

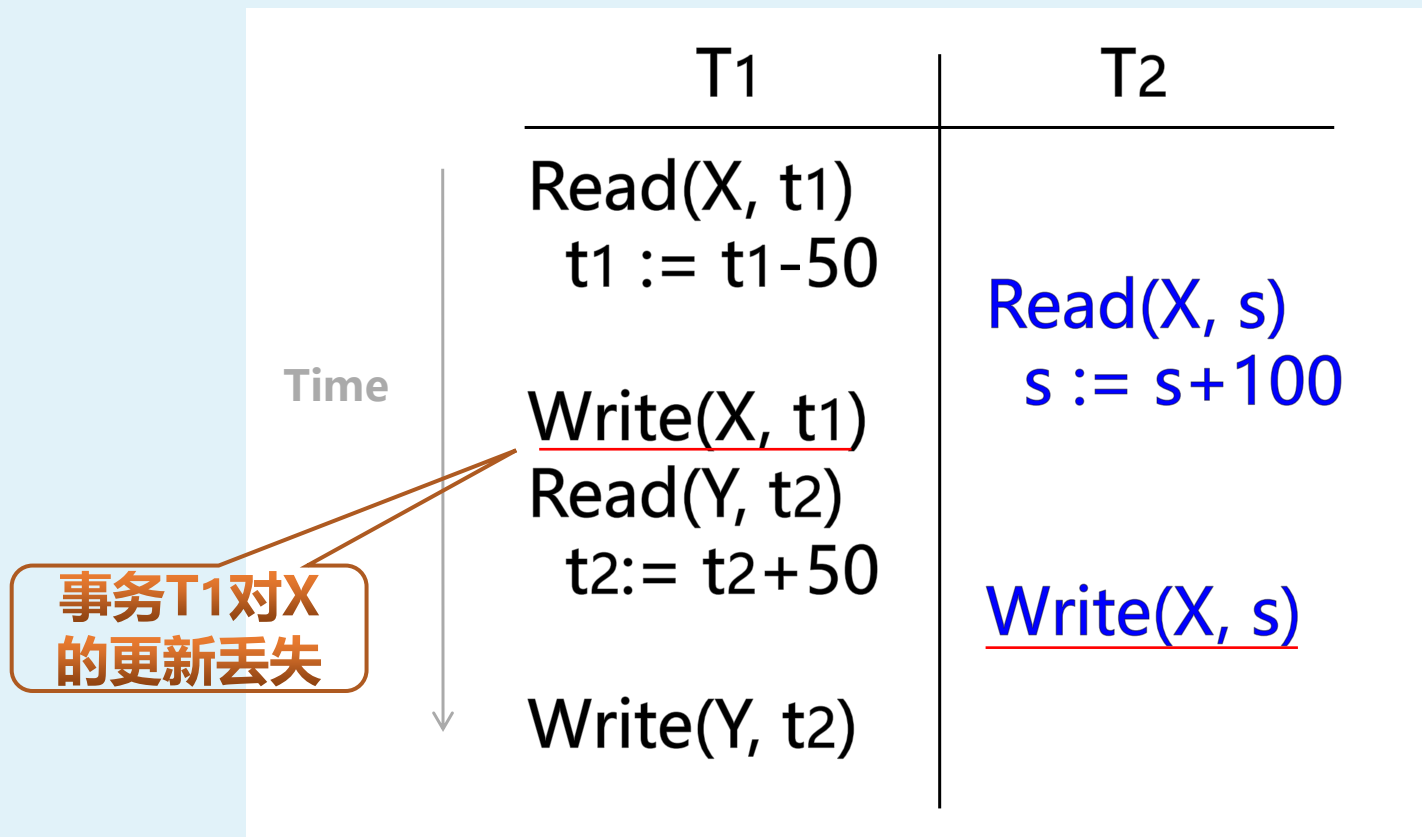
# 死锁

The background of the slide features a series of overlapping, wavy lines in various shades of blue and white, creating a sense of motion and depth. The lines are more densely packed in some areas, forming a grid-like pattern, while in others, they are more sparse and flowing. The overall effect is a modern, technological aesthetic.



