数据库原理及应用

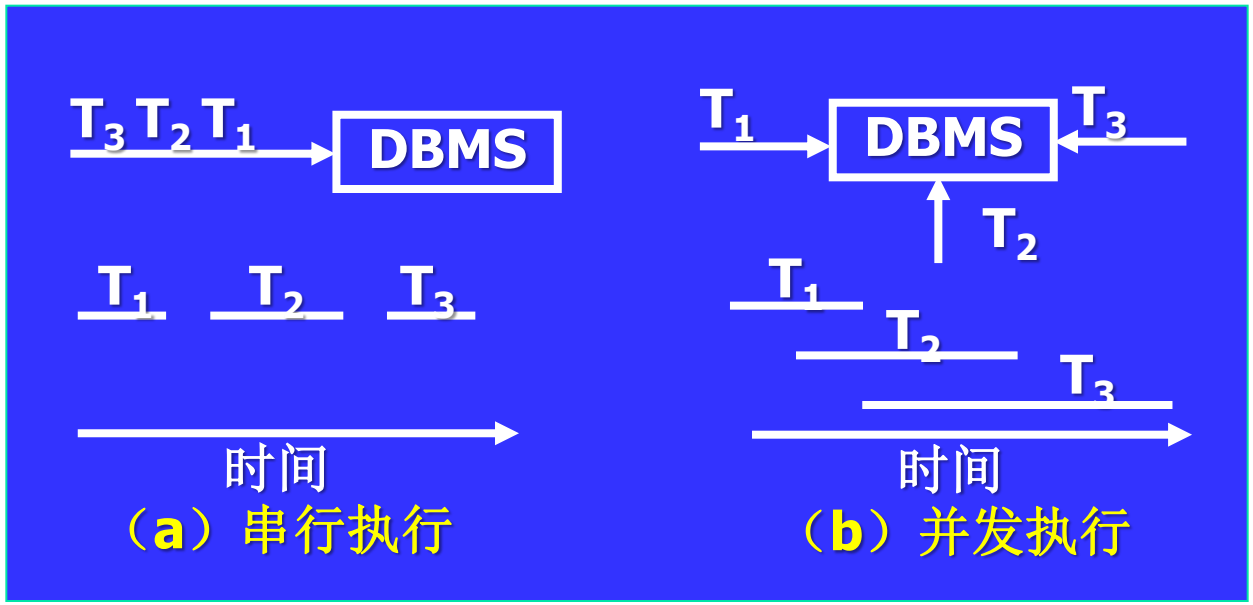
——并发控制

要求

1 了解并发操作可能产生的数据不一致性

2 掌握并发控制的技术：封锁机制、三级封锁协议

并发



目的： （1） 提高系统利用率

（2） 缩短事务的响应时间

并发

多用户并发存取数据库时就会产生多事务同时存取同一数据情形。若不控制，可能会存取和存储不正确的数据，造成数据库的不一致性。

注意： 并发操作下，事务的调度是随机的

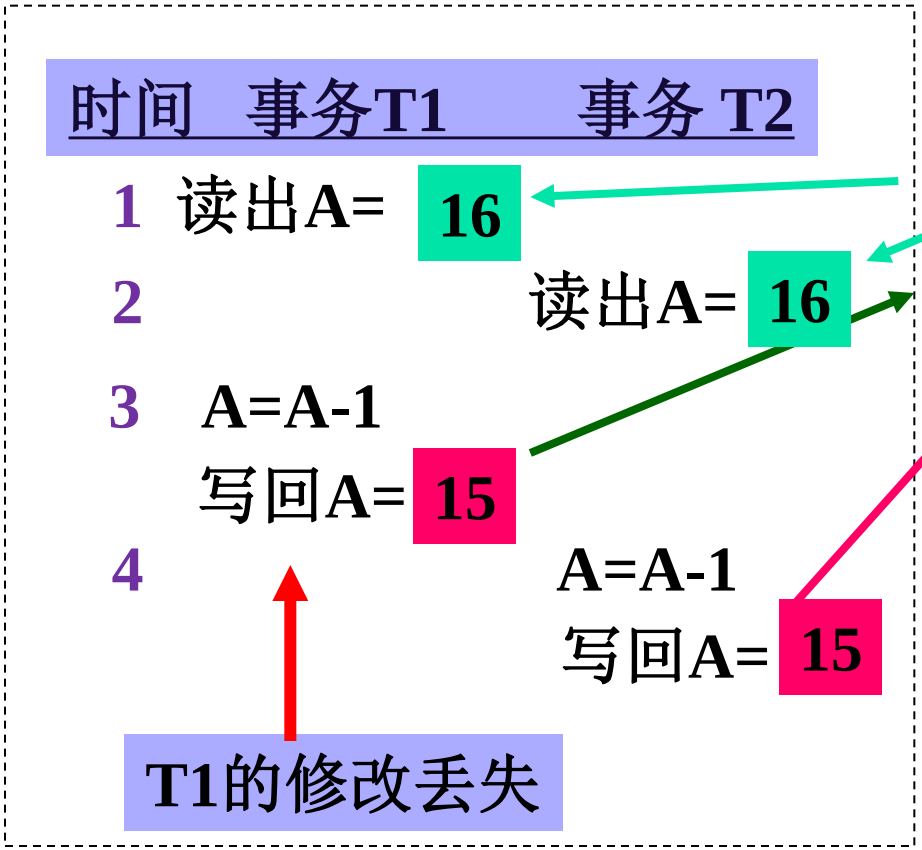
并发操作带来的数据不一致性包括：

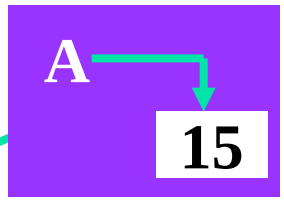
丢失更新、不可重现、读脏数据

并发引起问题

(1) 丢失更新（Lost update）

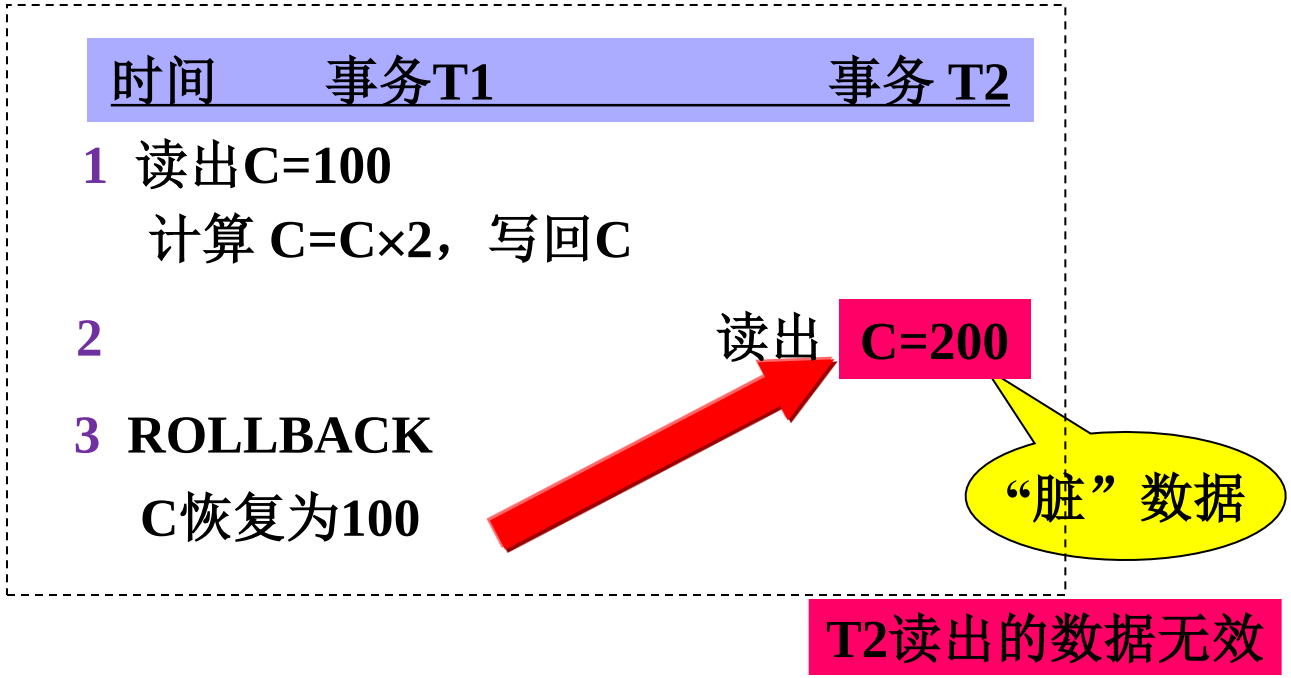
两事务读同数据并修改，先写回的数据修改丢失





并发引起问题

(2) 读“脏”数据（dirty read）T1修改某数据，T2读取同一数据；T1被撤消，T2读的是“脏”数据



并发引起问题

(3) 读值不可重现（**unrepeatable read**）**T1读数据，T2读并修改同一数据；T1为校对再读此数据，得到不同结果**。

时间	事务T1	事务 T2
