事务-并发控制与恢复

汇报人：王俊

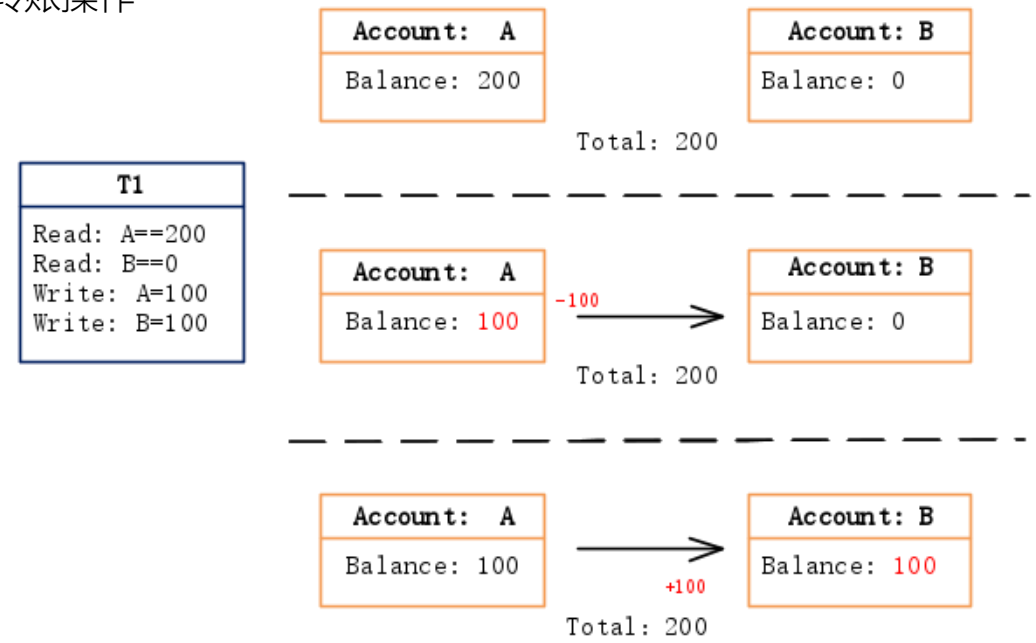
DSC
分布式存储与计算实验室

2022.09.14

事务基础

- 事务的引入：

经典例子：两个人之间转账操作



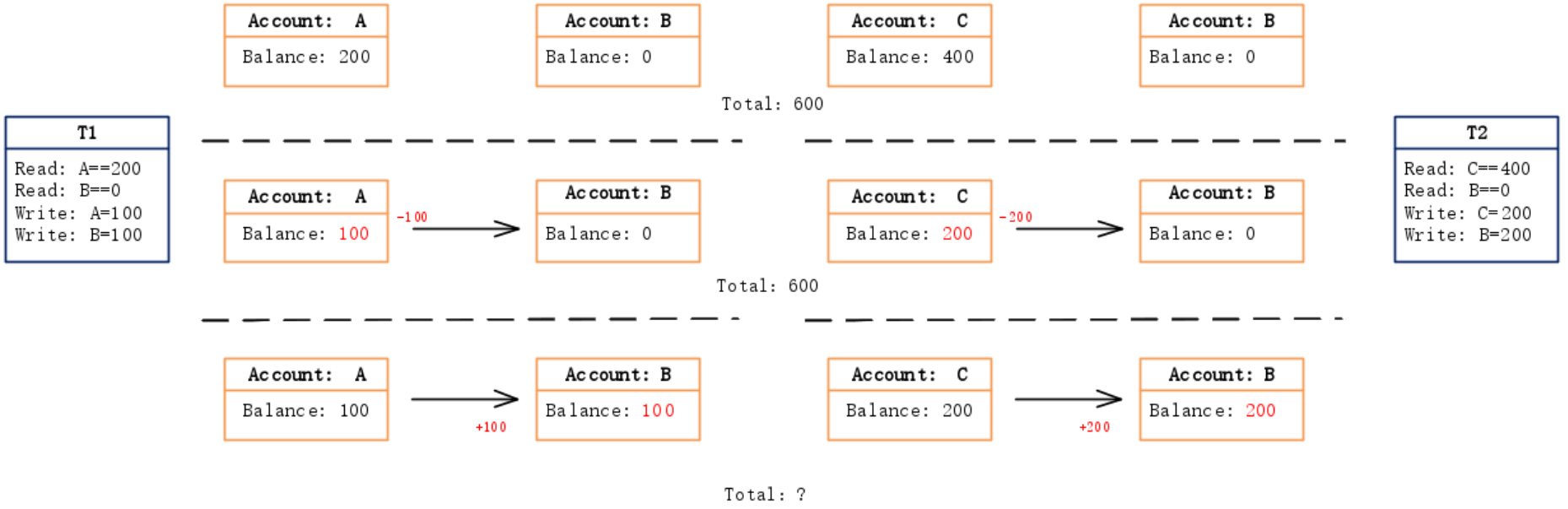
如果要保证正确性，需要满足什么条件？

1. 步骤1和2必须同时成功 或 同时失败
2. 转账操作必须是永久性的

事务基础

- 事务的引入：

例子：多人同时转账



如果要保证正确性，需要满足什么条件？

1. 步骤1和2必须同时成功 或 同时失败 —— 一致性 C
2. 转账操作必须是永久性的 —— 持久性 D
3. 多个同时进行的操作之间不能相互影响 —— 隔离性 I

事务基础

- 事务：是数据库执行过程中的一个逻辑单位，由一个有限的数据库操作序列构成。

- 特性：ACID

1. 原子性

   Write Ahead Log(WAL)

2. 持久性

   WAL

3. 隔离性

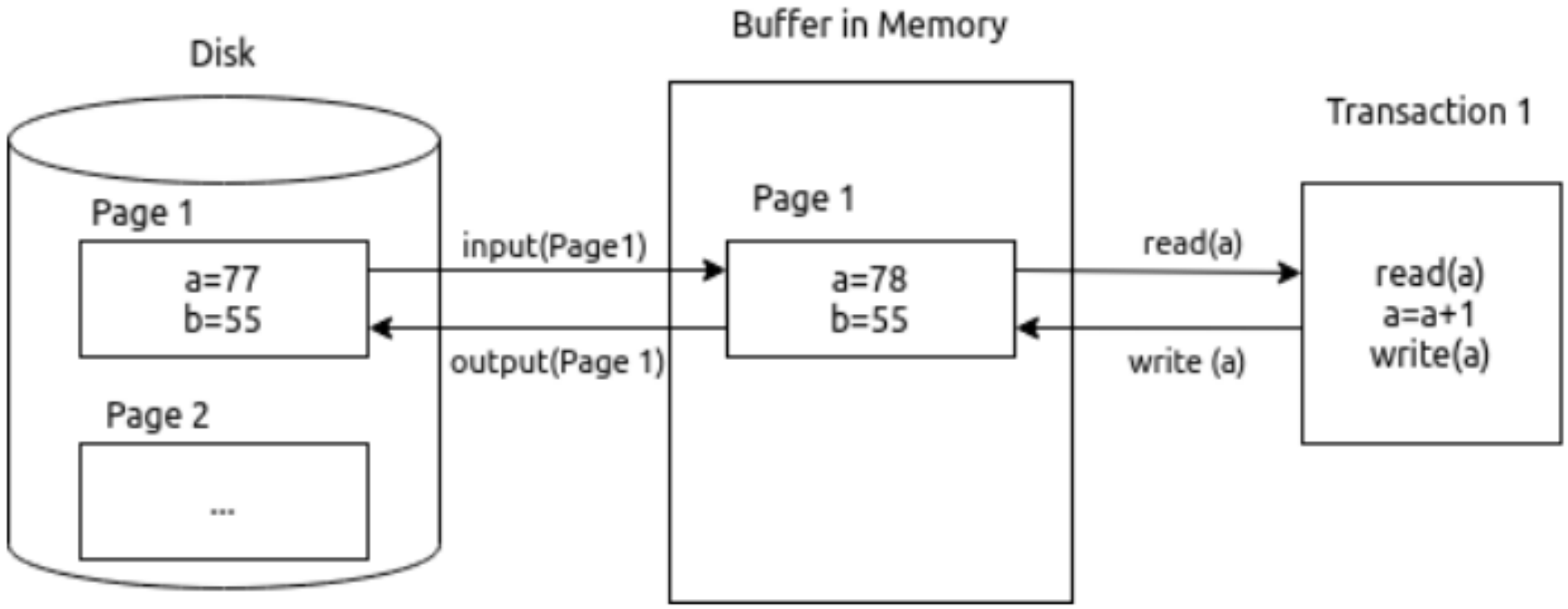
   并发控制算法

4. 一致性

   ???

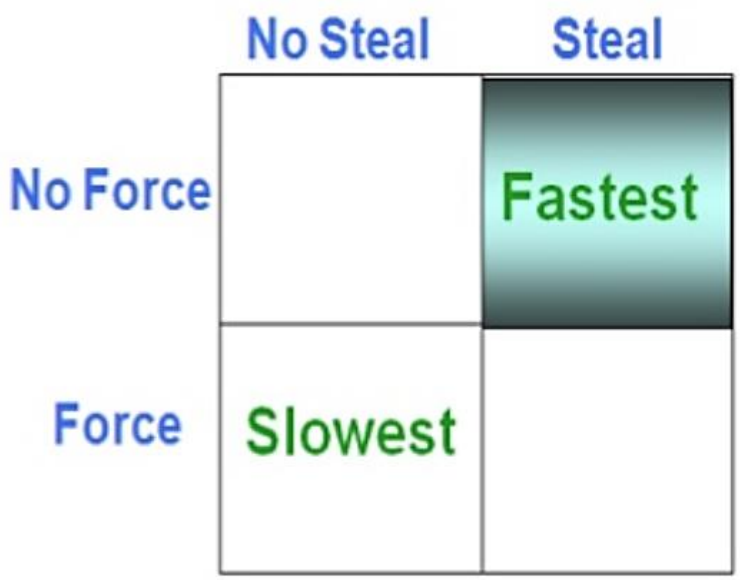
原子性&持久性

数据写入流程



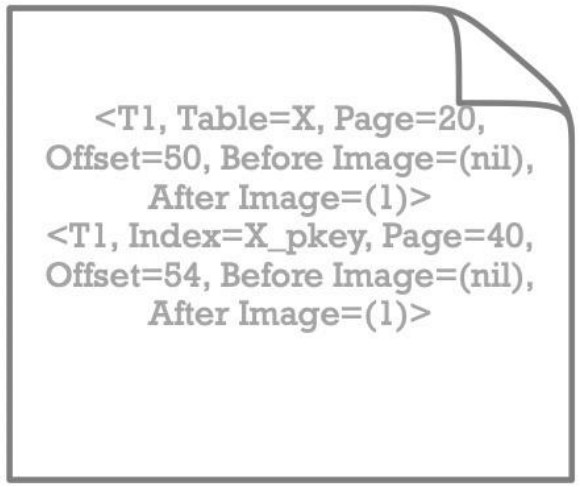
原子性&持久性

- **STEAL**：允许将未提交事务的改动刷盘。
- **NO-STEAL**：与 STEAL 相反。
- **FORCE**：必须在事务提交时把该事务的所有改动刷盘。
- **NO-FORCE**：与 FORCE 相反。

