事务管理(并发控制技术1)

单 位：重庆大学计算机学院

火车站的调度

- 一个火车站只能同时停靠K辆火车，如果火车数量增多，如何保证火车停靠能够不发生冲突？

主要学习目标

- 锁
- 两阶段封锁协议

前测小问题

- 两个事务同时并发执行，会出现什么问题？

一 并发控制

1)什么是并发?

- 案例1：

T1	T2
Read(A)	Read(A)
A:=A-50	Temp:=A*0.1
Write(A)	A:=A-temp
Read(B)	Write(A)
B:=B+50	Read(B)
Write(B)	B:=B+temp
commit	Write(B)
	commit

2)并发的重要性?

1. 提高吞吐量和资源利用率
2. 减少等待时间

一 并发控制

3）不加控制的并发会怎么样?

T1	T2
Read(A)	
A:=A-50	
Write(A)	
Read(B)	
B:=B+50	
Write(B)	
commit	
	Read(A)
	Temp:=A*0.1
	A:=A-temp
	Write(A)
	Read(B)
	B:=B+temp
	Write(B)
	commit

T1	T2
	Read(A)
	Temp:=A*0.1
	A:=A-temp
	Write(A)
	Read(B)
	B:=B+temp
	Write(B)
	commit
Read(A)	
A:=A-50	
Write(A)	
Read(B)	
B:=B+50	
Write(B)	
commit	

一 并发控制

3）不加控制的并发会怎么样?

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

一 并发控制

3）不加控制的并发会怎么样?

案例2

- 更新数据库中某表的操作。

```
Update product
set price= price*1.02
```

- 具体怎么操作?
- 如果有一个读操作同时执行会怎么样?

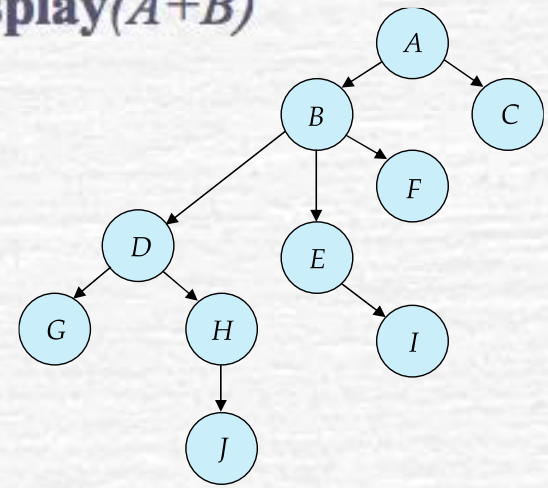
```
Select price from product
```

有效的并发

1）实现有效并发的手段？

T_2: lock-S(*A*);
read (*A*);
unlock(*A*);
lock-S(*B*);
read (*B*);
unlock(*B*);
display(*A*+*B*)

T_1: lock-X(*B*);
read (*B*);
B:=B-50;
write (*B*);
unlock(*B*);



基于图
基于时间戳
快照隔离

案例3

一 并发控制

有效的并发

2)两种锁的区别是什么？为什么需要s锁？

给数据项加锁的方式有多种，在这一节中，我们只考虑两种：

1. **共享的**(shared)：如果事务 T_i 获得了数据项 Q 上的**共享型锁**(shared-mode lock)(记为 S)，则 T_i 可读但不能写 Q。
2. **排他的**(exclusive)：如果事务 T_i 获得了数据项 Q 上的**排他型锁**(exclusive-mode lock)(记为 X)，则 T_i 既可读又可写 Q。

T_1: lock-X*(B)*;
read *(B)*;
B:=B-50;
write *(B)*;
unlock*(B)*;

T_2: lock-S*(A)*;
read *(A)*;
unlock*(A)*;
lock-S*(B)*;
read *(B)*;
unlock*(B)*;
display*(A+B)*

二 锁与封锁协议

2.1 锁

为何引入锁，不同类型锁的作用？

给数据项加锁的方式有多种，在这一节中，我们只考虑两种：

1. **共享的**(shared)：如果事务 T_i 获得了数据项 Q 上的**共享型锁**(shared-mode lock)(记为S)，则 T_i 可读但不能写 Q。
2. **排他的**(exclusive)：如果事务 T_i 获得了数据项 Q 上的**排他型锁**(exclusive-mode lock)(记为X)，则 T_i 既可读又可写 Q。

指多大对象？ 可大可小(大到整个数据库，小到一个记录或属性值)

(分析吃自助餐场景：勺子＋四处看看)

引入锁机制：
是为了保证数据的一致性(事务的隔离性)和提高系统的并发处理能力!
或者说，为保了证应用的有效性(两人不会订到同一位置火车票，查看到一个不存在的成绩--如教务处刚误输一个学生的成绩后马上又删除)

何谓锁相容矩阵，S锁带来多大并发能力？

S锁明显增强了并发能力：
因为可能**95%**以上的应用是读数据，更新应用的频率非常小。
没有**S**锁，系统的发处理能力会大大降低，仅比串行调度略好。

(自助餐看多，而动手少)