# 引言

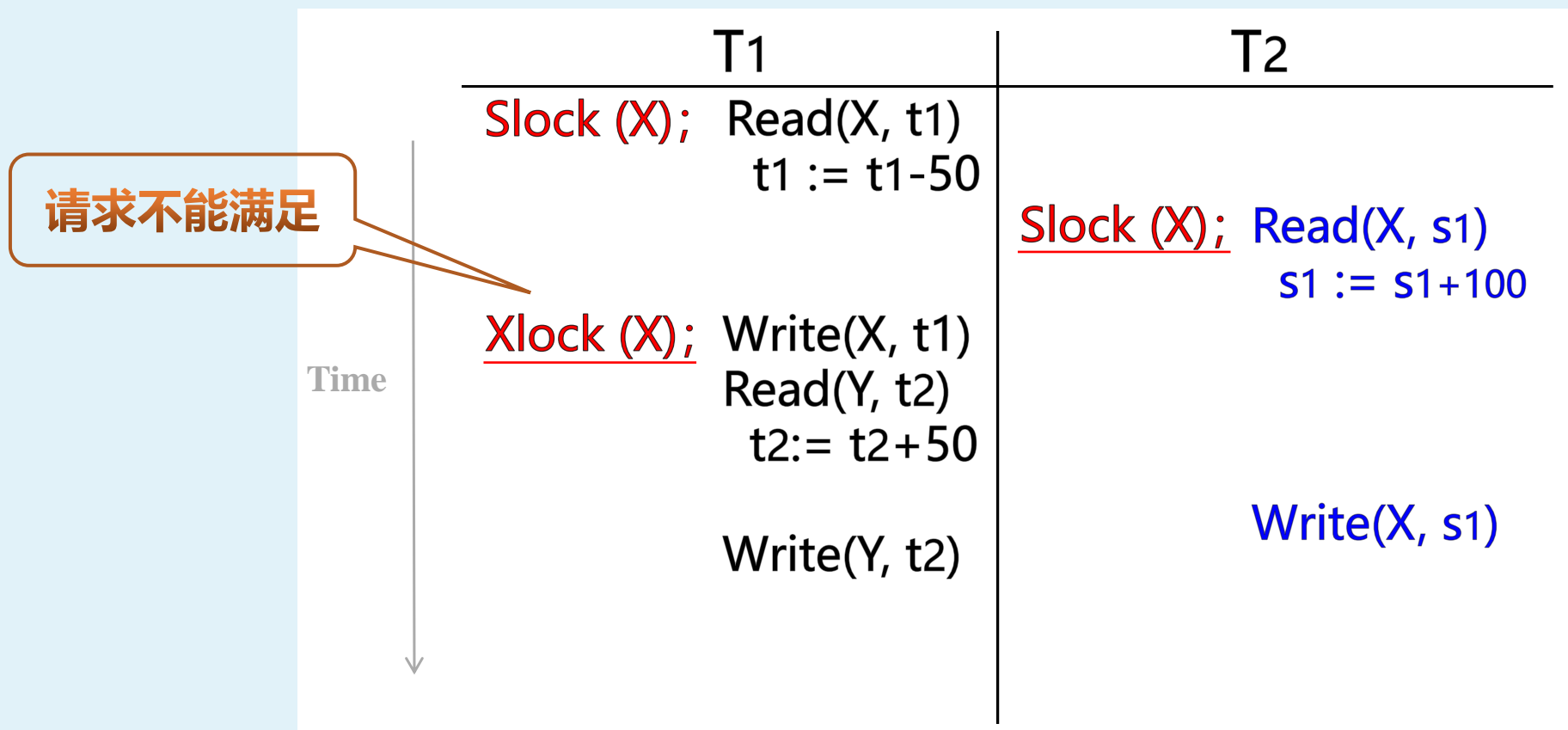
## 更新丢失





# 引言

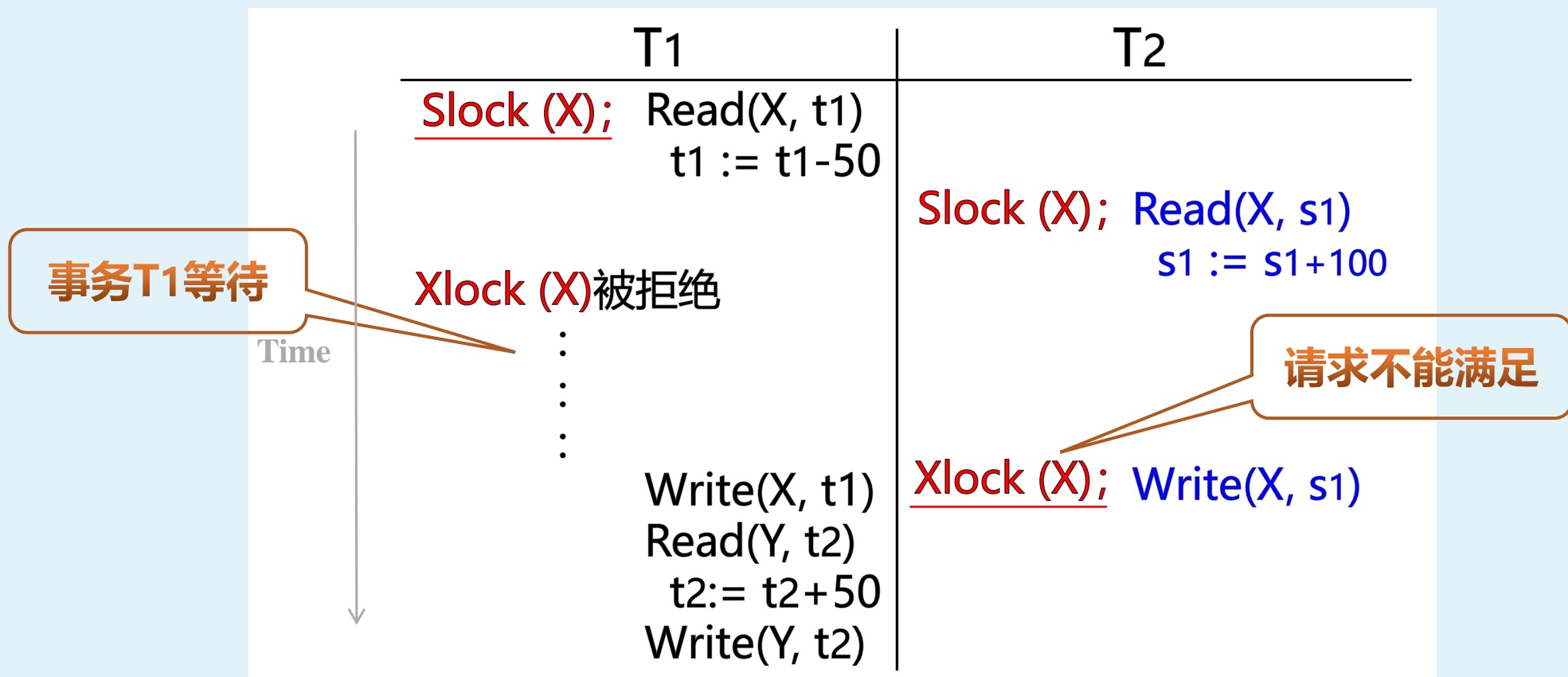
## 利用严格两阶段封锁协议解决“更新丢失”问题





# 引言

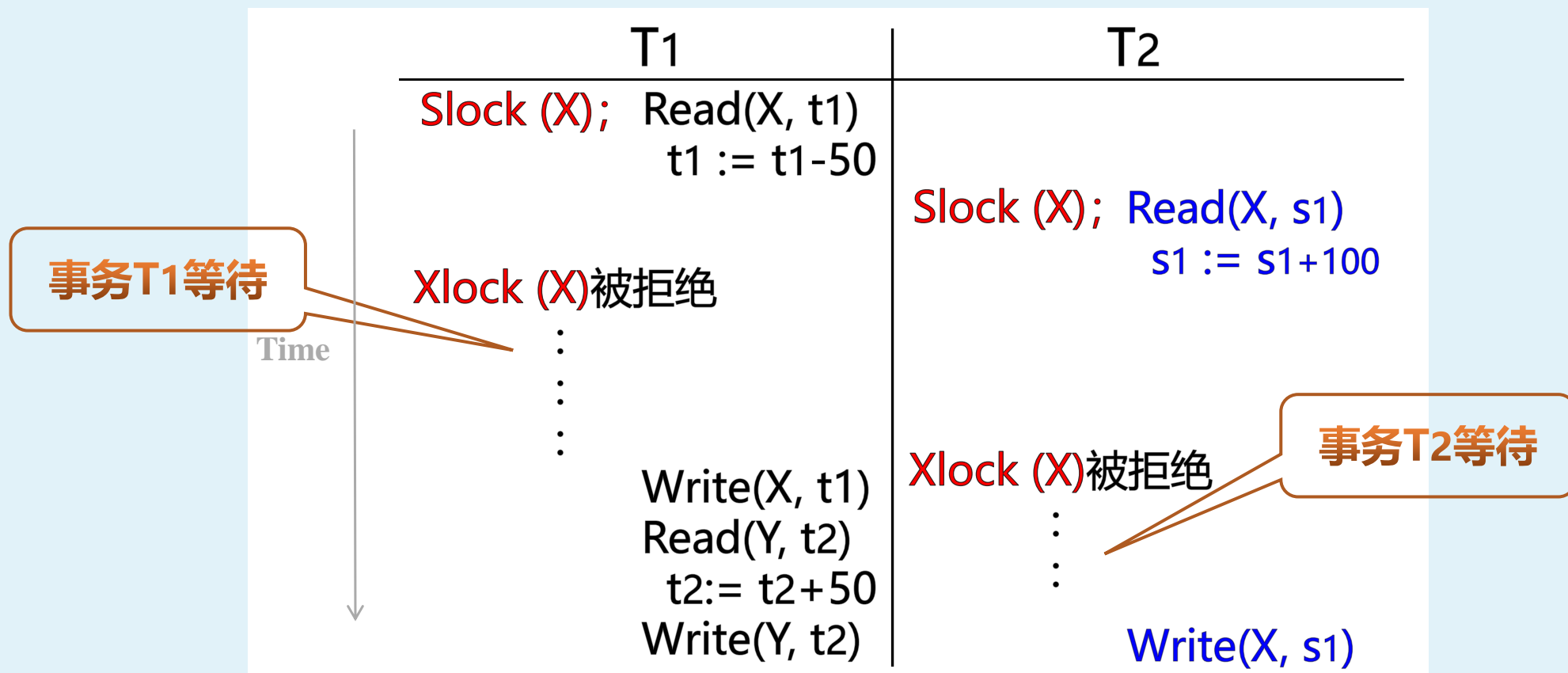