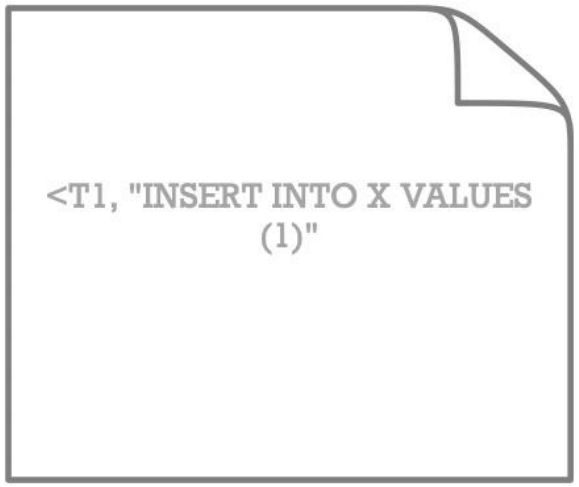


原子性&持久性

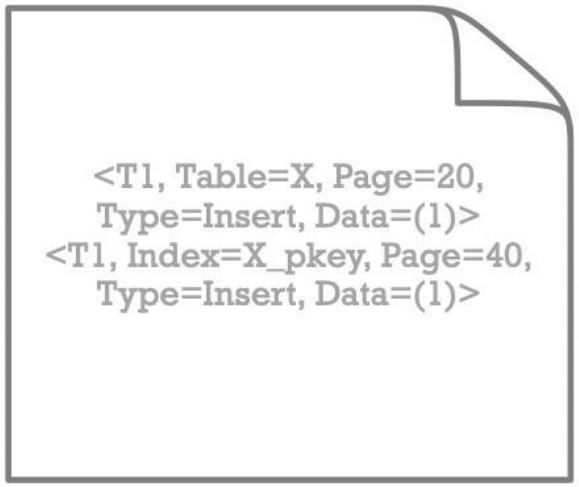
INSERT INTO X VALUES(1);



Physical Log



Logical Log

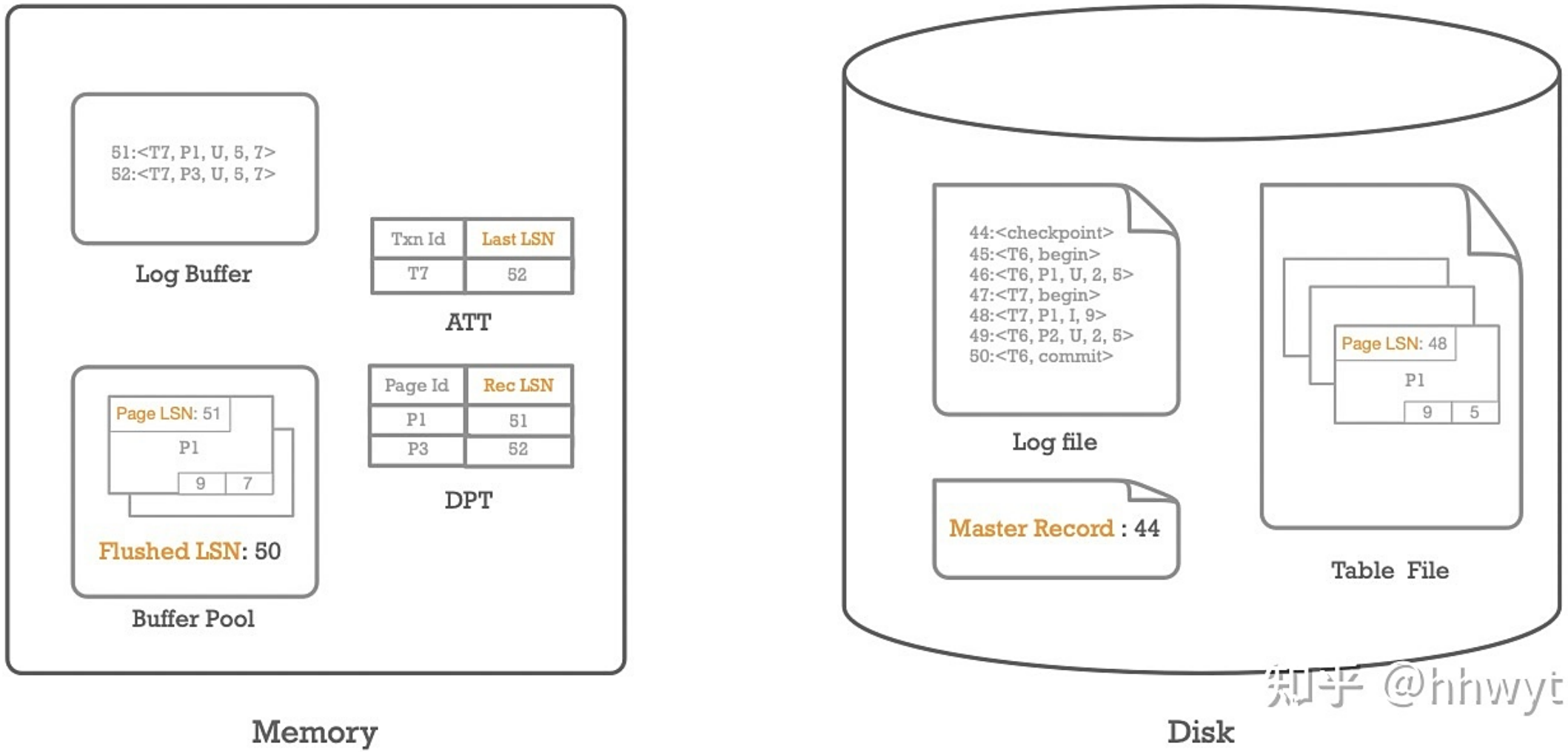


Physiological Log

原子性&持久性

Log 格式	数据量	幂等	Redo 速度	Undo 速度
Physical Log	大	是	快	快
Logical Log	小	否	不支持 Redo	慢
Physiological Log	中	否	快	快

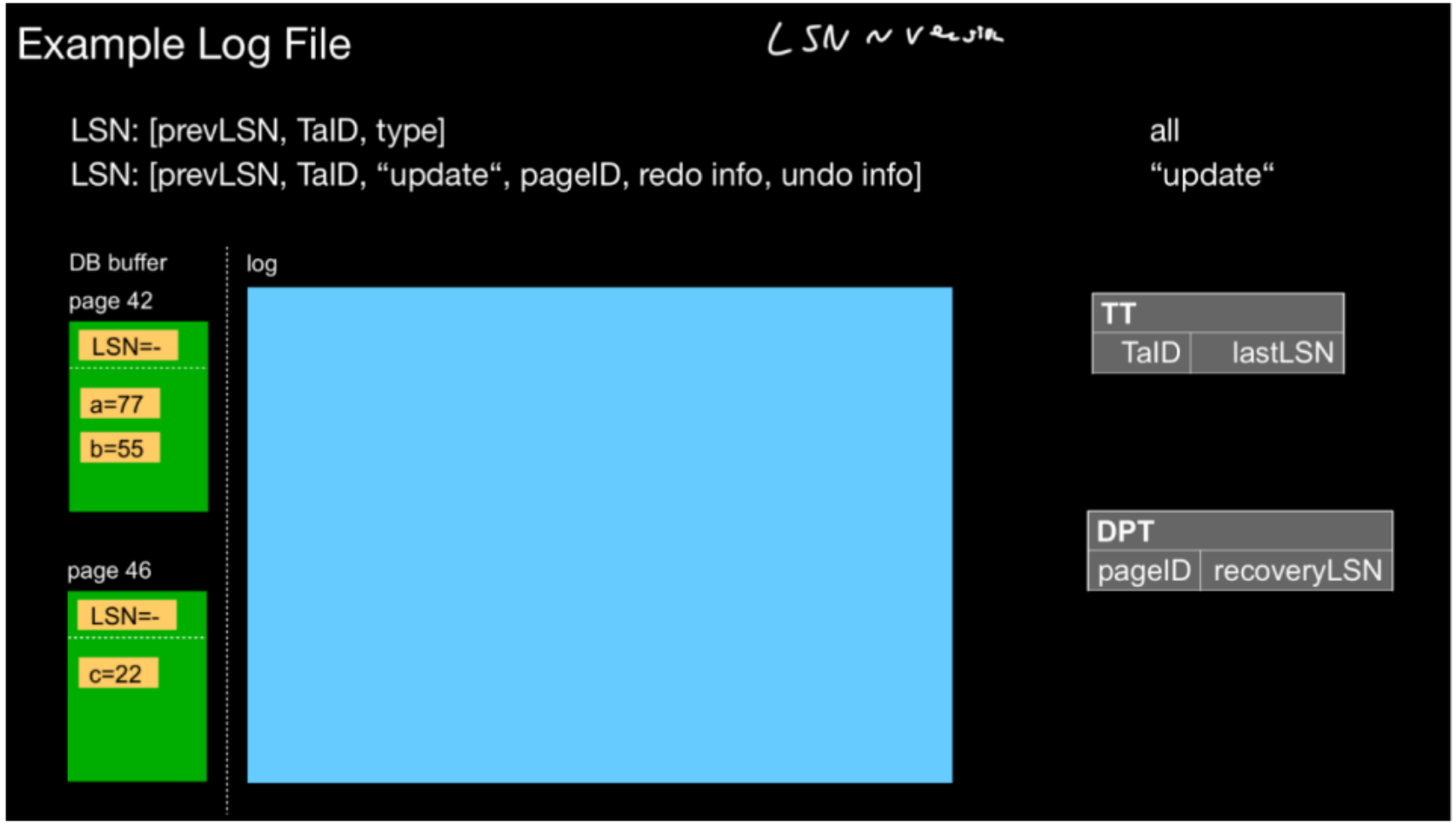
原子性&持久性



原子性&持久性


Time
---------------->
需要撤销事务
最早可能的修改
最早可能出
现的脏页面
CRASH
LOGS(WAL)
1. Analysis
2. Redo
3. Undo

