第6章 事务管理

清华大学计算机系 李国良

liguoliang@tsinghua.edu.cn
http://dbgroup.cs.tsinghua.edu.cn/ligl

目录

01 事务概览及其概念

02 事务的特性

03 可串行化调度

04 事务的隔离级别

事务概览及其概念

- **事务的定义：事务是多个数据库操作组合成的一个不可分割的、同时成功或失败的工作单元。**
  - 如下为一转账事务

```
START TRANSACTION;
UPDATE account SET balance = balance + 100.00 WHERE name = '小王';
UPDATE account SET balance = balance - 100.00 WHERE name = '小明';
COMMIT;
```

  - START TRANSACTION 表示事务开始
  - COMMIT 表示事务提交

事务概览及其概念

- **事务的定义：事务是多个数据库操作组合成的一个不可分割的、同时成功或失败的工作单元。**
  - 如下为一转账事务
  - 事务中途也可以主动撤销该事务

```
START TRANSACTION;
UPDATE account SET balance = balance + 100.00 WHERE name = '小王';
UPDATE account SET balance = balance - 100.00 WHERE name = '小明';
ROLLBACK;
```

  - ROLLBACK 表示事务回滚，撤销该事务
    - 也有数据库使用ABORT，其功能和ROLLBACK一致

目录

01 事务概览及其概念

02 事务的特性

03 可串行化调度

04 事务的隔离级别

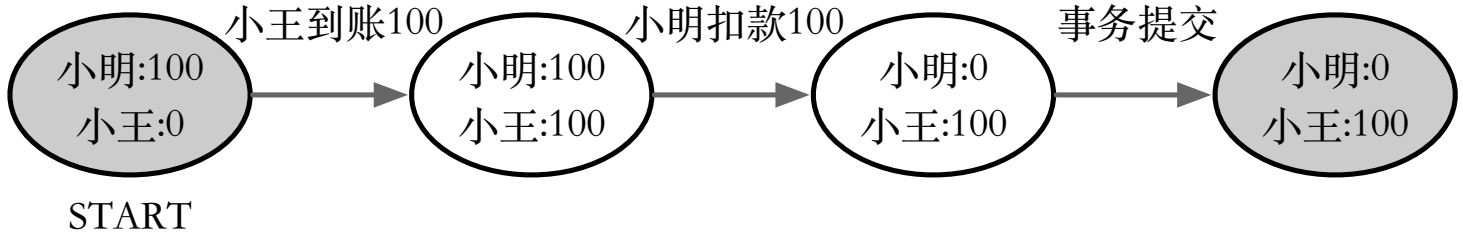
事务的特性

□ 数据库事务包含ACID四大特性。

- 原子性（atomicity）：**不可分割**，事务单元全部成功或者全部失败
- 一致性（consistency）：**正确一致**，一个正确（一致）的状态转移到另一个正确的状态
- 隔离性（isolation）：**互不干扰**，多个事务在并发执行的过程中所得到的结果，和串行执行得到的结果是一致
- 持久性（durability）：**永久保持**，执行结果不会丢失

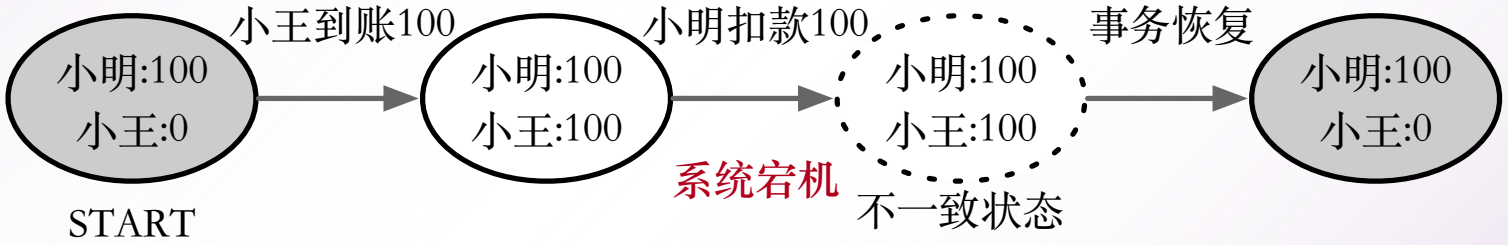
原子性

- **事务的原子性规定了事务是一个不可再分的基本操作单元**
  - “事务全部成功或者失败”就是事务原子性的要求。
  - “不可再分”意味着不管是单条语句还是多条语句组合的事务，所有语句需要同时生效，或者同时被数据库拒绝。
  - 数据库在事务的执行过程中发生故障而无法完成事务，需要撤销已经完成的部分语句，使数据库不会处于只有部分语句执行成功的状态。
- **以之前的转账操作为例**

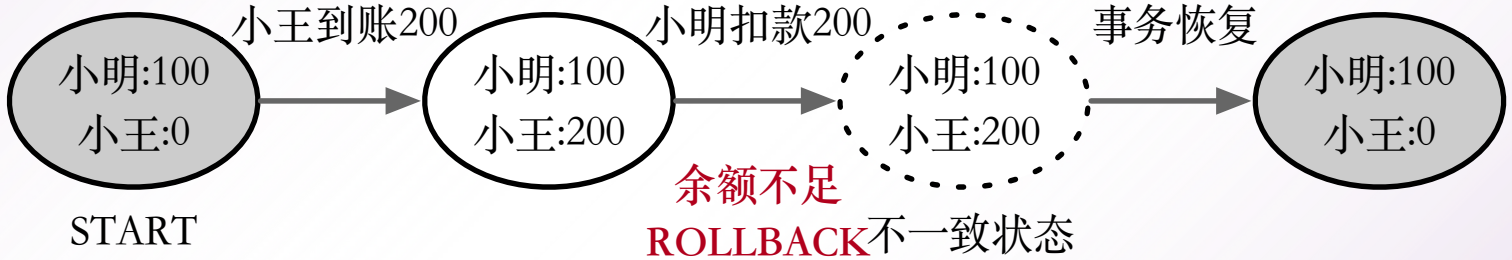


原子性

- 给小明扣款100元时发生系统故障，该操作不成功，需要恢复所有的数据。

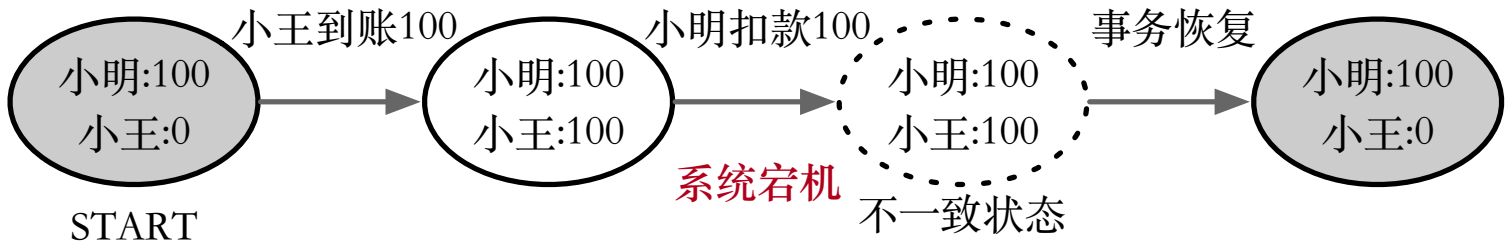


- 给小明扣款200元时余额不足，主动回滚（ROLLBACK），需要恢复小王到账的200元



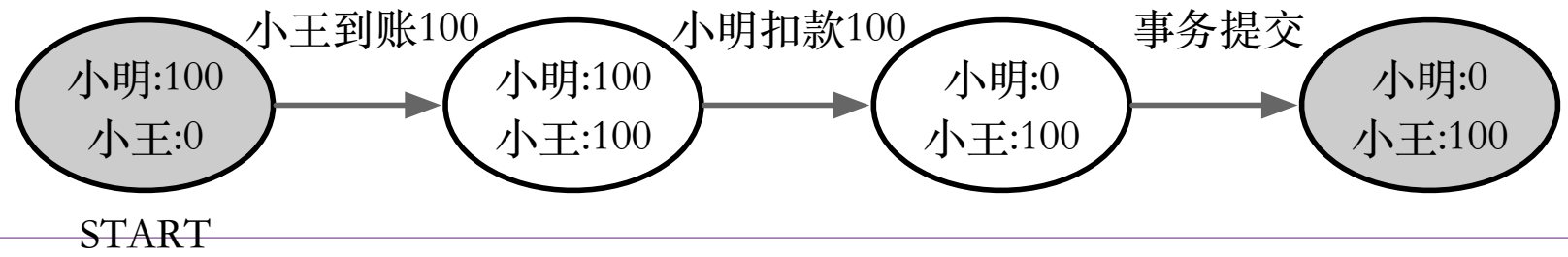
一致性

- **事务的执行会改变系统状态，一致性是指事务的执行会保证数据库从一个正确（一致）的状态转移到另一个正确的状态。**
  - 一致状态：比如数据库可以存储负数，但银行业务中用户的存款不能出现负数，存在负存款账户的数据库状态是不正确的状态。
  - 下图中可以设置检查条件“小明和小王的存款和为100元”来对转账事务进行一致性的验证，宕机后的状态不满足一致性。



持久性

- **持久性是指，一旦事务完成提交，即使数据库发生故障，该事务的执行结果也不会丢失，可以被正确地恢复，仍然对后续事务可见 。**
  - 下图描述的转账事务中，一旦转账事务提交，小明的100元就会永远地转给小王
  - 即使后续系统发生故障，该100元也不会丢失，也不会转移回小明的账户上。

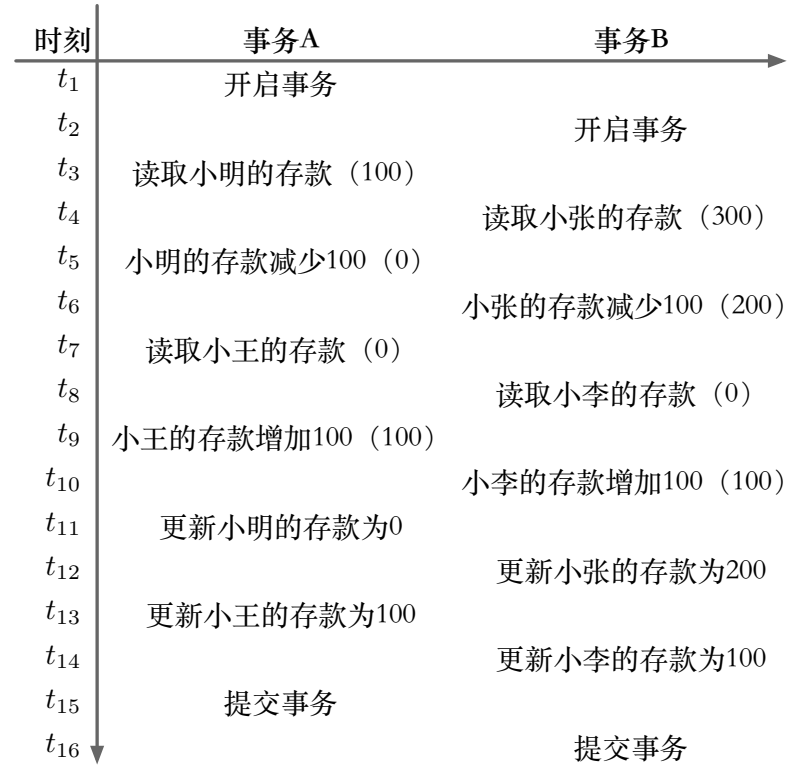


隔离性

- **事务的隔离性是指多个事务在并发执行的过程中所得到的结果，和串行执行得到的结果是一致的，保证了事务并发处理中事务的正确性**
- **并发控制是实现事务隔离性的手段**
  - 串行处理不存在事务之间的相互干扰，但是不能充分利用计算资源，效率低下。
  - 并行处理可以提升事务并发性，但是要解决冲突问题

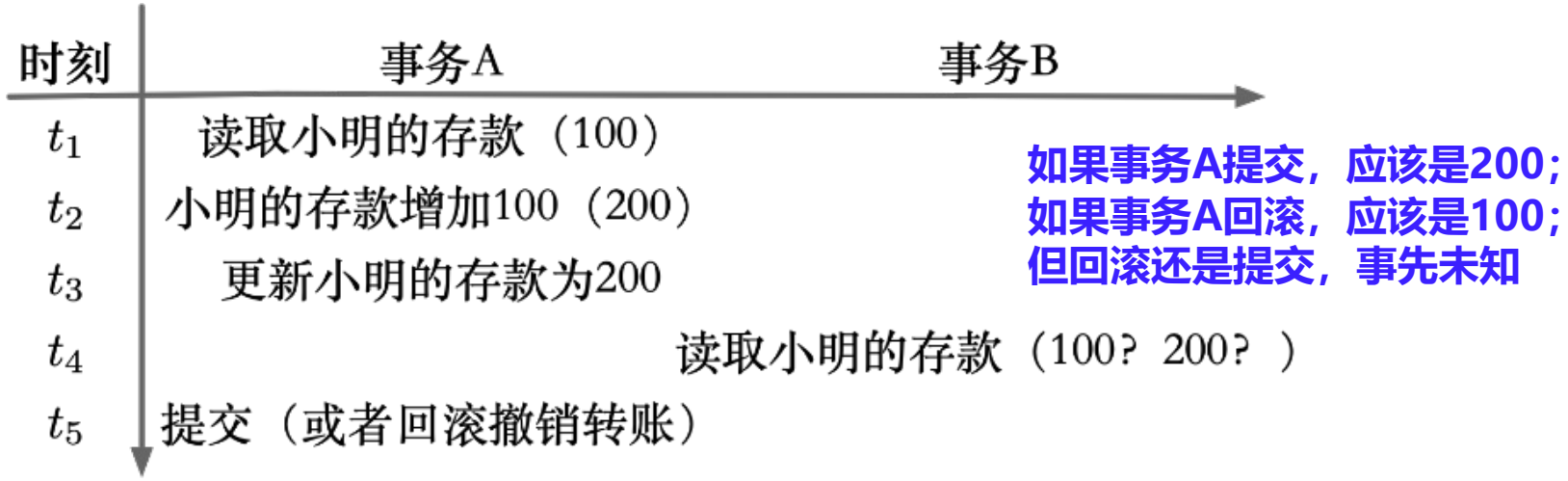
隔离性

- **考虑银行场景中如下两个事务，事务A先被请求：**
  - 事务A，小明给小王转账100元。
  - 事务B，小张给小李转账100元。
- **由于事务的每条操作所需的执行时间不同，用时刻表示一个事务操作的顺序。**
  - 事务B时刻t4执行了一个读取小张存款的操作，该操作需要从磁盘读取数据，阻塞了事务的执行，使CPU在时刻t5处于空闲状态。
  - 在时刻t5执行事务A的小明存款减少100元的操作。同时执行事务A和事务B可以利用闲置资源，提高计算和存储资源的利用效率。
  - **事务A和事务B涉及不同的储户资源，没有互相的干扰。可以并发执行。**



隔离性：写读冲突

- **银行场景中新来了两个事务，事务A先被请求：**
  - 事务A，小明给账户存款100元。
  - 事务B，家长查询小明存款。
- **事务B与事务A在并发过程中存在写读冲突：导致脏读（dirty read）**
  - 事务A更新了小明的余额数据，使其从100元变成200元，但是事务A还未提交。
  - 此时事务B要读取小明的余额数据，数据库应该读取原先的数据（100），还是事务A修改但是未提交的数据（200）？