	S	X
S	true	false
X	false	false

2.2 封锁协议

什么是封锁协议，加锁后的调度是什么样？

指一组加锁规则！

转账

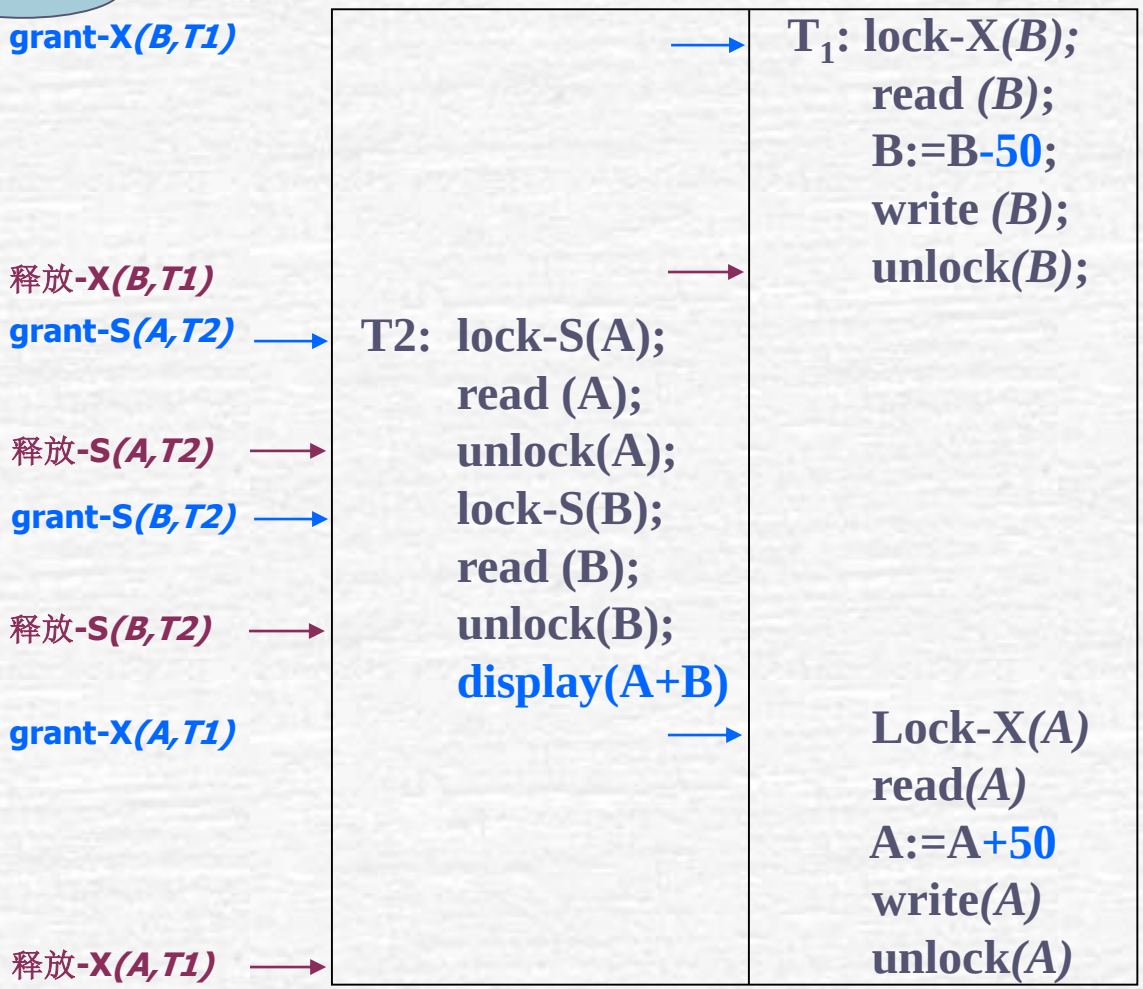
```
T1: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    unlock(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(A).
```

图 15-2 事务 T_1

查总账

```
T2: lock-S(A);
    read(A);
    unlock(A);
    lock-S(B);
    read(B);
    unlock(B);
    display(A + B).
```

图 15-3 事务 T_2



（并发控制器）

图25-4

2.2 封锁协议(续)

图25-4

这是一个可串行化调度?