## 利用严格两阶段封锁协议解决“更新丢失”问题





# 引言

## 利用严格两阶段封锁协议解决“更新丢失”问题

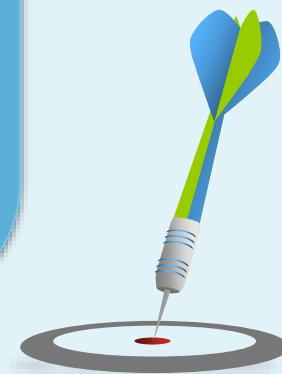






# 讲授内容

- ◆ 1 死锁
  - 活死锁（活锁）
  - 死死锁（死锁）
- ◆ 2 死锁的预防和检测





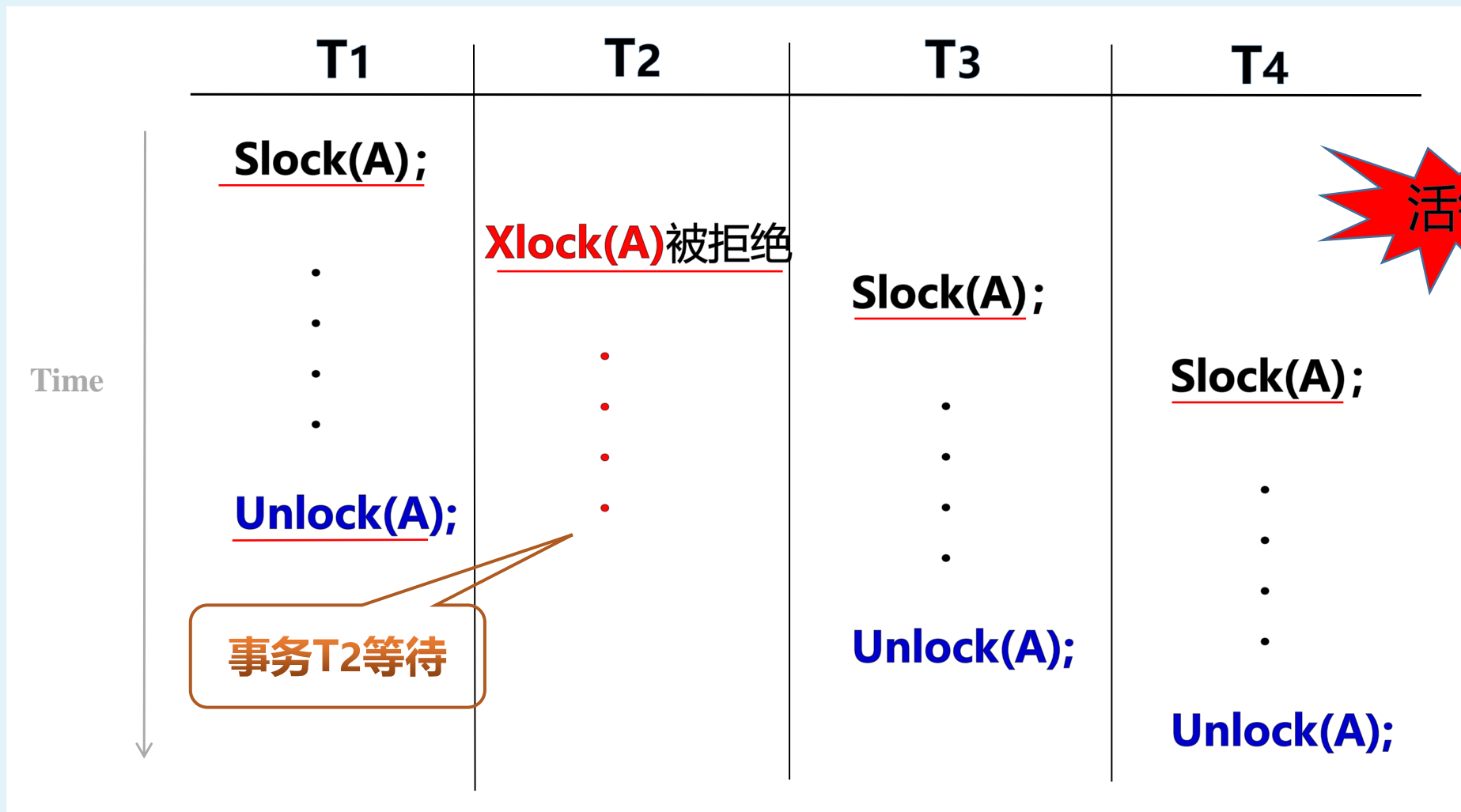
