# 第三部分 事务的并发控制（5章内容）

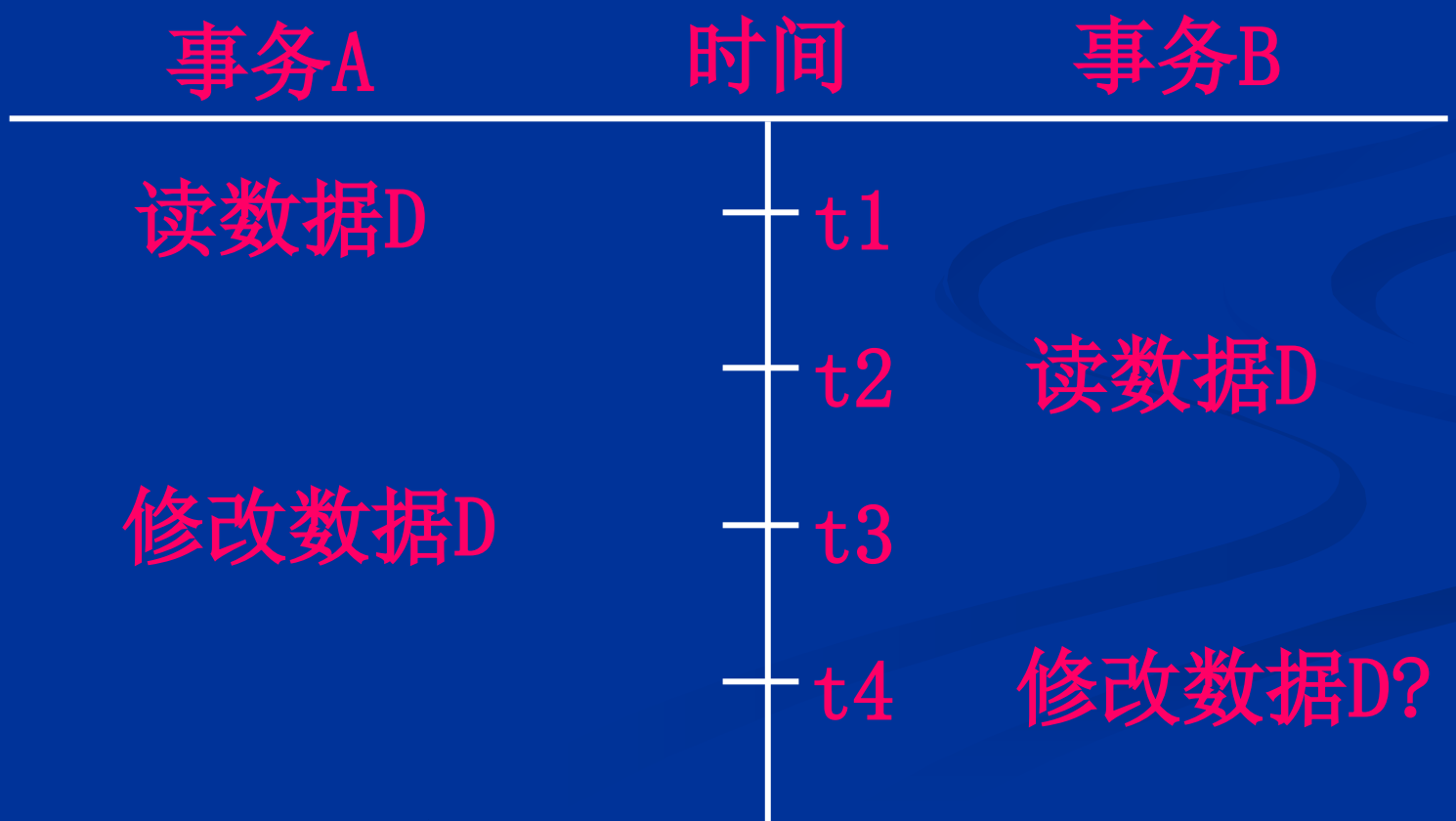
1. 并发控制的概念
2. 基于封锁的调度
3. 基于时间戳的调度
4. 基于有效性检验的调度
5. 多版本并发调度
6. 逻辑层事务的控制

# §1. 事务并发的概念

在某一时间段内，多种事务同步存取相同的数据库数据。

(1) 并发操作时因为不能隔离而产生的问题：

①丢失修改



## ②读入“脏”数据

# ③不可反复的读

（2）调度：若T = { T1 , T2 , ……, Tn }是一组并发执行的事务，则T中各事务的主要操作按时间排序的一种序列，记S = $\{\Sigma_{T,}, \angle_T\}$，是T 的一种调度。

a. 串行调度：若对于一种调度S 中的任意两个事务Ti和Tj, i≠j, Σi<Σj,或Σj<Σi.

b. 一致性调度:

假如执行一种调度S ，使数据库从一种一致性状态转入另一种一致性状态，则称S 是一种一致性调度。

注： 数据库状态的“一致性”与业务规则有关。

例：

例1 设： $T_1$、$T_2$两个事务定义如下：

| $T_1$: | $T_2$: |
| --- | --- |
| Read(x); | Read(x); |
| x := x +10; | x := x - 20; |
| Write(x); | Write(x); |
| Read(y); | Read(y); |
| y := y –15; | y := y * 2; |
| Write(y); | Write(y); |
| Commit; | Commit; |

$T_1$、$T_2$的两个调度$S_1$、$S_2$如下：

$S_1$ = { $R_1(x)$; $x := x + 10$; $W_1(x)$;

$R_1(y)$; $y := y - 15$; $W_1(y)$; $C_1$

$R_2(x)$; $x := x - 20$; $W_2(x)$;