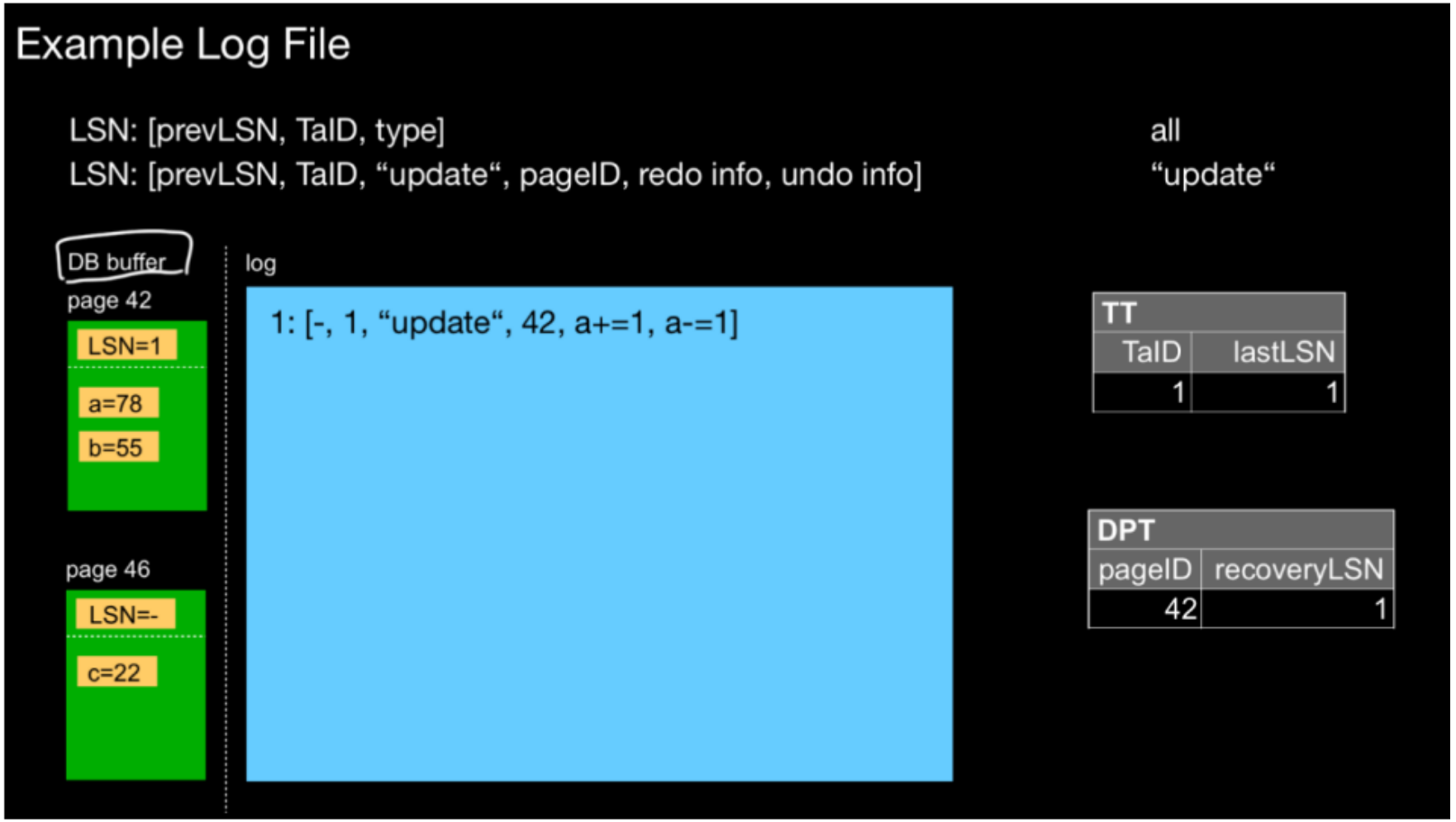
原子性&持久性


Example Log File
LSN: [prevLSN, TaID, type]
LSN: [prevLSN, TaID, “update“, pageID, redo info, undo info]
all
“update“
DB buffer
page 42
LSN=-
a=77
b=55
page 46
LSN=-
c=22
log
TT
TaID
lastLSN
DPT
pageID
recoveryLSN

初始状态

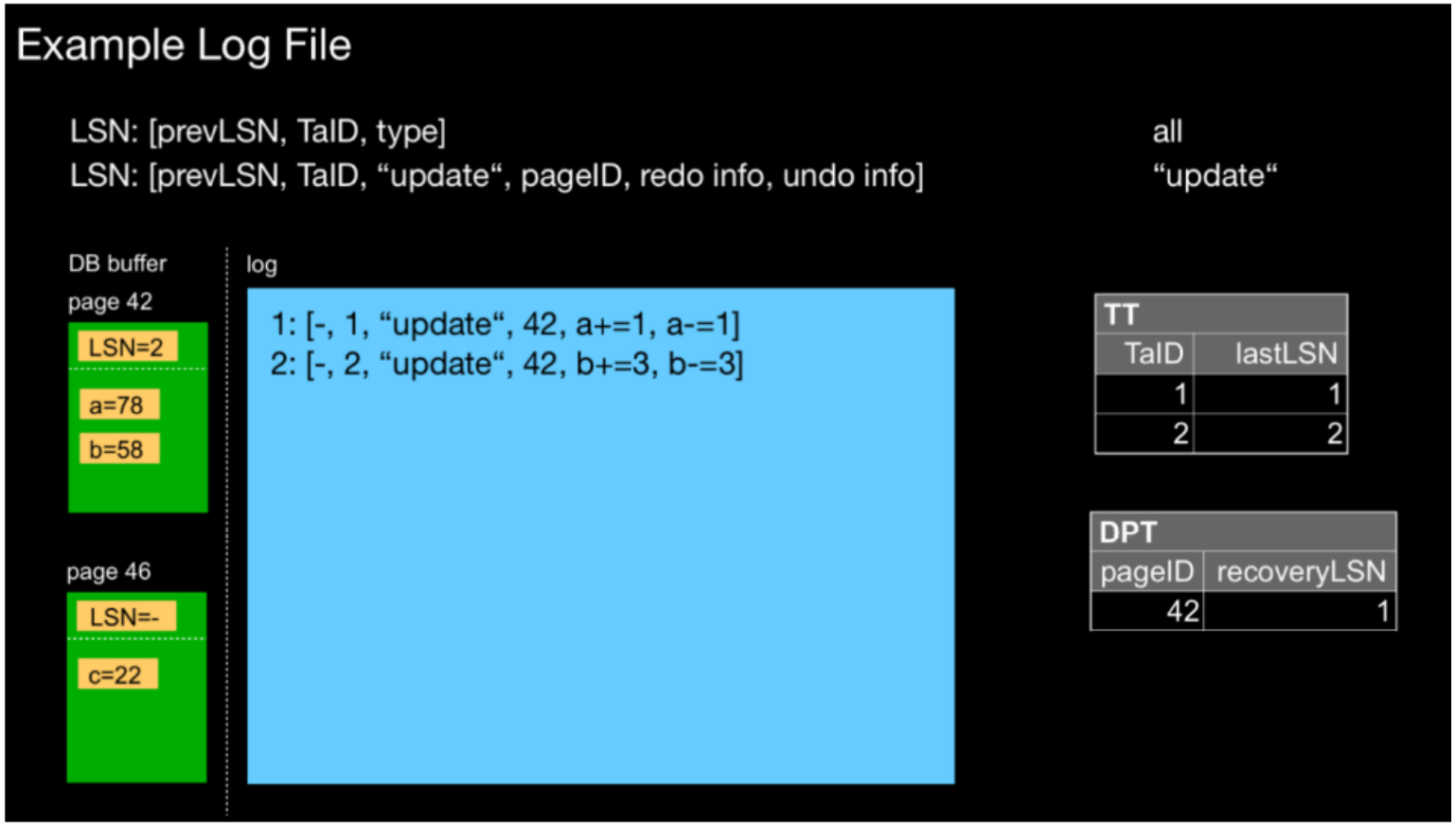
原子性&持久性

1. 在 TT 插入 T1。
2. 写更新日志。
3. 将 Page 42 插入 DPT表
4. 更新缓冲区，设置 PageLSN 为1，同将 a+=1



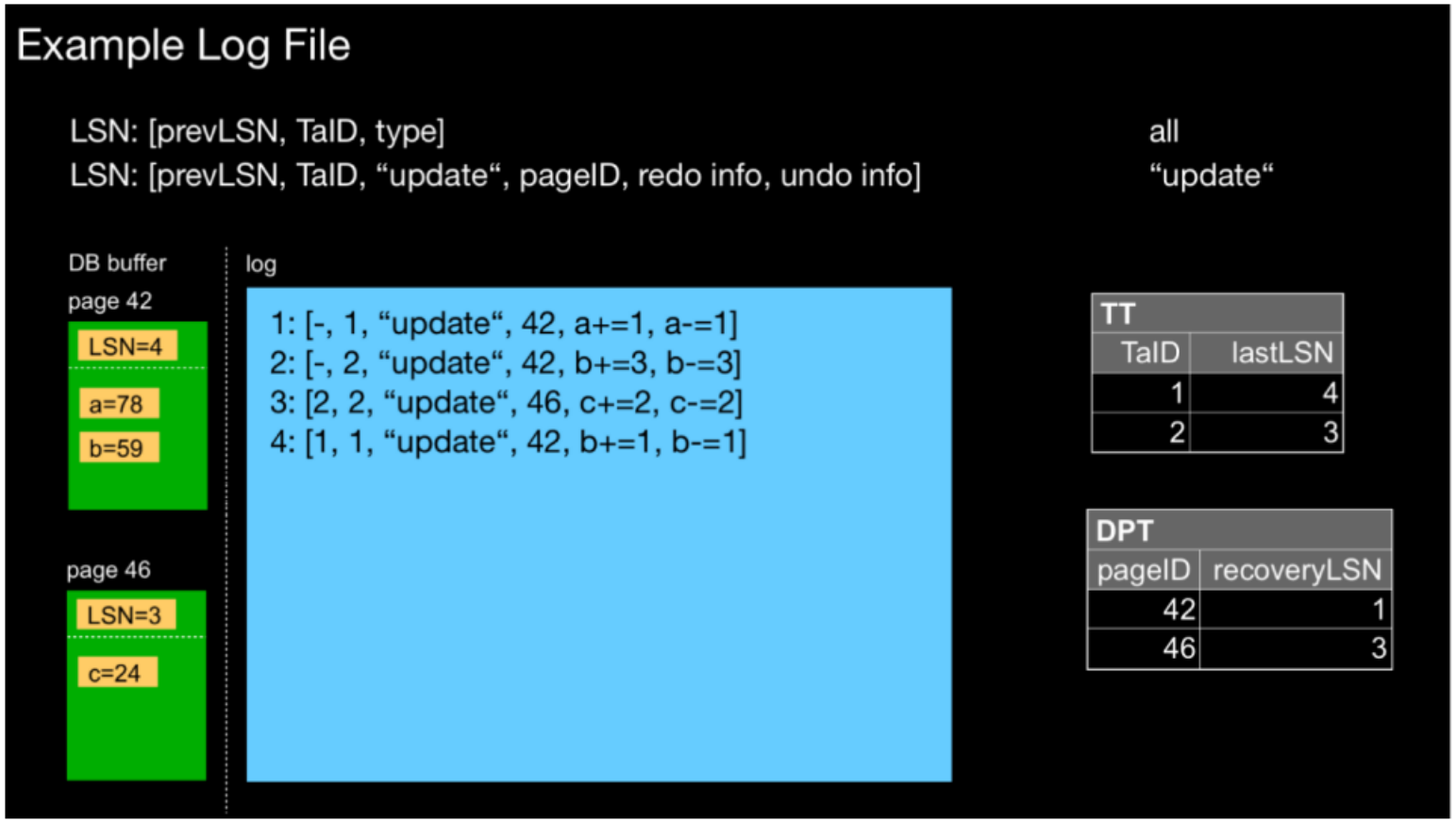
原子性&持久性

1. 在 TT 插入 T2。
2. 写更新日志。
3. 因为 Page 42 已在 DPT，所以不变。
4. 更新缓冲区，设置 PageLSN 为2，同将 b+=3