若：A is 100,B is 200
则 T_2调度将会显示：
A+B=250

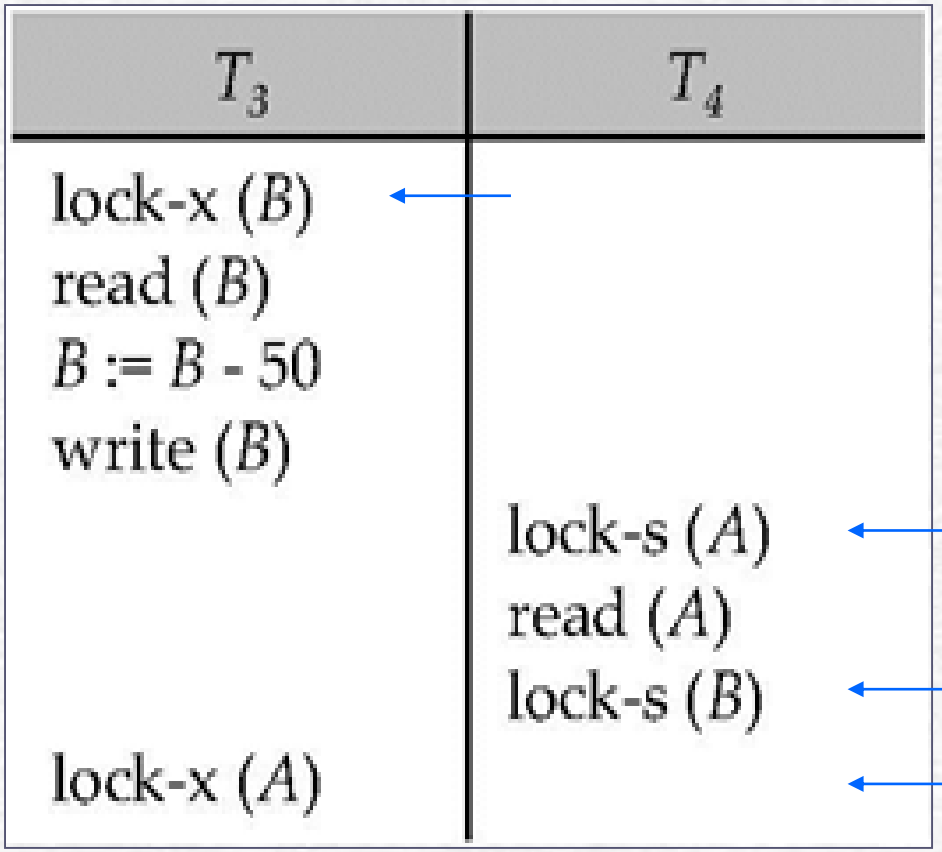
而串行调度<T1,T2>后
T2显示A+B=300

且串行调度<T2,T1>后
T2显示A+B=300

故:不是可串行化调度！
（调度优先图有环）
T2读到T1未提交数据！

仅有封锁未必能保证调度的可串行化！

什么是死锁,如何引起,有何危害?



grant-X*(B,T3)*-成功

grant-S*(A,T4)* -成功

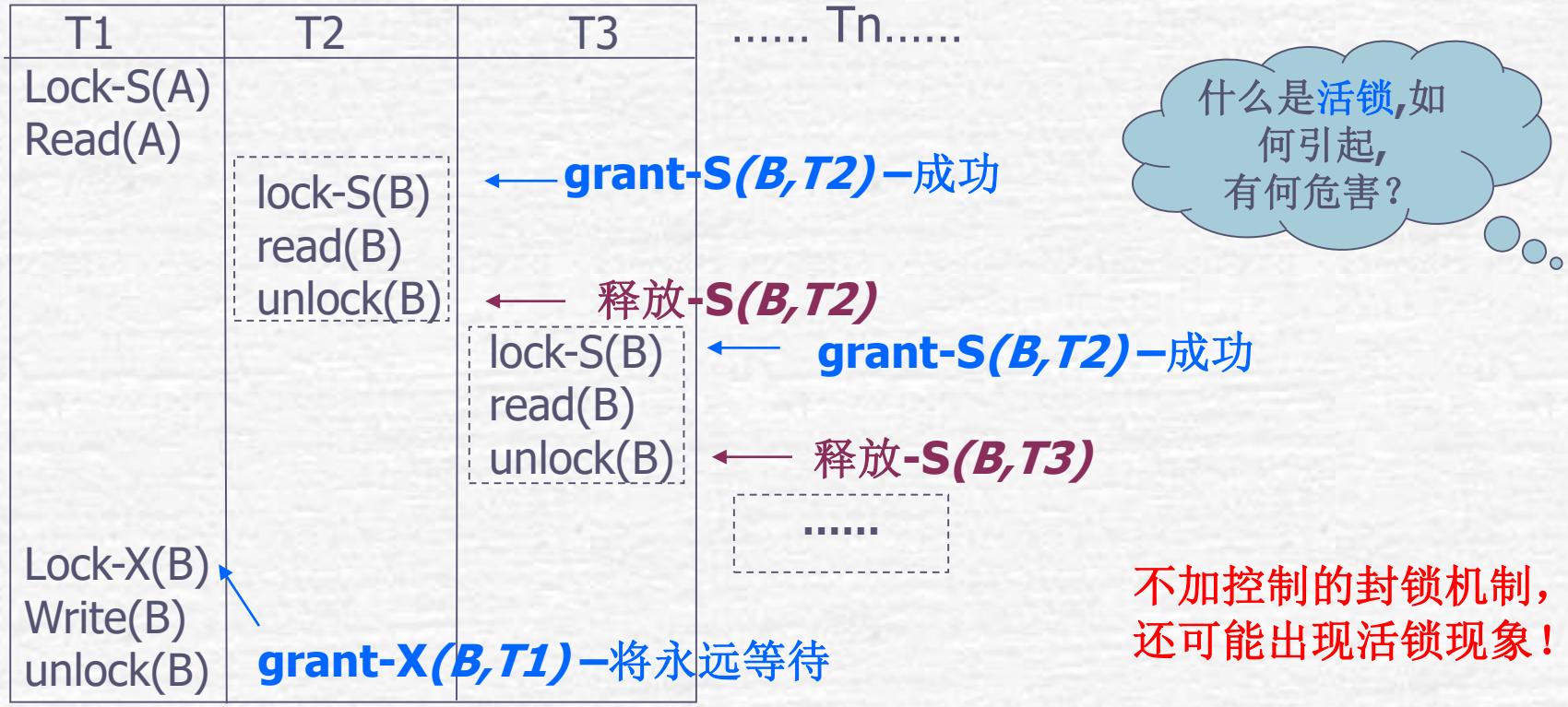
grant-S*(B,T4)* –等待T3释放

grant-X*(A,T3)* -等待T4释放

形成两个事务“相互等待”对方释放资源各自才能往下继续做的僵局!