$R_2(y)$; $y := y * 2$; $W_2(y)$; $C_2$ }

$S_2$ = { $R_2(x)$; $x := x - 20$; $W_2(x)$;

$R_2(y)$; $y := y * 2$; $W_2(y)$; $C_2$

$R_1(x)$; $x := x + 10$; $W_1(x)$;

$R_1(y)$; $y := y - 15$; $W_1(y)$; $C_1$ }

这两个调度均为串行调度

（2）可串行化调度的概念：

设：T = { T1 , T2 , ……,Tn }是一组并发执行的事务，S 是T 的一种串行调度，而S ′ 是T 的一种调度。若S ′ 是与S 有着相同一致性状态的一种调度，则称S ′ 是一种可串行化调度。

例3 T1 、T2 的一种可串行化调度S3 ：

```
S3 = {
        R1 (x); x := x +10; W1 (x);
        R2 (x); x := x  -  20; W2 (x);
        R1 (y); y := y  - 15; W1 (y); C1
       R2 (y); y := y * 2; W2 (y); C2
     }
```

例4 T1 、T2 的另一种可串行化调度S4 ：

```
S4 = {
R2 (x); x := x  -  20; W2 (x);
R1 (x); x := x +10; W1 (x);
R2 (y); y := y * 2; W2 (y); C2
R1 (y); y := y  - 15; W1 (y); C1
}
```

例5 T1 、T2 的一种不可串行化调度S5 ：

```
S5 = {
R1 (x); x := x +10; W1 (x);
R2 (x); x := x  -  20; W2 (x);
R2 (y); y := y * 2; W2 (y); C2
R1 (y); y := y  - 15; W1 (y); C1
}
```

（3）冲突可串行化：Ti={I1,I2…In},Tj={J1,J2,…,Jm}

则考虑两条指令Ii,Jk的下列情况：

Ii=read(Q),Jk=read(Q)　顺序无关紧要

Ii=read(Q),Jk=write(Q)　顺序主要

Ii=write(Q),Jk=read(Q)　顺序主要

Ii=write(Q),Jk=write(Q)　顺序主要

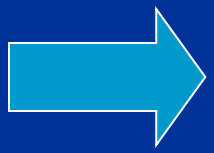
$I_i, J_k$是冲突的

某一并行调度S经过非冲突指令转换成串行调度，且其该串行调度的执行成果一致，则S是冲突可串行化的。

| T1 | T2 |
| --- | --- |
| Read(A) | |
| Write(A) | |
| | Read(A) |
| Read(B) | |
| | Write(A) |
| Write(B) | |
| | Read(B) |
| | Write(B) |

| T1 | T2 |
| --- | --- |
| Read(A) | |
| Write(A) | |
| | Read(A) |
| Read(B) | |
| | Write(A) |
| Write(B) | |
| | Read(B) |
| | Write(B) |