原子性&持久性

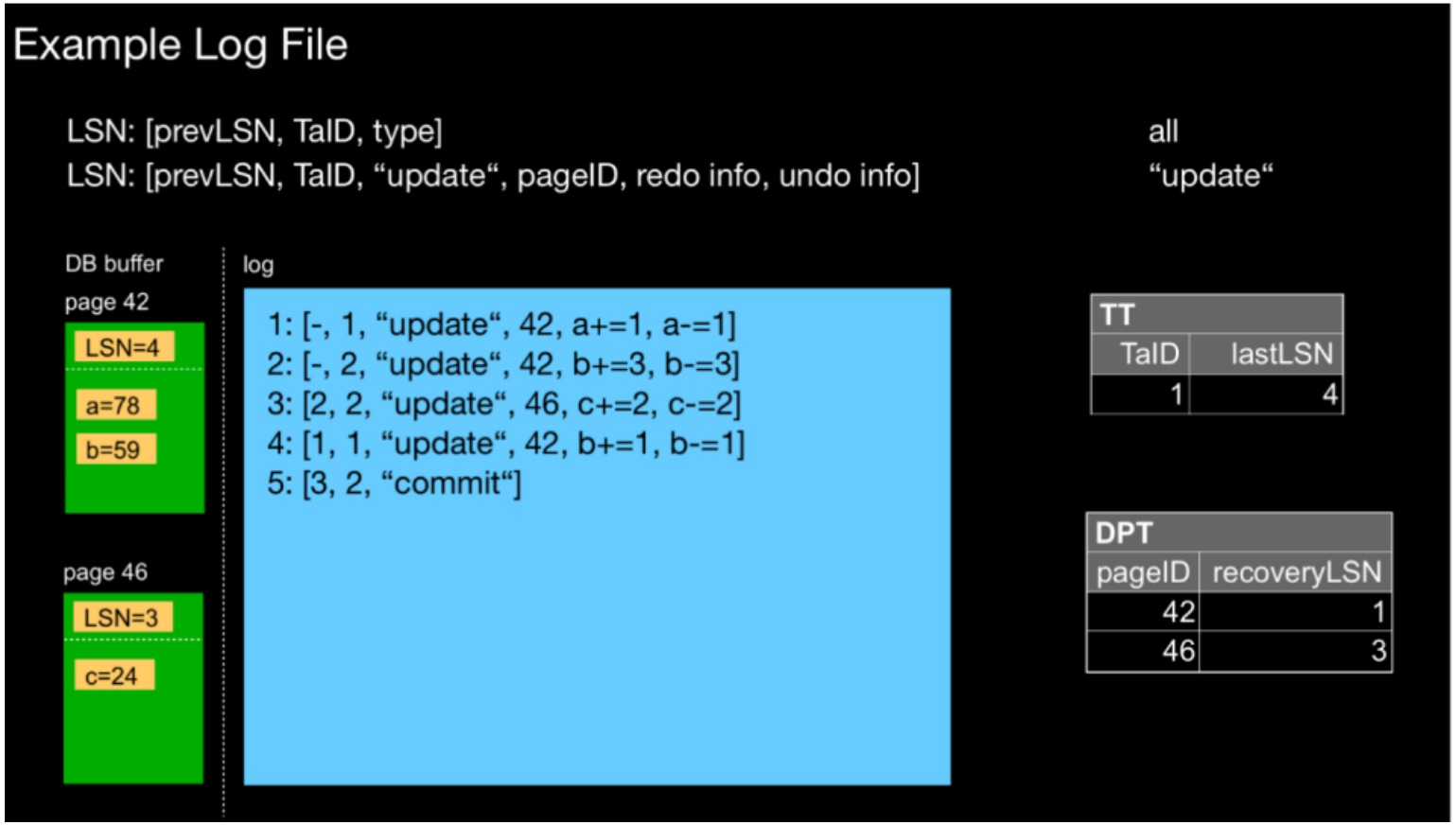
执行两条更新操作，由于 LSN3 更新 Page46，所以将 Page46 加入 DPT 表。