不加控制的封锁机制，可能引起死锁现象发生!

2.4 活锁



Starvation饿死(活锁) is also possible if concurrency control manager is badly designed. For example:

1) **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.**

2) **The same transaction is repeatedly rolled back due to deadlocks.**

出现一个事务永远(长时间)等待某一数据项被其它事务释放后才能进行封锁的现象!

三 两阶段封锁协议

3.1 两阶段封锁协议

两阶段封锁协议的主要特点？

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)	

图 15-4 调度 1

两阶段封锁协议要求每个事务分两个阶段提出加锁和解锁申请。

1. **增长阶段**(growing phase)：事务可以获得锁，但不能释放锁。
2. **缩减阶段**(shrinking phase)：事务可以释放锁，但不能获得新锁。

封锁点

```
T3: lock-X(B);
    read(B);
    B := B − 50;
    write(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(B);
    unlock(A).
```

图 15-5 事务 T_3

封锁点

```
T4: lock-S(A);
    read(A);
    lock-S(B);
    read(B);
    display(A + B);
    unlock(A);
    unlock(B).
```

图 15-6 事务 T_4

事务**T1-T4**都是两阶段封锁协议吗，封锁点在何处？

T1和T2不是两阶段封锁协议，

T3和T4都是两阶段封锁协议！

两阶段封锁协议的主要用途？

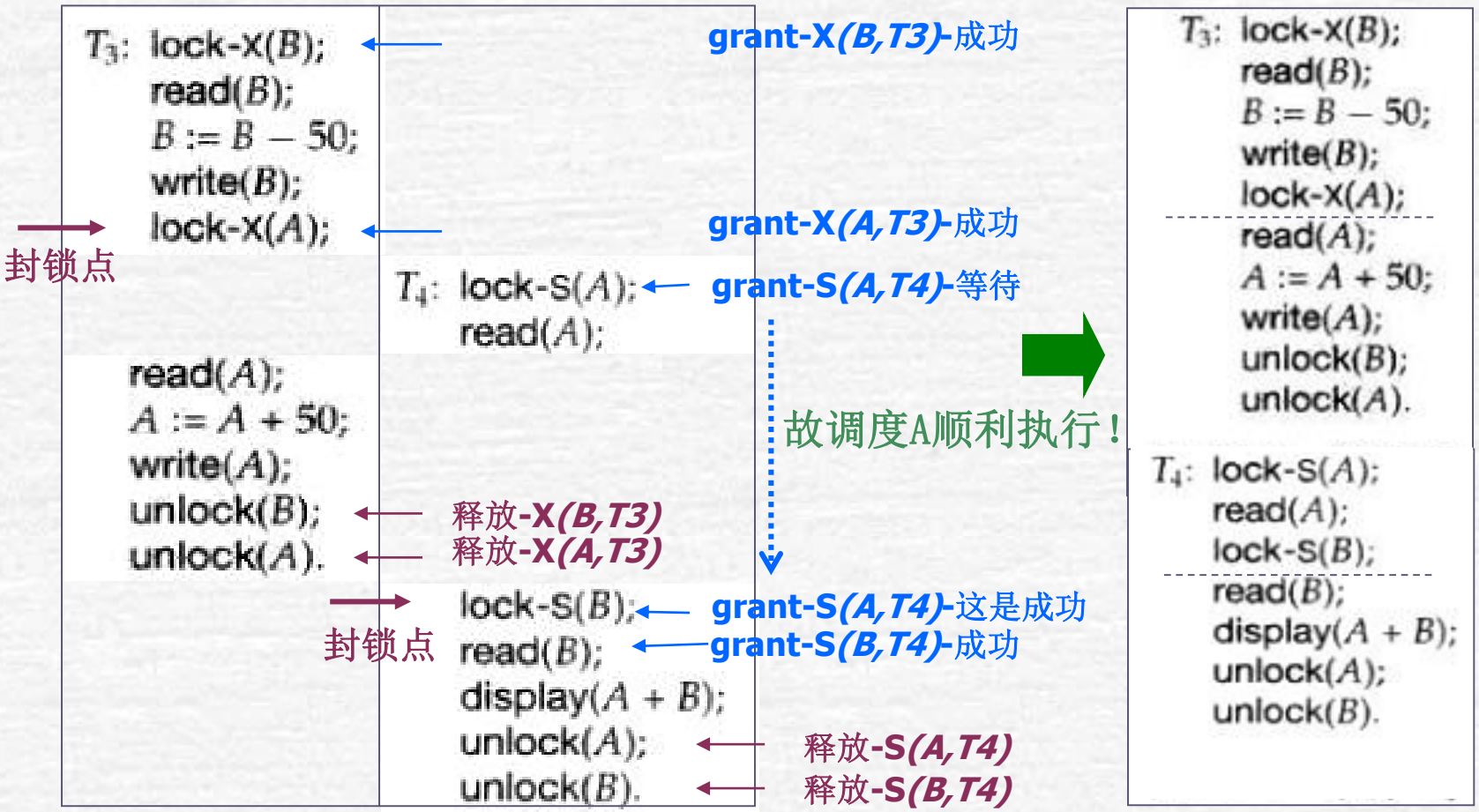
p. 376 我们可以证明两阶段封锁协议保证冲突可串行化。对于任何事务，在调度中该事务获得其最后加锁的位置(增长阶段结束点)称为事务的**封锁点**(lock point)。这样，多个事务可以根据它们的封锁点进行排序，实际上，这个顺序就是事务的一个可串行化顺序。我们将此证明留为习题(见实践习题15.1)。