| T1 | T2 |
| --- | --- |
| Read(A) | |
| | Write(A) |
| Write(A) | |



非冲突可串行的调度

* 有关冲突可串行性调度的讨论

1. 构造冲突可串行性调度
   允许不同事务上的非冲突操作互换执行顺序。

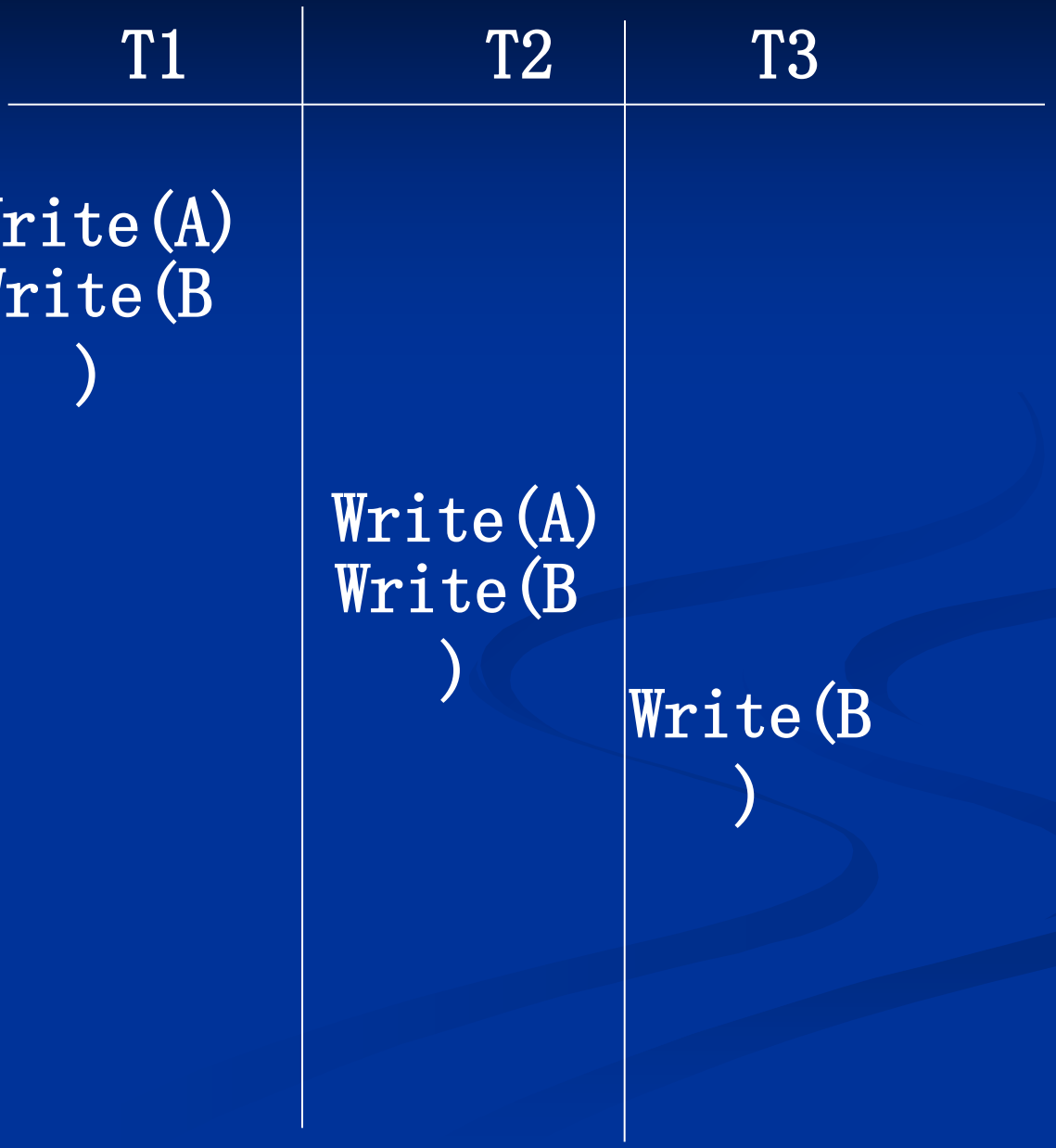
2. 冲突可串行调度肯定是一致性调度。

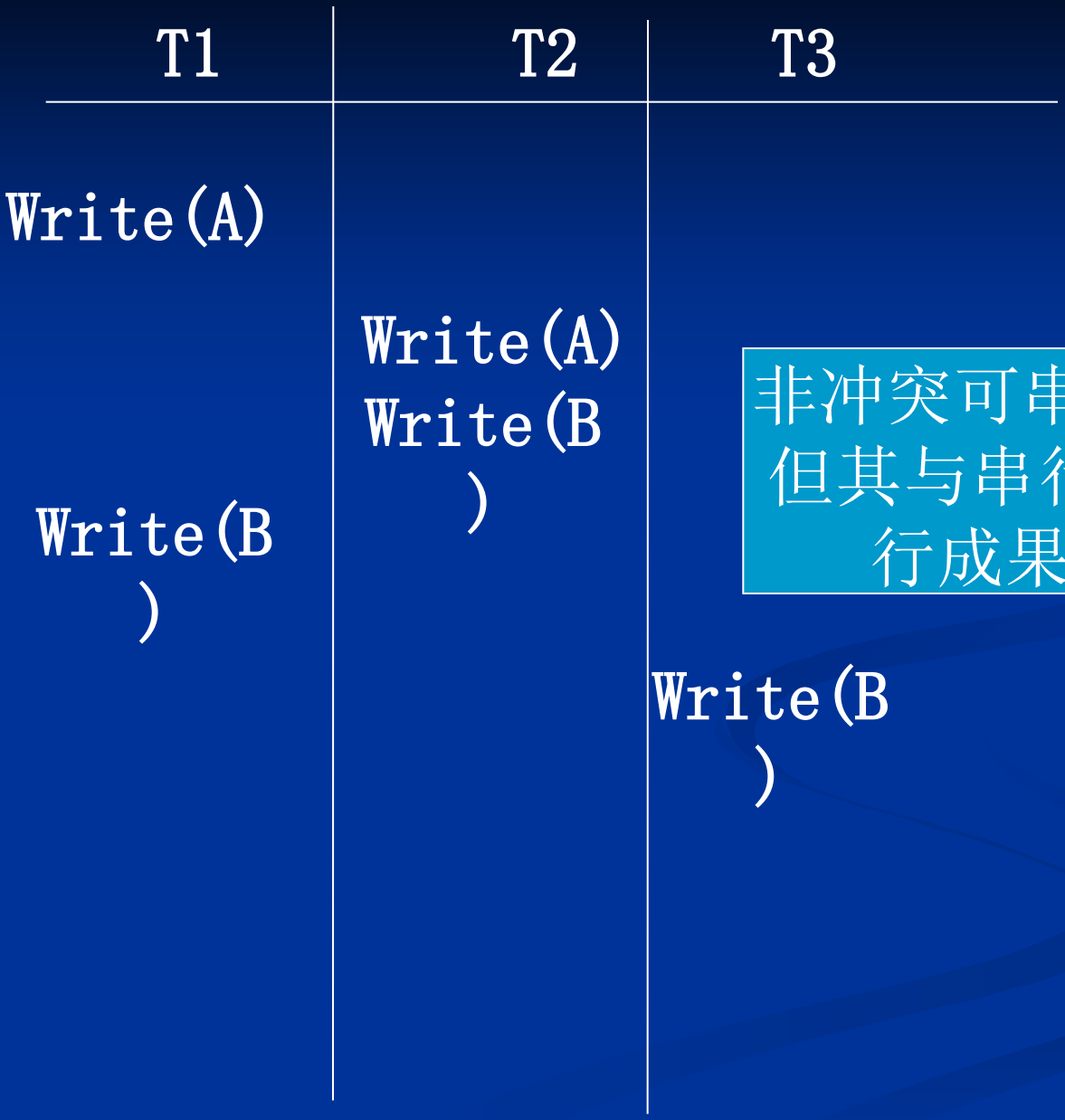
3. 冲突可串行性不是可串行性的必要条件。即，可串行性不一定是冲突可串行性。存在其他可串行调度策略！（如，视图可串行性调度）。

| T1 | T2 |
| --- | --- |
| Read(A) | |
| A=A-50 | |
| Write(A) | |
| | Read(B) |
| | B=B-10 |
| | Write(B) |
| Read(B) | |
| B=B+50 | |
| Write(B) | |
| | Read(A) |
| | A=A+10 |
| | Write(A) |