## 活锁

- 并发事务中有部分事务因封锁请求得不到满足而长期等待, 但其他事务仍然可以继续运行下去的状态。

饿死



## 活锁







## 活锁的预防

- 采用**先来先服务**的策略，当多个事务申请封锁同一数据对象时，按申请封锁的先后次序满足事务的封锁请求，来避免发生活锁。



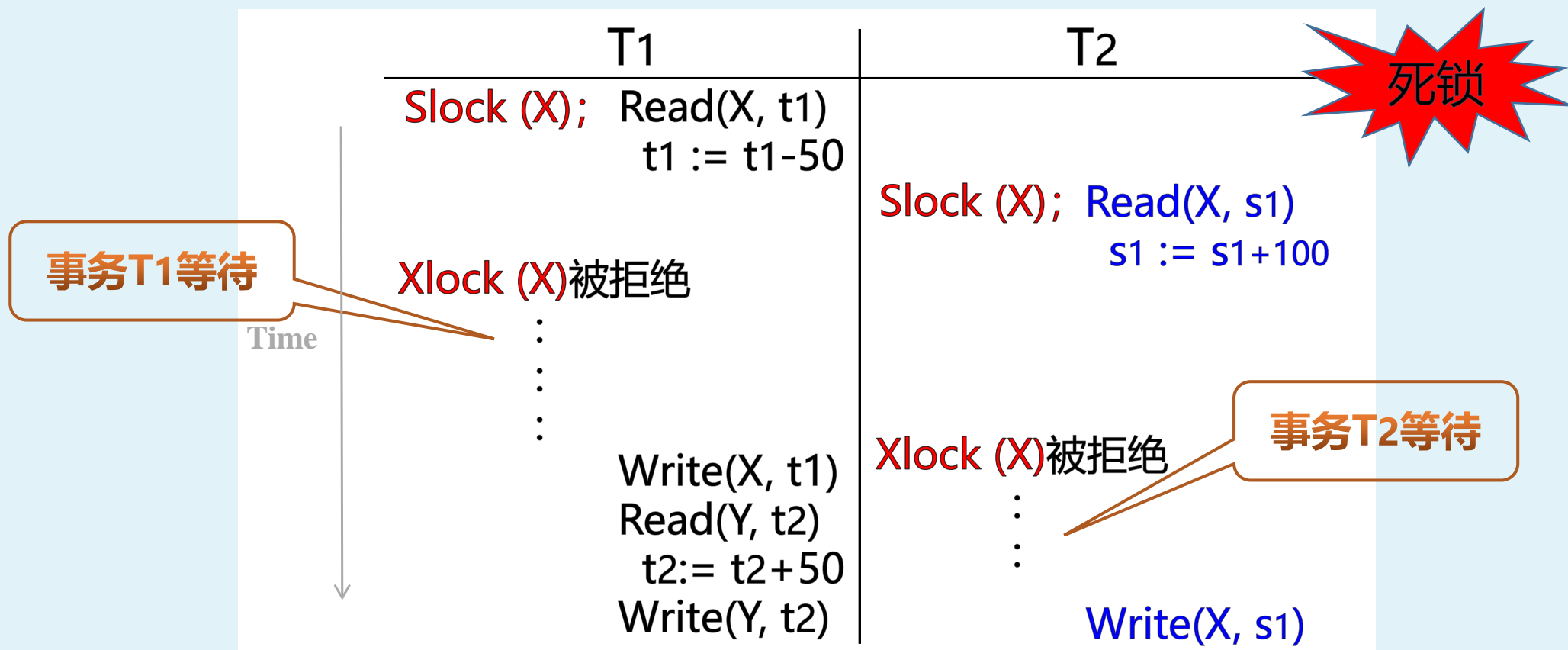