两阶段所要求释放锁的操作必须在事务的末尾吗？

不是必须的！如T3中的unlock(B)可以紧跟在Lock-X(A)之后。

示例分析一：两阶段封锁协议&可串行化

两阶段封锁调度P等价于何串行调度？

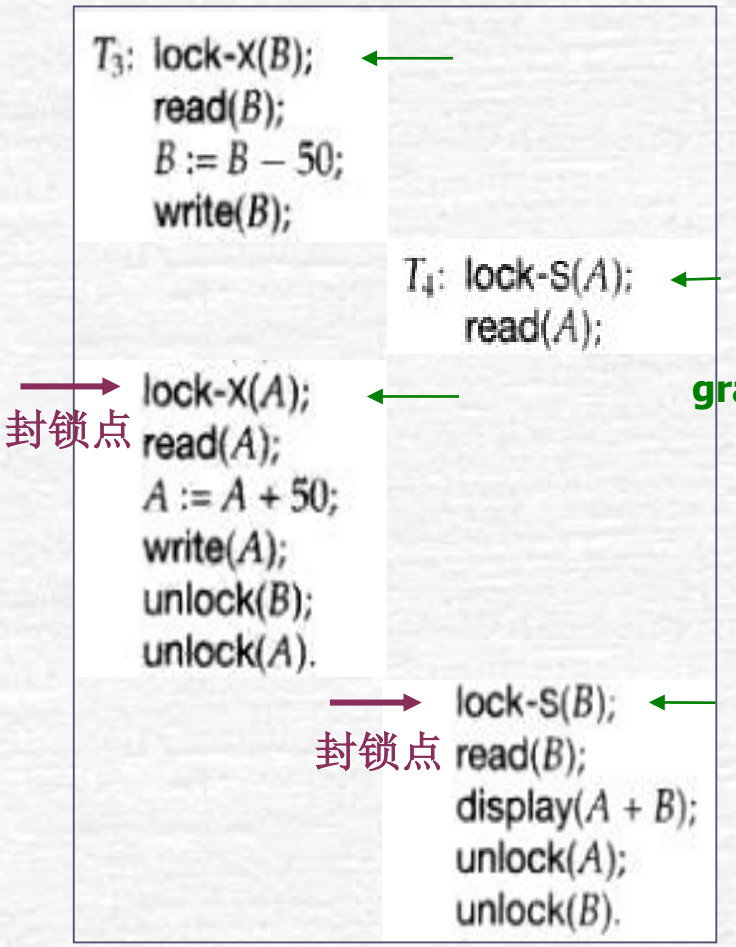


两阶段封锁-并发调度P

与P等价的串行调度Q

示例分析二：两阶段封锁&死锁

两阶段封调度Q能够顺利执行完成？



两阶段封锁-并发调度Q

grant-X(B,T3)-成功

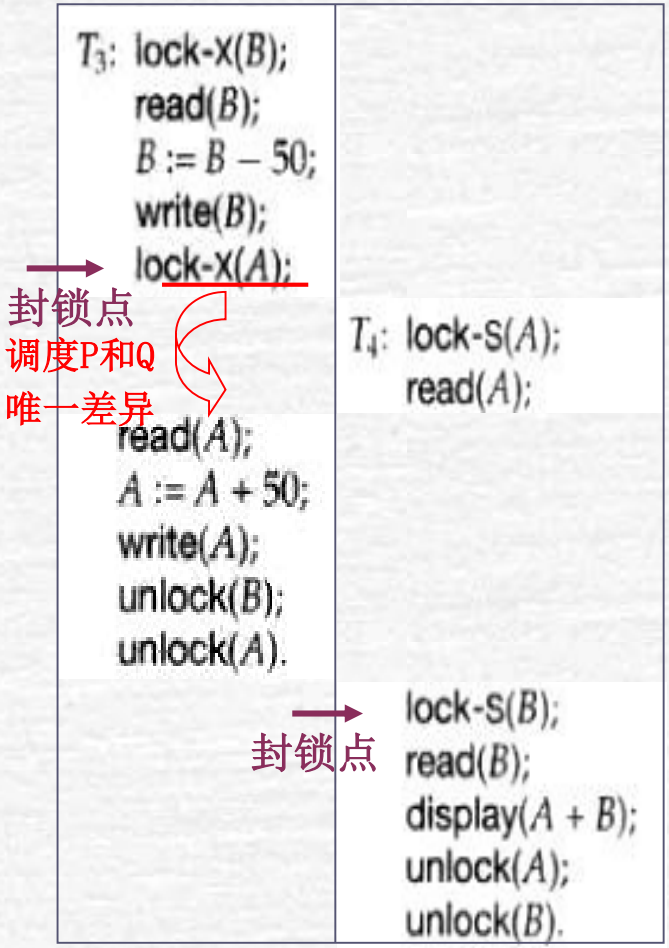
grant-S(A,T4)-成功

grant-X(A,T3)-等待T4

grant-S(B,T4)-等待T3

P还算并行调度吗？

当然算！如事务T4还可能处理其它数据C, D, 可与事务T3并行做.



两阶段封锁-并发调度P

调度Q形成死锁！

调度P和Q有何差异？

调度P顺利执行！

3.2 可串行化与两阶段封锁的关系

Q是冲突可串行化调度吗?

等价于串行调度 **T1→T2→T3**

调度Q符合两段锁协议吗?

不符合！

- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used.