非冲突可串行的调度，但其与串行调度的执行成果相同

串行调度：

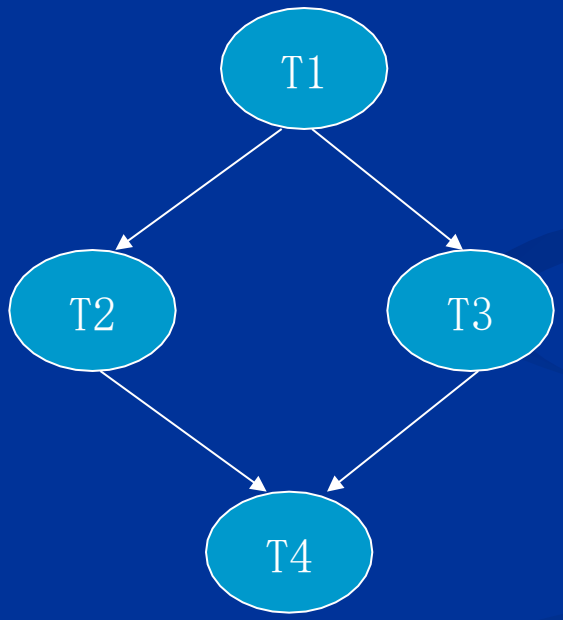


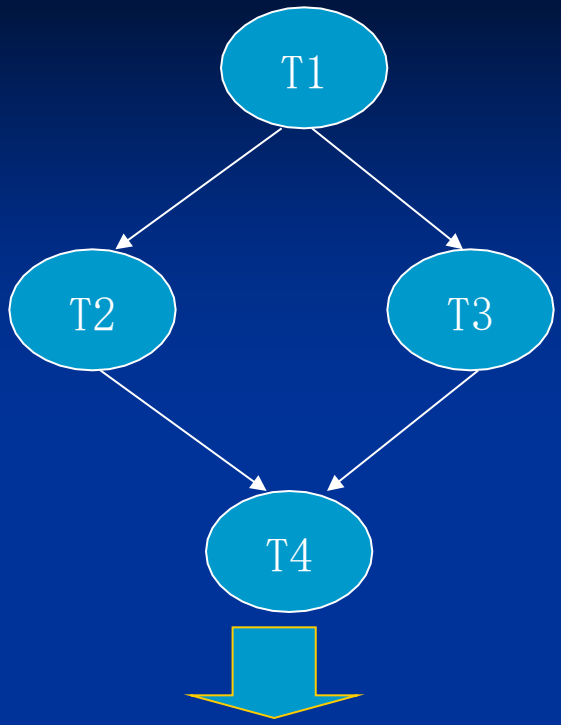

T1
T2
T3
Write(A)
Write(A)
Write(B)
Write(B)
非冲突可串行的调度，但其与串行调度的执行成果相同！

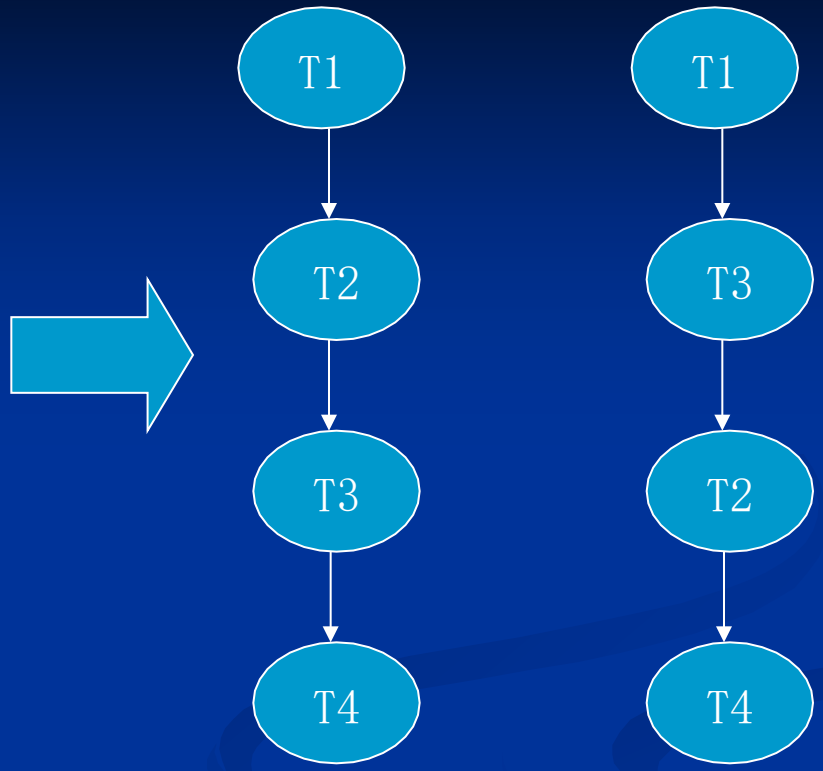
## 冲突可串行性的判断：优先图

优先图由节点和有向弧两种元素构成，节点表达事务，有向弧体现事务的先后顺序。

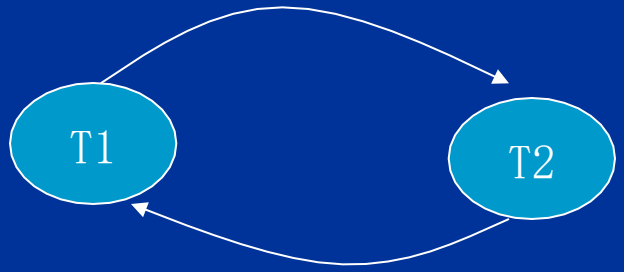
事务的先后顺序由冲突动作的先后顺序决定。






可串行化调度




不可串行化调度

# §2 基于锁的协议

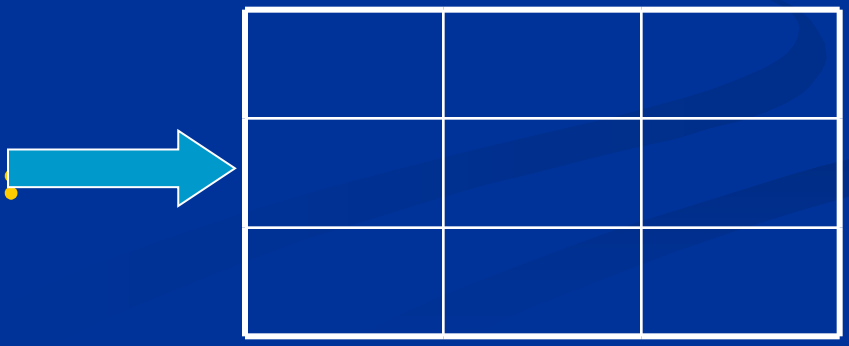
## （1）锁的基本模式：

- 共享锁（Shared---S锁）：允许执行读操作
- 排它锁（Exclusive---X锁）：允许执行读/写操作

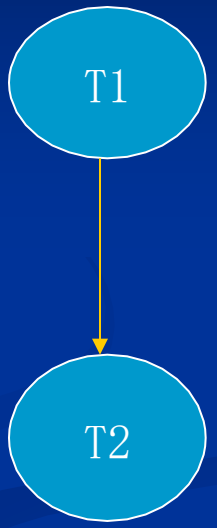
## （2）锁的调度策略：

- 解除一种数据对象的排它锁之前，其他事务不能对它加任何锁。
- 一种数据对象允许加几种共享锁，但不能在共享锁之上，加排它锁。

## （3）锁的相容矩阵：



| T1 | T2 |
| --- | --- |
| Lock-X(B) | |
| Read(B) | |
| B=B-50 | |
| Write(B) | |
| Lock-X(A) | |
| Read(A) | Lock-S(A) |
| A=A+50 | Read(A) |
| Write(A) | Lock-S(B) |
| Unlock(B) | Read(B) |
| unlock(A) | Display(A+B) |
| | unlock(A) |
| | Unlock(B) |

