Lock

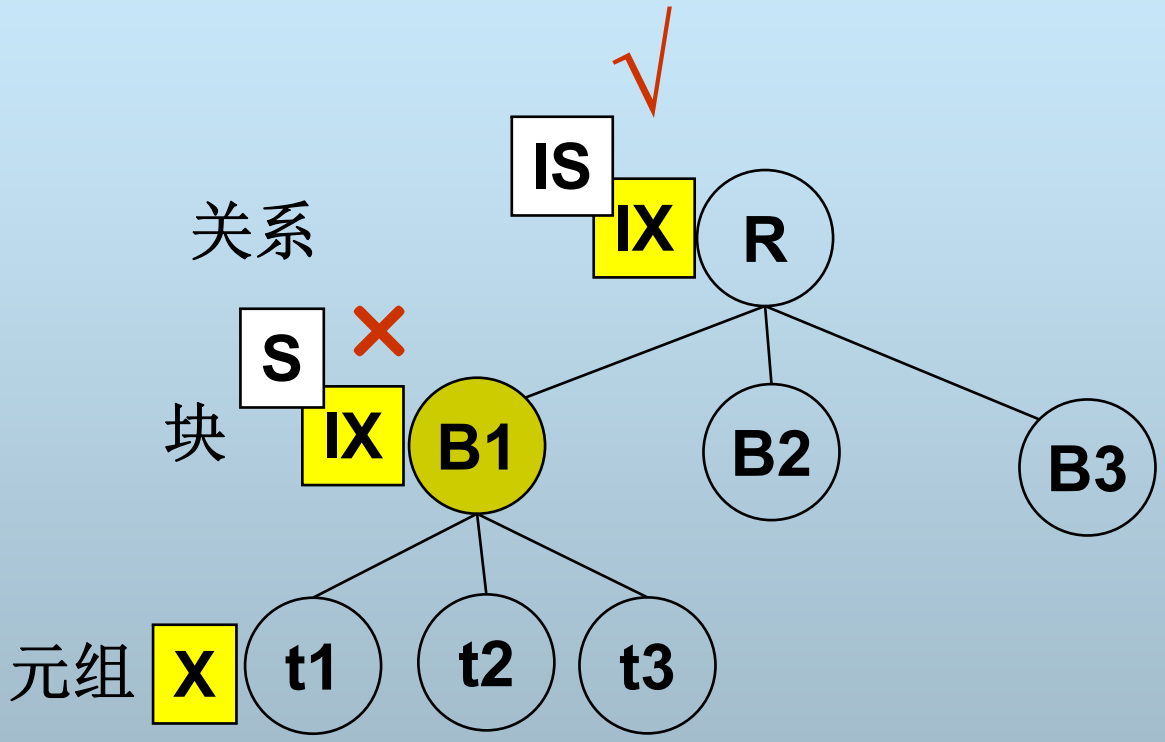
- 如果对某个结点加IS(IX)锁，则说明事务要对该结点的某个下层结点加S (X)锁；
- 对任一结点P加S(X)锁，必须先对从根结点到P的路径上的所有结点加IS(IX)锁

9、Intension Lock

Want to exclusively lock t1



9、Intension Lock



9、Intension Lock

- **Compatibility Matrix**

	IS	IX	S	X
IS	✓	✓	✓	×
IX	✓	✓	×	×
S	✓	×	✓	×
X	×	×	×	×