两阶段封锁协议保证冲突可串行化，冲突可串行化未必符合两阶段封锁！

T2	T3	T1
Lock-S(A)		
Read(A)	冲突操作	
Unlock(A)		
	lock-X(A)	
	write(A)	
	unlock(A)	
		lock-x(B)
	冲突操作	write(B)
		unlock(B)
Lock-X(B)		
Write(B)		
unlock(B)		

案例：并行调度**Q**

3.3 级联调度&两阶段封锁协议 p. 376

调度P和Q的执行效果有何不同?

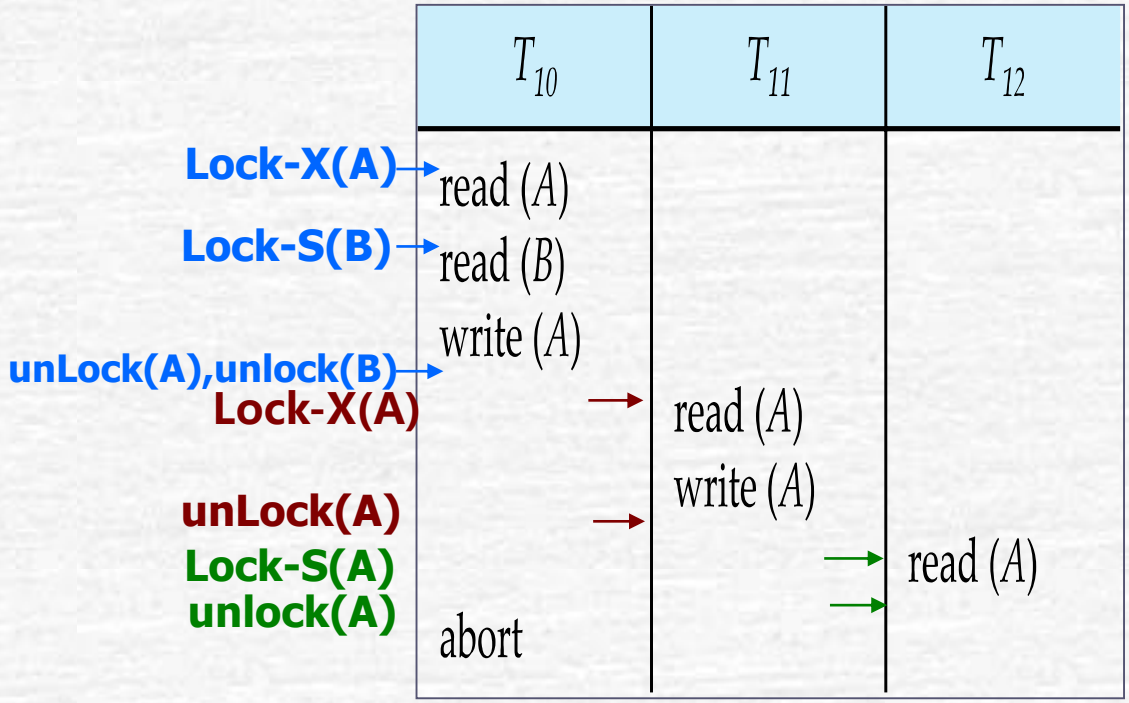
除了调度可串行化外，调度还应该是无级联的。
在两阶段封锁协议下，级联回滚可能发生。

T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
abort	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)

唯一差别

调度P：符合两阶段封锁协议

若T5撤销不会级联卷回



但执行效果却完全不同

调度Q：符合两阶段封锁协议

若T10撤销引发级联卷回

(因T11 T12读了T10写的数据)