## 死锁

- 并发事务中的事务各自拥有某数据库对象上“锁”，并去申请其他事务对某数据库对象所持有的“锁”，因申请得不到满足而产生的**循环等待**状态。



# 死锁

利用严格两阶段封锁协议解决“更新丢失”问题





## 死锁的预防与检测

- 预防死锁

- 一次封锁法：申请到要访问的所有数据对象上的锁
- 顺序封锁法：按某种顺序申请数据对象上的锁
- 事务等待图法

- 检测并解除死锁

- 超时回滚法：事务的执行时间超过设置的时限即回滚
- 事务等待图法



## 死锁的预防与检测

- 事务等待图是一个有向图。

$$G = (T, U)$$

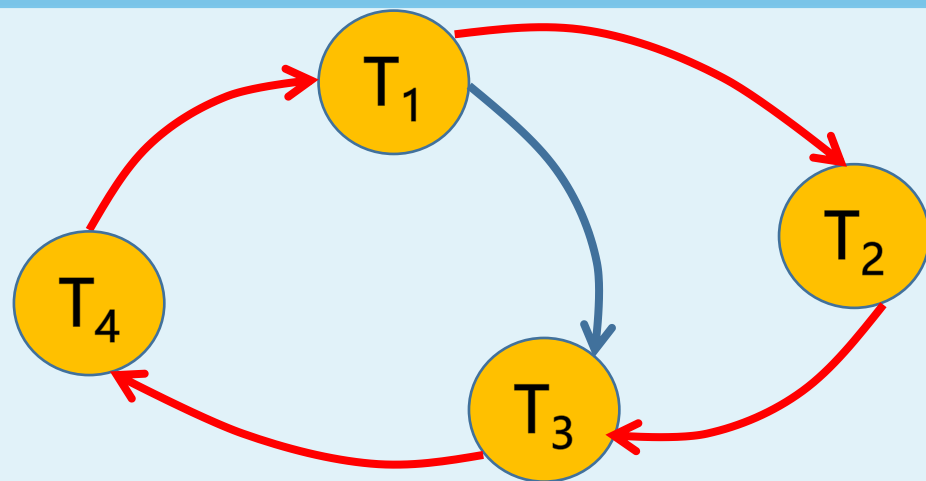
- T为结点的集合，每个结点表示正在运行的事务
- U为边的集合，每条边表示事务间等待锁的情况