原子性&持久性

T2 提交，将 T2 从 TT 移除。

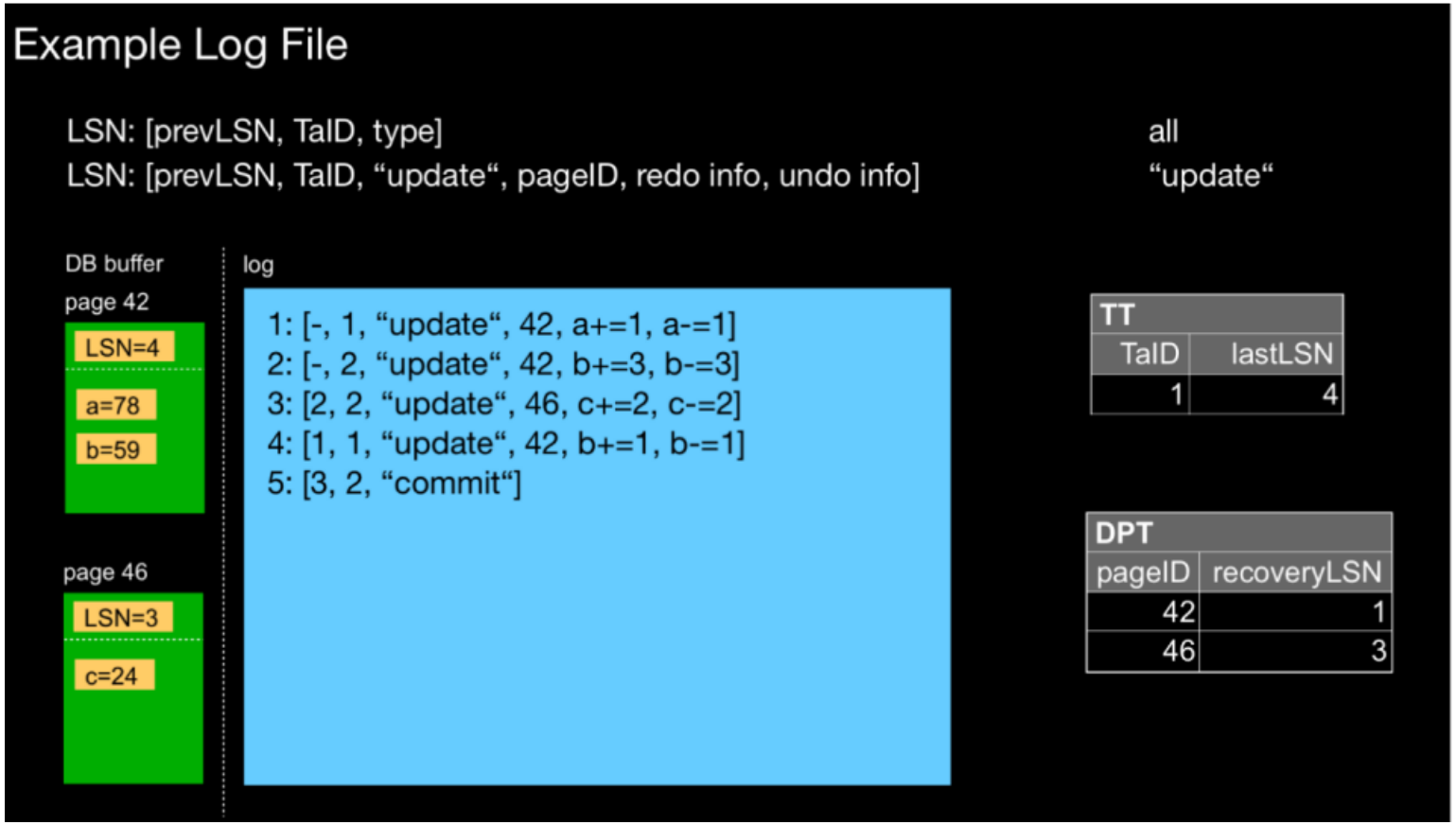
此时系统崩溃。



原子性&持久性

T2 提交，将 T2 从 TT 移除。

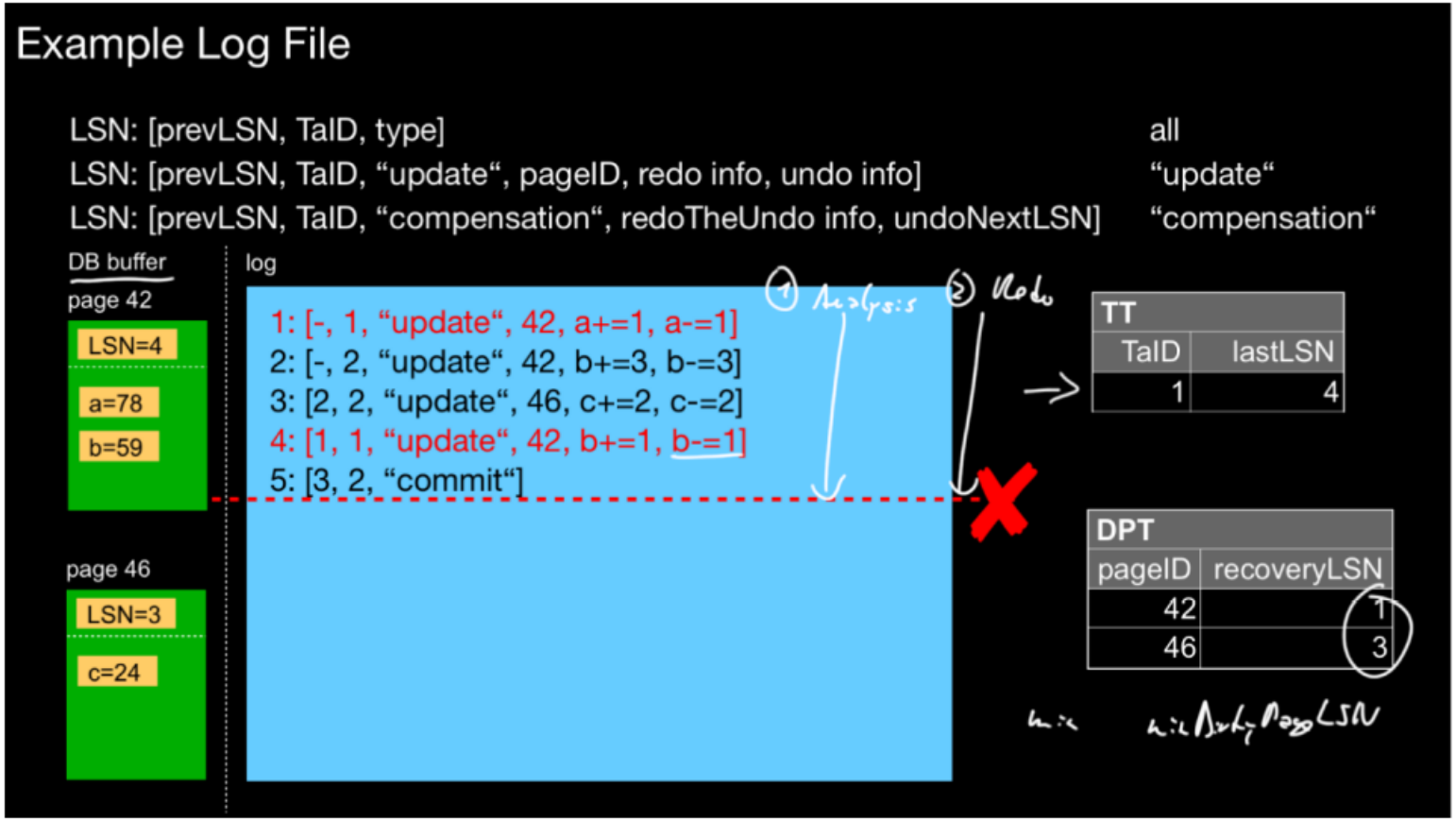
此时系统崩溃。



原子性&持久性

T2 提交，将 T2 从 TT 移除。

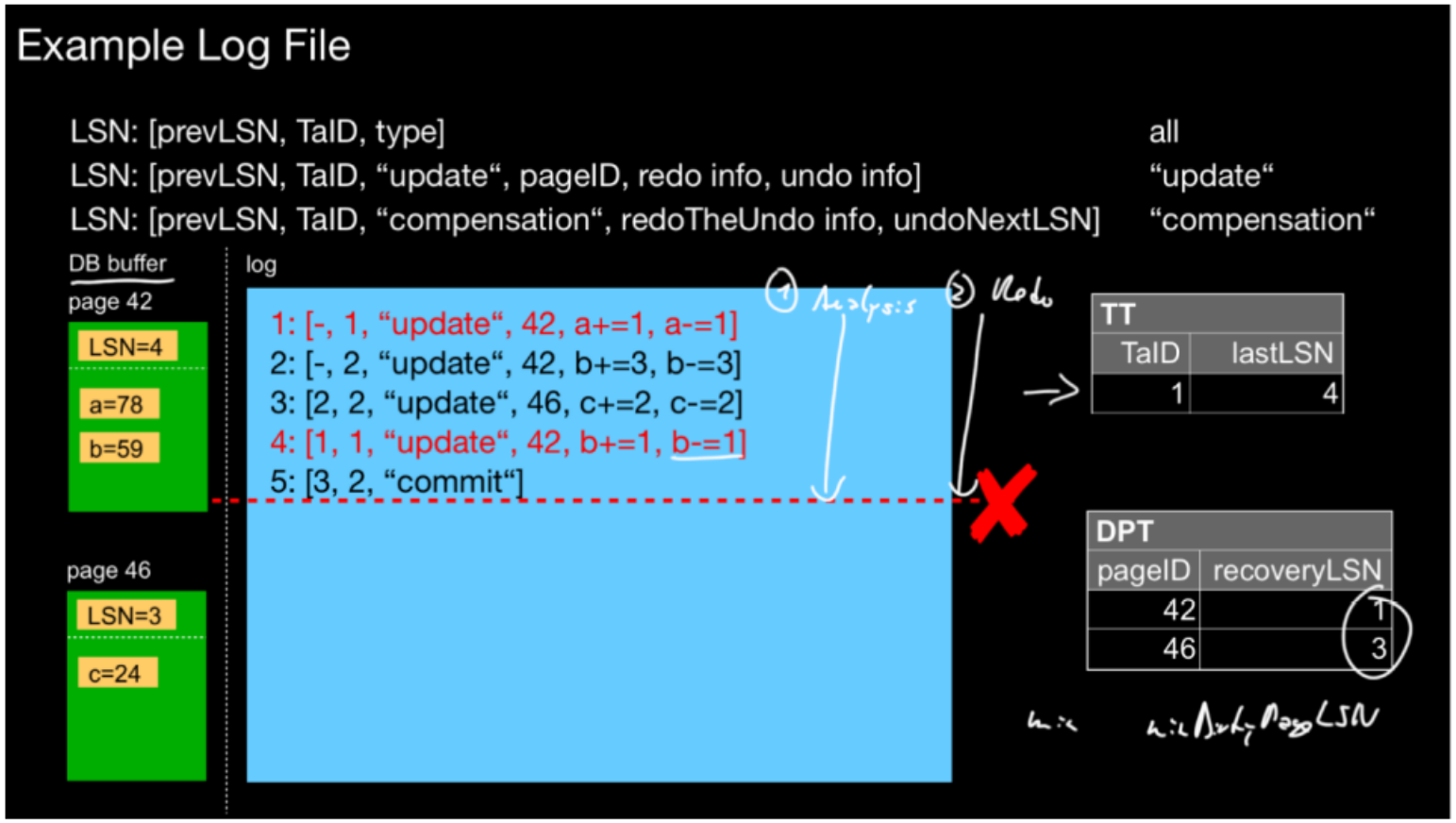
此时系统崩溃。



原子性&持久性

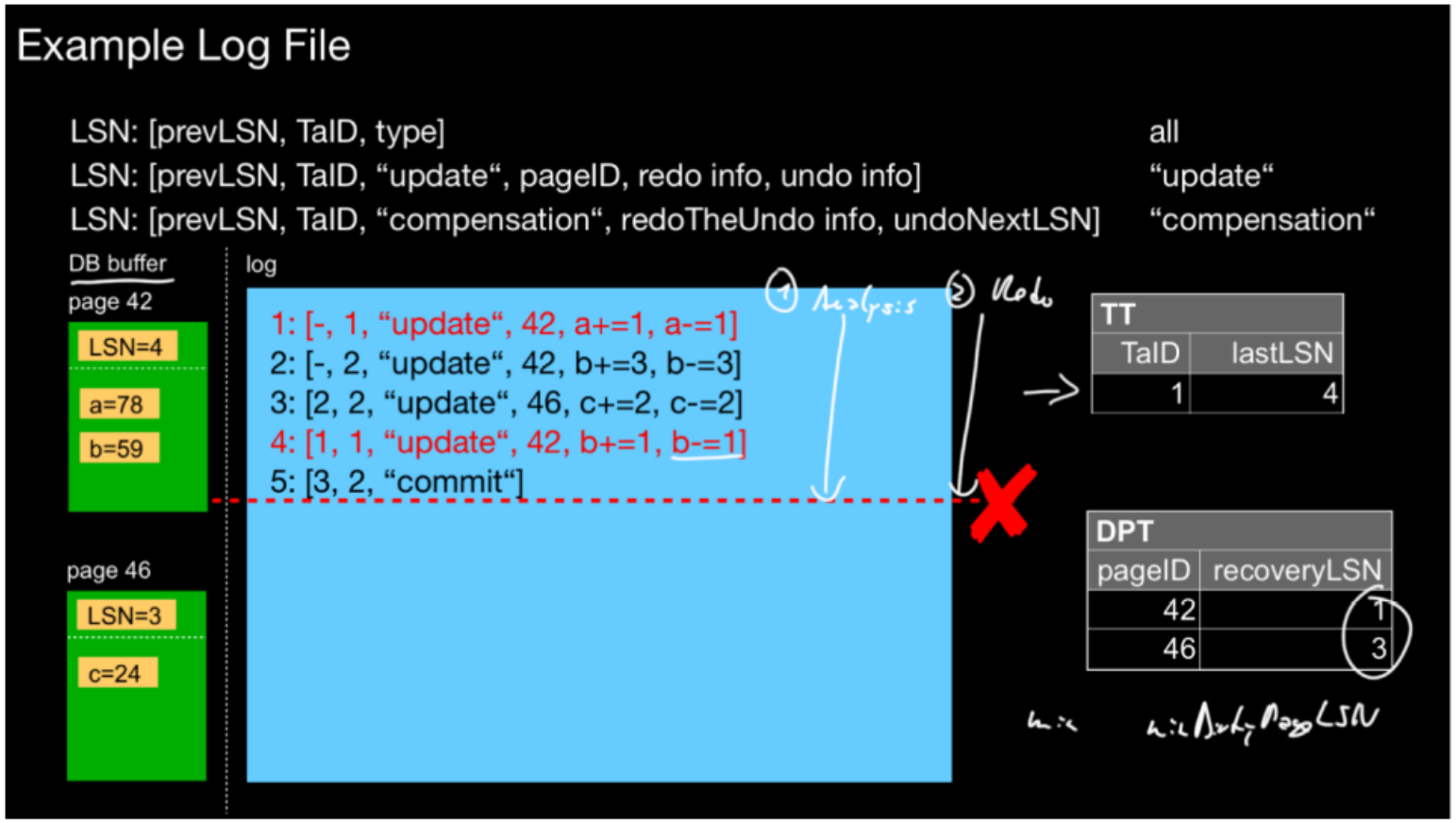
- ATT：
  - 遇到 Ti 的 Begin Log，就把 Ti 加入 TT，同时把状态设为 Undo Candidate。
  - 遇到 Ti 的 Commit Log，将 Ti 从 TT 中移除。
  - 遇到 Ti 的其他 Log，更新 Ti 的 Last LSN。
- DPT：

  遇到任意 Redo Log，如果 Page
  - 不在 DPT 中：将它加入 DPT，同时记录 Rec LSN。
  - 已经在 DPT 中：无需处理。



原子性&持久性

- Redo
  - 找到 DPT 中最小的 Rec LSN，将它作为起始点，顺序扫描 Log 并处理来重放历史。
  - 当且仅当 Log LSN > Page LSN，一个 Log 对应的更新操作才能在对应的 Page 上 Redo。
- Undo
  - 找到 ATT 中最大的 Last LSN，Undo 它对应的事务，
  - 把该事务从 ATT 中移除。
  - 重复上述步骤，直到 ATT 为空。



原子性&持久性

模糊检查点

- 写日志
- 为TT和DPT表创建一份副本（写日志时的版本），可以使用COW实现，以减小代价
  - TT_{ckpt} 和 DPT_{ckpt}
  - TT_{now} 和 DPT_{now}
- 事务继续运行，执行更新操作时修改 TT_{now} 和 DPT_{now}
- 准备好信息后，写日志
- 写入磁盘后，更新磁盘上的MasterRecord=日志的LSN

隔离性

- 隔离级别

Table 4. Isolation Types Characterized by Possible Anomalies Allowed.

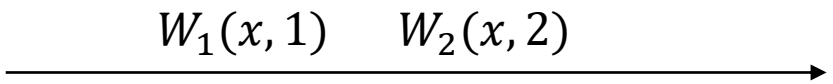
Isolation level	**P0** Dirty Write	**P1** Dirty Read	**P4C** Cursor Lost Update	**P4** Lost Update	**P2** Fuzzy Read	**P3** Phantom	**A5A** Read Skew	**A5B** Write Skew
READ UNCOMMITTED == Degree 1	Not Possible	Possible	Possible	Possible	Possible	Possible	Possible	Possible
READ COMMITTED == Degree 2	Not Possible	Not Possible	Possible	Possible	Possible	Possible	Possible	Possible
Cursor Stability	Not Possible	Not Possible	Not Possible	Sometimes Possible	Sometimes Possible	Possible	Possible	Sometimes Possible
REPEATABLE READ	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible	Possible	Not Possible	Not Possible
Snapshot	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible	Sometimes Possible	Not Possible	Possible
ANSI SQL SERIALIZABLE == Degree 3 == Repeatable Read Date, IBM, Tandem, ...	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible

隔离性

- Dirty Write（脏写）：

脏写是两个事务同时写一个 key 发生冲突的现象，可能出现的异常有两点：

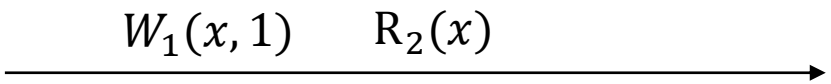
- 当 T1 先成功提交，T2 随后发生 rollback 时，应该回滚到哪个值是不明确的；
- 如果还写了其他的 key，可能会破坏约束的一致性。