3.4 严格两阶段封锁协议 p. 376

什么是严格两阶段封锁协议，该调度是吗？

严格两阶段封锁协议有何用途？

可保证调度不会出现级联回滚！

T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)
abort		

图 15-8 在两阶段封锁下的部分调度

（符合两阶段封锁协议）

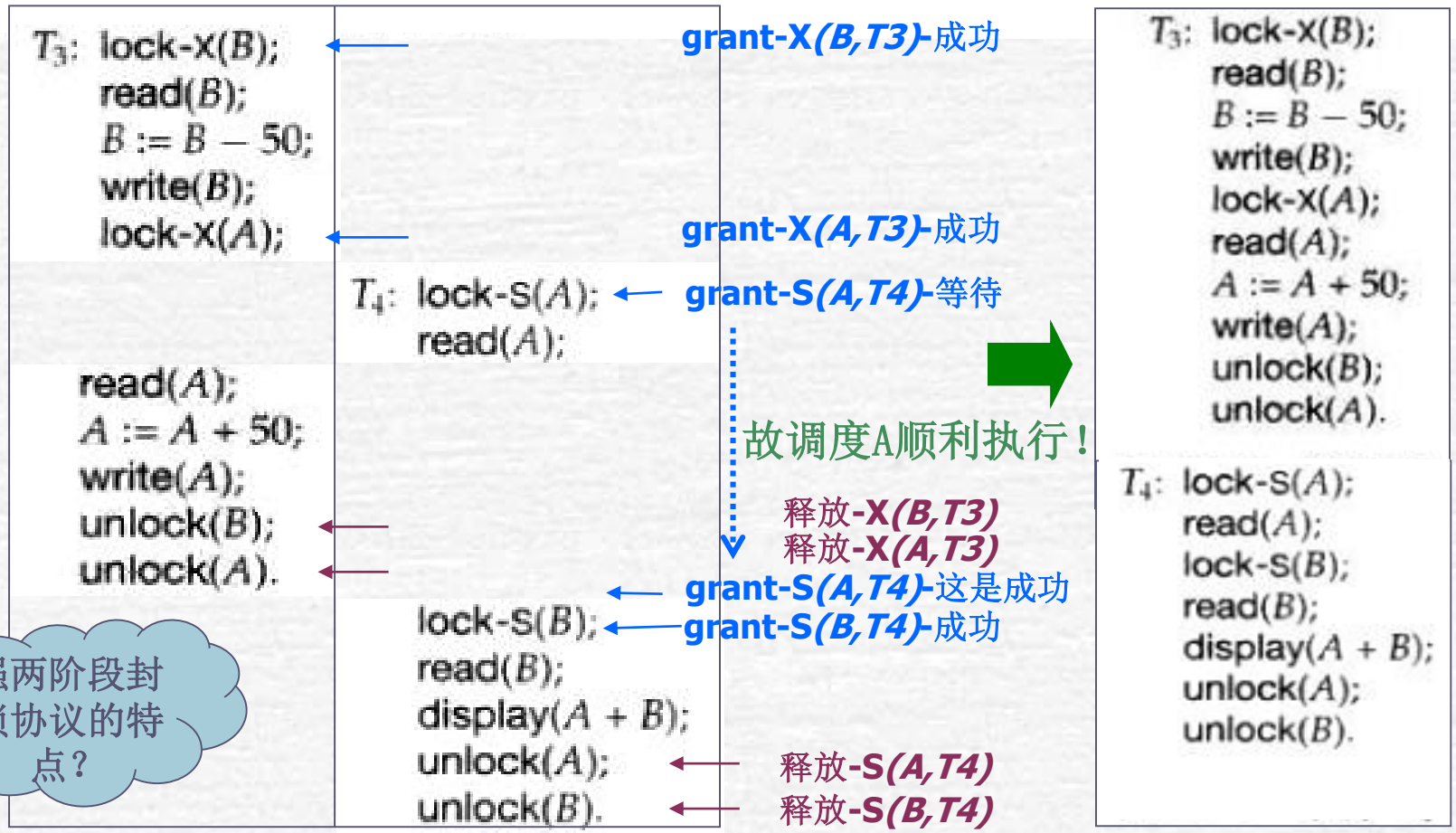
（T_5撤销时引发T_6，T_7级联卷回）

级联回滚可以通过将两阶段封锁修改为**严格两阶段封锁协议**（strict two-phase locking protocol）加以避免。这个协议除了要求封锁是两阶段之外，还要求事务持有的所有排他锁必须在事务提交后方可释放。这个要求保证未提交事务所写的任何数据在该事务提交之前均以排他方式加锁，防止其他事务读这些数据。

事务提交之前不得释放排他锁

3.5 强两阶段封锁协议 p. 376

另一个两阶段封锁的变体是**强两阶段封锁协议**(rigorous two-phase locking protocol)，它要求事务提交之前不得释放任何锁。我们很容易验证在强两阶段封锁条件下，事务可以按其提交的顺序串行化。



两阶段封锁-并发调度P

与P等价的串行调度Q

什么是带锁转换的两段锁协议?

- 带锁转换的两阶段封锁协议:
  - 第一阶段:
    - 能申请lock-S
    - 能申请lock-X
    - 能将lock-S 转换为lock-X (upgrade升级)
  - 第二阶段:
    - 能释放 lock-S
    - 能释放 lock-X
    - 能将lock-X 转换为(downgrade降级)
- 这个协议确保了冲突可串行化.

假设：T8和T9都符合两段锁协议.