## 死锁的预防与检测

- 事务等待图是一个有向图。
- 事务等待图动态反映并发事务申请等待锁的情况。
- 若事务等待图中存在**环路**，表示系统中存在死锁。

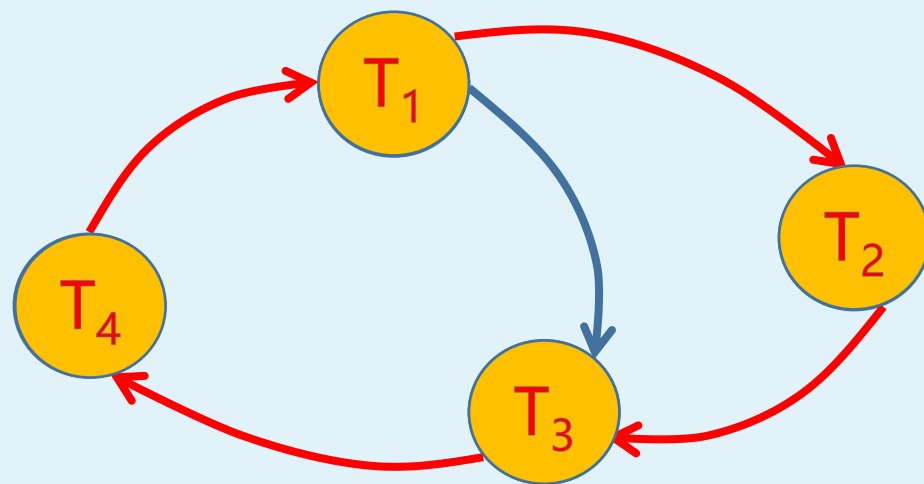






## 死锁的预防与检测

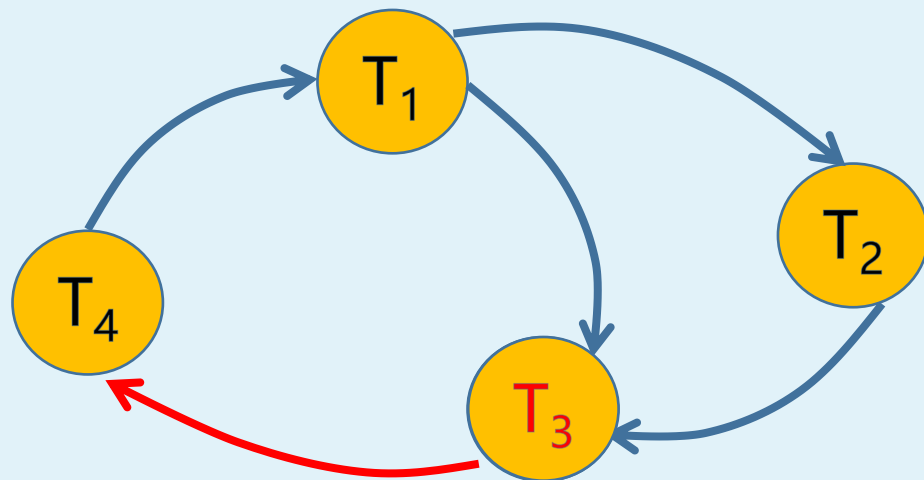
- 若检测到事务等待图中存在**环路**，系统选择环路中一个撤销代价最小的事务，将其回滚。