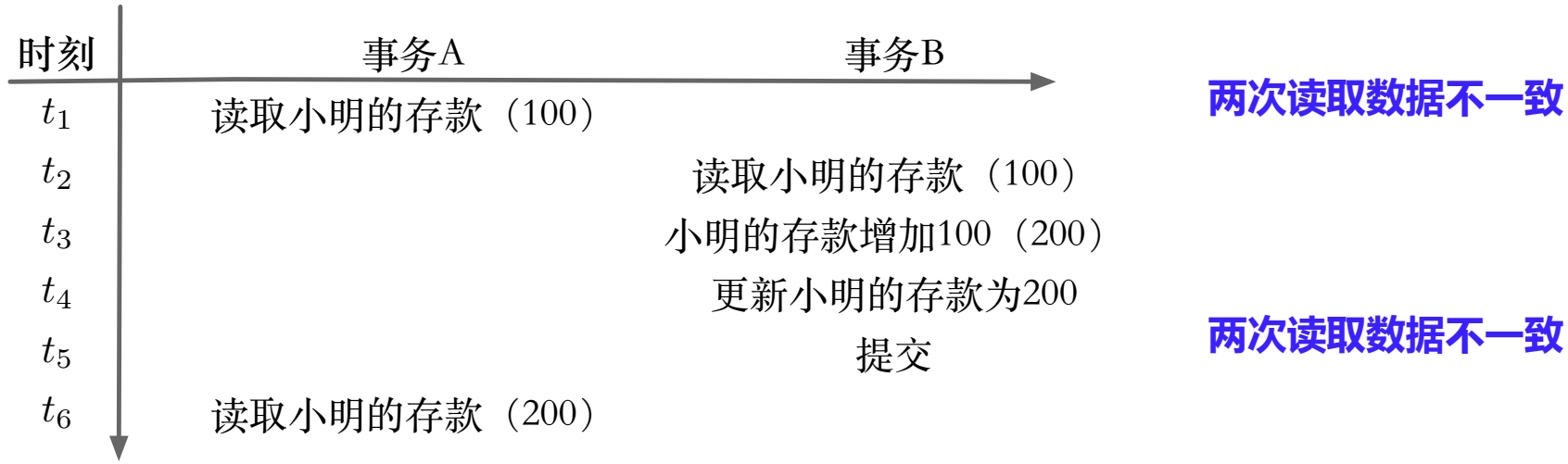


如果事务A提交，应该是200；
如果事务A回滚，应该是100；
但回滚还是提交，事先未知

隔离性：读写冲突

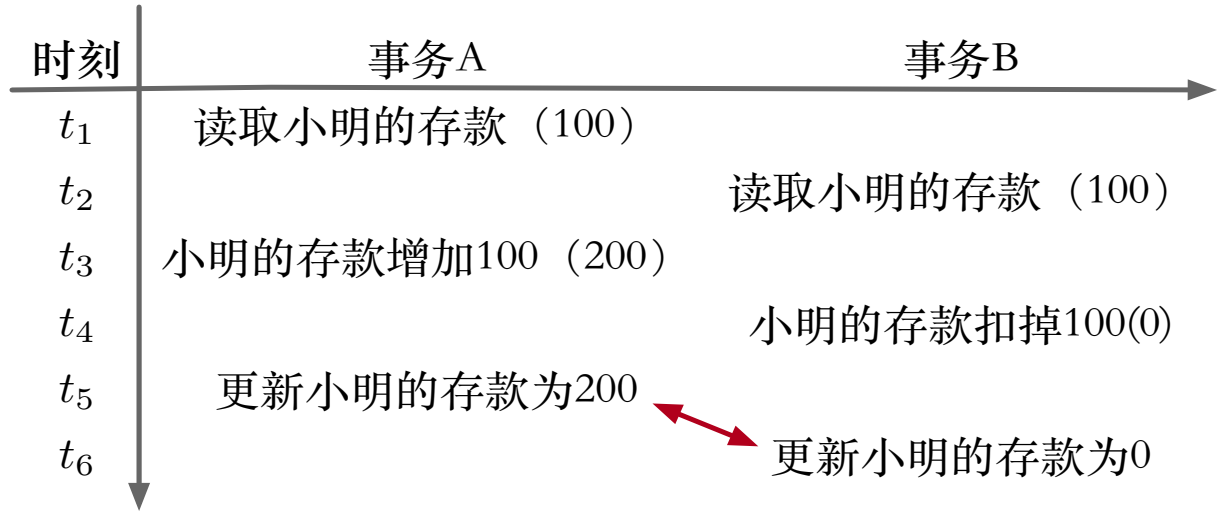
- **银行场景中新来了两个事务，事务A先被请求：**
  - 事务A，小明两次查询账户存款。
  - 事务B，小明家长给小明存款100元。

- **事务A与事务B在并发过程中存在读写冲突：导致不可重复度（unrepeatable read）**
  - 事务A两次读取数据不一致。
  - 事务B中间修改了数据造成其他事务的不可重复读。



隔离性：写写冲突

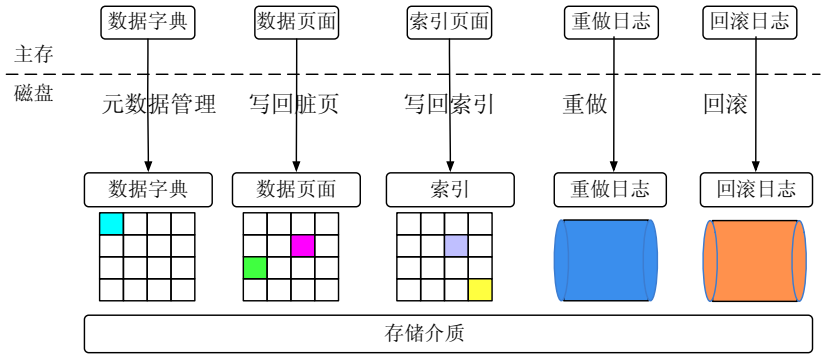
- **银行场景中新来了两个事务，事务A先被请求：**
  - 事务A，小明发工资100元。
  - 事务B，家长用小明银行卡消费了100元。
- **事务B与事务A在并发过程中存在写写冲突：导致脏写（dirty write）**
  - 两个事务串行执行，正确的结果是小明的存款余额为100元。
  - 事务A尝试更新余额为200元
  - 事务B尝试更新余额为0元

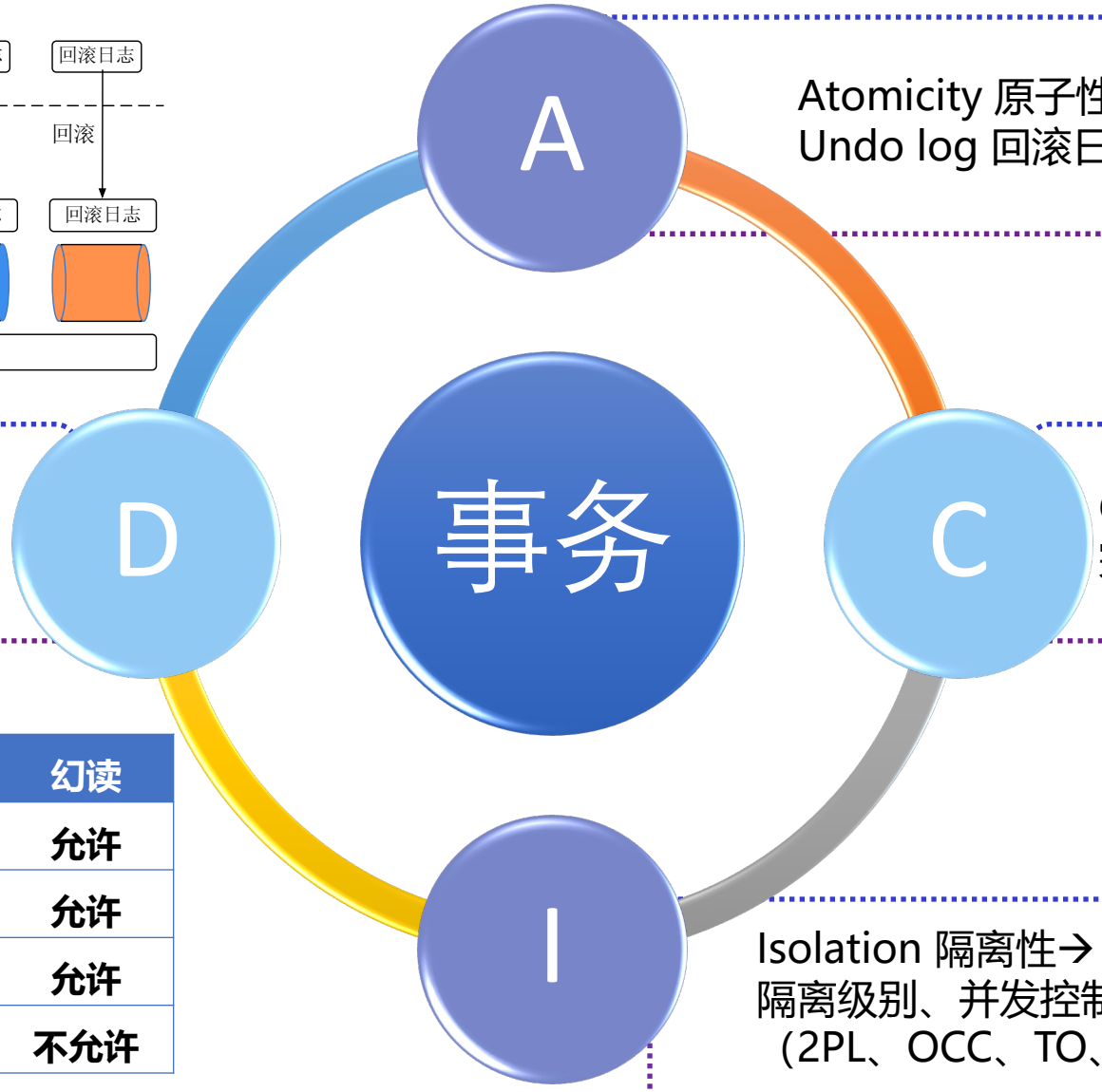


隔离性

- **事务的隔离性是指多个事务在并发执行的过程中所得到的结果，和某种串行执行得到的结果是一致的，保证了事务并发处理中事务的正确性。**
- **并发控制是实现事务隔离性的手段。**
  - 并发控制的目标和基本实现原理
    - 串行调度（serial schedule）：慢、资源浪费
    - 可串行化调度（serializability）：快、并发调度，但需要避免冲突
      - 终态可串行化调度（final state serializability）
      - 冲突可串行化调度（conflict serializability）
      - 视图可串行化调度（view serializability）
  - 事务的四大隔离级别实现机制→第八章并发控制的具体算法

事务特性总结





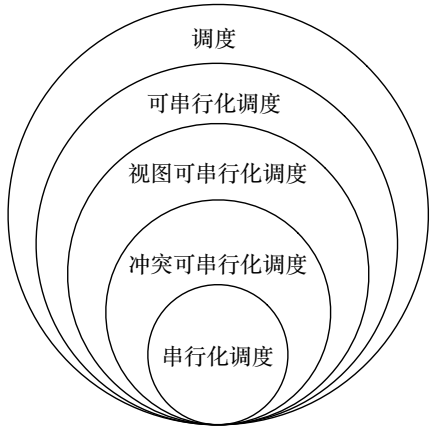
Atomicity 原子性→
Undo log 回滚日志

Consistency 一致性→
完整性约束+AID技术

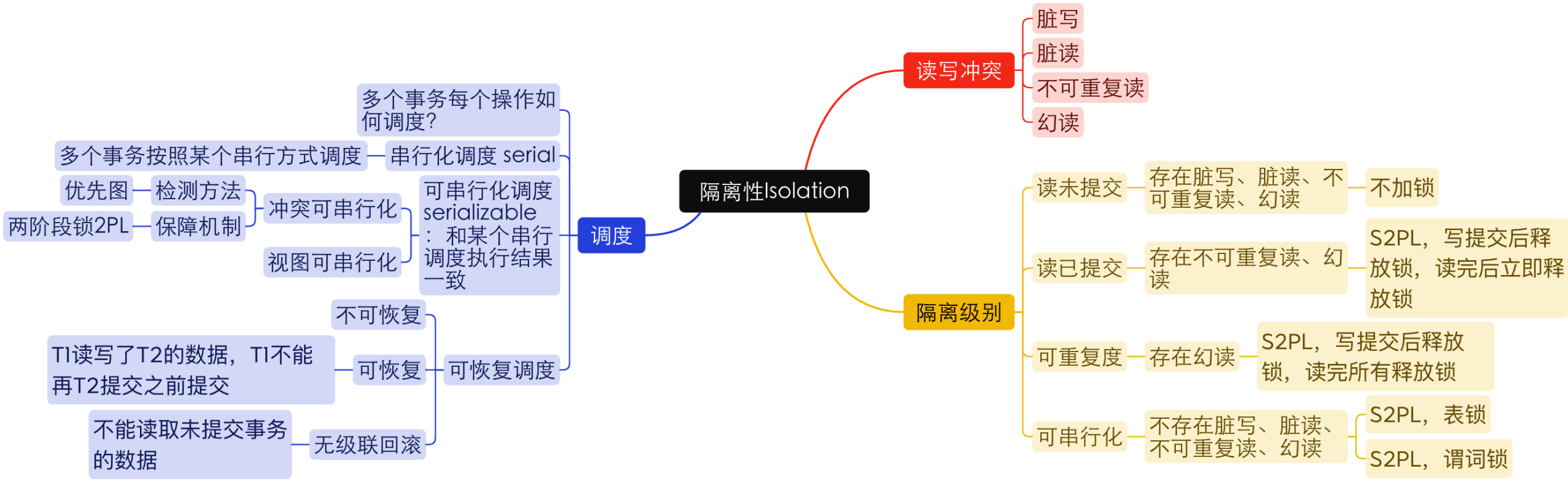
Isolation 隔离性→
隔离级别、并发控制
（2PL、OCC、TO、MVCC）

Durability 持久性→
Redo log 重做日志

隔离级别	脏读	不可重复读	幻读
读未提交	允许	允许	允许
读已提交	不允许	允许	允许
可重复读	不允许	不允许	允许
可串行化	不允许	不允许	不允许



隔离性


隔离性Isolation
读写冲突
脏写
脏读
不可重复读
幻读
隔离级别
读未提交
存在脏写、脏读、不可重复读、幻读
不加锁
读已提交
存在不可重复读、幻读
S2PL，写提交后释放锁，读完后立即释放锁
可重复度
存在幻读
S2PL，写提交后释放锁，读完所有释放锁
可串行化
不存在脏写、脏读、不可重复读、幻读
S2PL，表锁
S2PL，谓词锁
调度
多个事务每个操作如何调度?
串行化调度 serial
多个事务按照某个串行方式调度
可串行化调度 serializable：和某个串行调度执行结果一致
冲突可串行化
检测方法
优先图
保障机制
两阶段锁2PL
视图可串行化
可恢复调度
不可恢复
可恢复
T1读写了T2的数据，T1不能再T2提交之前提交
无级联回滚
不能读取未提交事务的数据

目录

01 事务概览及其概念

02 事务的特性

03 可串行化调度

04 事务的隔离级别

调度

- **为了更好地解释并发控制过程中，数据库对事务的处理流程，首先引入一个概念：调度。**
- **调度为事务的并发过程中，决定事务中每个操作的执行顺序。**
  - 读操作read(X)和写操作write(X)分别表示从数据库中读取记录X的数据和将数据写入记录X中。
  - 使用读操作和写操作来描述事务的执行过程。
  - 调度为读操作和写操作组成的序列。
    - 更新操作可以被替换为3个操作。

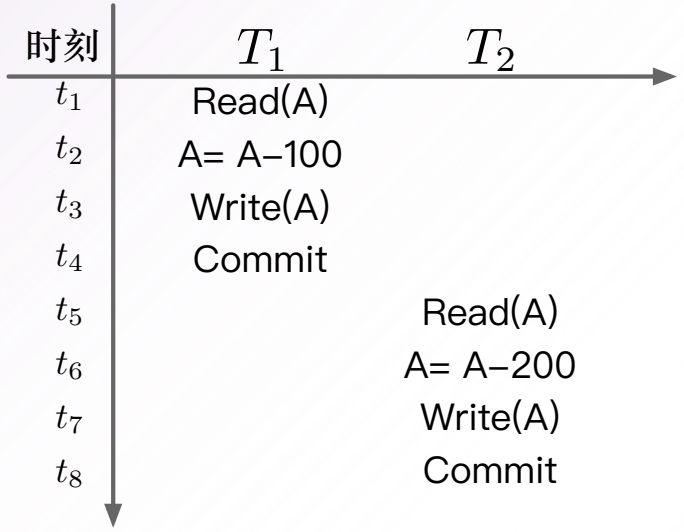
```
UPDATE account SET balance = balance - 100.00 WHERE name = '小明';
```

```
read(小明的余额）；
小明的余额 = 小明的余额 - 100;
write(小明的余额）；
```

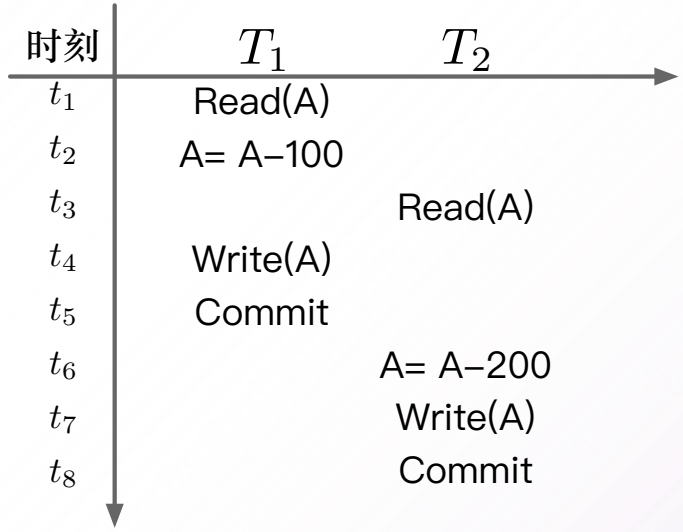
调度

考虑一个场景

- A表示小明的余额，初始为300元，T_1和T_2分别表示小明消费100元和消费200元的扣款事务。经过两次扣款之后小明的余额应该为0元。



(a) 调度1：串行调度



(b) 调度2:产生不一致结果

调度1是事务的串行执行，也被称为串行调度

调度2为一个并发调度，但是产生了不正确的结果

调度

- **串行调度(serial schedule): 事务串行执行**
- **调度定义下的并发控制:**
  - 给定一个并发调度S，存在一个串行调度S'， 在任何数据库状态下，**按照调度S和调度S'执行后所产生的结果都是相同的**。
  - 此时调度S被称之为**可串行化调度**（serializable schedule）。

