数据库原理

The Theory of Database System

第六章 数据库保护

中国矿业大学计算机学院

第六章 数据库保护

6.1 事务

6.2 数据库恢复

6.3 并发控制

6.4 数据库安全性

6.5 数据库完整性

6.3 并发控制

6.3.1 并发控制概述

6.3.2 封锁机制

6.3.3 并发调度的可串行性

6.3.4 封锁的粒度

6.3.1 并发控制概述

多事务执行方式

(1) 串行方式

- 每个时刻只有一个事务运行，其他事务必须等到这个事务结束以后方能运行；
- 不能充分利用系统资源，发挥数据库共享资源的特点；

(2) 并发方式

- 事务的并行执行是这些并行事务的并行操作轮流交叉运行；
- 是单处理机系统中的并发方式，能够减少处理机的空闲时间，提高系统的效率；

事务并发执行带来的问题

- 可能会存取和存储不正确的数据，破坏事务的隔离性和数据库的一致性
- DBMS必须提供并发控制机制
- 并发控制机制是衡量一个DBMS性能的重要标志之一

并发操作带来的数据不一致性

- 丢失修改（**lost update**）
- 不可重复读（**non-repeatable read**）
- 读“脏”数据（**dirty read**）

1. 丢失修改

- 丢失修改是指事务1与事务2从数据库中读入同一数据并修改
- 事务2的提交结果破坏了事务1提交的结果，导致事务1的修改被丢失。

T_1	T_2
① 读A=16	
②	读A=16
③ A←A-1	
写回A=15	
④	A←A-1
	写回A=15

2. 不可重复读

不可重复读是指事务1读取数据后，事务2执行更新操作，使事务1无法再现前一次读取结果。

T_1	T_2
① 读A=50 读B=100 求和=150	
②	读B=100 B←B*2 写回B=200
③ 读A=50 读B=200 求和=250 (验算不对)	

三类不可重复读

事务1读取某一数据后：

1. 事务2对其做了修改。 — 读不一致
2. 事务2删除了其中部分记录。 ┐
3. 事务2插入了一些记录。 ┘ 幻影读现象

3. 读“脏”数据

- 事务1修改某一数据，并将其写回磁盘
- 事务2读取同一数据后
- 事务1由于某种原因被撤消，这时事务1已修改过的数据恢复原值
- 事务2读到的数据就与数据库中的数据不一致，是不正确的数据，又称为“脏”数据。

读“脏”数据

T_1	T_2
① 读C=100 C←C*2 写回C	
②	读C=200
③ ROLLBACK C恢复为100	

并发控制机制的任务

- 对并发操作进行正确调度
- 保证事务的隔离性
- 保证数据库的一致性

6.3.2 封锁机制

一、什么是封锁

二、基本封锁类型

三、封锁协议

四、活锁和死锁

一、什么是封锁

- 封锁就是事务T在对某个数据对象（例如表、记录等）操作之前，先向系统发出请求，对其加锁。
- 加锁后事务T就对该数据对象有了一定的控制，在事务T释放它的锁之前，其它的事务不能更新此数据对象。
- 封锁是实现并发控制的一个非常重要的技术。

二、基本封锁类型

- DBMS通常提供了多种类型的封锁。一个事务对某个数据对象加锁后究竟拥有什么样的控制是由封锁的类型决定的。
- 基本封锁类型
  - 排它锁（eXclusive lock，简记为X锁）
  - 共享锁（Share lock，简记为S锁）

排它锁

- 排它锁又称为写锁
- 若事务T对数据对象A加上X锁，则只允许T读取和修改A，其它任何事务都不能再对A加任何类型的锁，直到T释放A上的锁

共享锁

- 共享锁又称为读锁
- 若事务T对数据对象A加上S锁，则其它事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁

三、封锁协议

- 在运用X锁和S锁对数据对象加锁时，需要约定一些规则：封锁协议（Locking Protocol）
  - 何时申请X锁或S锁
  - 持锁时间、何时释放
- 不同的封锁协议，在不同的程度上为并发操作的正确调度提供一定的保证
- 常用的封锁协议：三级封锁协议

1级封锁协议

- 事务T在修改数据R之前必须先对其加X锁，直到事务结束才释放
  - 正常结束（COMMIT）
  - 非正常结束（ROLLBACK）
- 1级封锁协议可防止丢失修改
- 在1级封锁协议中，如果是读数据，不需要加锁的，所以它不能保证可重复读和不读“脏”数据。

1级封锁协议

T_1	T_2
① Xlock A	
② 获得A=16	
	Xlock A
③ A←A-1	等待
写回A=15	等待
Commit	等待
Unlock A	等待
④	获得A=15
	A←A-1
	写回A=14
⑤	Commit
	Unlock A

没有丢失修改

1级封锁协议

T_1	T_2
① Xlock A	
② 获得A=16 A←A-1 写回A=15	
③	读A=15
④ Rollback Unlock A	

读“脏”数据

1级封锁协议

T_1	T_2
① 读A=50	
读B=100	
求和=150	
②	Xlock B
	获得B=100
	B←B*2
	写回B=200
	Commit
	Unlock B
③ 读A=50	
读B=200	
求和=250	
(验算不对)	

不可重复读

2级封锁协议

- 1级封锁协议+事务T在读取数据R前必须先加S锁，读完后即可释放S锁
- 2级封锁协议可以防止丢失修改和读“脏”数据。
- 在2级封锁协议中，由于读完数据后即可释放S锁，所以它不能保证可重复读。

2级封锁协议

T_1	T_2
① Slock A 获得 读A=50 Unlock A ② Slock B 获得 读B=100 Unlock B ③ 求和=150	Slock B 读B=100 Xlock B 获得Xlock B B←B*2

T_1 (续)	T_2 (续)
④ Slock A 获得 读A=50 Unlock A Slock B 获得 读B=200 Unlock B 求和=250 (验算不对)	写回B=200 Commit Unlock B

不可重复读

3级封锁协议

- 1级封锁协议 + 事务T在读取数据R之前必须先对其加S锁，直到事务结束才释放

- 3级封锁协议可防止丢失修改、读脏数据和不可重复读。

3级封锁协议

可重复读

T_1	T_2
① Slock A	
读A=50	
Slock B	
读B=100	
求和=150	
②	Slock B
	读B=100
	Xlock B
③ 读A=50	等待
读B=100	等待
求和=150	等待
Commit	等待
Unlock A	等待
Unlock B	获得Xlock B
④	B←B*2
	写回B=200
	Commit
	Unlock B

3级封锁协议

不读“脏”数据

T_1	T_2
① Slock C	
读C= 100	
Xlock C	
C←C*2	
写回C=200	Slock C
②	等待
	等待
	等待
③ ROLLBACK	等待
(C恢复为100)	获得Slock C
Unlock C	读C=100
④	Commit C
	Unlock C

封锁协议小结

- 三级协议的主要区别
  - 什么操作需要申请封锁
  - 何时释放锁（即持锁时间）

封锁协议小结(续)

	X锁		S锁		一致性保证		
	操作结束释放	事务结束释放	操作结束释放	事务结束释放	不丢失修改	不读"脏"数据	可重复读
1级封锁协议		√			√		
2级封锁协议		√	√		√	√	
3级封锁协议		√		√	√	√	√