## 死锁的预防与检测

- 若利用事务等待图预防死锁，可回滚所提请求将导致等待图中出现环路的任一事务。

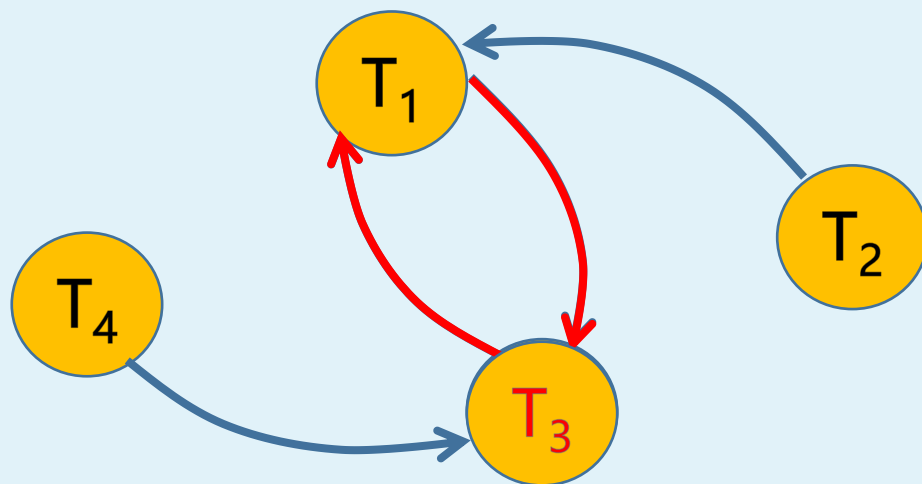




## 死锁的预防与检测

- 对该并发事务的非串行调度，采用严格的两阶段封锁协议进行并发控制。说明在调度的执行过程中，事务等待图的变化，判断是否会发生死锁，给出解锁方案。

$r_1(A)$ ;  $r_3(B)$ ;  $w_1(C)$ ;  $w_3(D)$ ;  $r_2(C)$ ;  $w_1(B)$ ;  $w_4(D)$ ;  $w_3(A)$ ;

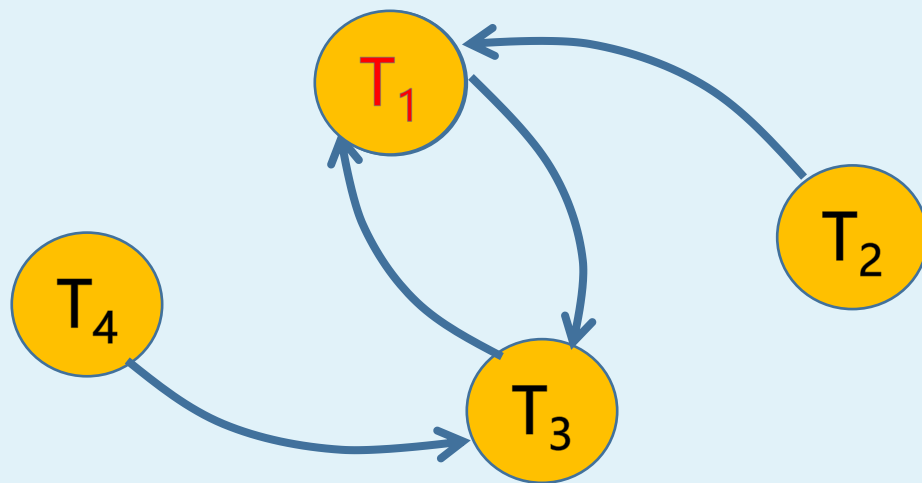




## 死锁的预防与检测



- 对该并发事务的非串行调度，采用严格的两阶段封锁协议进行并发控制。说明在调度的执行过程中，事务等待图的变化，判断是否会发生死锁，给出解锁方案。

$r_1(A)$ ;  $r_3(B)$ ;  $w_1(C)$ ;  $w_3(D)$ ;  $r_2(C)$ ;  $w_1(B)$ ;  $w_4(D)$ ;  $w_3(A)$ ;





## 小结

-  并发事务因竞争不到所需的共享锁资源，处于长期等待或相互等待的**死锁**状态，造成系统性能的降低。
-  大多数DBMS通过判断**事务等待图**是否形成**环路**来避免或解除死锁。