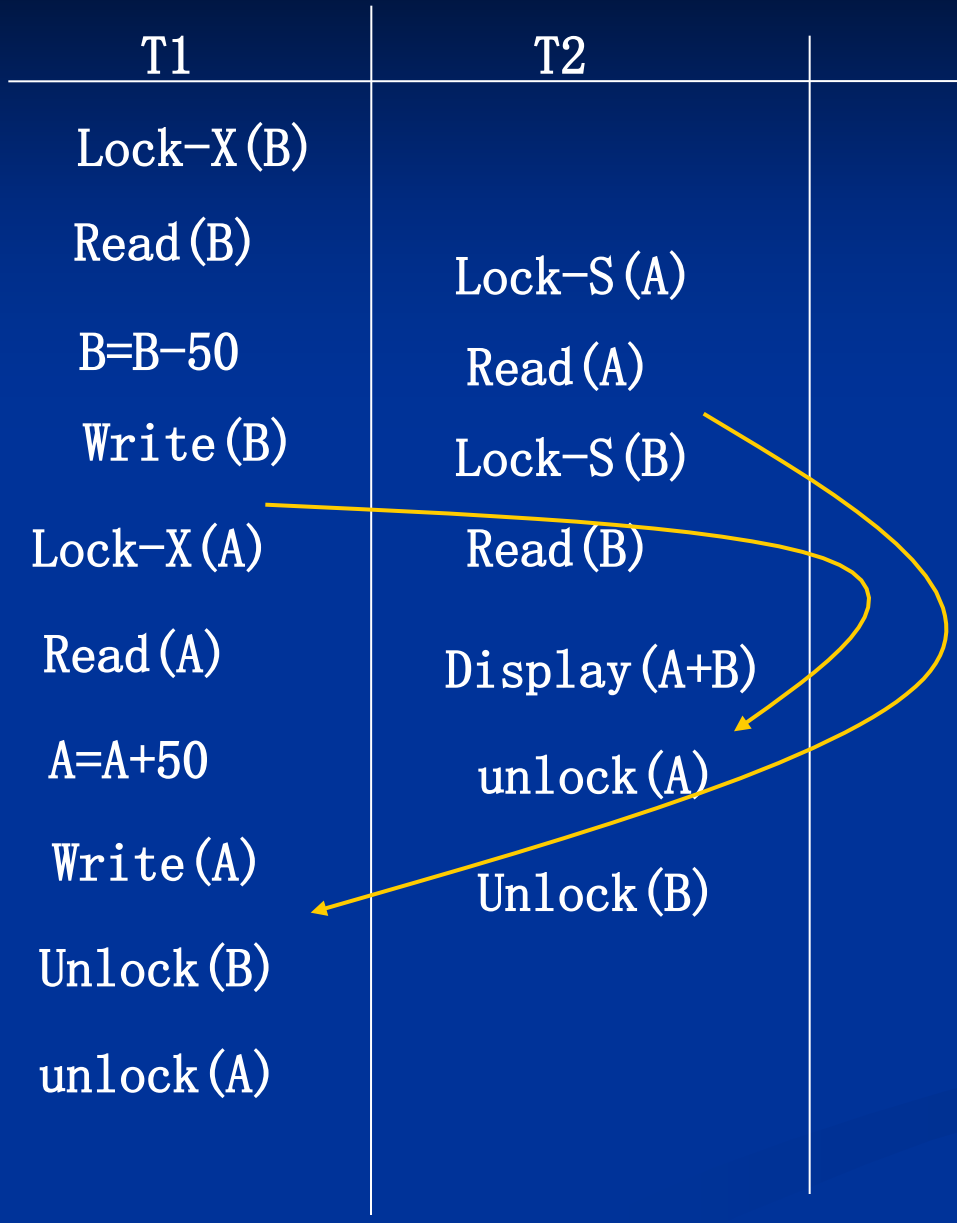


# （4）加锁产生的问题：

## ①不正确加锁产生错误成果:B=200, A=100

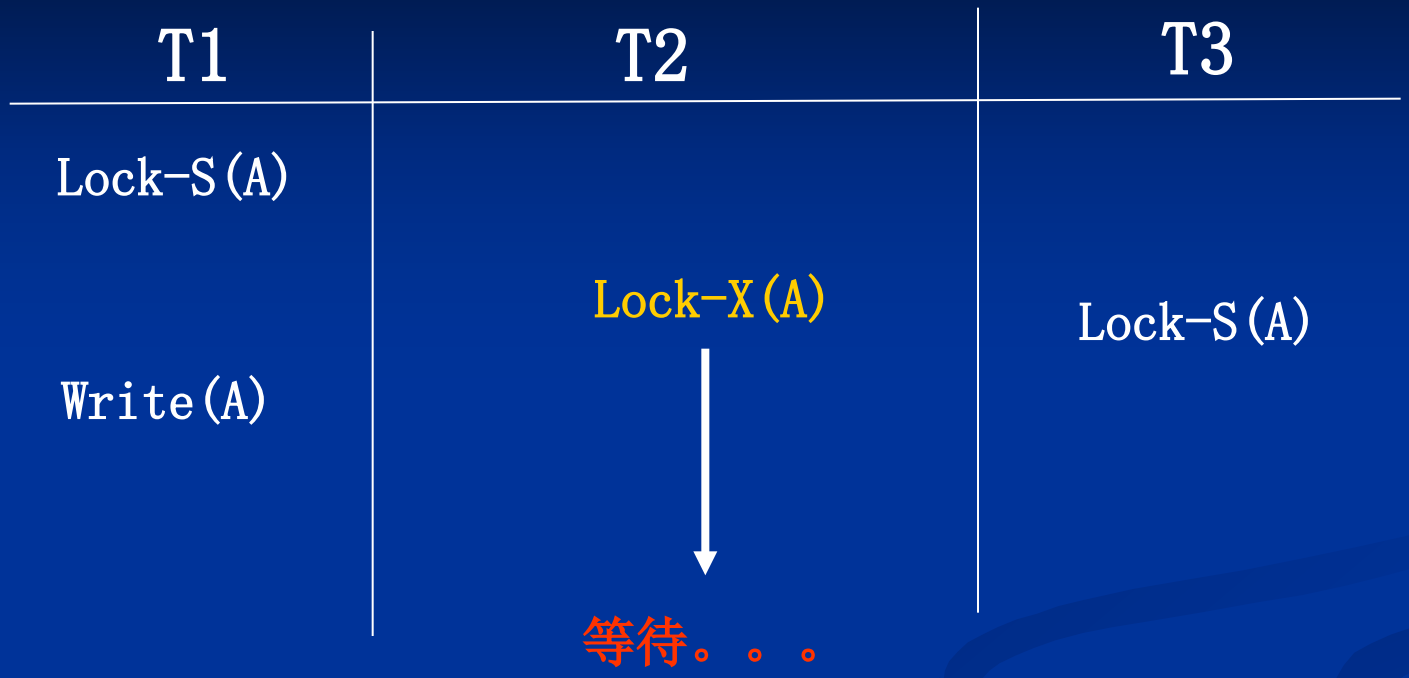
| T1 | T2 | 管理器 |
| --- | --- | --- |
| Lock-X(B) | | grant-X(B, T1) |
| Read(B) | | |
| B=B-50 | | |
| Write(B) | | |
| Unlock(B) | | |
| | Lock-S(A) | grant-S(A, T2) |
| | Read(A) | |
| | unlock(A) | |
| | Lock-S(B) | grant-S(B, T2) |
| | Read(B) | |
| | Unlock(B) | |
| | Display(A+B) | |
| Lock-X(A) | | grant-X(A, T1) |
| Read(A) | | |
| A=A+50 | | |
| Write(A) | | |
| unlock(A) | | |