- **可串行化调度的数量十分巨大，且难以校验，数据库中一般通过找到可串行化调度的子集（充分条件），**即找到能够提前确认是可串行调度的并发调度，进而提升调度效率。

调度

T1 begin
A=A+100
B=B-100
T1 commit

T2 begin
A=A*1.05
B=B*1.05
T2 commit

初始余额

A	B
$1000	$1000

串行调度 T_1,T_2:

T_1 A+=100 B-=100

T_2 A*=1.05 B*=1.05

串行化

A	B
$1155	$945

其他调度:

T_1 A += 100 B -= 100

T_2 A *=1.05 B*=1.05

非串行化

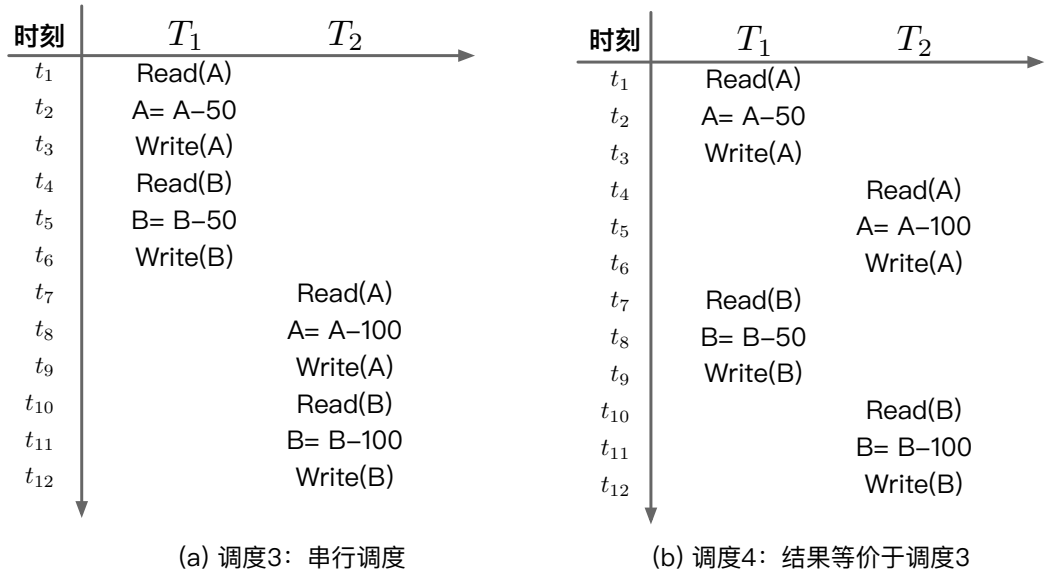
结果不一致

A	B
$1155	$950

可串行化调度

场景：

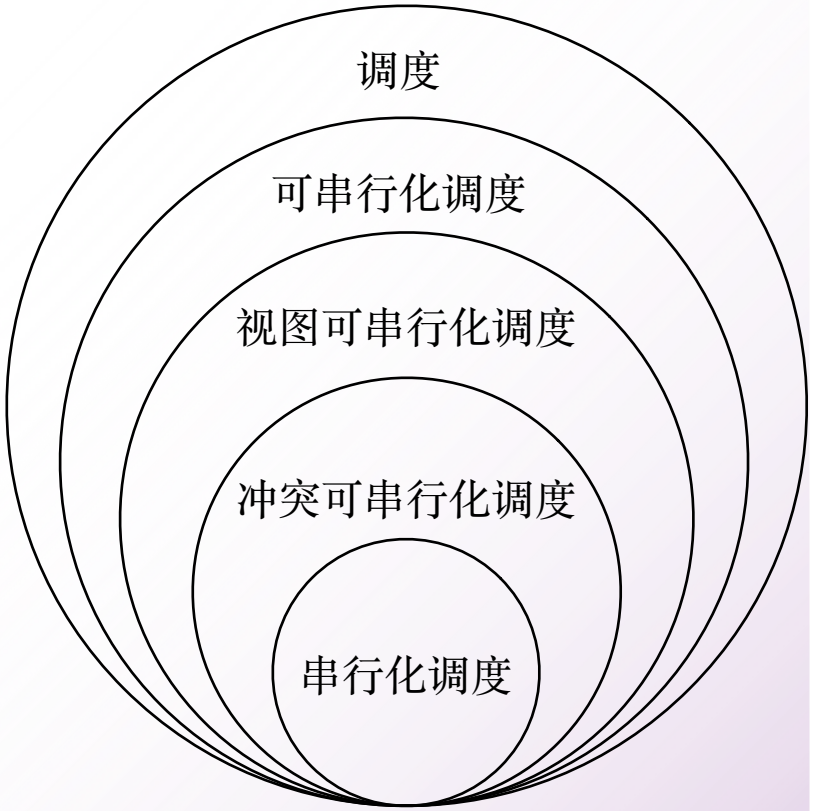
- 小明仍然在超市进行了两笔消费，分别为100元和200元。小明有超市的会员卡，可以抵扣一半的金额。
  - 小明的存款用A表示，初始为300。
  - 会员卡的数据用B表示，初始为300。
  - 两笔消费分别为A减去50，B减去50(事务T_1)；
  - A减去100，B减去100（事务T_2）。
- 调度3描述的是该事务的串行调度。
- 调度4则描述了一种可能的并发调度。
- 调度4在任何情况下与调度3都能产生相同的结果。是一种可串行化调度



(a) 调度3：串行调度　　(b) 调度4：结果等价于调度3

可串行化调度

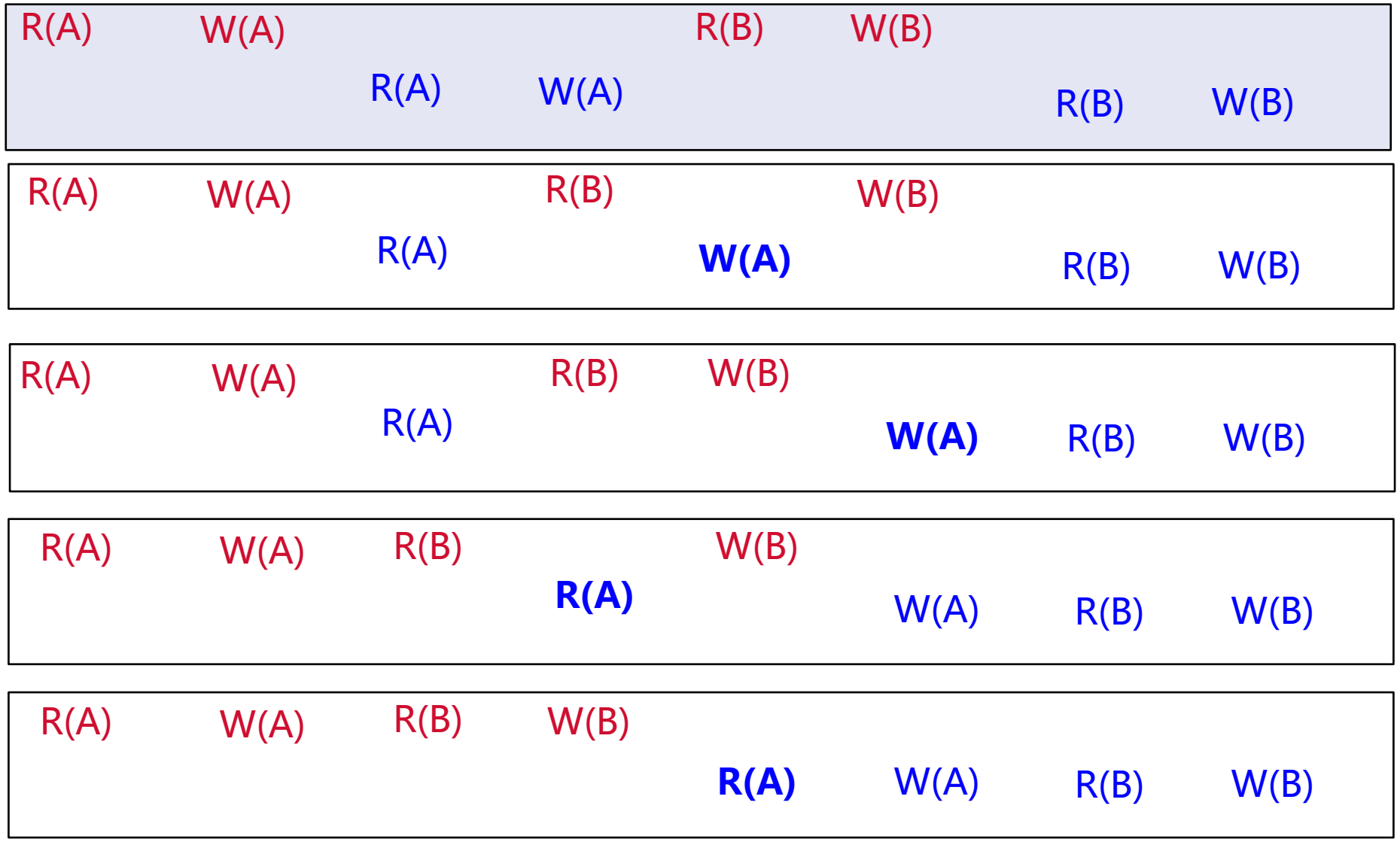
- 如何判断一个调度是否可串行化？如何实现可串行化调度？
- 验证代价较大，一般找一个容易验证的可串行化调度方案
  - 终态可串行化
    - 判读最终执行结果的状态
  - 冲突可串行化
    - 判断操作之间是否冲突
  - 视图可串行化
    - 判断操作流程是否一致
      - 初始化读的一致性
      - 读写顺序的一致性
      - 最终写的一致性



冲突可串行化调度

- **定义：冲突可串行化调度，是从冲突的角度出发，针对一个调度*S*去发现其等价的串行调度*S'*来确定*S*是一个可串行化调度。**
  - 一次操作交换定义为交换事务调度序列中相邻的两个操作，一个交换操作可以将一个调度A变成另外一个调度B。
  - 当交换不会影响两个调度的一致性时，定义该交换得到的两个调度是等价的，该交换为等价交换。
    - 等价操作
      - 交换连续两个相同数据读取操作的顺序:$R_{T1}(A)$ $R_{T2}(A)$
      - 交换连续两个不同数据读写操作的顺序:$R_{T1}(A)$ $W_{T2}(B)$；$W_{T1}(A)$ $R_{T2}(B)$；$W_{T1}(A)$ $W_{T2}(B)$
    - 非等价操作
      - 交换连续两个相同数据的读写、写写操作: $R_{T1}(A)$ $W_{T2}(A)$；$W_{T1}(A)$ $R_{T2}(A)$；$W_{T1}(A)$ $W_{T2}(A)$

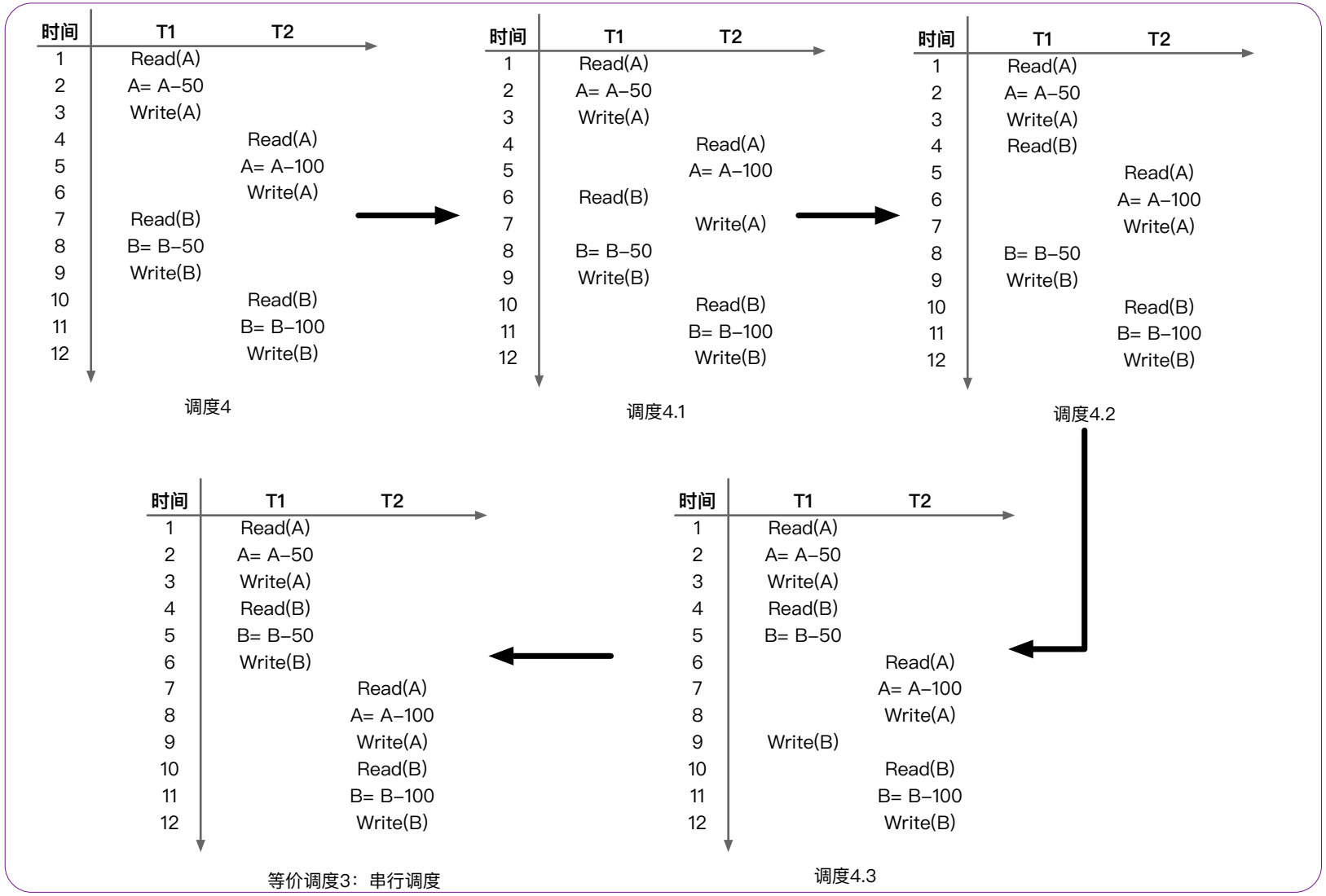
冲突可串行化



冲突可串行化调度

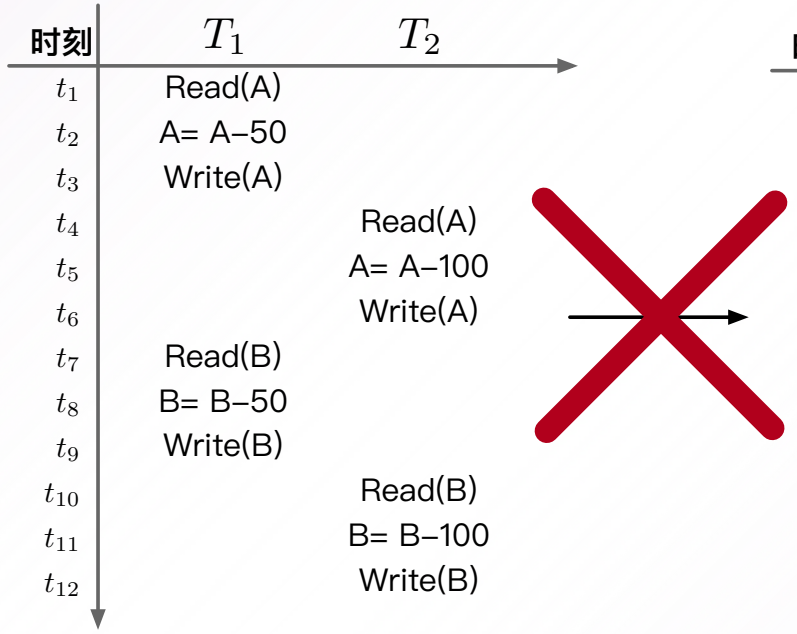
等价交换

- 交换时刻t_6和时刻t_7的write(A)和read(B)，两个操作不涉及同一个数据。
- 一直将调度中的read(B)换到T_1的操作write(A)之下。
- 通过持续地将T_1中的操作与T_2的操作向前交换， 调度4变成了调度3。
- 调度4是等价于串行调度3的可串行化调度。

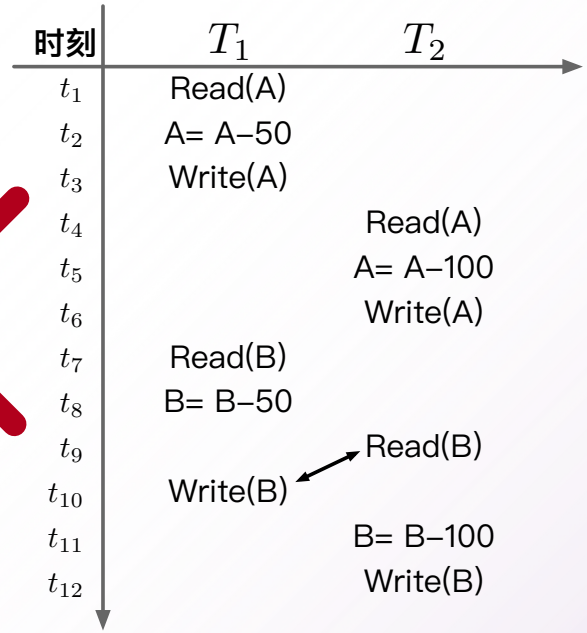


冲突可串行化调度

- **不等价交换，产生冲突。**
  - 交换调度4中t_9和t_{10}的两条调度write(B)和read(B)，导致T_2读取不到T_1写入B的值。
  - T_2会根据错误的B值再次写入错误的B值。
  - 冲突操作：交换后会导致错误的操作，T_1的write(B)和T_2的read(B)为冲突操作。