事务的隔离性级别

- 事务在并发执行时应该相互独立互不干扰，即事务的隔离性。
- 事务的隔离性级别，又称事务的一致性级别，是一个事务必须与其它事务实现隔离的程度，是事务可接受的数据不一致程度。

四种隔离级别

①读未提交：最低的隔离级别，在这种事务隔离级别下，一个事务可以读到另外一个事务未提交的数据，不允许丢失修改，接受读脏数据和不可重复读现象。

②读已提交：若事务还没提交，其他事务不能读取该事务正在修改的数据。不允许丢失修改和读脏数据，接受不可重复读现象。

四种隔离级别(续)

③可重复读：事务多次读取同一数据对象的结果一致。不允许丢失修改、读脏数据和读不一致，接受幻影读现象。

④可串行化：最高级别的隔离性，保证可串行化，不允许丢失修改、读脏数据、读不一致以及幻影读现象的发生。

查看当前事务的隔离级别：

```
select @@transaction_isolation;
```

设置事务的隔离级别：

```
set [global | session] transaction
        isolation level 隔离级别名称;
```

四种隔离级别

隔离级别	脏读	不可重复读	幻读
读未提交（Read uncommitted）	V	V	V
读已提交（Read committed）	x	V	V
可重复读（Repeatable read）	x	x	V
可串行化（Serializable）	x	x	x

四、活锁和死锁

- 封锁技术可以有效地解决并行操作的一致性问题，但也带来一些新的问题
  - 活锁
  - 死锁

活锁

- 事务T1，T2申请数据对象A，T1先给A加锁，T1释放A上的锁后，事务T3又给A加锁，T2等待，这样，A始终被其他事务封锁，事务T2可能长时间得不到A，这种情况称为活锁。

T_1	T_2	T_3	T_4
lock R	.	.	.
.	lock R	.	.
.	等待	Lock R	.
Unlock	等待	.	Lock R
.	等待	Lock R	等待
.	等待	.	等待
.	等待	Unlock	等待
.	等待	.	Lock R
.	等待	.	.

如何避免活锁

采用先来先服务的策略：

当多个事务请求封锁同一数据对象时

- 按请求封锁的先后次序对这些事务排队
- 该数据对象上的锁一旦释放，首先批准申请队列中第一个事务获得锁。

死锁

事务T1已经封锁A，而又想申请封锁B，而此时事务T2已经封锁B，而又想申请封锁A，这样，T1等待T2释放B，而T2等待T1释放A，使得T1、T2均无法继续执行下去，这种情况称为死锁。

T_1	T_2
Slock R_1	.
.	.
.	Slock R_2
.	.
Xlock R_2	.
等待	Xlock R_1
等待	等待
等待	等待
.	.

解决死锁的方法

两类方法

1. 预防死锁

2. 死锁的诊断与解除

*1. 死锁的预防

- 产生死锁的原因是两个或多个事务都已封锁了一些数据对象，然后又都请求对已为其他事务封锁的数据对象加锁，从而出现死等待。

- 预防死锁的发生就是要破坏产生死锁的条件

*死锁的预防（续）

预防死锁的方法

- 一次封锁法
- 顺序封锁法

*（1）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
- 一次封锁法存在的问题：降低并发度
  - 扩大封锁范围
  - 将以后要用到的全部数据加锁，势必扩大了封锁的范围，从而降低了系统的并发度

*（2）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
  - 数据库系统中可封锁的数据对象极其众多，并且随数据的插入、删除等操作而不断地变化，要维护这样极多而且变化的资源的封锁顺序非常困难，成本很高

死锁的预防（续）

- 结论
  - 在操作系统中广为采用的预防死锁的策略并不很适合数据库的特点
  - DBMS在解决死锁的问题上更普遍采用的是诊断并解除死锁的方法

2. 死锁的诊断与解除

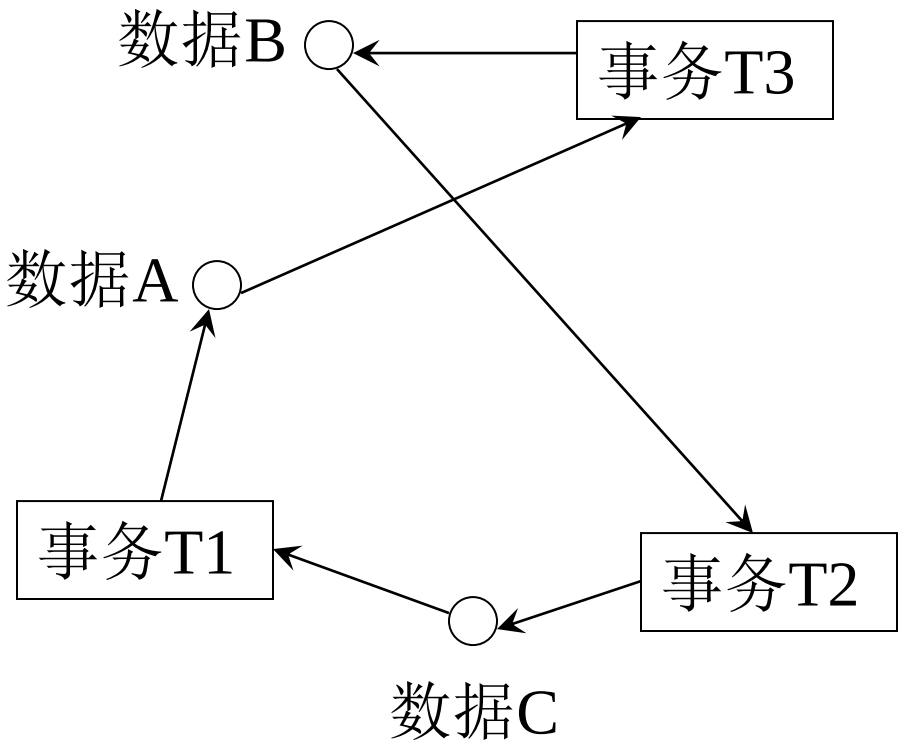
- 允许死锁发生
- 解除死锁
  - 由DBMS的并发控制子系统定期检测系统中是否存在死锁
  - 一旦检测到死锁，就要设法解除

检测死锁：超时法

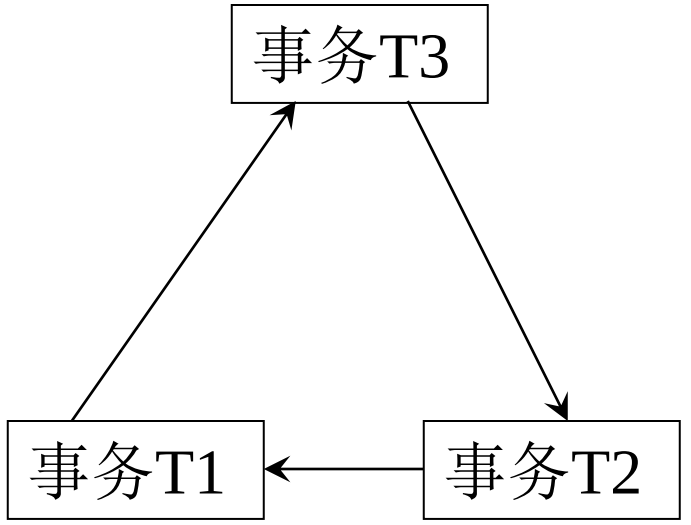
- 如果系统等待时间超过了规定的时限，就认为发生了死锁
- 优点：实现简单
- 缺点
  - 有可能误判死锁
  - 时限若设置得太长，死锁发生后不能及时发现