# ②产生死锁现象：




两个事务相互等待数据，进入死锁状态！

# ③产生饿死现象：

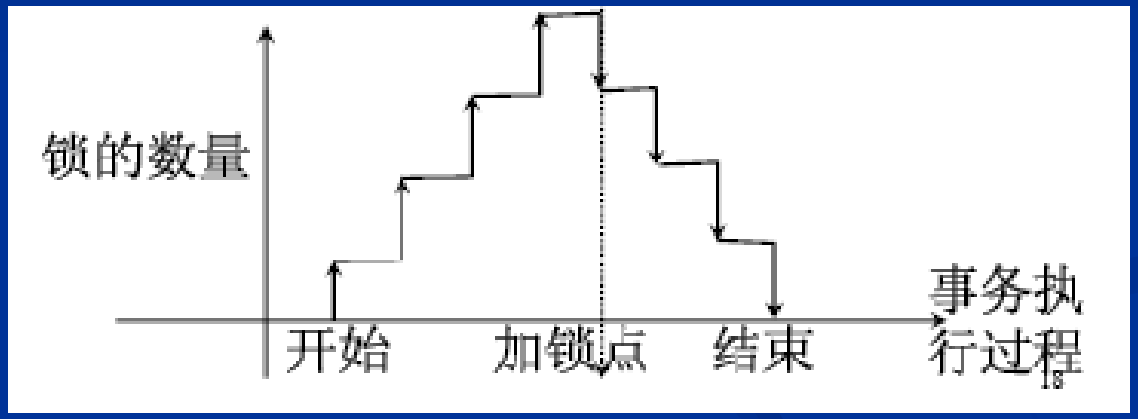



饿死现象经过下列条件来防止：

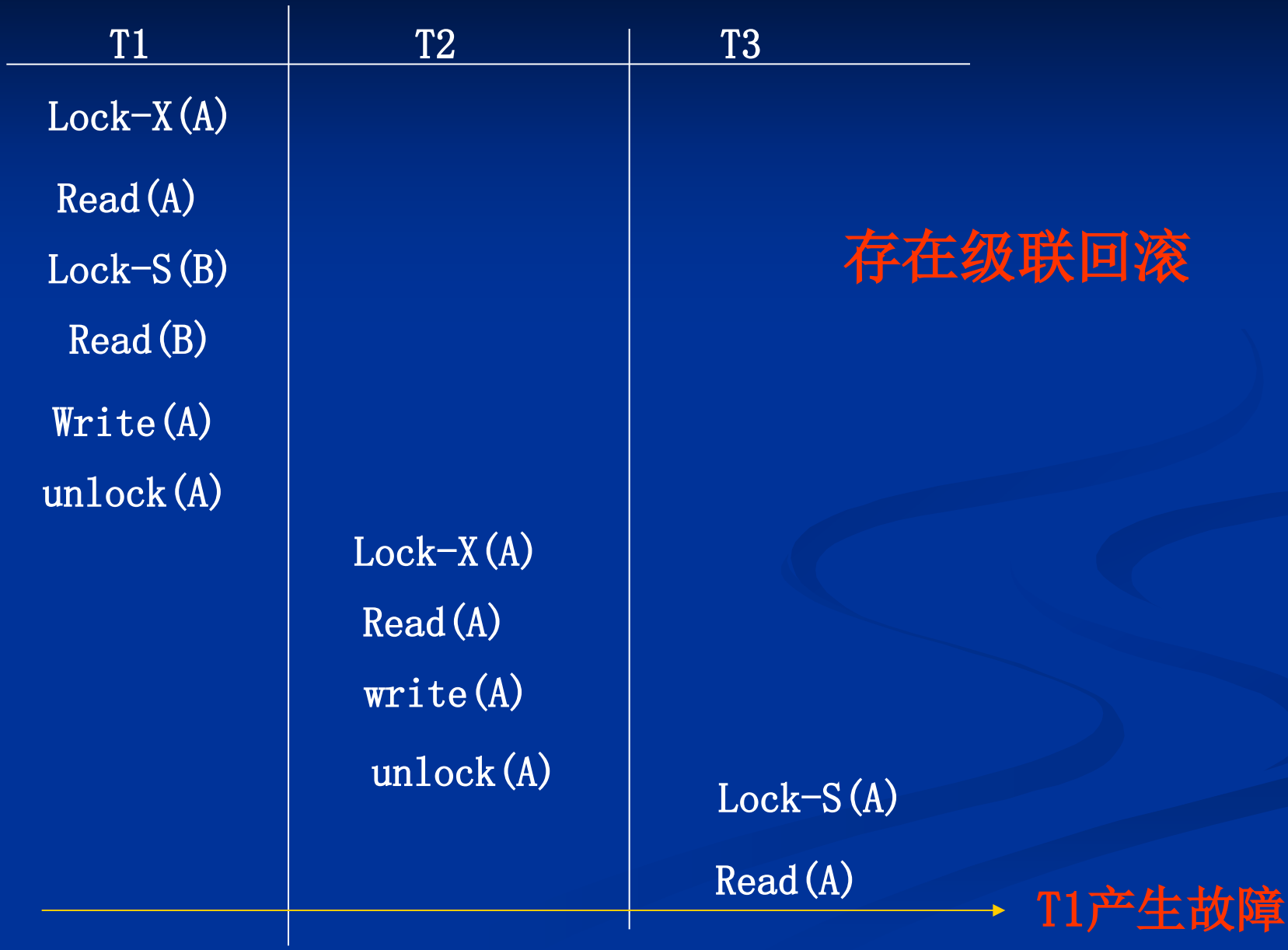
- 不存在在数据项Q上持有与M型锁冲突的锁的其他事务
- 不存在等待对数据Q加锁且先于T申请加锁的事务

## （5）两阶段锁协议：

全部事务分两个阶段提出加锁和解锁祈求：
增长阶段：事务能够取得锁，但不能解锁
缩减阶段：事务解锁，但不能取得锁




遵照了两阶段锁协议的并发控制算法所产生的调度是冲突可串行调度。即处理了第一种问题，但依然存在死锁现象，级联回滚可能发生。


T1
T2
T3
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)
存在级联回滚
T1产生故障

# （6）严格两阶段锁协议：防止级联回滚

| T1 | T2 | T3 |
| --- | --- | --- |
| Lock-X(A) | | |
| Read(A) | | |
| Lock-S(B) | | |
| Read(B) | | |
| Write(A) | | |
| unlock(A) | | |
| commit | | |
| | Lock-X(A) | |
| | Read(A) | |
| | write(A) | |
| | unlock(A) | |
| | commit | |
| | | Lock-S(A) |
| | | Read(A) |