(a) 调度4：结果等价于调度3



(b) 调度4.4—个不等价的变换

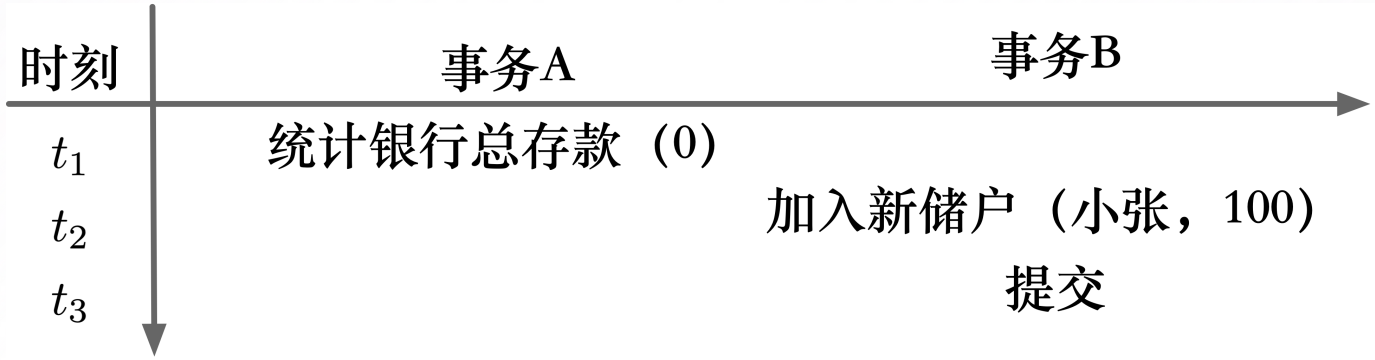
冲突可串行化调度的验证

- **当并发事务处理同一个数据X时，可能会发生三种冲突：**
  - （1）读写冲突（read(X)-write(X)，RW），表示调度中T_1先读取了数据X，然后T_2再写入数据X。T_1的 read(X) 和T_2的write(X)的先后顺序改变可能会改变数据库的结果（即T_1本来不应该读到T_1写入的数据X，但是交换后读到T_2写入的数据X）。
  - 以图中调度为例：

    (a) 事务A：统计银行的总存款为0元。

    (b) 事务B：加入新储户小张，存款为100元。

当t_1、t_2改变先后顺序时，事务A的统计的银行总存款数会发生变化。



冲突可串行化调度的验证

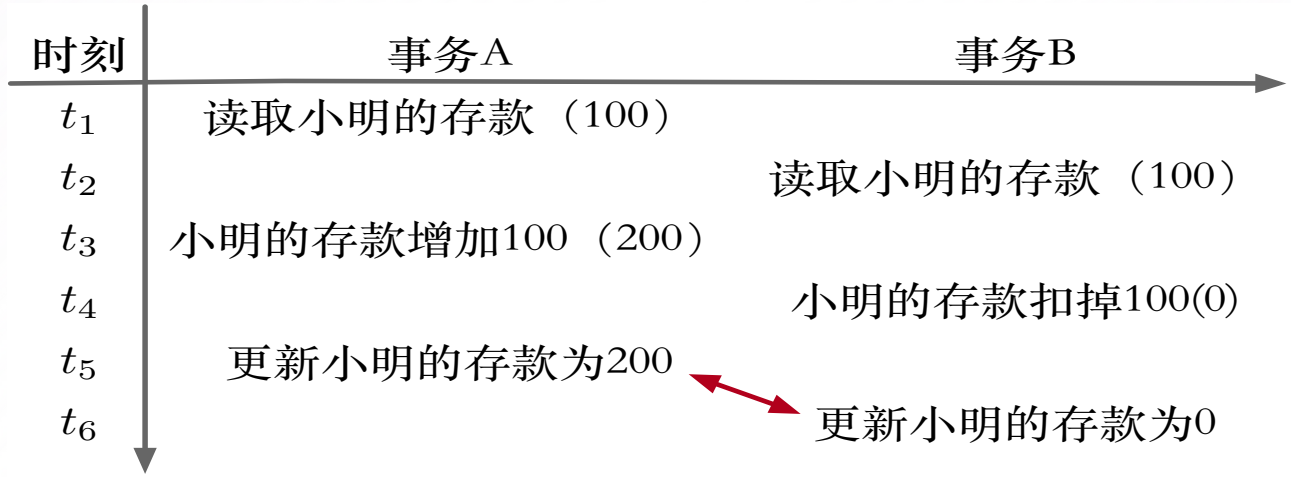
当并发事务处理同一个数据X时，可能会发生三种冲突：

- （2）写写冲突（write(X)-write(X)，WW），表示调度中T_1先写入了数据X，然后T_2再写入了数据X。T_1的write(X)和T_2的write(X)的先后顺序改变会改变数据库的结果（即本来是T_2写入了数据X，但是交换后是T_1写入了数据X）。
- 以图中调度为例：

  （a）事务A：小明发工资100元。

  （b）事务B：家长用小明银行卡消费了100元。

当交换t_5、t_6两个操作的写入顺序时，会影响小明的存款结果。



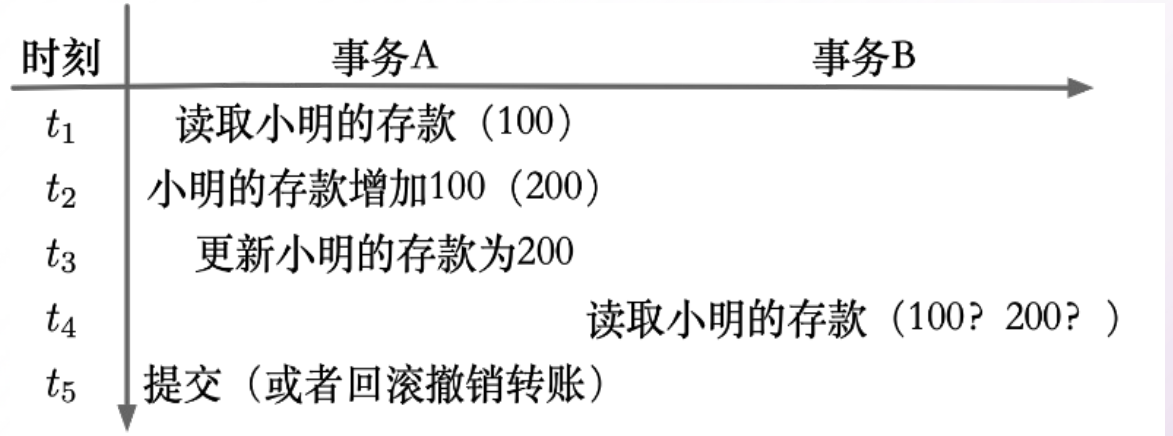
冲突可串行化调度的验证

□ 当并发事务处理同一个数据X时，可能会发生三种冲突：

- （3）写读冲突（write(X)-read(X)，WR），表示调度中T_1先写入了数据X，T_2再读取数据X。T_1的 write(X)和T_2的read(X)的先后顺序改变会影响数据库的结果（即T_2本来应该读到T_1写入的数据X，但是交换后T_2不能读到了T_1写入的数据X）。
- 以图中调度为例：

（a）事务A：小明给账户存款100。

（b）事务B：朋友查询小明存款。



时刻t_3事务A将小明的余额更新为200元

时刻t_4事务B读取了小明的存款

当t_3与t_4的操作交换时，事务B读取到的结果会发生改变。

冲突可串行化调度的验证

口 冲突可串行化调度验证的原则：

- 当并发的两个事务之间不存在冲突时，可以很容易的通过交换操作将当前调度S'转化成等价的串行调度。
- 对于存在冲突的并发事务调度S'，在保证冲突操作的次序不变的前提下，通过仅交换不冲突的操作而转化为等价的串行调度S时，称调度S'和调度S冲突等价，调度S'为冲突可串行化调度 。

时刻	T_1	T_2
t_1	Read(A)	
t_2	A= A–50	
t_3	Write(A)	
t_4	Read(B)	
t_5	B= B–50	
t_6	Write(B)	
t_7		Read(A)
t_8		A= A–100
t_9		Write(A)
t_{10}		Read(B)
t_{11}		B= B–100
t_{12}		Write(B)

(a) 调度3：串行调度

时刻	T_1	T_2
t_1	Read(A)	
t_2	A= A–50	
t_3	Write(A)	
t_4		Read(A)
t_5		A= A–100
t_6		Write(A)
t_7	Read(B)	
t_8	B= B–50	
t_9	Write(B)	
t_{10}		Read(B)
t_{11}		B= B–100
t_{12}		Write(B)

(b) 调度4：结果等价于调度3

冲突可串行化调度的验证

□ 冲突可串行化调度验证的原则：

- 当并发的两个事务之间不存在冲突时，可以很容易的通过交换操作将当前调度S'转化成等价的串行调度。
- 对于存在冲突的调度S'，可以通过仅交换不冲突的操作而转化为等价的串行调度S时，称调度S'和调度S冲突等价，调度S'为冲突可串行化调度 。

□ 结论：调度S'中T_i，T_j的某两个操作存在冲突（比如读写、写写冲突），且T_i的冲突操作在T_j冲突操作之前时。若调度S'存在冲突等价的串行调度S，则在串行调度S中T_i一定在T_j之前。

- 根据冲突和冲突等价的定义，一旦操作发生冲突，则不可对其进行交换，因此无法交换T_i，T_j的冲突操作，因此调度保序。

冲突可串行化调度的验证

- **优先图：利用冲突关系，可以将并发的事务构建成一个图来验证事务的并发调度S’是否是冲突可串行化调度。**
  - 若T_i与事务T_j存在冲突（读写冲突、写写冲突、写读冲突），且T_i的冲突操作在T_j冲突操作之前，则T_i向T_j连一条有向边（T_i，T_j）。
- **冲突可串行化的充分必要条件：优先图无环**

时间	T1	T2
1	Read(A)	
2	A= A–50	
3	Write(A)	
4		Read(A)
5		A= A–100
6		Write(A)
7	Read(B)	
8	B= B–50	
9	Write(B)	
10		Read(B)
11		B= B–100
12		Write(B)

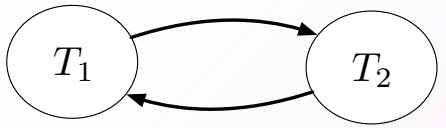
调度4

时间	T1	T2
1	Read(A)	
2	A= A–50	
3	Write(A)	
4		Read(A)
5		A= A–100
6		Write(A)
7	Read(B)	
8	B= B–50	
9		Read(B)
10	Write(B)	
11		B= B–100
12		Write(B)

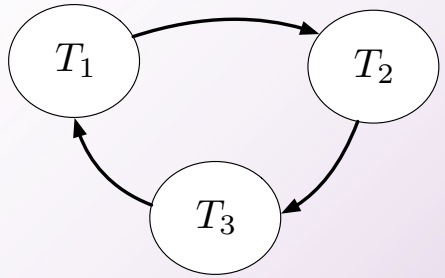
调度4.4一个不等价的变换



(a) 调度4的优先图



(b) 调度4.4的优先图



(c) 一个复杂的非冲突可串行的优先图

冲突可串行化调度的验证

必要性：

- 若优先图G存在环，则调度S'等价的串行调度S中T_i需要在T_j之前并且T_j需要在T_i之前，存在矛盾。

充分性：

- 对于有向无环图的判定，可以使用经典的拓扑排序算法。若图G无环，使用拓扑排序可以得到图G的一个拓扑序列C。
  - 拓扑序列C则是图中顶点的序列，且保证序列中排在后面的顶点在图G中不会指向序列中在它之前的顶点。$<T_1, T_2>$则是优先图的一个拓扑序列
- 拓扑序列C为该调度S'等价的串行调度S。

时间	T1	T2
1	Read(A)	
2	A= A−50	
3	Write(A)	
4		Read(A)
5		A= A−100
6		Write(A)
7	Read(B)	
8	B= B−50	
9	Write(B)	
10		Read(B)
11		B= B−100
12		Write(B)

调度4



(a) 调度4的优先图

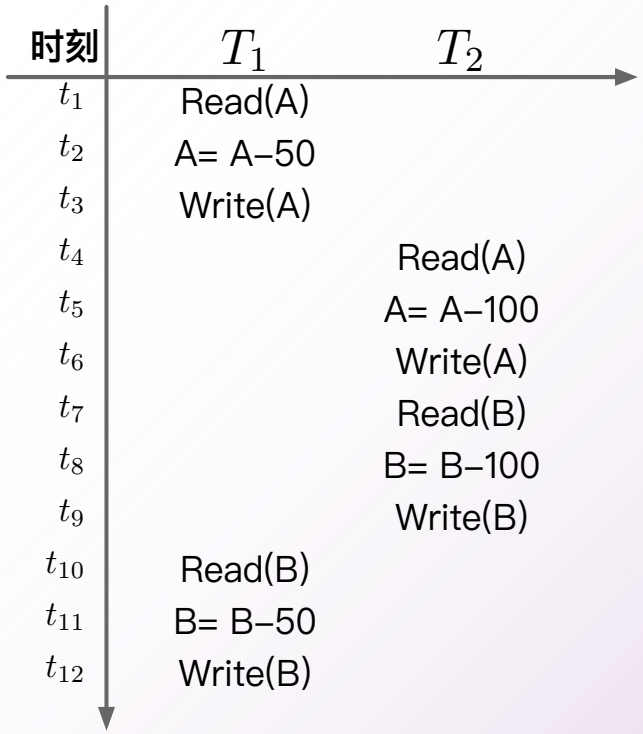
冲突可串行化调度的验证

口拓扑序列C为该调度S'等价的串行调度S的证明：

- （1）C_i表示等价的串行调度的第i个事务。
- （2）按照拓扑序列C的顺序，将调度S'中事务C_1对应的所有操作移到剩余事务（C_2, C_3等）前面。
  - 由于后续的事务与C_1不存在冲突（没有指向C_1的有向边）操作，所有操作都可以被交换。
- （3）将事务C_2的操作等价交换到C_1之后。
  - 原理如（2）
- 重复移动C_3,C_4等，直到调度S'成为串行调度C（S）。由于都是等价交换，因此调度S'等价的串行调度S

冲突可串行化调度的验证

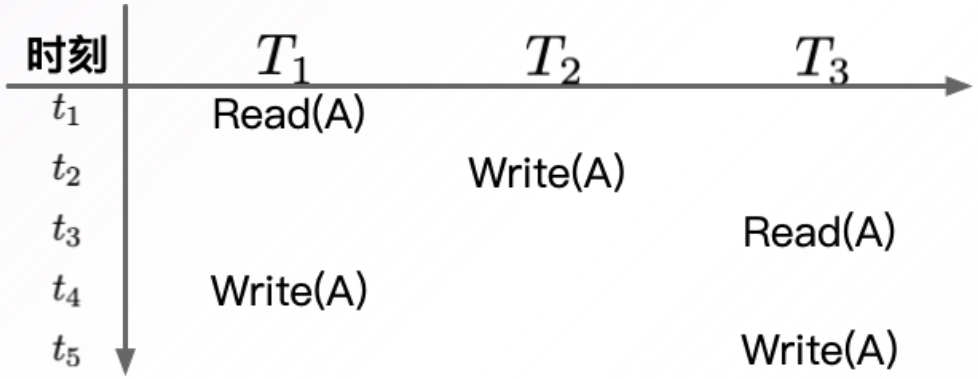
- **冲突可串行化调度是可串行化调度的一种，是易于验证和实现的可串行化调度**
  - 调度5展示了一种非冲突可串行化的可串行化调度。
    - 等价于串行调度< T_1, T_2>
  - 冲突可串行化调度的验证复杂度较低，而且可以通过两阶段封锁2PL实现，因此是数据库中通常采用的一种方法。
  - 还有些调度可验证，但是验证复杂度过高的可串行化调度—**视图可串行化调度。**



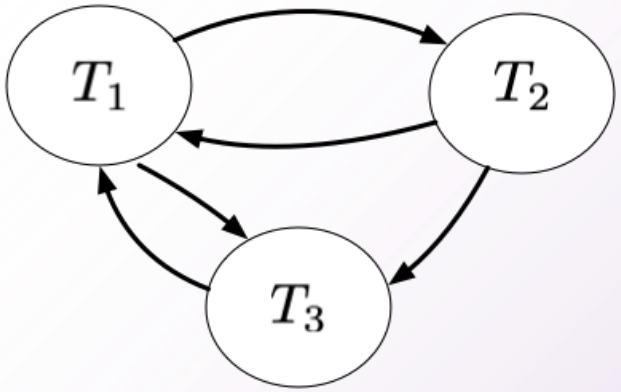
调度5：一个非冲突可串行化的可串行调度

视图可串行化调度

- 除了冲突可串行化调度，还有一种基于读写操作的冲突定义的可串行化调度被称为视图可串行化调度。
- 视图可串行化调度包含的调度范围比冲突可串行化调度更广，但是判断难度也更大。
  - 调度6及其优先图中可以发现，其优先图存在环，因此调度6不是冲突可串行化调度。
  - 调度6等价于串行调度<T_1,T_2,T_3>，因此调度6是可串行化调度。
  - （注意事务T1的Write（A）是无效写，不会被T_2,T_3读到，因此该调度是可串行化的）