T8：read(a1)
read(a2)
.....
read(a_n)
write(a1)

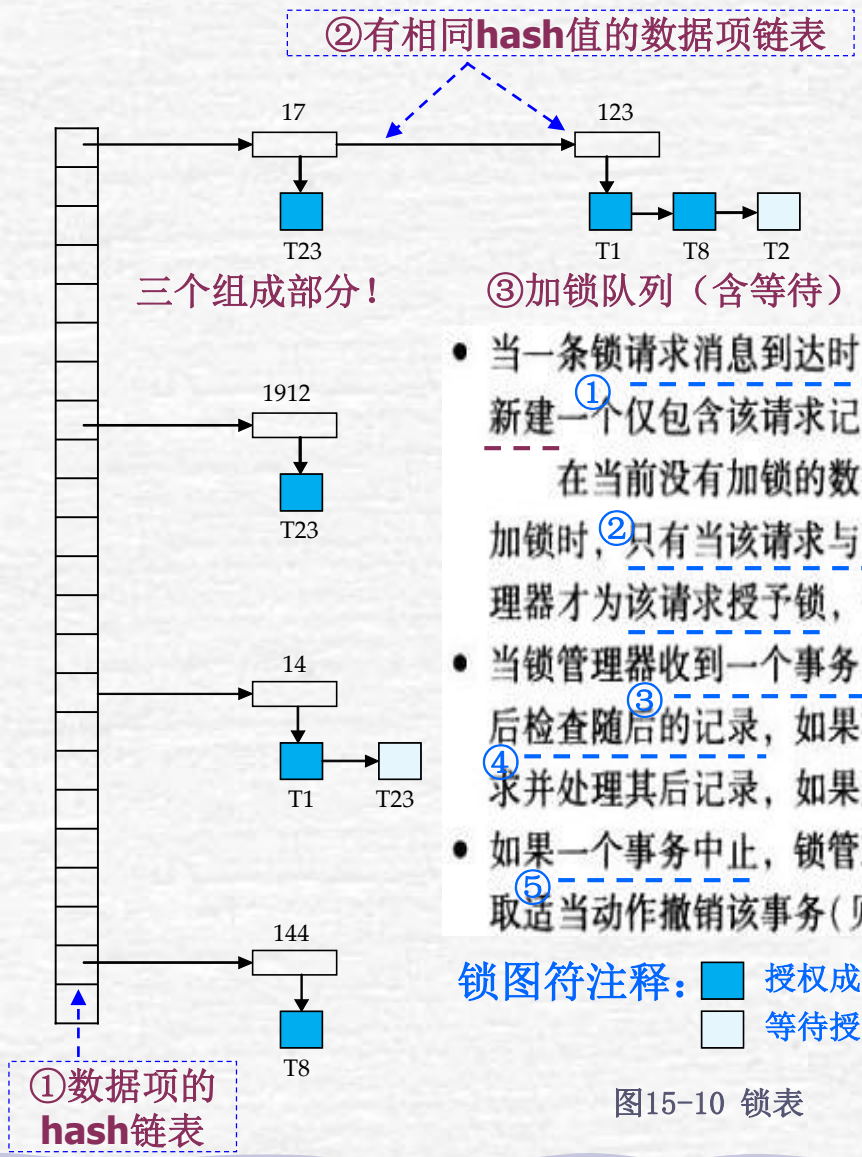
T9：read(a1)
read(a2)
display(a1+a2)

因T8必须对a1加X锁，两事务的并发度不高！(并发相当于串行执行)

T8	T9
Lock-S(a1)	
	Lock-S(a1)
Lock-S(a2)	
	Lock-S(a2)
Lock-S(a3)	
Lock-S(a4)	
	unLock-S(a1)
	unLock-S(a2)
.....	
Lock-S(a_n)	
update(a1)	← S锁升级为X锁

图15-9. 带所转换的协议.
（直到修改数据时才升级锁）
提高了事务的并发度！

四 封锁机制的实现



锁管理器的作用，主要组成部分？

锁管理器的基本工作原理？

- 当一条锁请求消息到达时，如果相应数据项的链表存在，在该链表末尾增加一个记录；否则，新建一个仅包含该请求记录的链表。

  在当前没有加锁的数据项上总是授予第一次加锁请求，但当事务向已被加锁的数据项申请加锁时，只有当该请求与当前持有的锁相容，并且所有先前的请求都已授予锁的条件下，锁管理器才为该请求授予锁，否则，该请求只好等待。
- 当锁管理器收到一个事务的解锁消息时，它将与该事务相对应的数据项链表中的记录删除，然后检查随后的记录，如果有，如前所述，就看该请求能否被授权，如果能，锁管理器授权该请求并处理其后记录，如果还有，类似地一个接一个地处理。
- 如果一个事务中止，锁管理器删除该事务产生的正在等待加锁的所有请求。一旦数据库系统采取适当动作撤销该事务（见 16.3 节），该中止事务持有的所有锁将被释放。 p. 378

锁图符注释：授权成功　等待授权

图15-10 锁表

课堂小测试

Read(A),Read(B),Read(C),C=A+B,Write(C)
使用两阶段封锁协议为该事务加锁。

课程总结与作业安排

- 基本知识：
  - 锁的概念和基本种类
  - 封锁协议
  - 两阶段封锁协议
  - 锁转换
- 扩展学习：
  - 如何在数据库中实现锁？
- 作业

  第15章习题： 15.2,15.3.