- Dirty Read（脏读）：

脏读是一个事务写 key，一个事务读相同的key 发生的现象，可能出现的异常有：

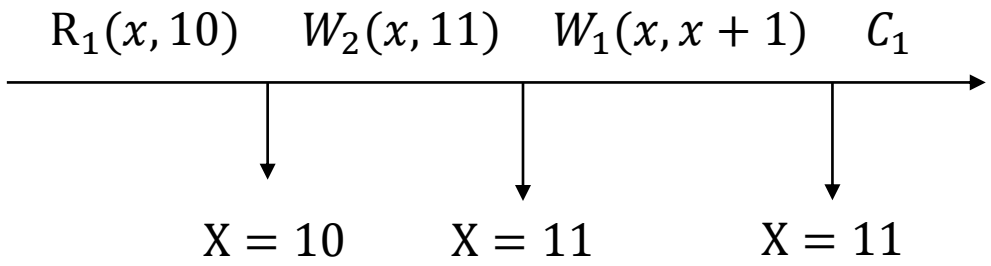
- 当 T1 发生rollback，T2 读到了不存在的值



隔离性

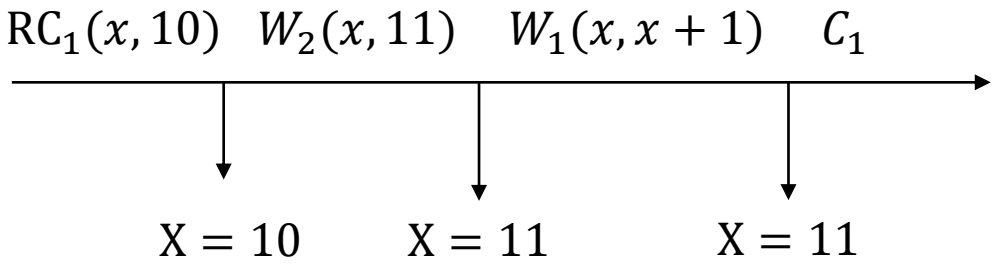
- Lost Update（更新丢失）：

  更新丢失指的覆盖掉其他事务的写现象



- Cursor Lost Update（游标更新丢失）：

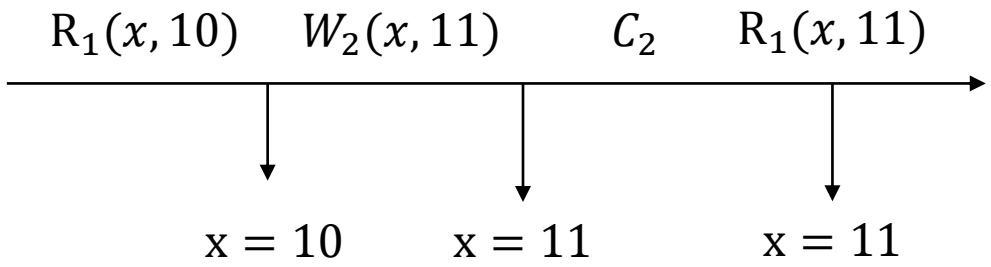
  与更新丢失一致，只是约束在了游标操作的时候。



隔离性

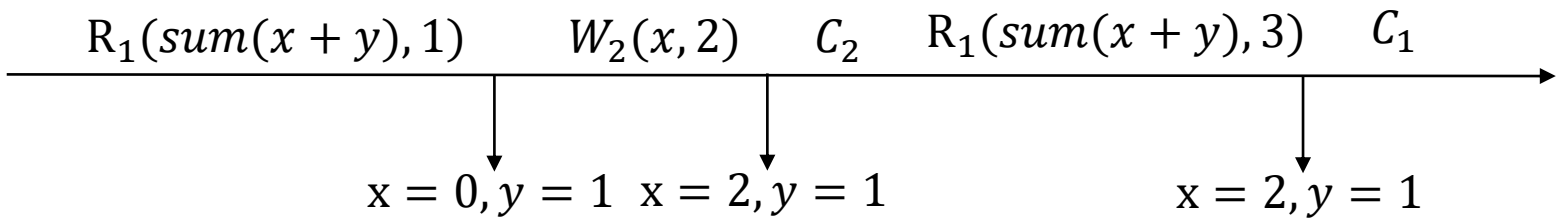
- Non-repeatable(Fuzzy) Read（不可重复读）：

  Non-repeatable Read 指的是两次相同key类型的读操作读到了不同的数据



- Phantom（幻读）：

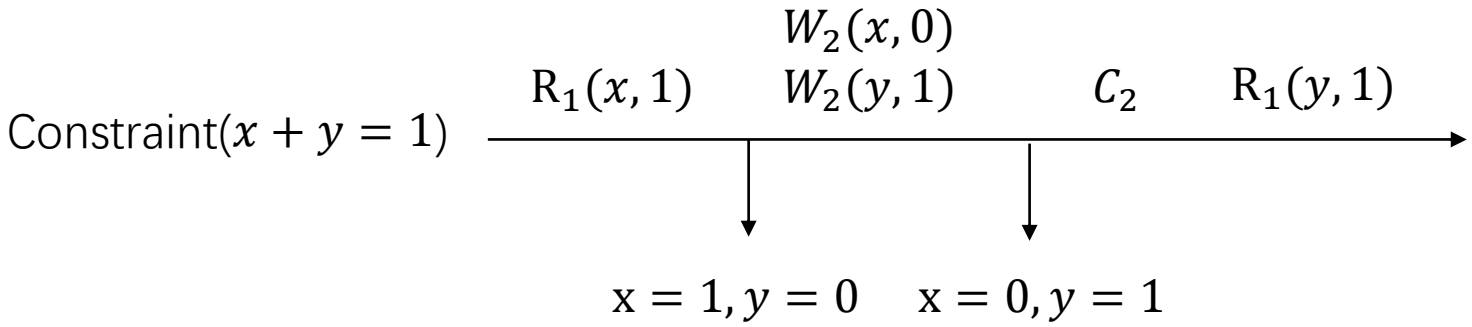
  两次 predicate（谓词） 类型的读操作读到了不同的数据。



隔离性

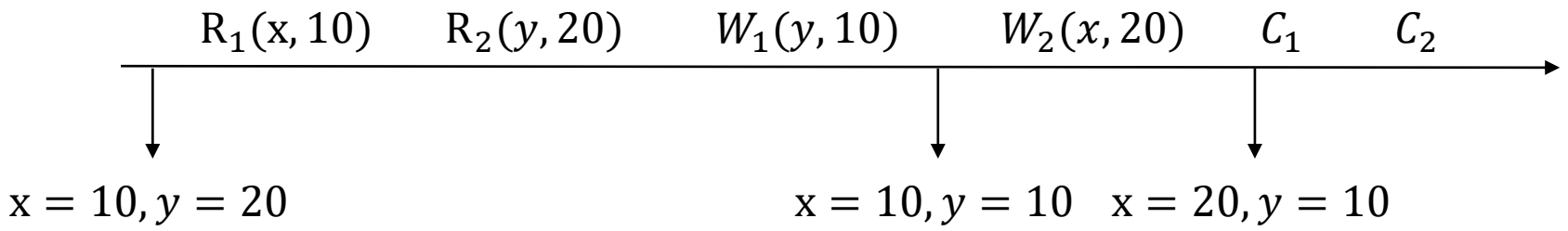
- Read Skew（读偏序）：

  Read Skew 的现象是因为读到两个状态的数据，导致观察到了违反约束的结果



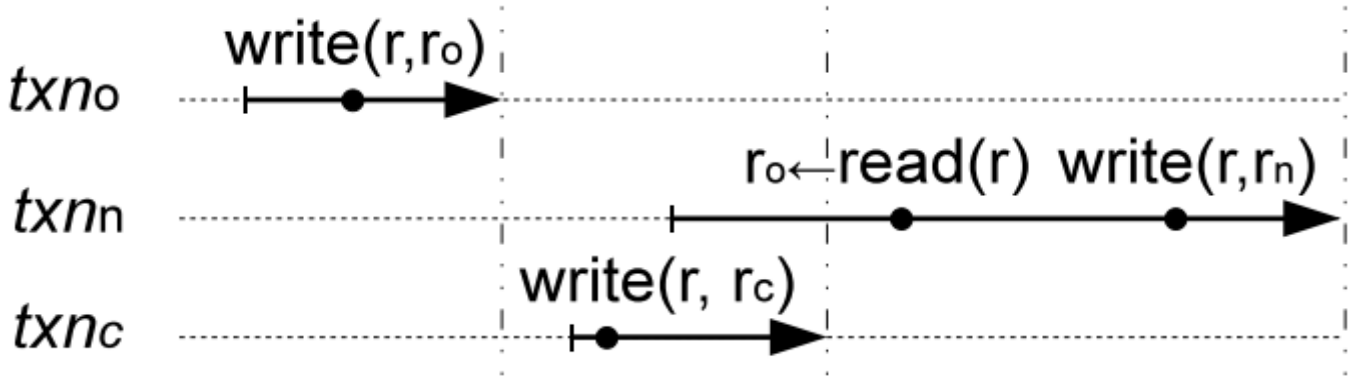
- Write Skew（写偏序）：

  Write Skew 是两个事务在写操作上发生的异常。



隔离性

Snapshot Isolation



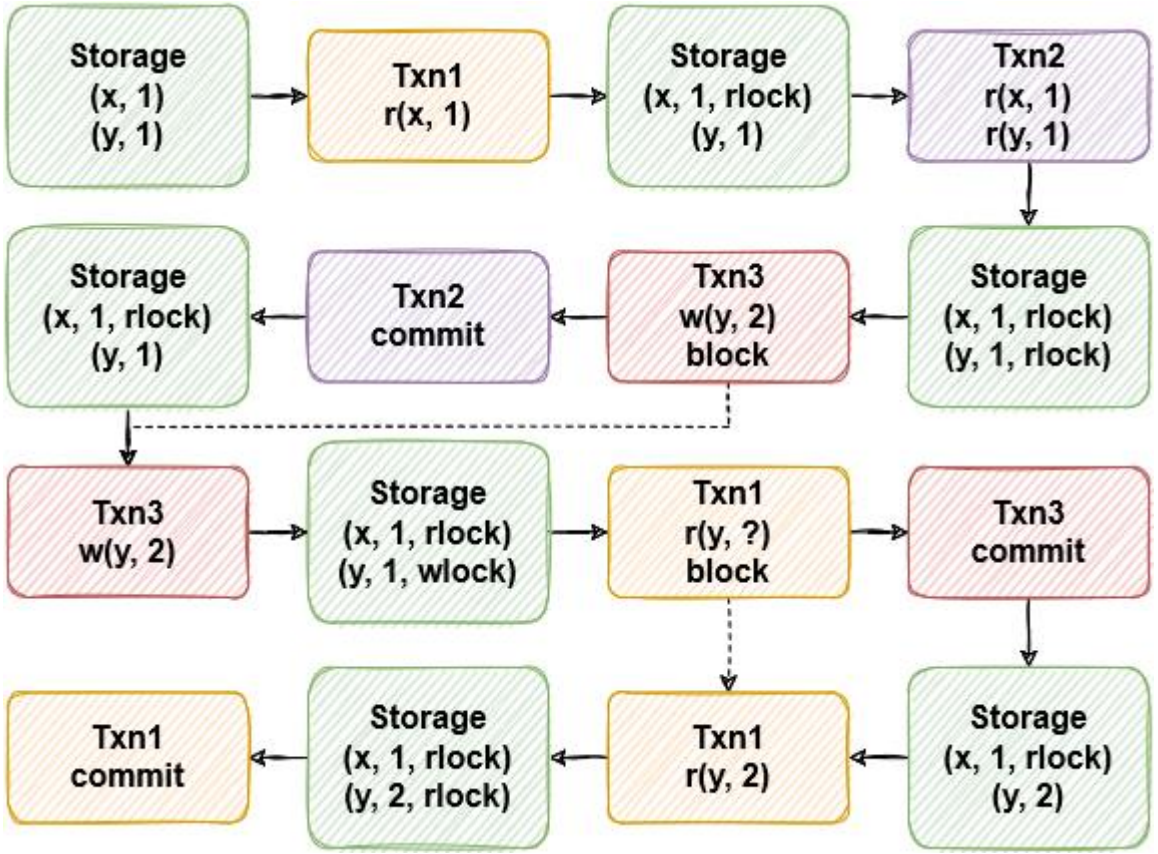
txn_i和txn_j发生冲突的条件

Spatial overlap: 同时对于同一行数据r进行了写入。

Temporal overlap： $T_s(txn_i) < \mathrm{T}_c(txn_j)\ and\ T_s(txn_j) < \mathrm{T}_c(txn_i)$