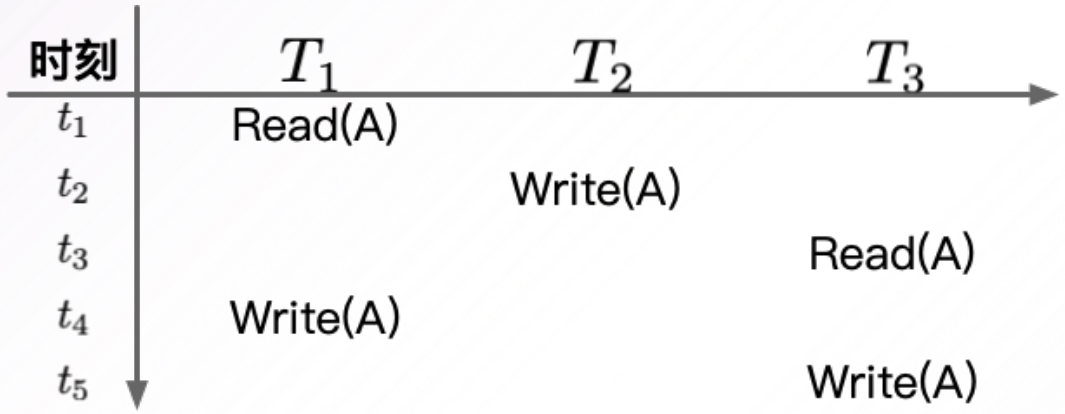
(a)调度6



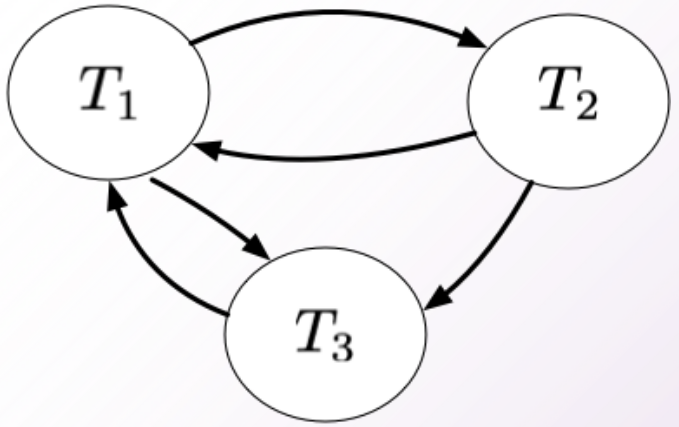
(b)优先图

视图可串行化调度定义

- 视图可串行化从视图等价的定义出发，针对一个调度S'去发现其等价的串行调度S来确定S'是一个视图可串行化调度。
- 满足下面3个条件的两个调度视图等价：对任意数据库的记录X
  - 1. 若调度S'中事务T_i读取了X的初始值，则调度S中T_i也需要读取X的初始值。
  - 2. 若调度S'中事务T_i读取了T_j更新的X值，则调度S中T_i也需要读取T_j更新的X值。
  - 3. 若调度S'中事务T_i最后写入了X的值，则调度S中T_i也要最后写入了X的值。



(a)调度6



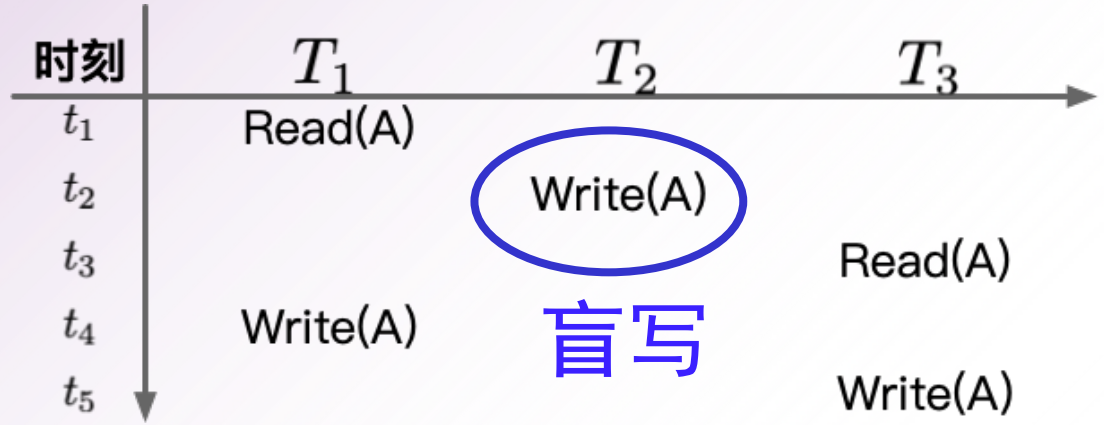
(b)优先图

视图可串行化 vs 冲突可串行化

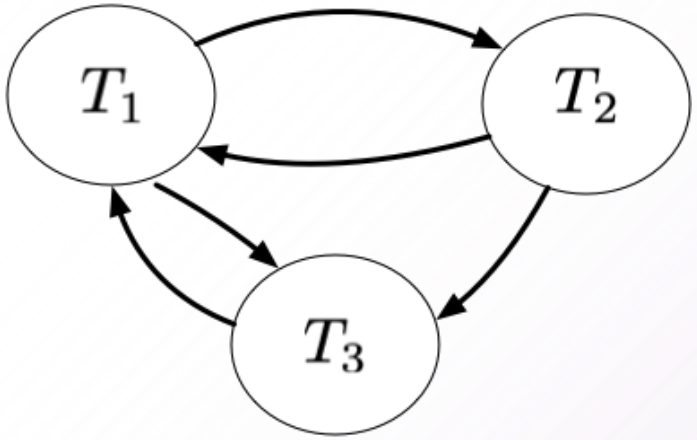
- **冲突可串行化调度肯定是视图可串行化调度的证明：**
  - 冲突可串行化调度S'等价于某个串行调度S。
  - 读与写冲突要求了调度S对于任意记录X的事务读写关系和S'一致，这满足了视图等价的条件1和条件2。
  - 写写冲突要求了调度S中最后一个记录X的写入事务无法移到其他写入X的事务之前，这满足了视图等价的条件3。
  - 因此冲突可串行化调度S'视图等价于串行调度S。
- **冲突可串行化调度一定是视图可串行化**
- **但视图可串行化调度不一定是冲突可串行化**
  - 一般是引入了盲写（blind write），即不需要读取数据的写

视图可串行化 vs 冲突可串行化

- 盲写：未读，直接写



(a)调度6

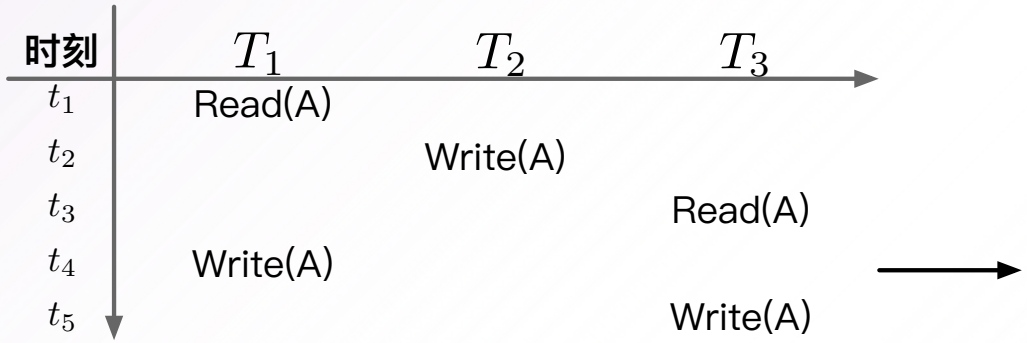


(b)优先图

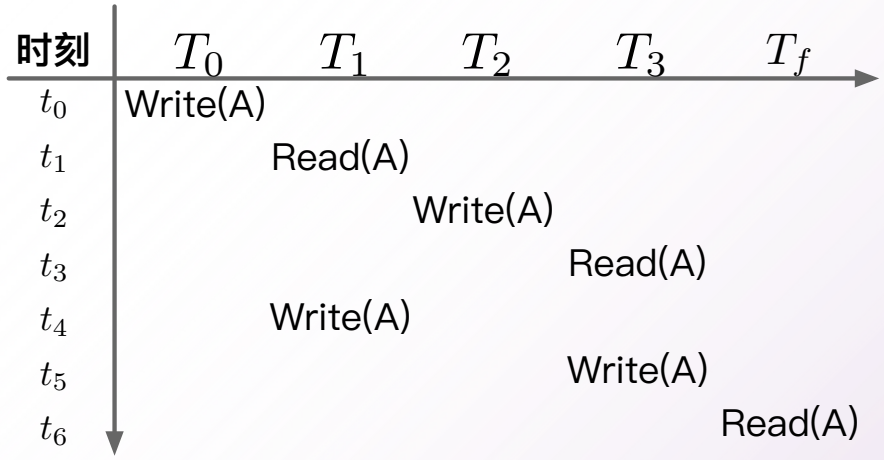
- 一个调度是视图可串行化，但不是冲突可串行化，一般由盲写造成

视图可串行化调度验证

- 视图可串行化使用一种带标记的优先图进行判断。
- 首先需要为调度S增加一个起始事务T_0和一个终止事务T_f
  - T_0全为写操作，为调度S所有数据的写操作，放置于所有事务之前（初始读）
  - T_f全为读操作，为调度S所有数据的读操作，放置于所有事务之后（最终写）



调度6



带T_0和T_f的调度6–1

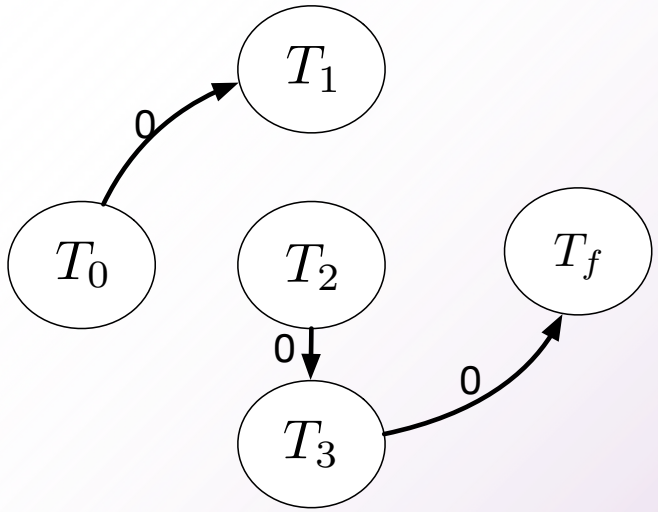
视图可串行化调度验证

构建带标记的优先图

– 若调度S中事务T_i读取了事务T_j写入的值，则从T_j构建指向T_i的有向边，标记为0。

时刻	T_0	T_1	T_2	T_3	T_f
t_0	Write(A)				
t_1		Read(A)			
t_2			Write(A)		
t_3				Read(A)	
t_4		Write(A)			
t_5				Write(A)	
t_6					Read(A)

带T_0和T_f的调度6–1



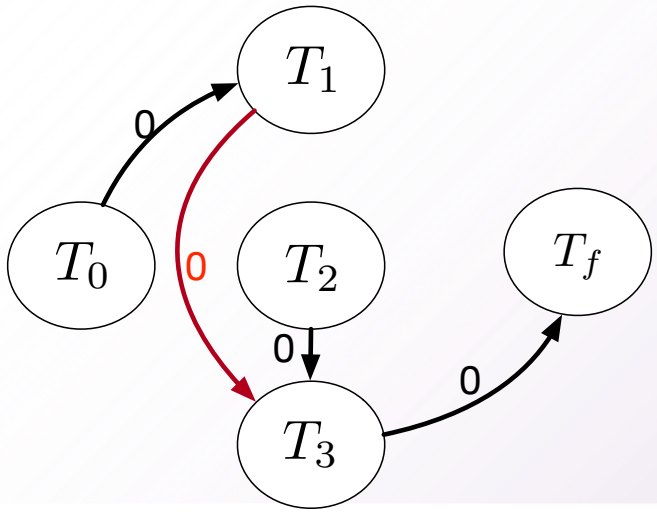
视图可串行化调度验证

构建带标记的优先图

- 若调度S中事务T_i读取了事务T_j写入的值，则从T_j构建指向T_i的有向边，标记为0。
- 对每个数据项A，枚举调度S中非T_0的所有写操作，枚举到的写操作属于的事务用T_k表示。对于事务T_k的写操作write(A)，若T_i读取了T_j写入A的值，且k，i，j都不相等时，增加带标记的边。
  - $T_i=T_f$时，由于T_f是终止事务，T_k必须在T_j之前，因此T_k引出一条标记为0的边指向T_j

时刻	T_0	T_1	T_2	T_3	T_f
t_0	Write(A)				
t_1		Read(A)			
t_2			Write(A)		
t_3				Read(A)	
t_4		Write(A)			
t_5				Write(A)	
t_6					Read(A)

带T_0和T_f的调度6-1



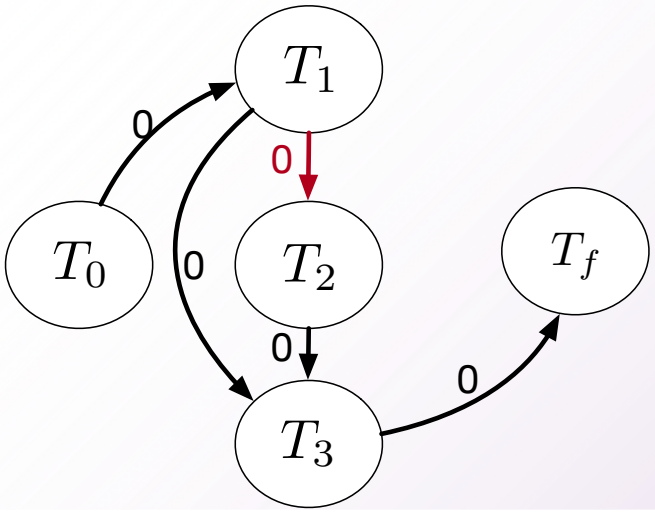
视图可串行化调度验证

口 构建带标记的优先图

- 若调度S中事务T_i读取了事务T_j写入的值，则从T_j构建指向T_i的有向边，标记为0。
- 对每个数据项A，**枚举调度S中非T_0的所有写操作**，枚举到的写操作属于的事务用T_k表示。对于事务T_k的写操作write(A)，若T_i读取了T_j写入A的值，且k，i，j都不相等时，增加带标记的边。
  - $T_i=T_f$时，由于T_f是终止事务，T_k必须在T_j之前，因此T_k引出一条标记为0的边指向T_j
  - $T_j=T_0$时，由于T_0是起始事务，T_k必须在T_i之后，因此T_i引出一条标记为0的边指向T_k

时刻	T_0	T_1	T_2	T_3	T_f
t_0	Write(A)				
t_1		Read(A)			
t_2			Write(A)		
t_3				Read(A)	
t_4		Write(A)			
t_5				Write(A)	
t_6					Read(A)