1	读出A=50，B=100 求和 =150	
2		读出B=100 计算 B= B×2，写回B
3	读出A=50，B= 200 求和 = 250	

T1读出B的值与原来的不符，验算结果不对

并发控制

产生**data**不一致的原因是并发操作*破坏*了事务的**隔离性**。*并发控制*用正确方式调度，使事务不相互干扰。

并发控制的技术是**封锁**

封锁

排它锁（简称X锁，又称互斥锁）

事务T对数据对象R加上X锁，则只允许T读、写R，禁止其它事务对R加任何锁。

共享锁（简称S锁）

事务T对数据对象R加上S锁，则T可以读R，但不可以写R，且其它事务可以对R加S锁、但禁止加X锁。

封锁

1、封锁协议

对数据加锁时，对何时申请锁、持锁时间、何时释放等的约定。

2、基本方法：

事务读写数据R前先上锁，否则需等待；

何时释放锁（Unlock）？

1级封锁协议：

T在修改数据R前须先加X锁，直到事务结束才释放。

没有丢失更新

作用：

防止丢失更新

保证事务是可恢复的

1级封锁协议：

T在修改数据R前须先加X锁，直到事务结束才释放。

时间	事务T1	事务 T2
	请求X锁	
	对C加X锁	
	读出C=100	
	计算 C=C×2	
	写回C	
		读出 C=200
	ROLLBACK	
	（C恢复为100）	
	释放X锁	

仍然读出“脏”数据

T1已对C加X锁，为什么T2仍能读取C？