隔离性

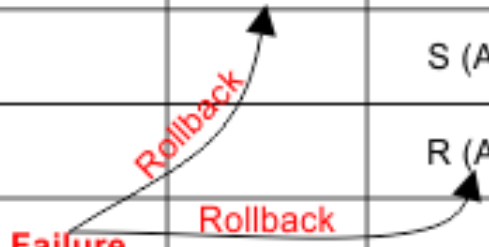
单机下的并发控制：2PL



隔离性

单机下的并发控制：2PL

Time	T1	T2	T3
Time1	S (A)		
Time2	R (A)		
Time3	W (A)		
Time4	U (A)		
Time5		S (A)	
Time6		R (A)	
Time7			S (A)
Time8			R (A)
	Failure		



2PL Cascading Rollback Problem

隔离性

单机下的并发控制：2PL

Strict 2PL	
T1	T2
s-lock(A)	
read(A)	
	s-lock(A)
x-lock(B)	
unlock(A)	
read(B)	
write(B)	
	read(A)
	unlock(A)
commit	
unlock(B)	
	s-lock(B)
	read(B)
	unlock(B)
	commit

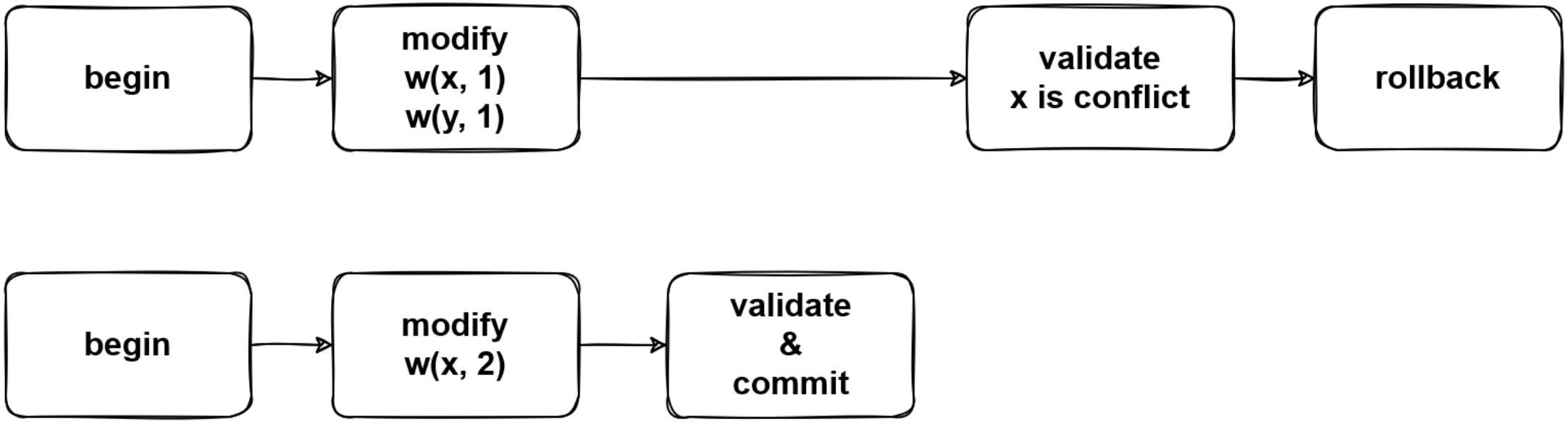
事务执行结束后才允许释放写锁

Rigorous 2PL	
T1	T2
s-lock(A)	
read(A)	
	s-lock(A)
x-lock(B)	
	read(A)
read(B)	
write(B)	
commit	
unlock(B)	
	s-lock(B)
	read(B)
unlock(A)	
	commit
	unlock(A)
	unlock(B)

事务执行结束后才能够释放读锁和写锁

隔离性

乐观并发控制算法：OCC



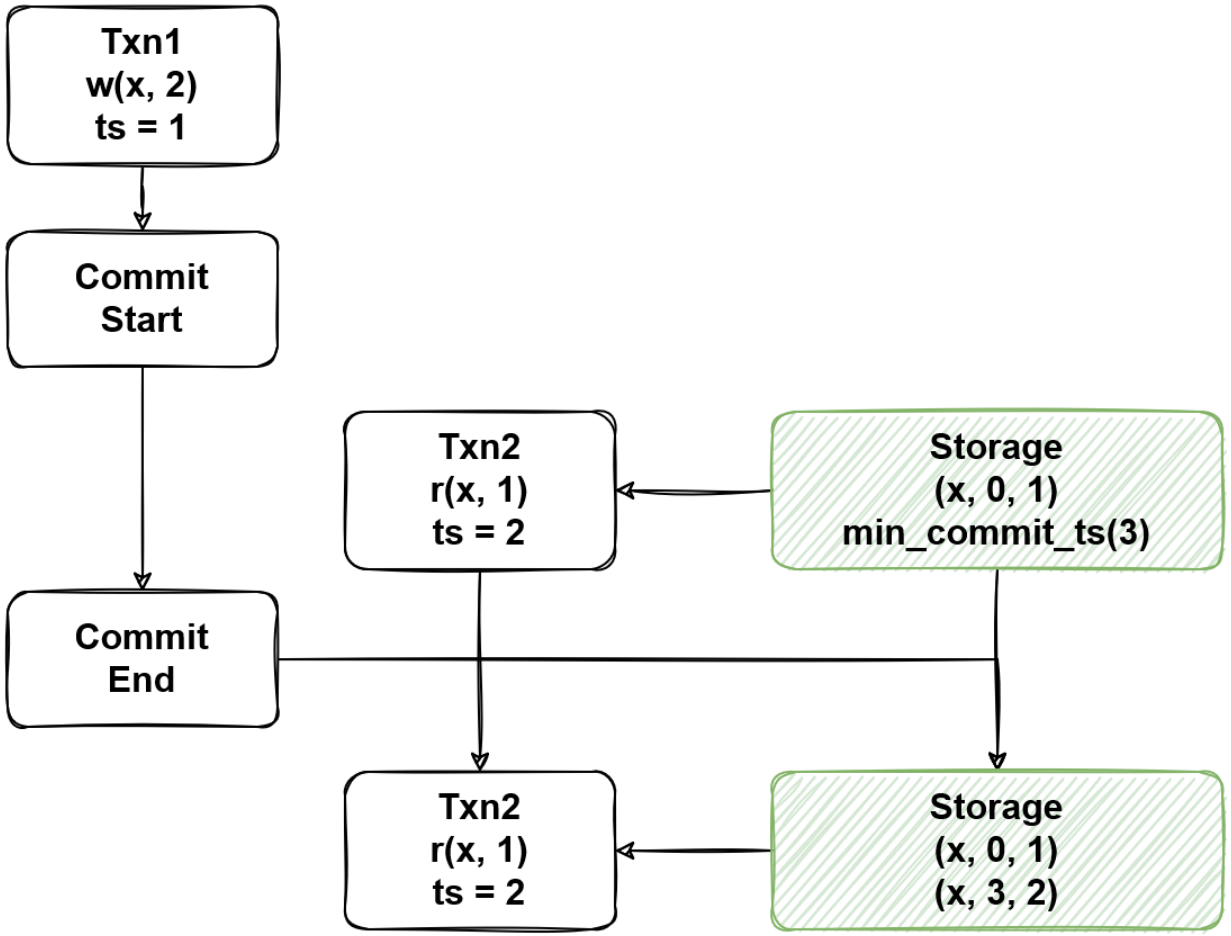
隔离性

多版本并发控制方法 : MVCC

Txn1	Txn2	Storage
		(x0, 1), (y0, 1)
r(x, 1, ts1)		(x0, 1), (y0, 1)
	w(x, 2, ts2) commit	(x0, 1), (x2, 2), (y0, 1)
r(x, 1, ts1)		(x0, 1), (x2, 2), (y0, 1)
commit		(x0, 1), (x2, 2), (y0, 1)
GC works		(x2, 2), (y0, 1)