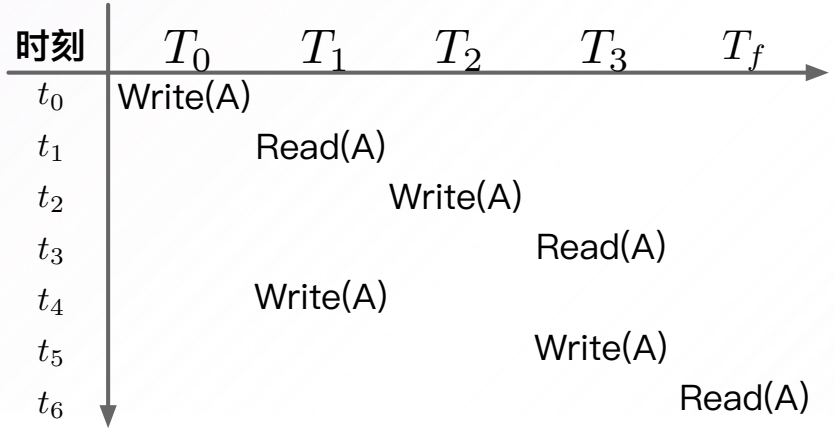
带T_0和T_f的调度6-1



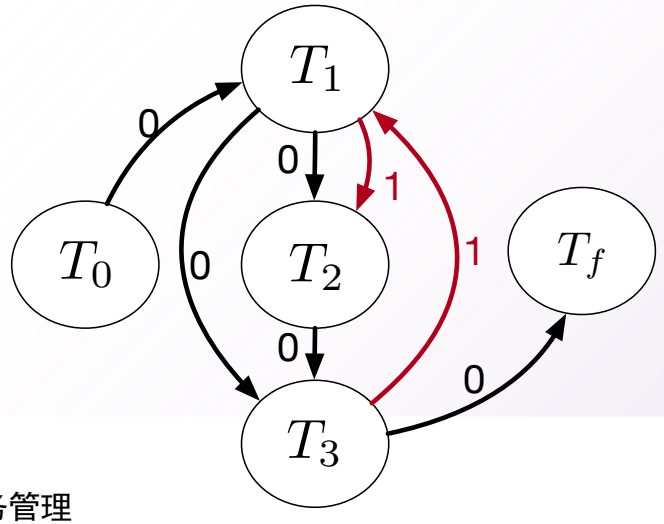
视图可串行化调度验证

构建带标记的优先图

- 若调度S中事务T_i读取了事务T_j写入的值，则从T_j构建指向T_i的有向边，标记为0。
- 对每个数据项A，**枚举S中非T_0事务的所有写操作**，枚举到的写操作属于的事务用T_k表示。对于T_k的写操作write(A)，若T_i读取了T_j写入A的值，且k，i，j都不相等，增加带标记的边。
  - $T_i=T_f$时，由于T_f是终止事务，T_k必须在T_j之前，因此T_k引出一条标记为0的边指向T_j。
  - $T_j=T_0$时，由于T_0是起始事务，T_k必须在T_i之后，因此T_i引出一条标记为0的边指向T_k。
  - 当T_i不等于T_f且T_j不等于T_0时，则T_k可以在T_j之前也可以在T_i之后。从之前没有选择过的标号中，选择一个标号x，从T_k连出标记为x的有向边指向T_j，并从T_i连出标记为x的有向边指向T_k。



带T_0和T_f的调度6–1

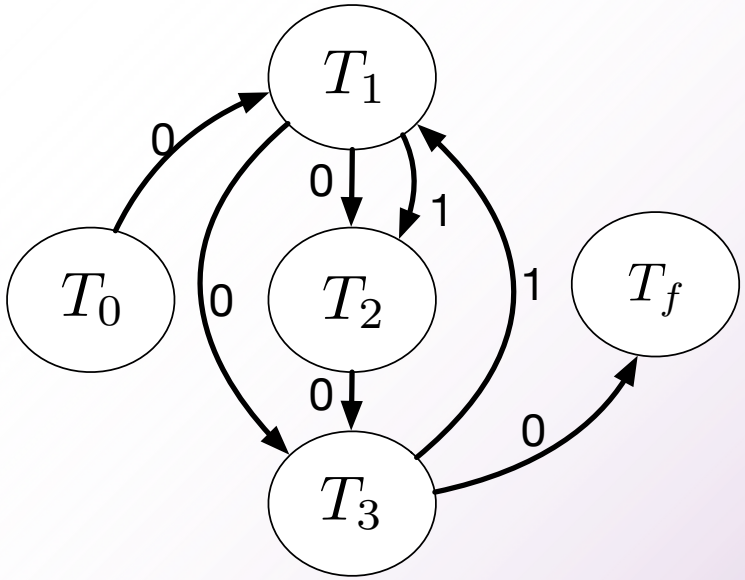


视图可串行化调度验证

口 带标记的优先图转为常规优先图

– 每种大于0的标记都有两条边，代表两种可能

- 写事务在冲突的写读操作之前
- 写事务在冲突的写读操作之后



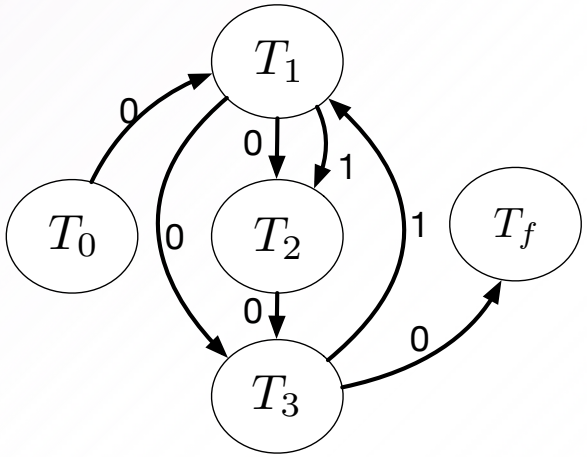
带标记的优先图

视图可串行化调度验证

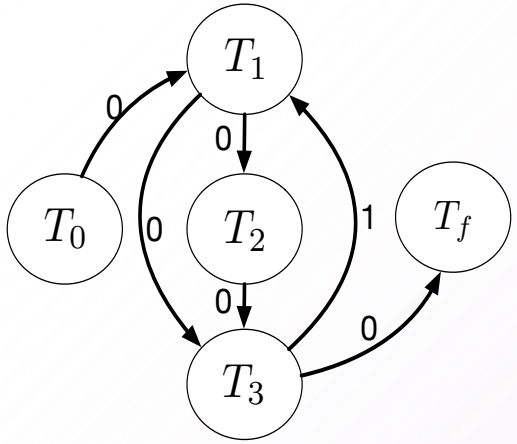
□ 带标记的优先图转为常规优先图

- 每种大于0的标记都有两条边，代表两种可能
  - 写事务在冲突的写读操作之前
  - 写事务在冲突的写读操作之后
- 枚举大于0的标记的边的两种可能性转换为普通优先图
  - 每次优先图判定为O(M)，M为边的数量
  - 若有N种标记，复杂度为O(2^N*M)

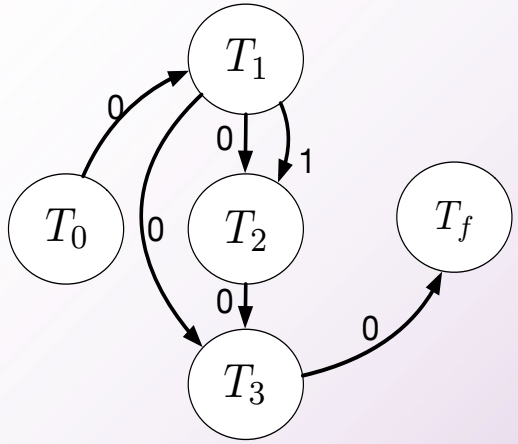
只要存在一种优先图无环，则视图可串行化



带标记的优先图



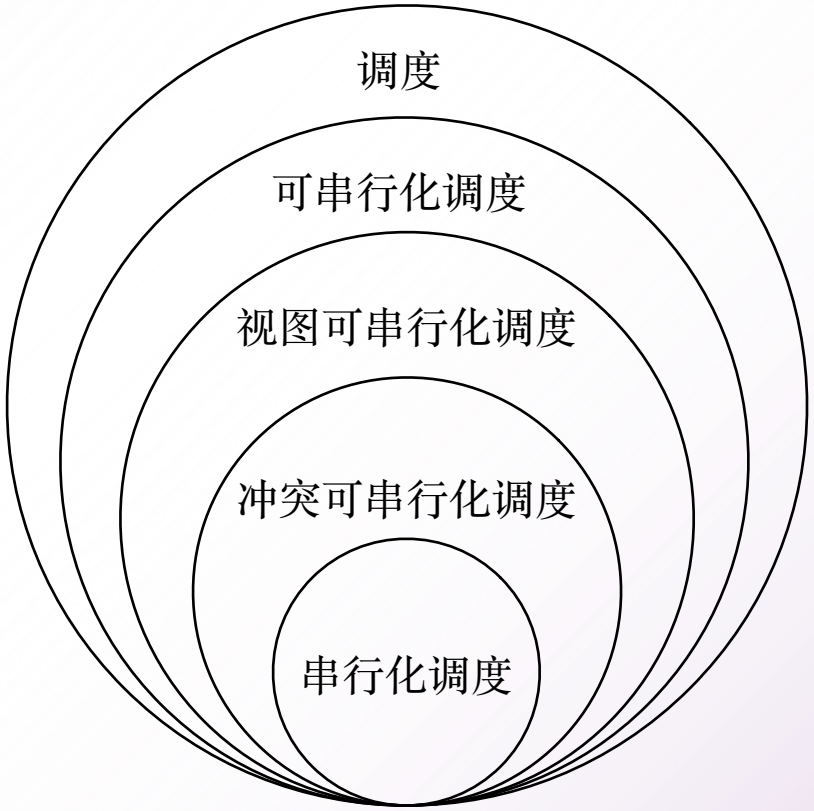
(a)优先图1



(b)优先图2

调度的包含关系

- **串行化调度**
  - 串行，并发低，时延高
- **冲突可串行化调度**
  - 并行，并发高，时延低
  - 事后易判断
  - 事前可保证 → 两阶段锁
- **视图可串行化调度**
  - 难判断
- **（终态）可串行化调度**
  - 难判断



可恢复调度

- 可串行化调度考虑ACID特性中的隔离性，旨在数据库正常运行的时候可以正确处理事务的并发。
- 然而当数据库发生故障需要回滚时，数据库需要高效可靠地恢复事务数据。

T1	T2
W(x)	
commit	
	R(x)
	abort

可恢复
recoverable

T1	T2
W(x)	
	R(x)
	commit
abort	

不可恢复
Not recoverable

T1	T2	T3
W(x)		
	W(x)	
		R(x)
abort		

级联回滚
Cascade rollback

可恢复调度

- T_2已经提价了事务，但T_2基于T_1写入的数据进行了进一步修改，当T_1被回滚时，T_2也应行回滚。
  - T_2在时刻t_{10}进行了提交，成为了结束的事务。（用户已经看到了修改的结果，不应被回滚）
    - 比如T_2事务提交表示购物已经完成，用户已经拿到了商品，此时难以回滚该事务。
  - T_2难以被回滚会造成事务T_1也难以被回滚。

- 不可恢复调度：读写了未提交事务修改的数据。
- 可恢复调度要求：当T_1未提交时，T_2（读写了T_1数据的事务）不应该提交。

时刻	T_1	T_2
t_1	Read(A)	
t_2	A= A–50	
t_3	Write(A)	
t_4		Read(A)
t_5		A= A–100
t_6		Write(A)
t_7		Read(B)
t_8		B= B–100
t_9		Write(B)
t_{10}		Commit;
t_{11}	Read(B)	
t_{12}	B = B–50	
t_{13}	回滚	

(a) 不可恢复调度,T_2已提交难以回滚

可恢复调度

- 可恢复调度：对于调度S，若T_j读取了T_i所写的数据，则需要保证T_j在T_i之后进行提交（即COMMIT操作），调度S是可恢复调度。

时刻	T_1	T_2
t_1	Read(A)	
t_2	A= A–50	
t_3	Write(A)	
t_4		Read(A)
t_5		A= A–100
t_6		Write(A)
t_7		Read(B)
t_8		B= B–100
t_9		Write(B)
t_{10}		Commit;
t_{11}	Read(B)	
t_{12}	B= B–50	
t_{13}	回滚	

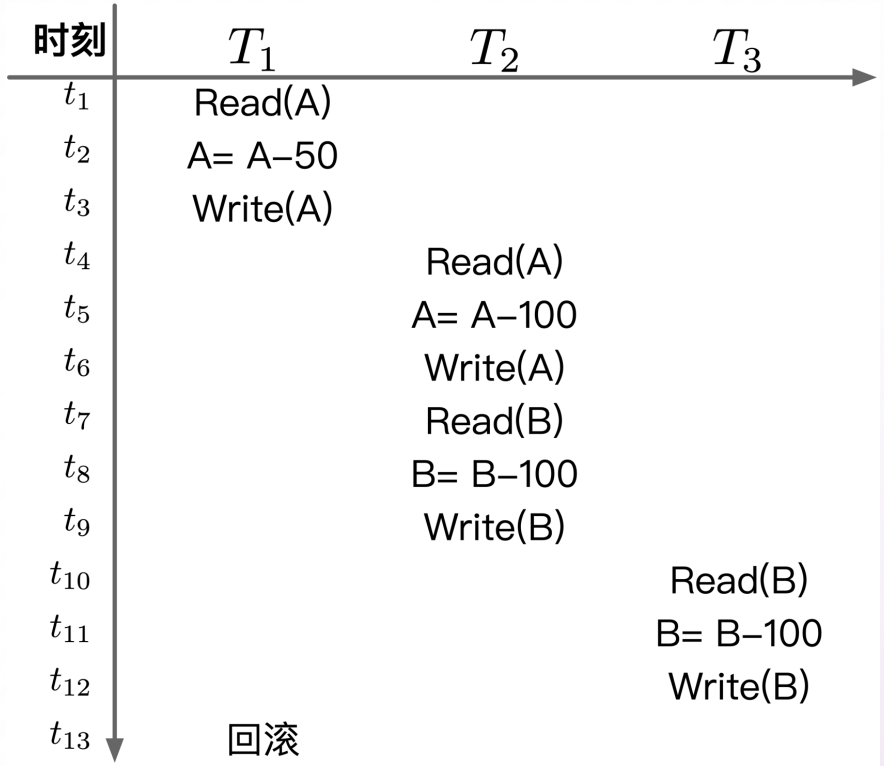
(a) 不可恢复调度，T_2已提交难以回滚

时刻	T_1	T_2
t_1	Read(A)	
t_2	A= A–50	
t_3	Write(A)	
t_4		Read(A)
t_5		A= A–100
t_6		Write(A)
t_7		Read(B)
t_8		B= B–100
t_9		Write(B)
t_{10}	Read(B)	
t_{11}	B= B–50	
t_{12}	Write(B)	
t_{13}	Commit;	
t_{14}		Commit;

(b) 可恢复调度，T_2等待T_1提交后提交

无级联调度

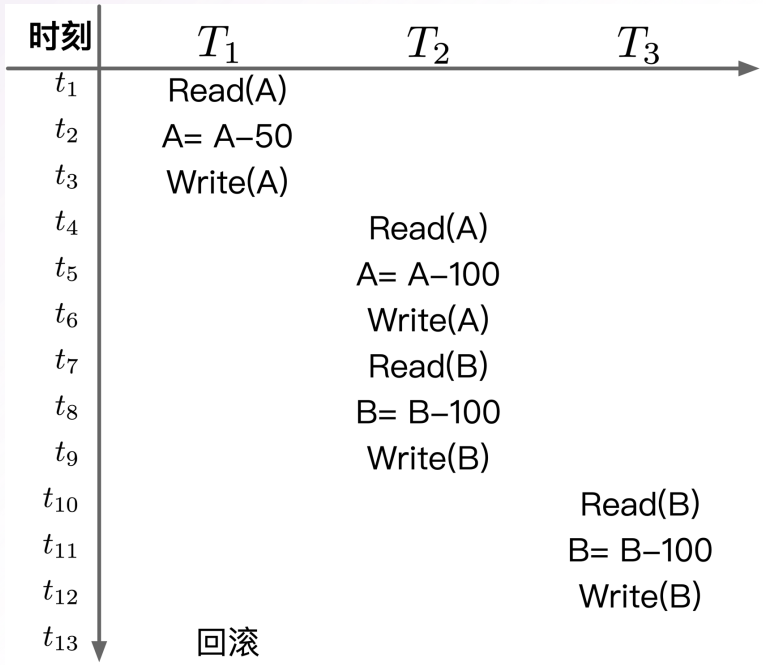
- 可恢复调度中在T_1进行回滚操作的时候，T_2需要一起回滚以进行数据库的恢复。
- 级联回滚：一个事务T_1回滚引起的另一个事务T_2回滚的现象为**级联回滚**。级联回滚是保证事务原子性的重要手段。
  - 较多的级联回滚有时会极大地影响数据库的性能。



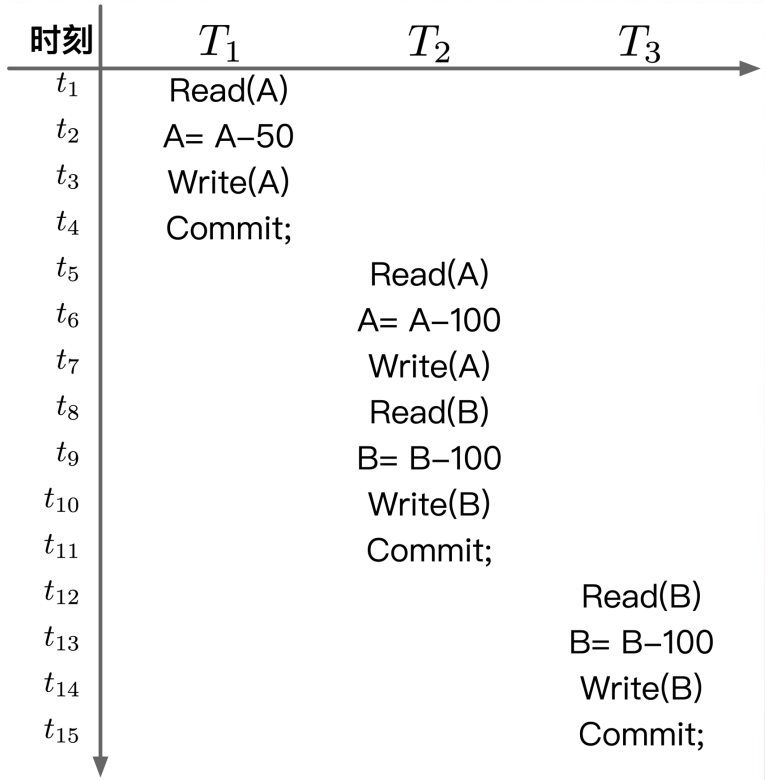
(a) 级联调度，若回滚则连续回滚3条事务

无级联回滚调度

- 无级联调度：当调度S中任意T_j读取某个T_i修改的数据X时，T_i都已经进行了提交，则调度S是无级联的。
- 无级联调度都是可恢复调度。



(a) 级联调度，若回滚则连续回滚3条事务



(b) 无级联调度，读操作在提交事务操作之后发生。

目录

01 事务概览及其概念

02 事务的特性

03 可串行化调度

04 事务的隔离级别

事务的隔离级别

- 数据库事务处理的性能依赖于事务的并发，事务的隔离性保障了事务并发处理过程的一致性。但严格的可串行化调度会影响事务的并发，甚至会退化成单纯的串行化调度，从而影响性能。