*等待图法

- 用事务等待图动态反映所有事务的等待情况
  - 事务等待图是一个有向图$G=(T, U)$
  - T为结点的集合，每个结点表示正运行的事务
  - U为边的集合，每条边表示事务等待的情况
  - 若T_1等待T_2，则T_1，T_2之间划一条有向边，从T_1指向T_2
- 并发控制子系统周期性地（比如每隔1 min）检测事务等待图，如果发现图中存在回路，则表示系统中出现了死锁。



事务已封锁及申请锁情况



等待图

死锁的诊断与解除（续）

★解除死锁

– 选择一个处理死锁代价最小的事务，撤消（undo）其已经做过的每笔操作，释放此事务持有的所有的锁，使其它事务能继续运行下去。

6.3.3 并发调度的可串行性

一、什么样的并发操作调度是正确的

二、如何保证并发操作的调度是正确的

一、什么样的并发操作调度是正确的

调度的种类：

（1）**串行调度：**如果调度S中的任意两个事务Ti和Tj，如果Ti的所有操作都先于Tj的所有操作，或者相反，则称S为串行调度。

（2）**并发调度：**如果在一个调度中，各个事务交叉地执行，这个调度称为并发调度。

在串行调度中每一个事务都是在下一个事务开始执行之前提交。因此，串行调度没有并发性，故每一个串行调度都是一个正确的执行。

一、什么样的并发操作调度是正确的

- 计算机系统对并行事务中并行操作的调度是的随机的，而不同的调度可能会产生不同的结果。
- 当且仅当其结果与按某一次序串行地执行它们时的结果相同。这种并发调度策略称为**可串行化**（Serializable）的调度。

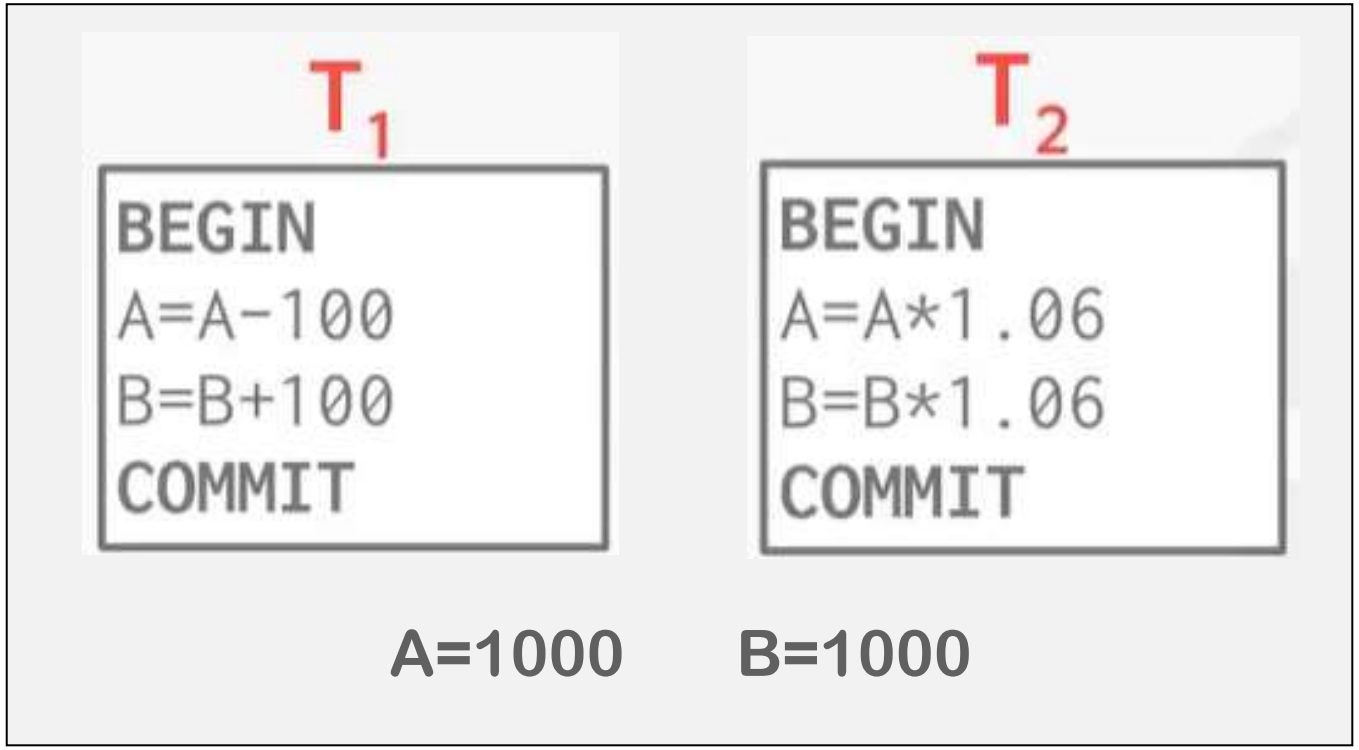
可串行化的调度

定义：多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行它们时的结果相同，称这种调度策略为**可串行化的调度**。

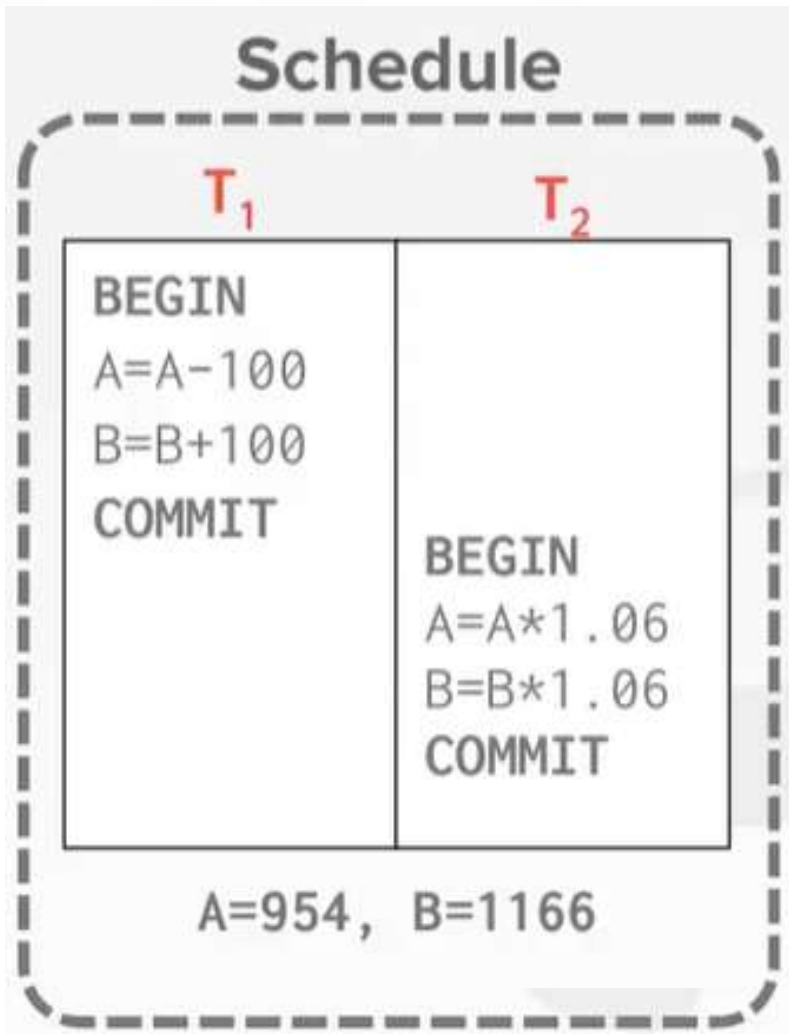
可串行化是作为并发调度正确与否的判定准则!