要求：事务持有的排它锁必须在事务提交后方可释放。

强两阶段锁协议：要求事务提交之前不得释放任何锁。大部分数据库系统采用严格两阶段锁协议或强两阶段锁协议。

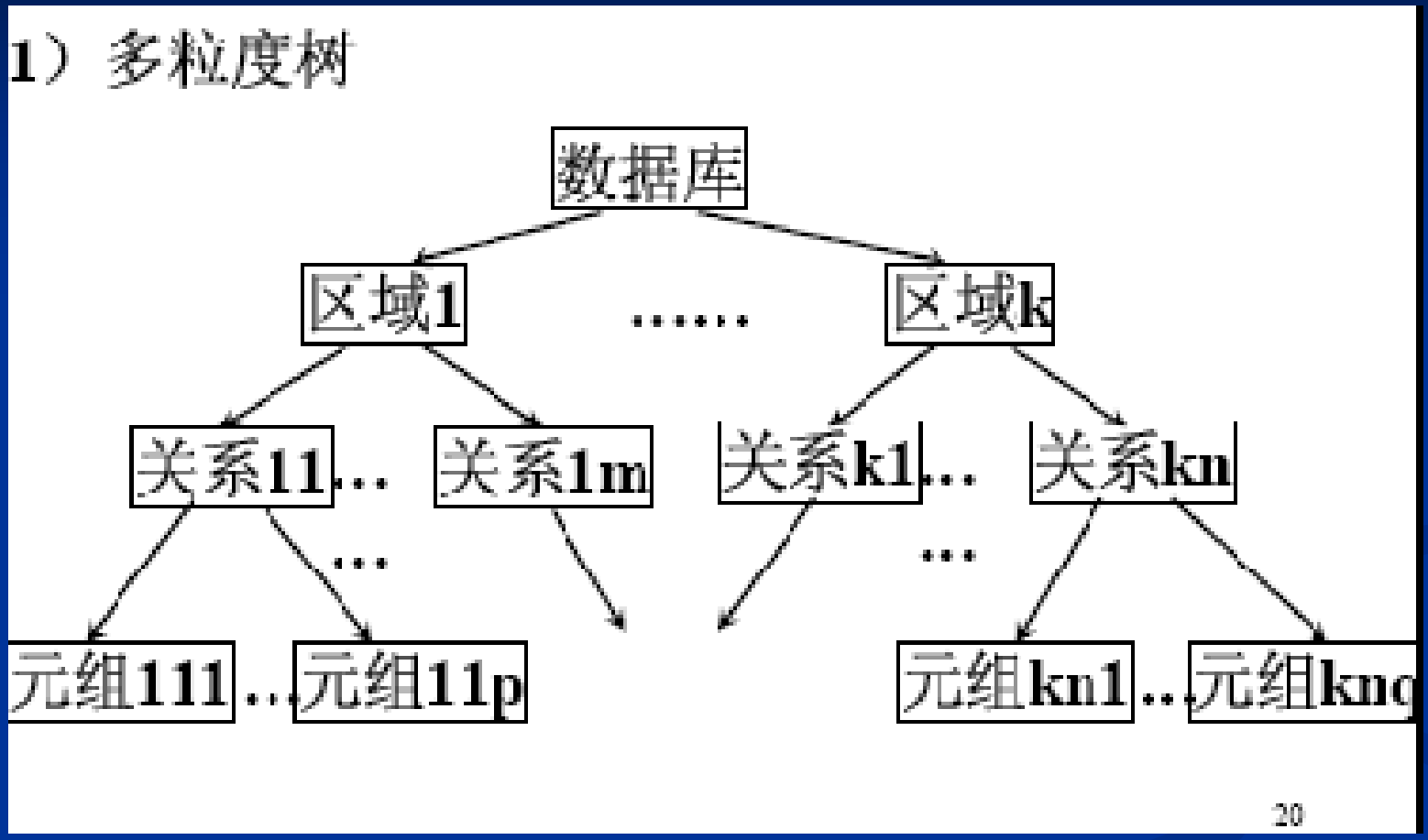
但这两种协议并行程度较低，有时为提升并行能力，采用锁转换机制，在写的时候将S锁转换成X锁，写完后再降为S锁。

```
T1:
Read(a1)
Read(a2)
Read(a3)
…
Write(a1)
```

```
T2:
Read(a1)
Read(a2)
display(a1+a2)
```

| T1 | T2 | |
|---|---|---|
| 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(an) | | |
| upgrade(a1) | | |

# （7）多粒度锁及意向锁：


1）多粒度树
数据库
区域1
……
区域k
关系11
关系1m
关系k1
关系km
元组111
元组11p
元组kn1
元组knq
20

多粒度加锁的特点：

- 显式加锁：树上每个结点都能够单独加锁。
- 隐式加锁：对目前结点加锁会造成隐式地对全部后裔结点加上同类型的锁。
- 检验锁冲突时，必须检验祖先、后裔结点！

对锁类型进行扩充：意向锁：
一种事务对一种数据对象显式加锁之前，必须对它的全部祖先结点加意向锁。

∴ 假如一种结点上有意向锁，则它的后裔结点必有被显式加锁。