- 为了提高事务对可串行化要求不高的场景的处理效率，数据库标准对事务处理的性能和一致性进行了平衡，定义了4种隔离级别，分别是
  - 读未提交（read uncommitted）
  - 读已提交（read committed）
  - 可重复读（repeatable read）
  - 可串行化（serializable）

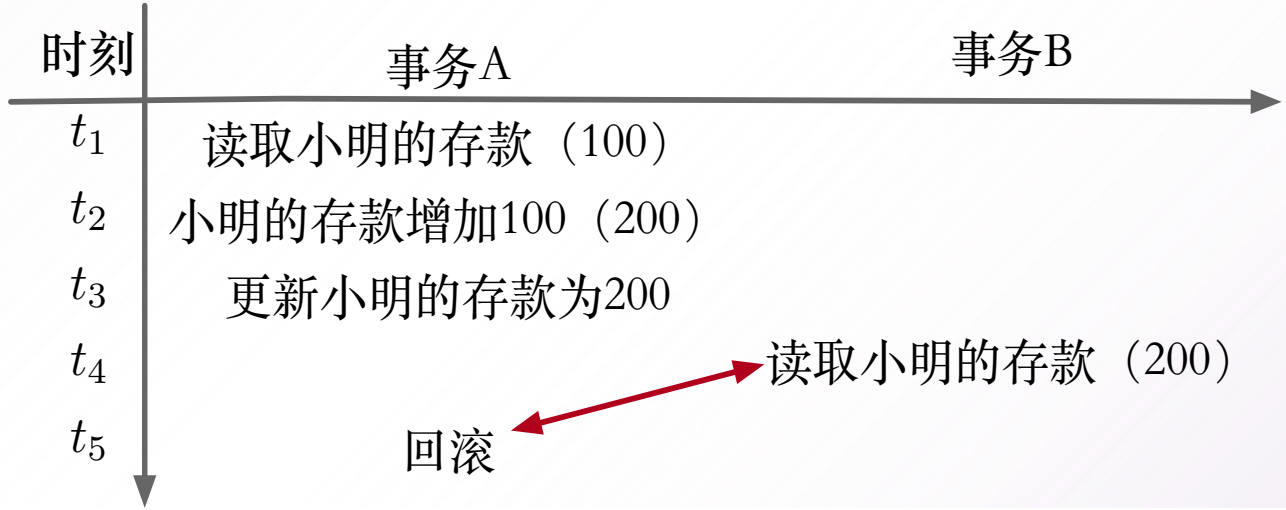
事务的隔离级别

- 每个隔离级别按照在该级别下禁止发生的异常现象来定义，这些异常现象包括：
  - 脏读(dirty read)：指一个事务在执行过程中读到其他尚未提交的写事务的修改数据；
  - 不可重复读(unrepeatable read)：指同一个事务先后两次读到的同一条记录的内容发生了变化，即两次分别读到被其他并发事务修改前和修改后的数据；
  - 幻读（phantom read）：指同一个事务内先后两次读到的数据不同，即两次分别读到其他并发写事务插入或者删除前后的数据。
- 隔离级别与允许的异常对应表格：

隔离级别	脏读	不可重复读	幻读
读未提交	允许	允许	允许
读已提交	不允许	允许	允许
可重复读	不允许	不允许	允许
可串行化	不允许	不允许	不允许

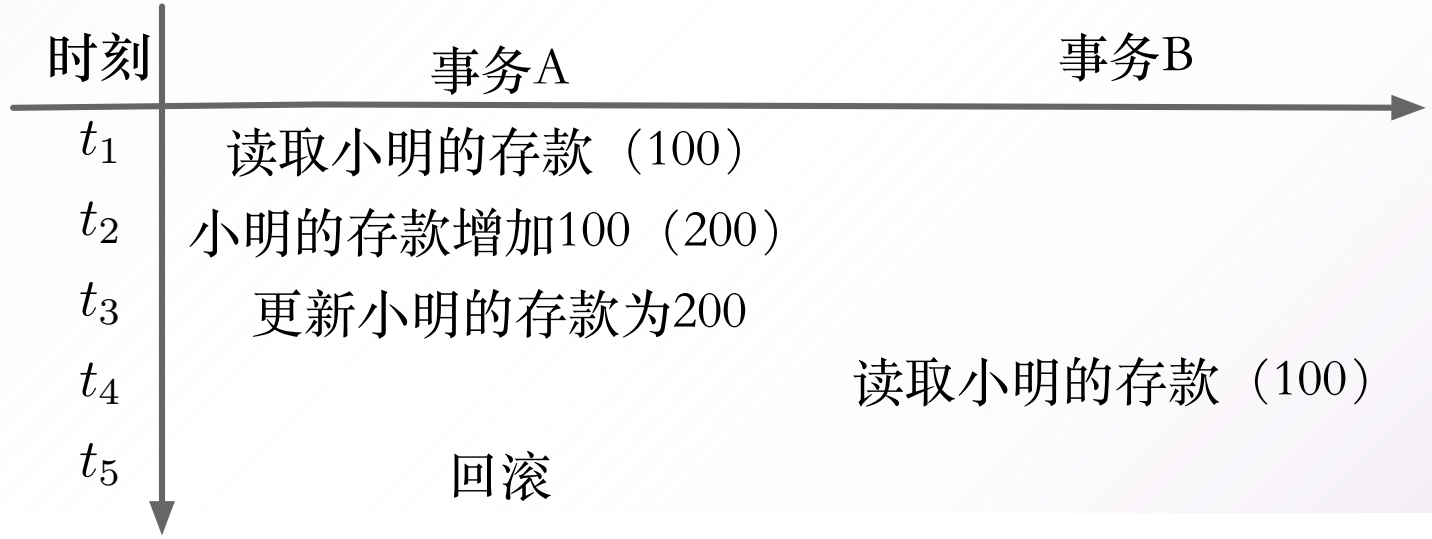
读未提交

- 读未提交这一隔离级别允许脏读、不可重复读和幻读三种异常情况的存在。
- 以脏读为例
  - 脏读允许一个事务在执行过程中读到其他尚未提交的写事务的修改数据；
  - 读未提交完全没有考虑到事务之间的影响，事务可能会读取到某些不正确的值（如将要回滚的值，或者中间状态的操作结果等）。
  - 事务A进行了回滚，事务B读取的小明的存款则为不正确的200元。因此，只有当业务场景中不同的事务之间没有相互影响时，才会考虑该隔离级别。



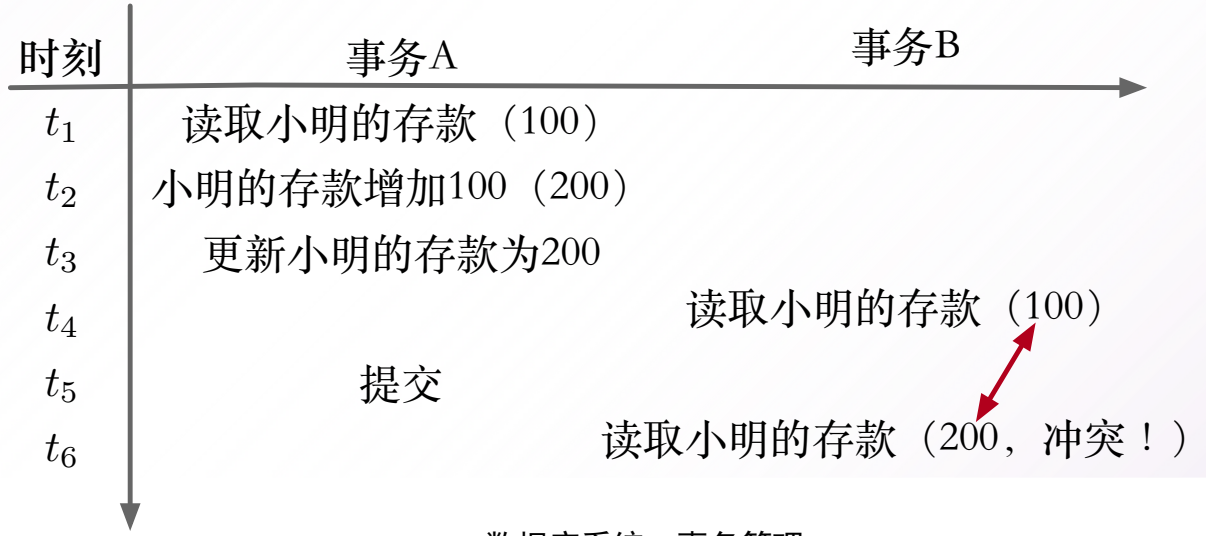
读已提交

- **数据库中为了防止脏读，提出了读已提交这一隔离级别：**
  - 读已提交要求事务只能读取到已经提交的值，忽略其他事务未提交的值。
    - 读已提交下，上述例子中读取到的值为100元。
  - 实现方法：通过MVCC，锁和时间戳等技术
    - 比如，通过MVCC来保证读到的数据为已经提交的版本。
  - 读已提交允许不可重复读和幻读



读已提交

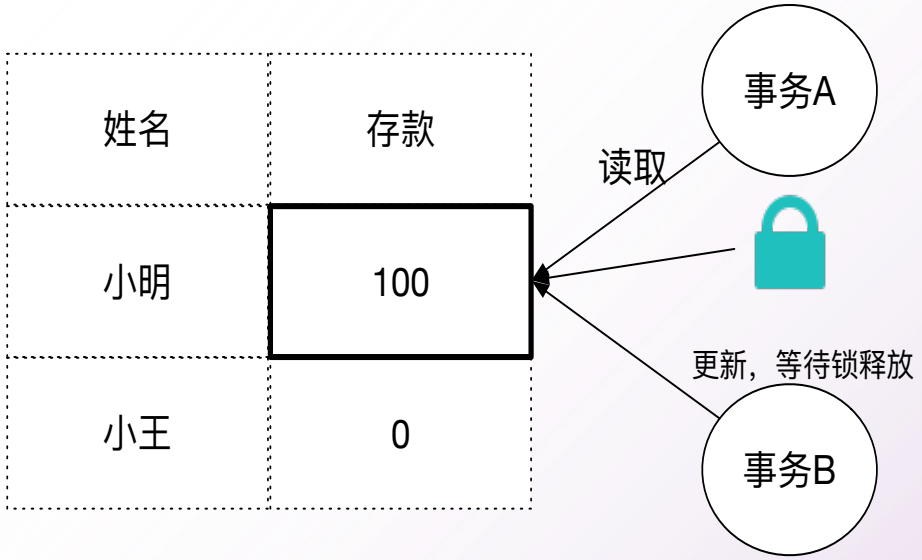
- **数据库中为了防止脏读，提出了读已提交这一隔离级别：**
  - 读已提交要求事务只能读取到已经提交的值，忽略其他事务正在处理但未提交的值。
    - 读已提交下，上述例子中读取到的值为100元。
  - 实现方法：通过MVCC，锁和时间戳等技术
    - 比如，通过MVCC来保证读到的数据为已经提交的版本。
  - 读已提交允许不可重复读和幻读
- **以不可重复读为例：**
  - 指同一事务先后两次读到的同一条记录不一致，分别读到被其他并发事务修改前后的数据。



可重复读

□ 数据库中为了防止不可重复读，提出了可重复读这一隔离级别：

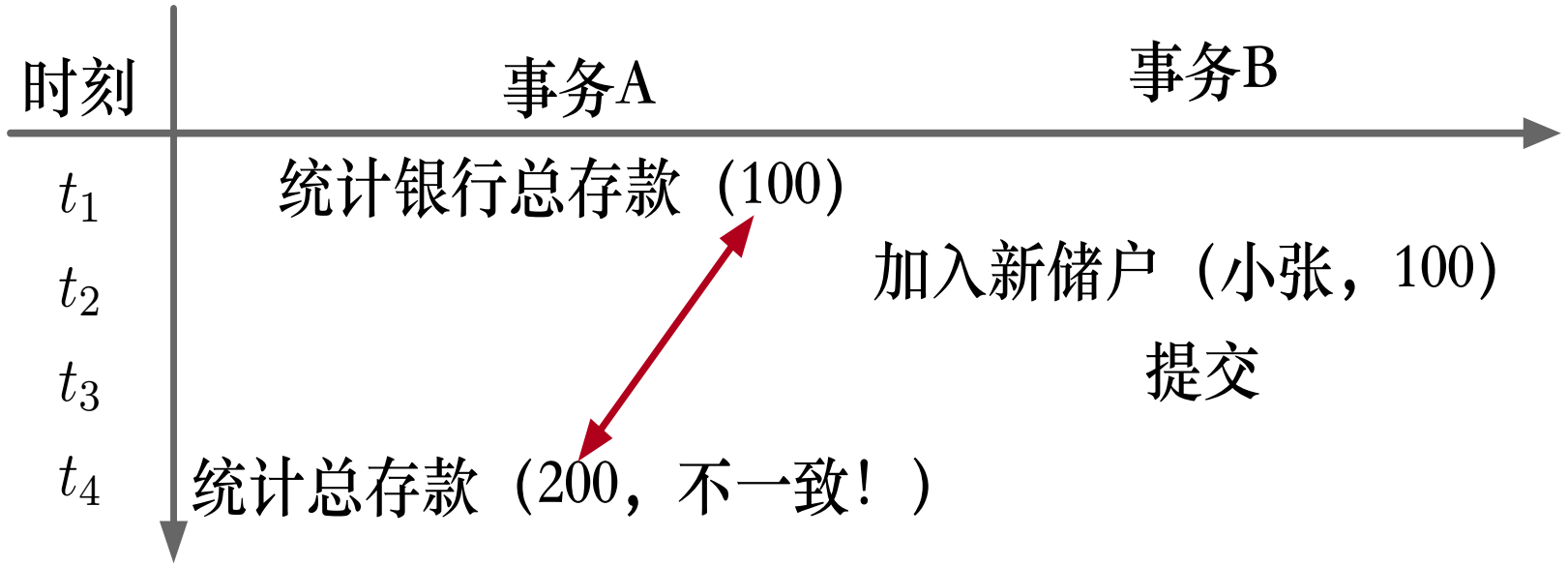
- 可重复读这一隔离级别要求能够读取已经提交的数据，且在同一个事务内，先后两次读到的同一条记录的内容不变。
- 实现方法：通过MVCC，锁和时间戳等技术
  - 比如，通过锁，锁住读取的数据，禁止修改
- 可重复读允许幻读



幻读

口幻读:

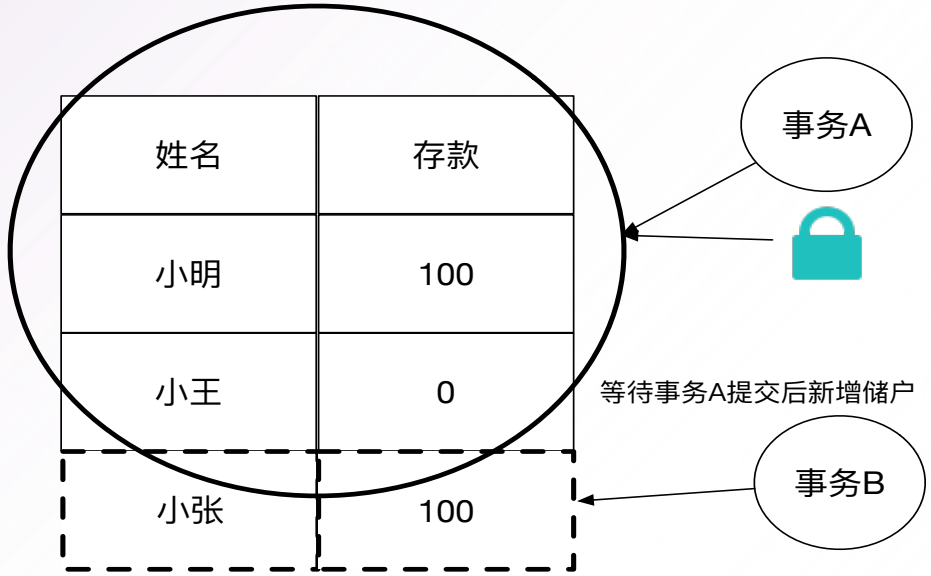
– 指同一个事务内先后两次读到的数据不同，即两次分别读到其他并发写事务插入或者删除前后的数据。