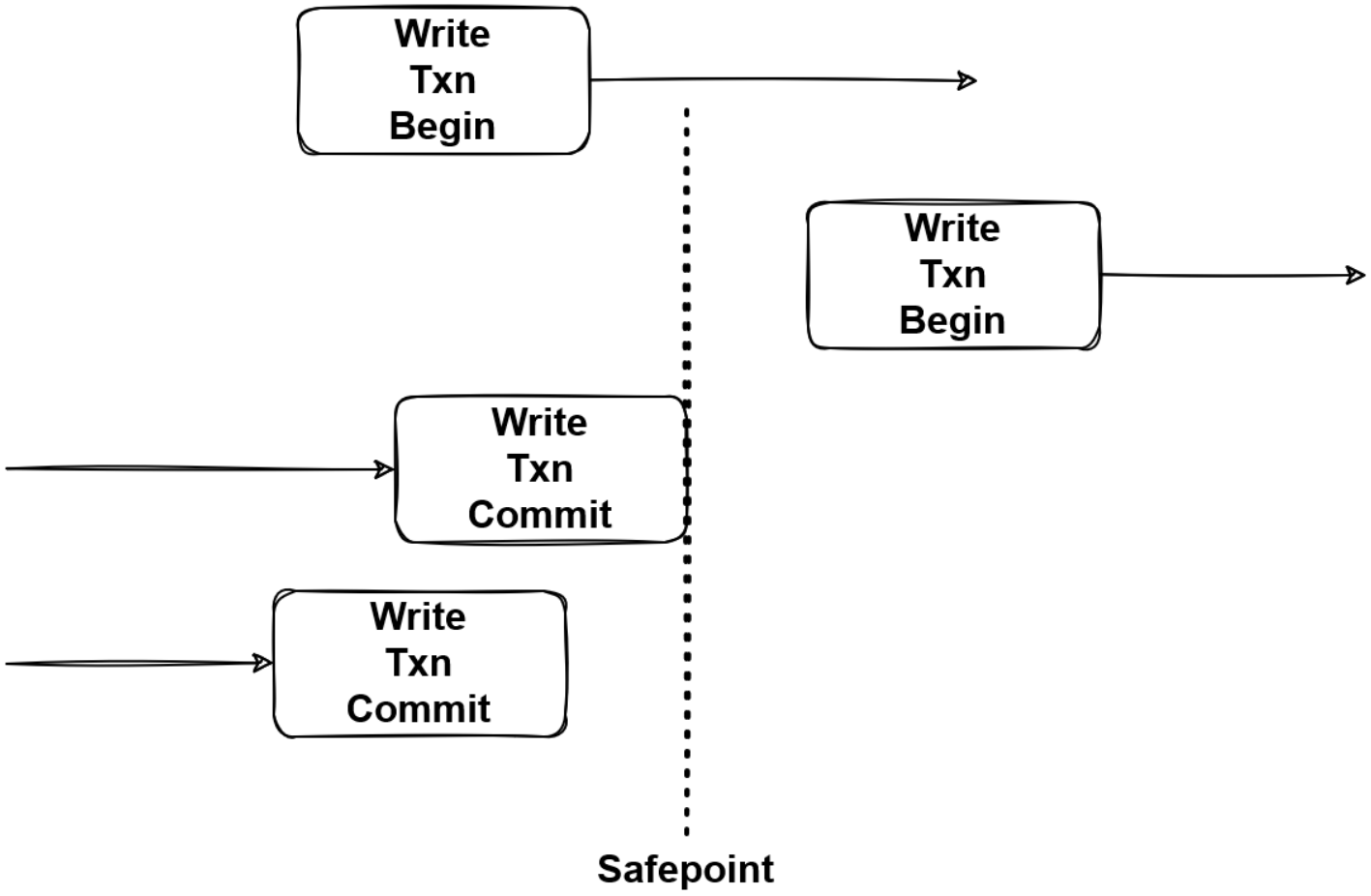
隔离性

多版本并发控制方法 : MVCC



隔离性

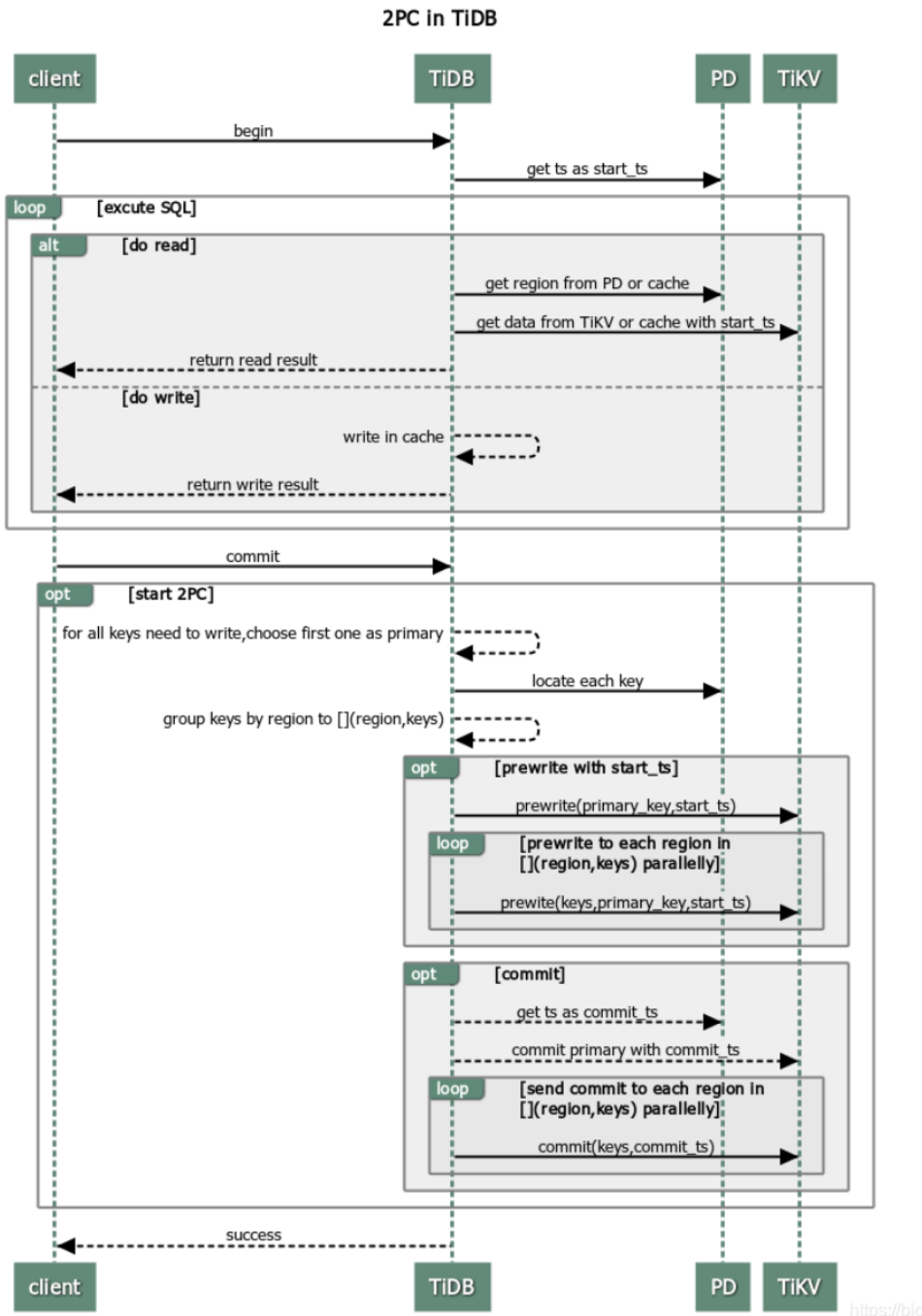
多版本并发控制方法 : MVCC



隔离性

分布式并发控制算法：Percolator

1. 获取事务的开始时戳
2. 从各个DataStore读取数据，并将write缓存到Client本地
3. 使用2PC协议进行提交
   - Prewrite阶段
   - Commit阶段



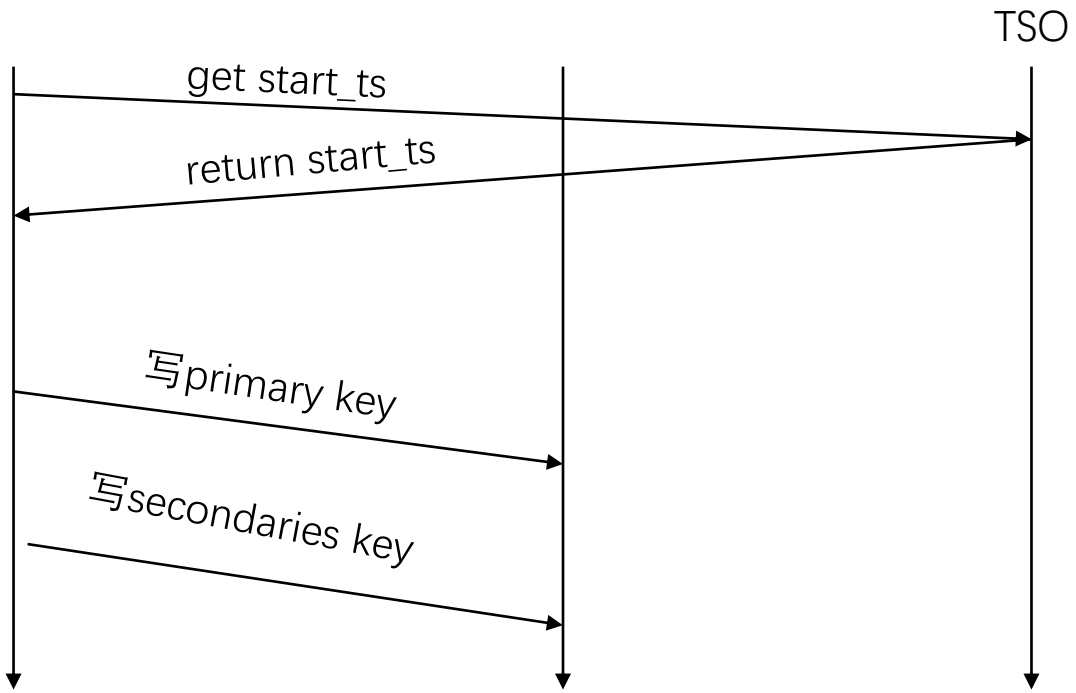
Large-scale Incremental Processing Using Distributed Transactions and Notifications

隔离性

分布式并发控制算法：Percolator

写入时会在写集合中随机选取一个key作为primary key，剩余key作为secondaires key

1. Write-write conflict：检查写入key在 [start_ts, max) 之间是否存在写入。
2. 检查 lock 列中该 Write 是否被上锁。
3. 以 start_ts 作为 Bigtable 的 timestamp，将数据写入 data 列。
4. 对 Write 上锁：以 start_ts 作为 timestamp，以 {primary.row, primary.col} 作为 value，写入 lock列 。{Priamry.row, Primary.col}

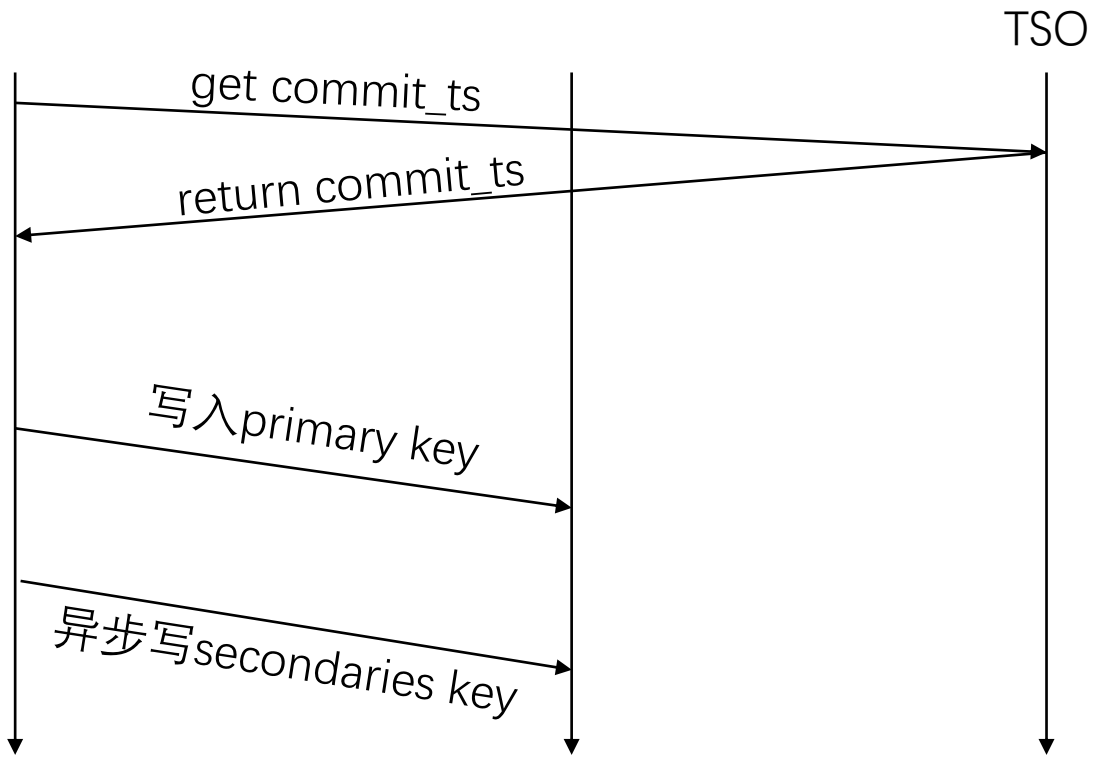


Large-scale Incremental Processing Using Distributed Transactions and Notifications

隔离性

分布式并发控制算法：Percolator

1. 从 TimeOracle 获取一个 timestamp 作为 commit_ts。
2. 先 Commit primary , 如果失败则 Abort 事务：
 2.1. 检测 lock 列 primary 对应的锁是否存在，
 2.2. 以 commit_ts 作为 timestamp，以 start_ts 作为 value 写 write 列。
 2.3. 删除 lock列中对应的锁。
3. 异步提交secondaries key



Large-scale Incremental Processing Using Distributed Transactions and Notifications