因为T2在读取C前未对C申请任何锁。

1级封锁协议：

T在修改数据R前须先加X锁，直到事务结束才释放。

时间	事务T1	事务 T2
	读出A=50，B=100 求和 =150	
		请求X锁 对B加X锁 读出B=100 计算 B=B×2，写回B Commit 释放X锁
	读出A=50，B= 200 求和 = 250	

仍然是
不可重复读

原因：T1在读取A、B时未申请任何锁。

2级封锁协议：

1级封锁协议 + T在读数据R前须加S锁，读完即释放。

时间	事务T1	事务 T2
	请求X锁	
	对C加X锁	
	读出C=100	
	计算 C=C×2,写回C	
		请求S锁
		等待
	ROLLBACK	
	（C恢复为100 ）	
	释放X锁	
		对C加S锁
		读出C=100
		释放S锁

作用：

防止丢失更新

保证事务可恢复

防止读“脏”数据

2级封锁协议：

1级封锁协议 + T在读数据R前须加S锁，读完即释放。

时间	事务T1	事务 T2
	对A加S锁	
	对B加S锁	
	读出A=50，B=100	
	求和 =150	
	释放A、B上的S锁	
		请求X锁
		对B加X锁
		读出B=100
		计算 B= B×2，写回B
		Commit
		释放X锁
	对A加S锁	
	对B加S锁	
	读出A=50，B= 200	
	求和 = 250	
	释放A、B上的S锁	