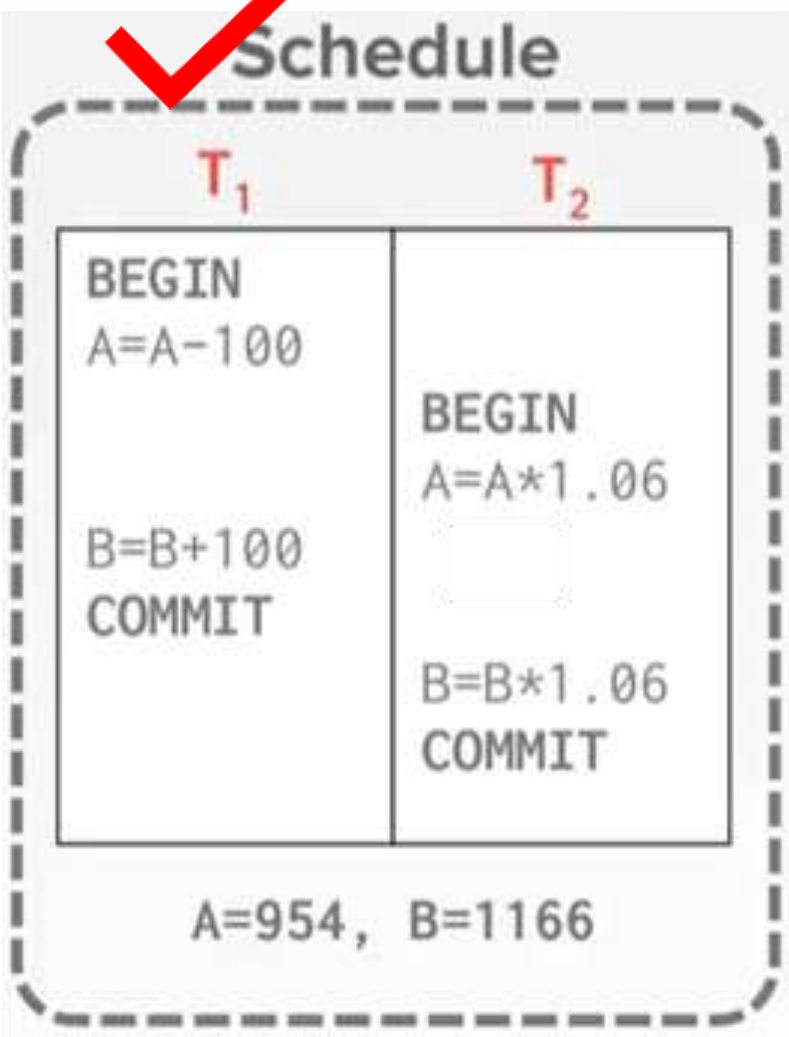
什么样的并发操作调度是正确的（续）

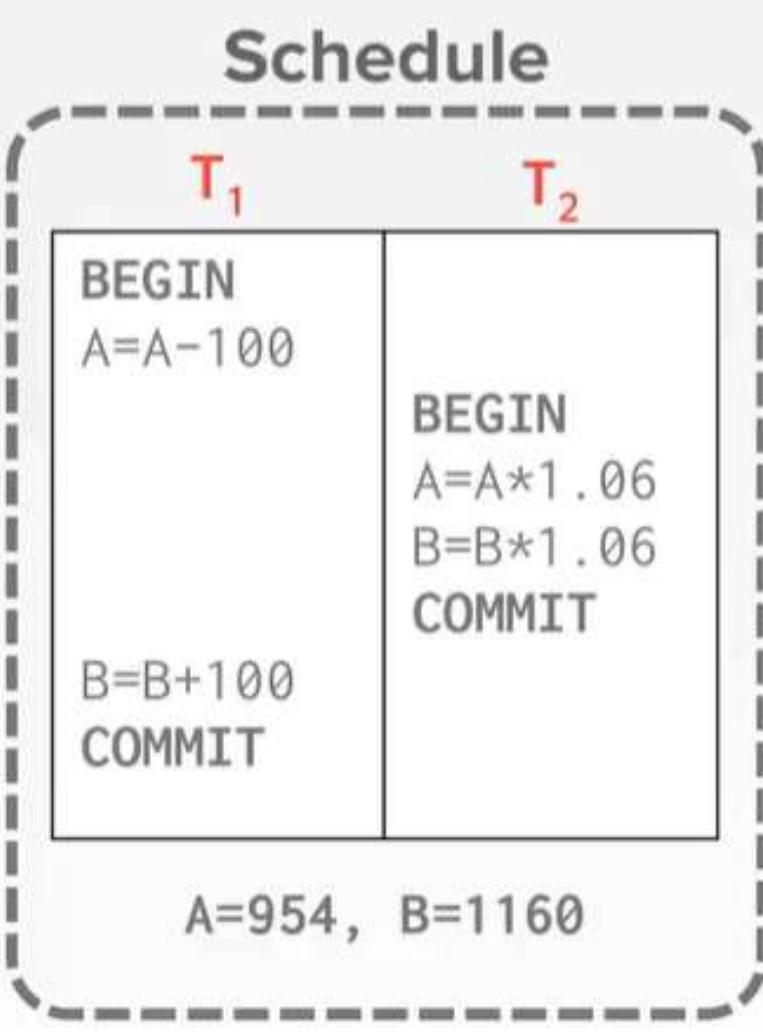
- **可串行性是并行事务正确性的唯一准则**



什么样的并发操作调度是正确的（续）



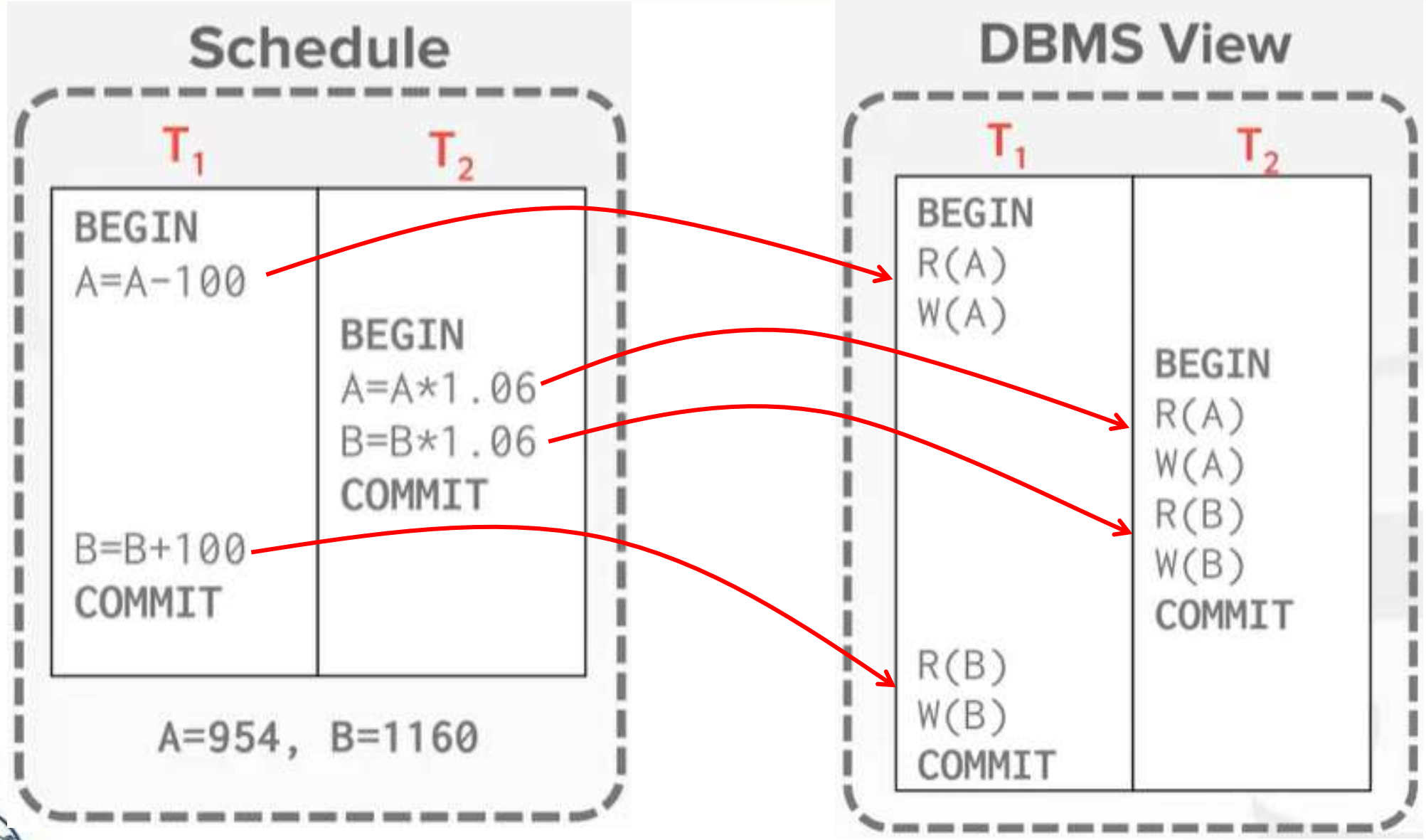




串行调度

并发调度

事务的表示方法


Schedule
T1
T2
BEGIN
A=A-100
BEGIN
A=A*1.06
B=B*1.06
COMMIT
B=B+100
COMMIT
A=954, B=1160
DBMS View
T1
T2
BEGIN
R(A)
W(A)
BEGIN
R(A)
W(A)
R(B)
W(B)
COMMIT
R(B)
W(B)
COMMIT

事务的表示方法

$R_i(X)$表示事务T_i的读X操作；

$W_i(X)$表示事务T_i的写X操作。

例：事务T1(Read(B)；A=B+1；write(A))，

事务T2(Read(A)；B=A+1；write(B))

可以表示成：

T_1：$R_1(B)W_1(A)$

T_2：$R_2(A)W_2(B)$

冲突操作