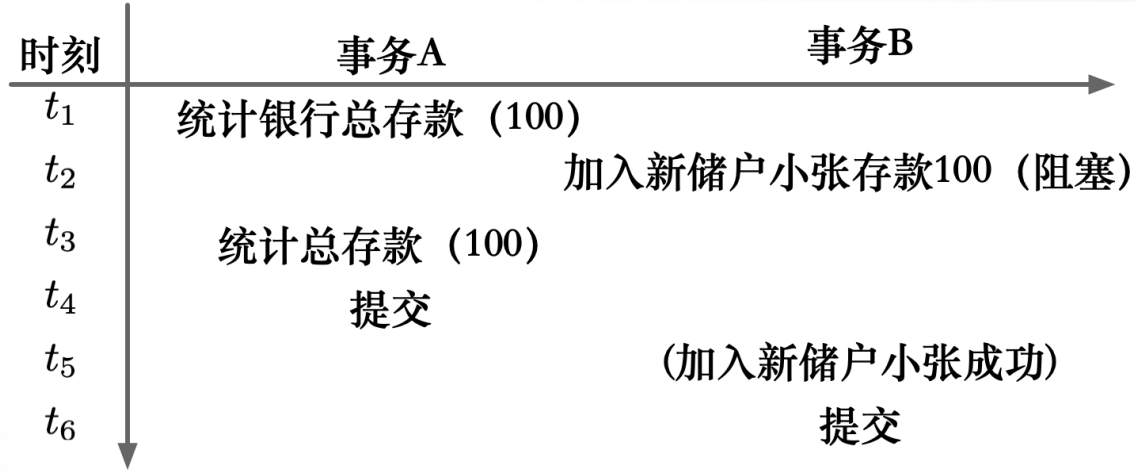


可串行化

□ 数据库中为了防止幻读，提出了可串行化这一隔离级别：

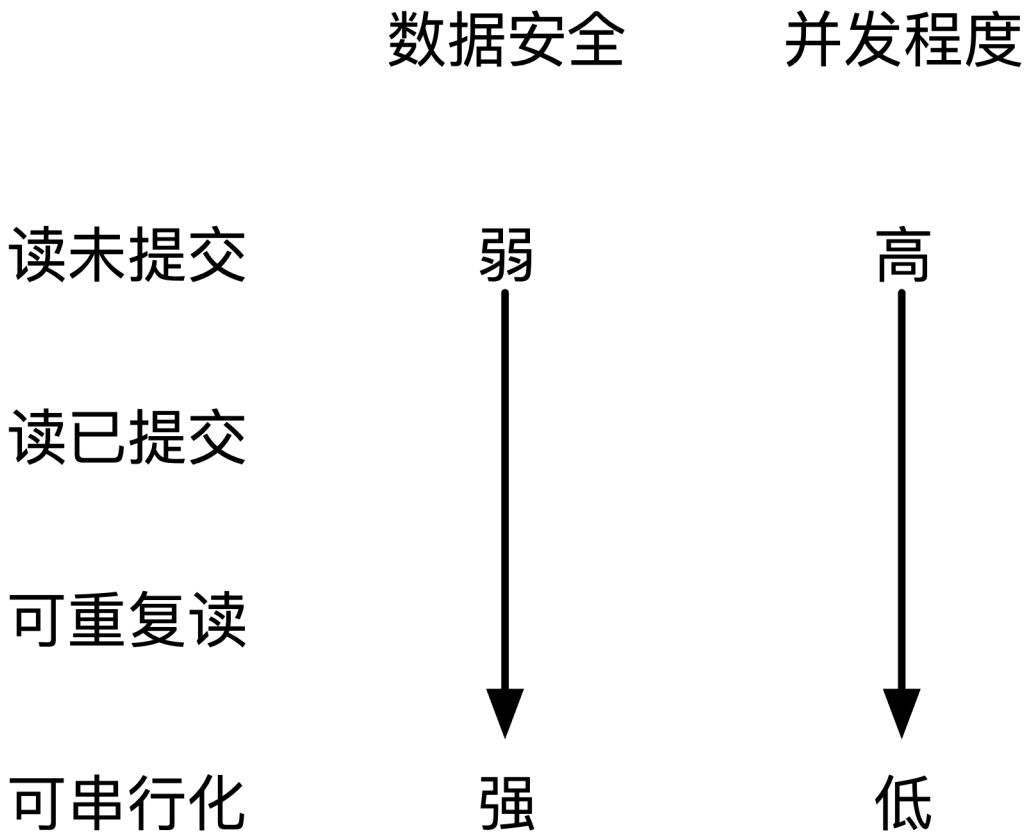
- 可串行化提供了最为严格的隔离级别，该级别要求事务的调度是一个可串行化调度，能够取得和串行执行一样的结果。
- 解决幻读：同样可以基于MVCC，两阶段锁和时间戳等技术。
  - 以锁为例





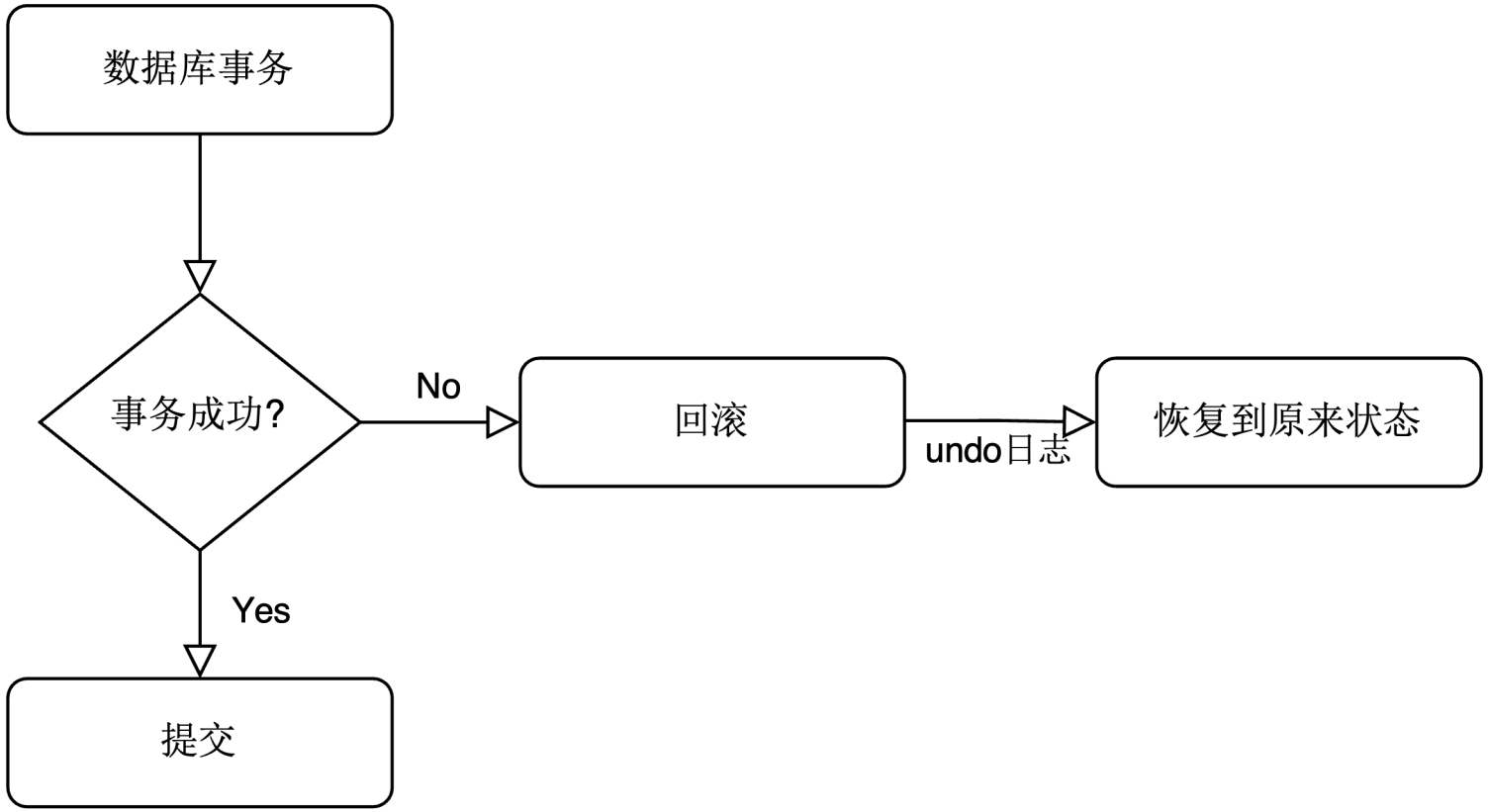
隔离级别的选择

□根据具体场景选择合适的隔离级别



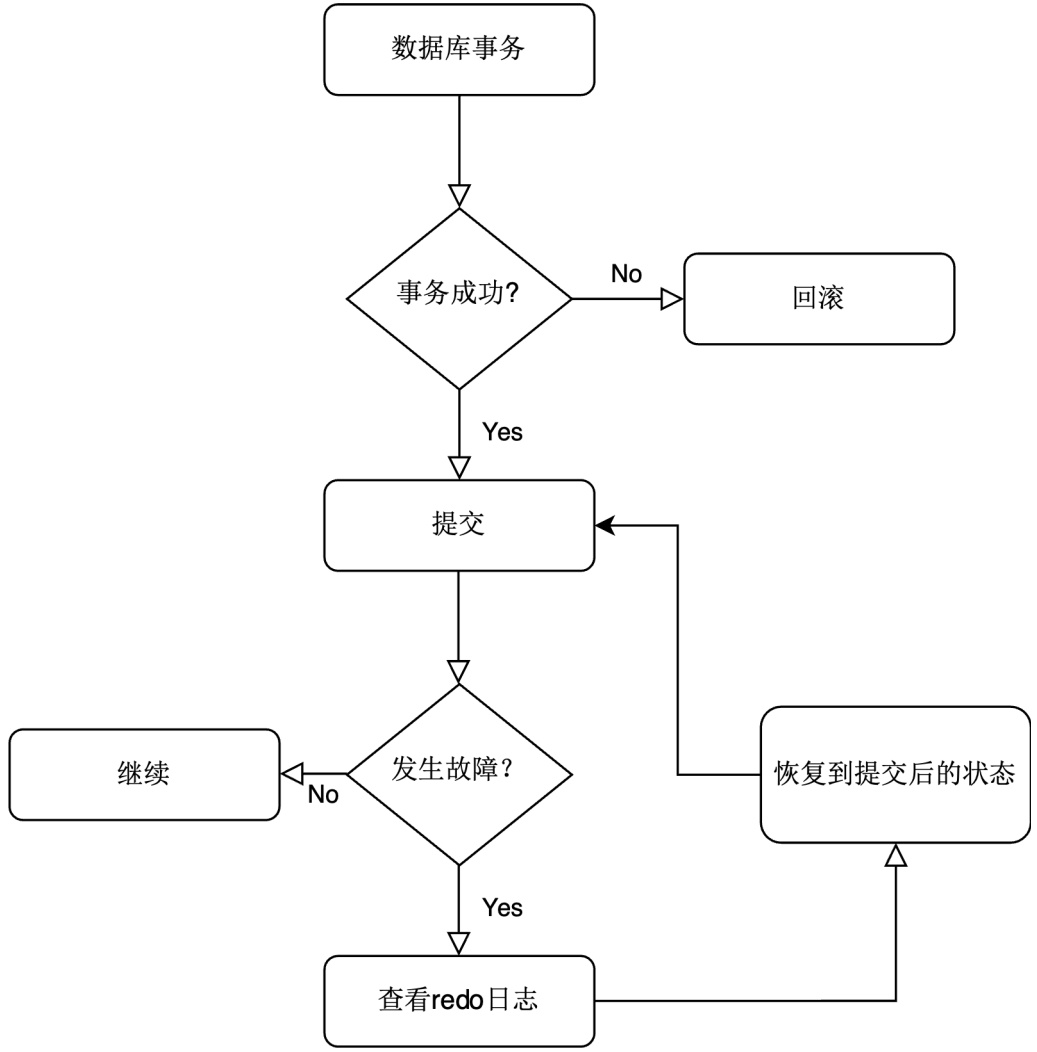
保证事务ACID的技术

- 原子性：数据库一般通过回滚日志（undo log）来保证原子性。即当事务回滚时，通过回滚日志恢复事务执行前的结果。



保证事务ACID的技术

- 持久性：数据库一般通过重做日志（redo log）来保证持久性。即当事务成功提交后发生故障时，通过重做日志恢复事务执行后的结果。



保证事务ACID的技术

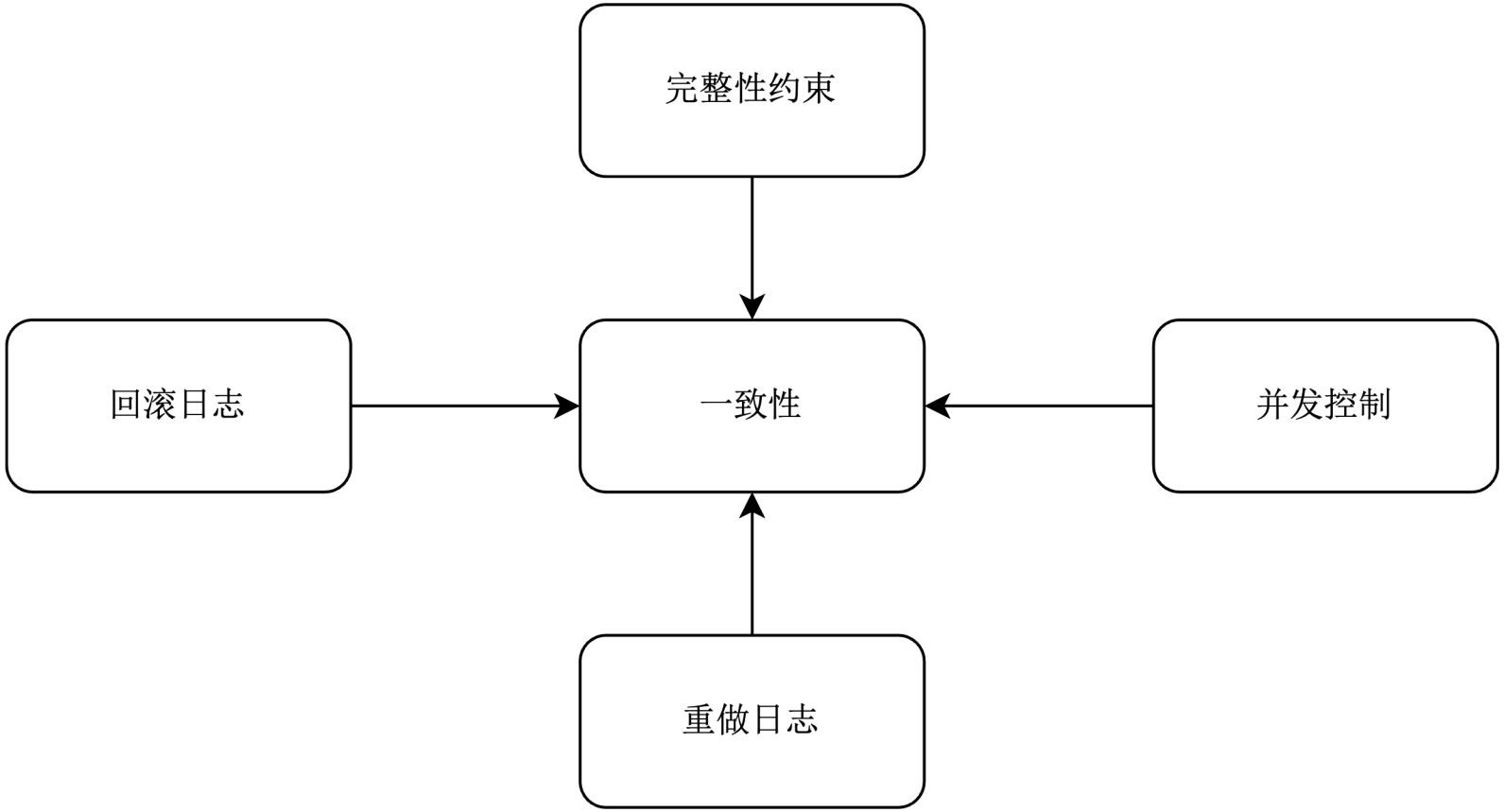
□ 隔离性：多个事务并发执行时，数据库通过并发控制协议来保证隔离性。

隔离级别	允许的异常现象	实现方式举例
读未提交	脏读、可重复读、幻读	一般不允许该级别
读已提交	可重复读、幻读	对数据项加锁，可以立刻释放读锁
可重复读	幻读	对数据项加锁，获取同一个时间戳或者缓存读取的数据
可串行化	均不允许	两阶段锁，对整张表加锁 或者 加谓词锁

快照隔离：基于快照的多版本隔离方法（Multi-Version Concurrency Control，MVCC）

保证事务ACID的技术

- 一致性：数据库通过回滚日志、重做日志、并发控制协议，以及数据库完整性约束来实现数据的一致性。



小结

- 本章介绍了事务这一数据库的重要功能。
- 本章介绍了事务的四大特性：原子性、持久性、隔离性、一致性。
- 本章介绍了在并发情况下可串行调度、冲突可串行化调度等概念。
- 本章介绍了数据库的四大隔离级别，以及幻读、不可重复读、脏读等异常现象。