仍然是
不可重复读

3级封锁协议：

1级封锁协议 +事务T读数据R前须加S锁，事务结束释放。

作用：

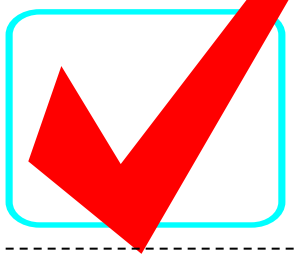
防止丢失修改

保证事务是可恢复的

防止读“脏”数据

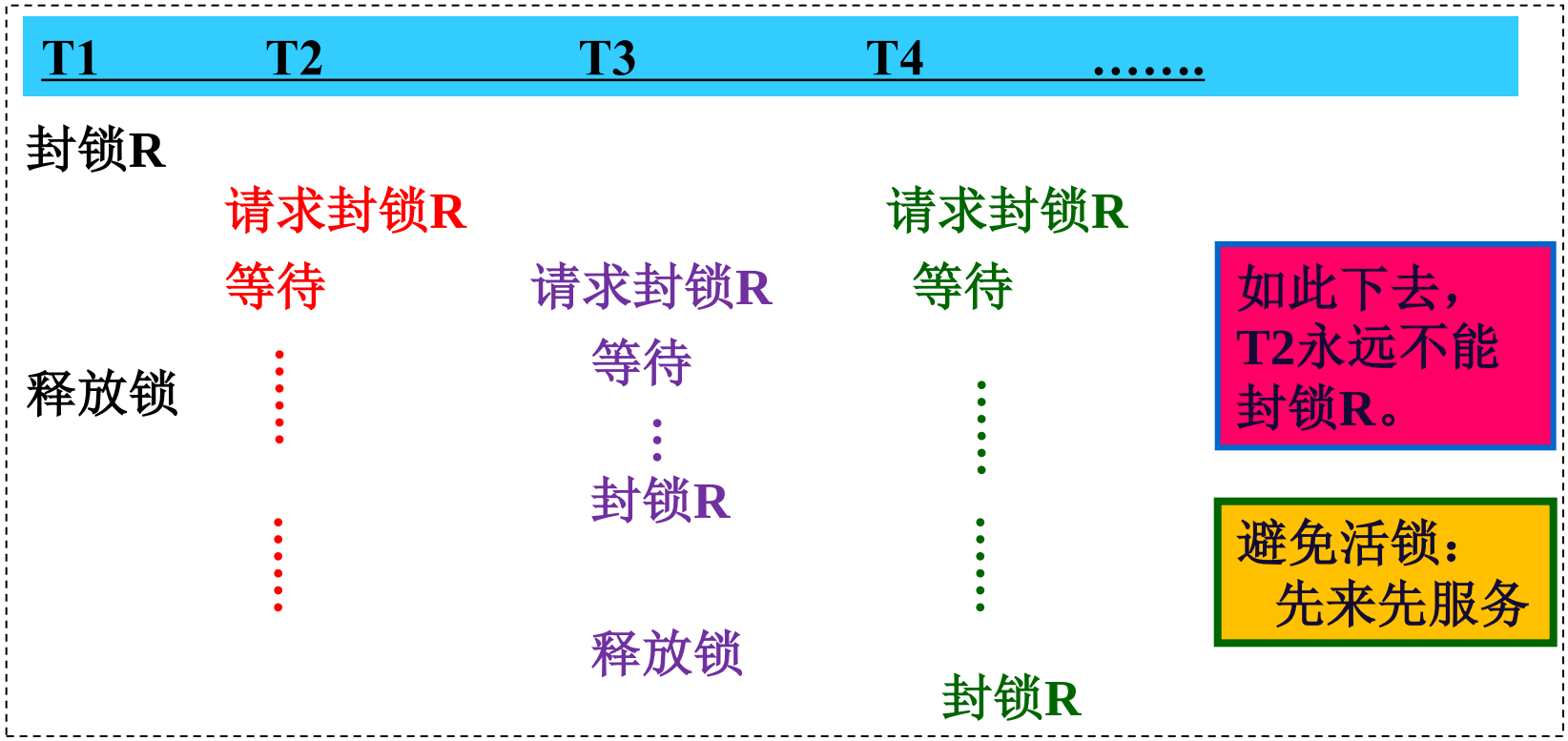
保证可重复读

时间	事务T1	事务 T2
	对A加S锁	
	对B加S锁	
	读出A=50，B=100	
	求和 =150	
		对B请求X锁
		等待
	读出A=50，B=100	
	求和 =150	
	Commit	
	释放A、B上的S锁	
		对B加X锁
		读出B=100
		计算 B= B×2
		写回B
		Commit
		释放B上X锁