定义：如果两个操作来自不同的事务，它们对同一数据单位进行操作，并且其中至少有一个是写操作，则称这两个操作是相互冲突的或冲突操作。

例：事务T_0：$W_0(X)W_0(Y)W_0(Z)$

事务T_1：$R_1(X)R_1(Z)W_1(X)$

则在这两个事务中有冲突操作：

$R_1(X)$与$W_0(X)$　　$W_1(X)$与$W_0(X)$　　$R_1(Z)$与$W_0(Z)$

调度

设τ={T_1，T_2，…T_n}是一事务集，τ的一个调度S是一拟序集（∑,<s)

其中:

1) ∑说明S执行的操作是T1，T2，…T n 的操作。

2) <s 说明调度S遵守每个事务的操作的内部执行次序

3) 每对冲突操作的执行次序由S决定。

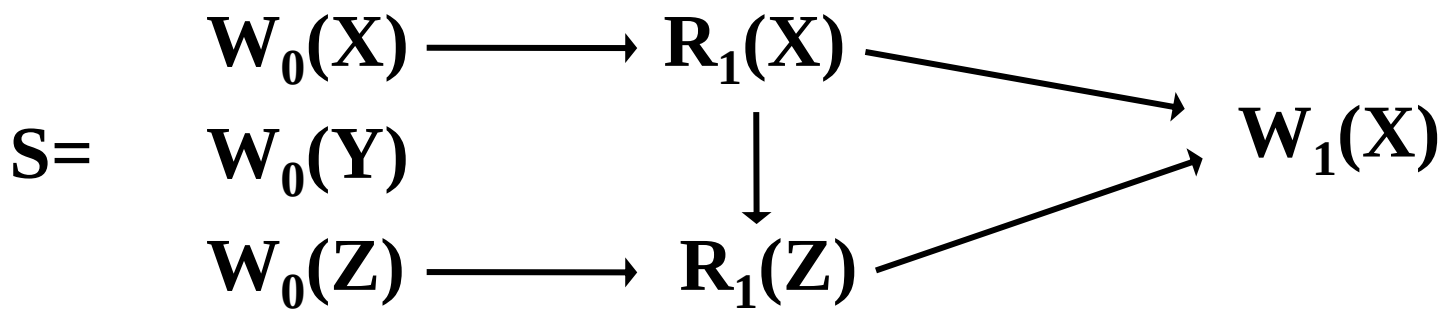
调度的图式方式

两个事务T_0，T_1的调度可以表示为：

$$S=(\{W_0(X),W_0(Y),W_0(Z),R_1(X),R_1(Z),W_1(X)\},$$

$$\{W_0(X)<R_1(X),\ W_0(Z)<R_1(Z),\ R_1(X)<R_1(Z),$$

$$R_1(X)<W_1(X),\ R_1(Z)<W_1(X)\})$$



调度的表式方式

T_1	T_2
	Read(A)
	Read(B)
Read(C)	
Write(C)	
Write(B)	
	Write(C)