封锁

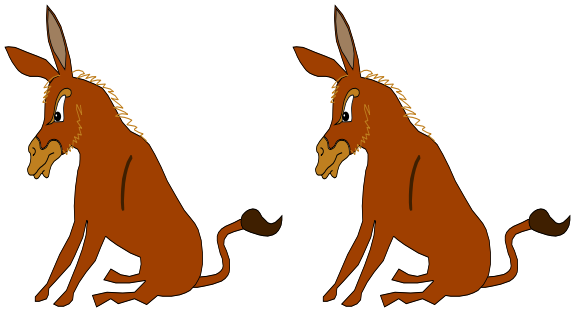
活锁： 数据不断处于上锁、开锁交替状态，某事务请求上锁，但始终没有得到机会而永久等待的情形。



封锁

死锁： 事务得到部分资源，又等待其他事务释放资源而出现的相互等待现象。

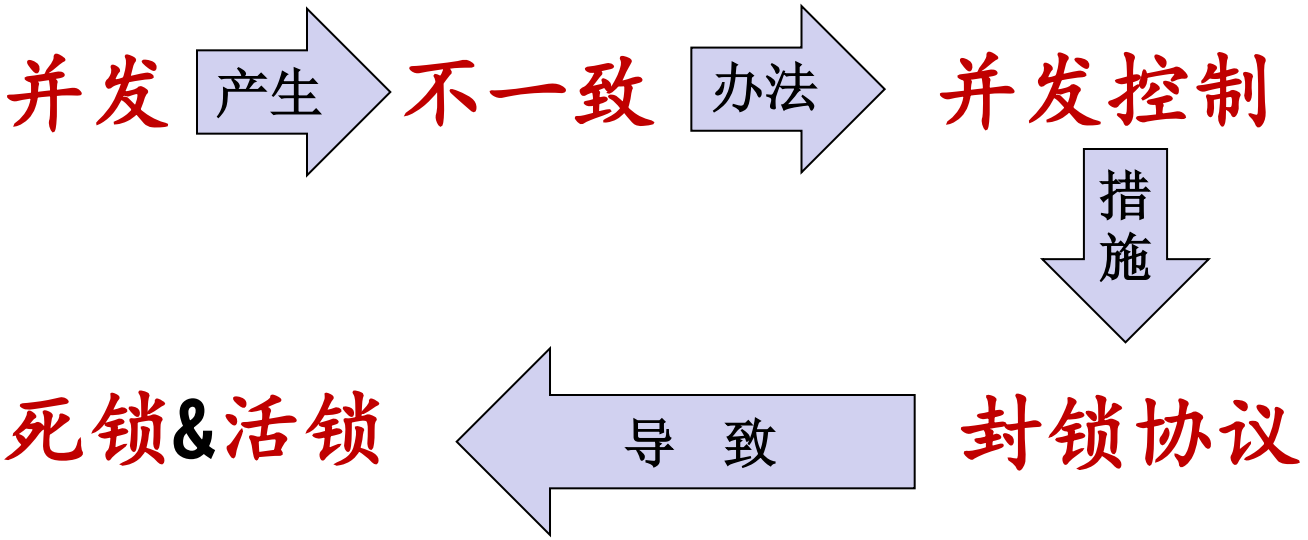
T1	T2
对R1上X锁	
	对R2上X锁
对R2请求X锁 等待	
	对R1请求X锁 等待
⋮	⋮



T1和T2将永远等待下去

死锁避免与诊断。敬请期待……

总结



谢谢！ ！ ！