冲突可串行化

一个调度Sc在保证：

（1）冲突操作的次序不变

（2）同一事务中操作次序不变

的情况下，通过交换两个事务不冲突操作的次序得到另一个调度Sc’，如果Sc’是串行的称调度Sc为冲突可串行化调度。

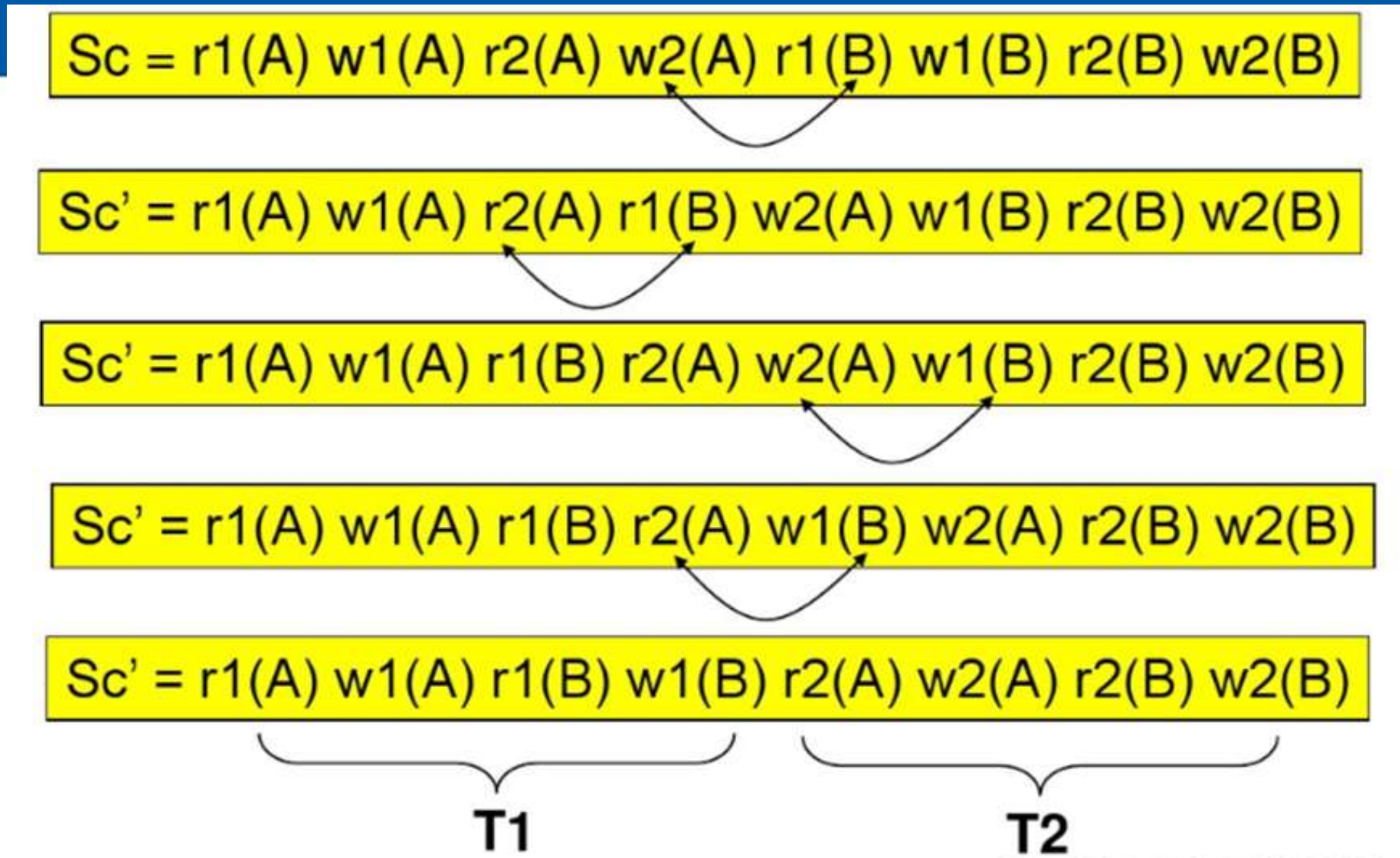
一个调度是冲突可串行化的，一定是可串行化调度，反之则不成立！

冲突可串行化的判定方法

【例】设事务T1和T2的一个调度

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

判断Sc是否是冲突可串行化的调度。



Sc'等价于串行调度T_1T_2，所以Sc是冲突可串行化的

冲突可串行化的判定方法

构造前趋图

设S是若干事务{T1,T2,...,Tn}的一个调度，S的前趋图G(V, E)是一个有向图，其构成规则如下：

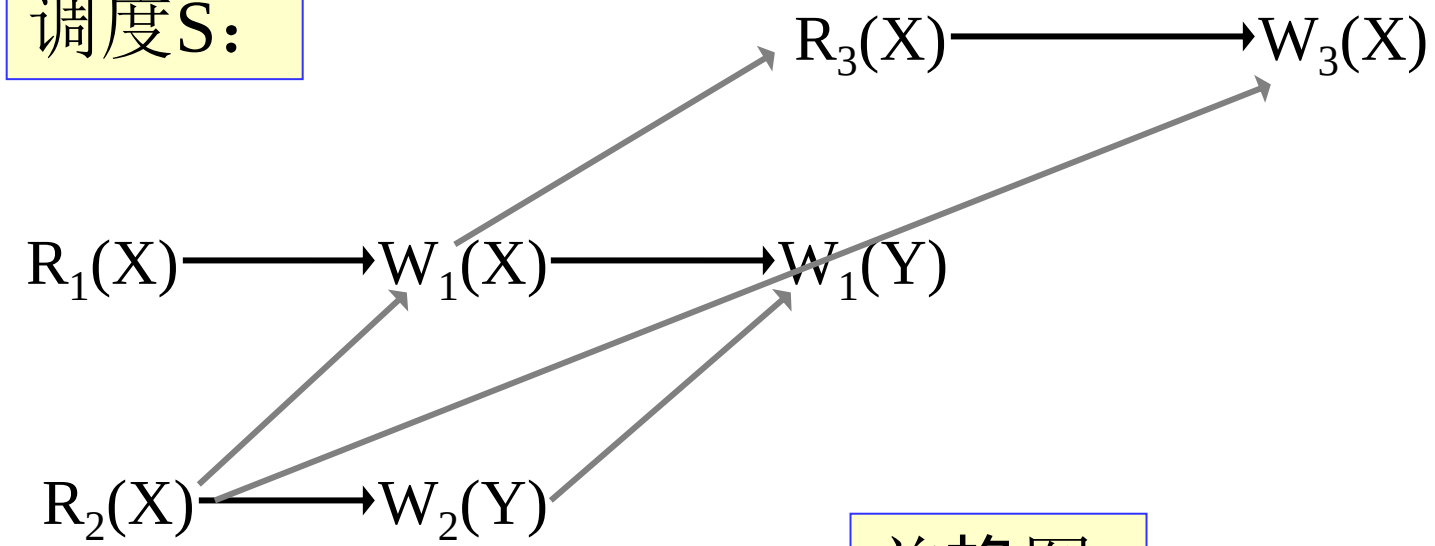
1）V是由所有参加调度的事务构成的节点

2）E是图中的一条有向边，如果O_i和O_j是冲突操作，且O_i先于O_j执行，则在图中有一条边$T_i \rightarrow T_j$。

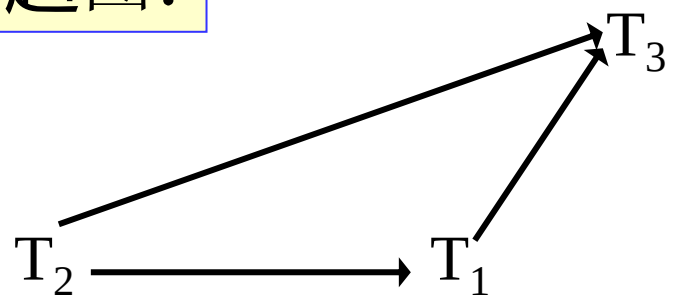
若一个调度的前趋图无环，则该调度是冲突可串行化的。

冲突可串行化的判定方法

调度S：



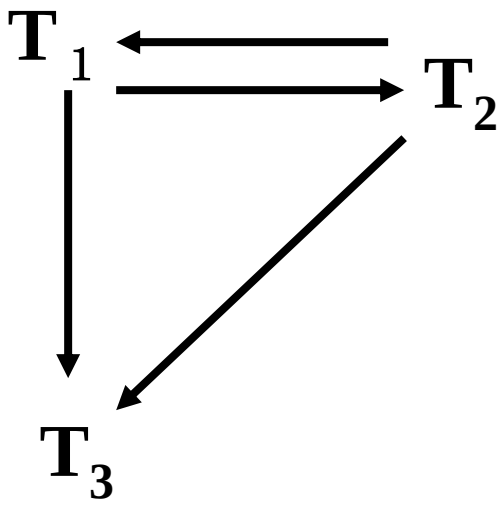
前趋图：



调度:

T_1	T_2	T_3
	Read(A)	
	Read(B)	
Read(C)		
Write(C)		
		Read(D)
	Write(C)	
		Read(C)
		Write(D)
Write(B)		

前趋图:



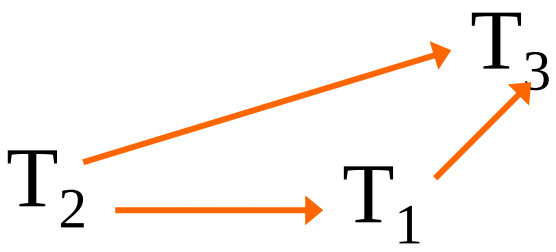
冲突可串行化的判定方法

等价的串行调度：如果前趋图是无环的，则S等价于前趋图的任一拓扑排序。

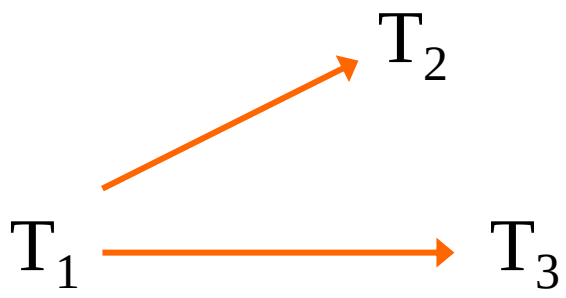
拓扑排序方法：

由于图中无回路，必有一个节点无入弧，将这个节点及其相连的弧删去，并把该节点存入先进先出的队列中。对剩下的图做同样的处理，直至所有节点移入队列中。按照队列中次序串行安排各事务的执行，就可以得到一个等价的串行调度。

冲突可串行化的判定方法



拓扑排序为：T2, T1, T3



拓扑排序为：T1,T3,T2

或为：T1,T2,T3

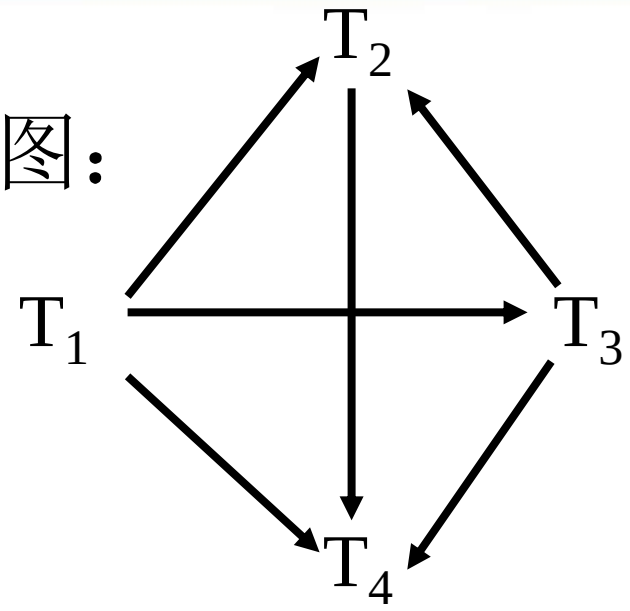
冲突可串行化的判定方法

根据如图调度S，判断其是冲突可串行化的吗？为什么？如果调度S是冲突可串行化的，请给出与之等价的一个串行调度序列。

T_1	T_2	T_3	T_4
		Write(y)	
Read(x)			
	Read(y)		
		Write(x)	
	Write(x)		
		Write(z)	
			Read(z)
			Write(x)

冲突可串行化的判定方法

根据调度S得到串行图：



由于图中不存在环，故调度S是可串行化的，与之等价的一个串行调度序列为：

$$T_1 \longrightarrow T_3 \longrightarrow T_2 \longrightarrow T_4$$

二、如何保证并发操作的调度是正确的

- 保证并发操作调度正确性的方法
- 封锁方法：

两段锁（Two-Phase Locking，简称2PL）协议

两段锁协议的内容

1. 在对任何数据进行读、写操作之前，事务首先要获得对该数据的封锁
2. 在释放一个封锁之后，事务不再获得任何其他封锁。

“两段”锁的含义

- 事务分为两个阶段
  - 第一阶段是获得封锁，也称为扩展阶段；
  - 第二阶段是释放封锁，也称为收缩阶段。

两段锁协议（续）

事务1的封锁序列：

Slock A ... Slock B ... Xlock C ... Unlock B ... Unlock A ... Unlock C；

事务2的封锁序列：

Slock A ... Unlock A ... Slock B ... Xlock C ... Unlock C ... Unlock B；

事务1遵守两段锁协议，而事务2不遵守两段协议。

两段锁协议（续）

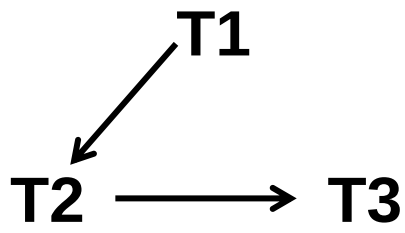
并行执行的所有事务均遵守两段锁协议，则对这些事务的所有并行调度策略都是可串行化的。

定理：任何一个遵从2PL协议的调度都是可串行化的。

- 事务遵守两段锁协议是可串行化调度的充分条件，而不是必要条件。
- 可串行化的调度中，不一定所有事务都必须符合两段锁协议。

不遵守2PL协议的可串行化调度

T1	T2	T3
Slock(X)		
Read(X)		
Unlock(X)		
	Slock(Y)	
	Read(Y)	
	Unlock(Y)	
		Slock(Y)
		Write(Y)
		Unlock(Y)
	Slock(X)	
	Write(X)	
	Unlock(X)	



T1 → T2 → T3

两段锁协议（续）

- 两段锁协议与防止死锁的一次封锁法
  - 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，因此一次封锁法遵守两段锁协议
  - 两段锁协议并不要求事务必须一次将所有要使用的数据全部加锁
  - 遵守两段锁协议的事务可能发生死锁

两段锁协议（续）

遵守两段锁协议的事务发生死锁

T_1	T_2
Slock B	
读B=2	
	Slock A
	读A=2
Xlock A	
等待	Xlock B
等待	等待

两段锁协议（续）

- 两段锁协议与三级封锁协议
  - 两类不同目的的协议
    - 两段锁协议
      - 保证并发调度的正确性
    - 三级封锁协议
      - 在不同程度上保证数据一致性
  - 遵守第三级封锁协议必然遵守两段锁协议

6.3.4 封锁粒度

- X锁和S锁都是加在某一个数据对象上的
- 封锁对象可以很大也可以很小

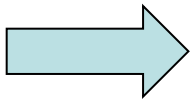
  例： 对整个表加锁

  对某个记录加锁

  – 逻辑单元，物理单元
- 封锁对象的大小称为封锁的粒度(Granularity)
- 多粒度封锁(multiple granularity locking)

  – 在一个系统中同时支持多种封锁粒度供不同的事务选择

Delete
From S
Where id=30;



事务层：

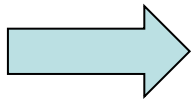
Slock(A)

R(A)

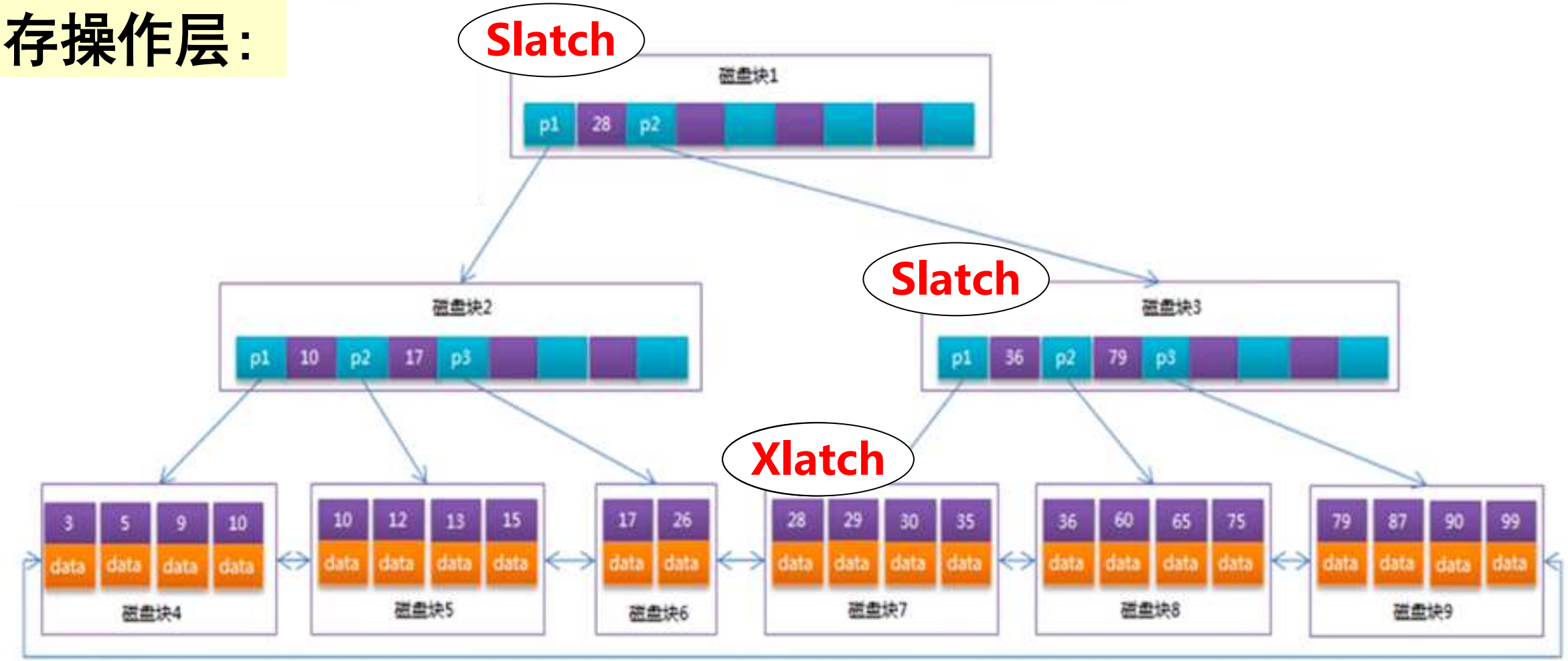
Xlock(A)

W(A)

Unlock(A)



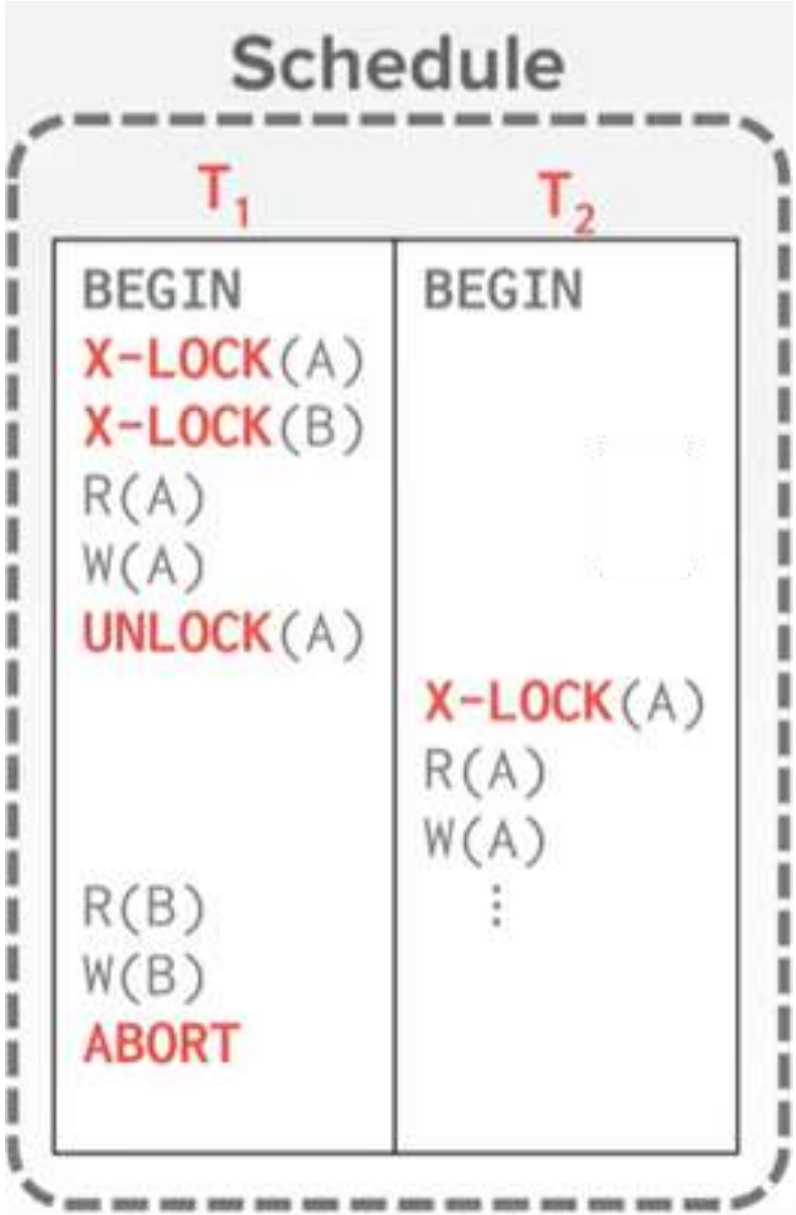
内存操作层：


Slatch
磁盘块1
p1
28
p2
Slatch
磁盘块2
p1
10
p2
17
p3
磁盘块3
p1
36
p2
79
p3
Xlatch
3
5
9
10
data
磁盘块4
10
12
13
15
磁盘块5
17
26
磁盘块6
28
29
30
35
磁盘块7
36
60
65
75
磁盘块8
79
87
90
99
磁盘块9

封锁技术带来的问题：

（1）死锁

（2）恢复时的级联回滚



本节重点

- 并发操作引发的问题
- 封锁的概念、封锁协议
- 事务的隔离级别
- 活锁和死锁
  - 解决死锁的方法
- 可串行化的调度
  - 2PL协议