# 第7讲　分布式数据库中的并发控制

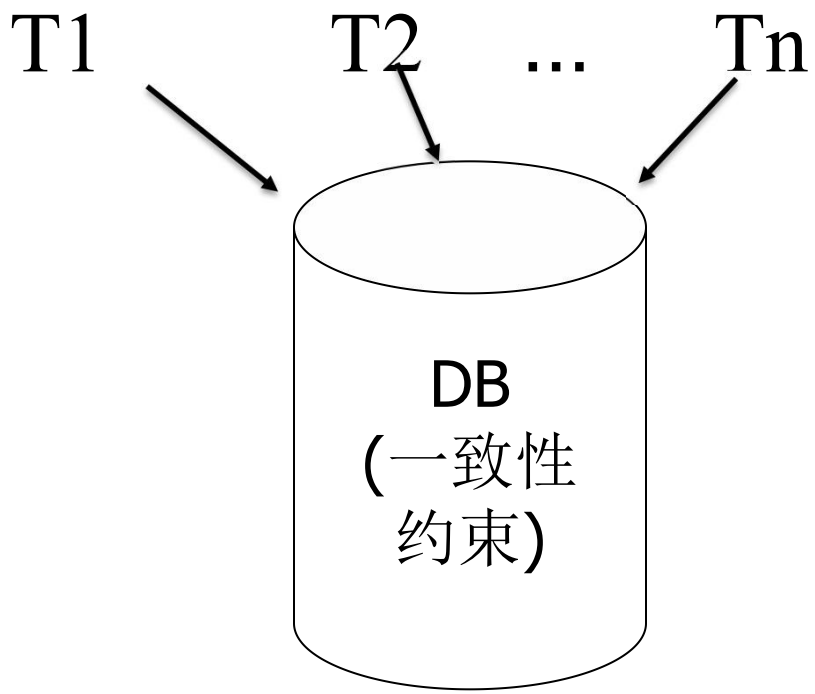
1. 并发控制的概念和理论
2. 分布式数据库系统并发控制的封锁技术
3. 分布式数据库系统并发控制的时标技术
4. 分布式数据库系统并发控制的多版本技术
5. 分布式数据库系统并发控制的乐观方法

# 1 并发控制的概念和理论

## 1.1 并发控制的概念

- 通常，数据库总有若干个事务在运行，这些事务可能并发地存取相同的数据，称为事务的并发操作。
- 当数据库中有多个事务并发执行时，系统必须对并发事务之间的相互作用加以控制，这是通过并发控制机制来实现的。
- 分布式数据库中的并发控制解决多个分布式事务对数据并发执行的正确性，保证数据库的完整性和一致性。比集中式并发控制更复杂。

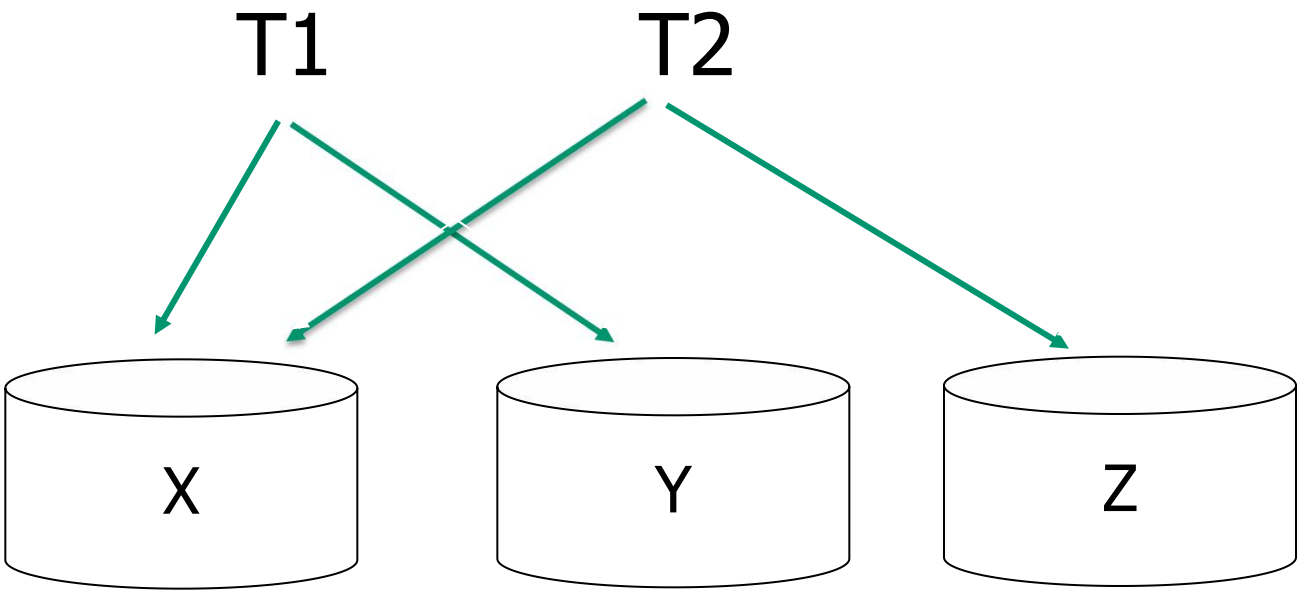
## 1.1 并发控制的概念




集中式DB环境

# 1 并发控制的概念和理论

## 1.1 并发控制的概念




分布式DB环境

# 1 并发控制的概念和理论

## 1.1 并发控制的概念

### 并发控制问题之一：丢失更新

| 时间 | 更新事务T1 | 数据库中X的值 | 更新事务T2 |
|---|---|---|---|
| $t_0$ | | 100 | |
| $t_1$ | FIND x | | |
| $t_2$ | | | FIND x |
| $t_3$ | x:=x-30 | | |
| $t_4$ | | | x:=x*2 |
| $t_5$ | UPDATE x | | |
| $t_6$ | | 70 | UPDATE x |
| $t_7$ | | 200 | |

注：其中FIND表示从数据库中读值，UPDATE表示把值写回到数据库
T1T2，结果140，T2T1,结果170，
得到结果是200，显然是不对的，T1在$t_7$丢失更新操作。

# 1 并发控制的概念和理论

## 1.1 并发控制的概念

### 并发控制问题之二：不一致分析

| 时间 | 更新事务T1 | 数据库中A的值 | 更新事务T2 |
|---|---|---|---|
| $t_0$ | | 100 | |
| $t_1$ | FIND x | | |
| $t_2$ | | | FIND x |
| $t_3$ | x:=x-30 | | |
| $t_4$ | UPDATE x | | |
| $t_5$ | | 70 | |

注：在时间t5事务T2仍认为x的值是100

# 1 并发控制的概念和理论

## 1.1 并发控制的概念

### 并发控制问题之三：依赖于未提交更新（读脏数据）

| 时间 | 更新事务T1 | 数据库中A的值 | 更新事务T2 |
|---|---|---|---|
| $t_0$ | | 100 | |
| $t_1$ | FIND x | | |
| $t_2$ | x:=x-10 | | |
| $t_3$ | UPDATE x | | |
| $t_4$ | | 90 | FIND x |
| $t_5$ | ROLLBACK | | |
| $t_6$ | | 100 | |

注： 事务T2依赖于事务T1的未完成更新

## 1.2 事务可串行化理论

- 调度
  - 一组事务的调度必须包含这些事务的所有操作
  - 调度中某个事务的操作顺序必须保持与该事务原有的顺序相同
- 串行调度
  - 一个事务的第一个动作是在另一个事务的最后一个动作完成后开始. 即调度中事务的各个操作不会交叉, 每个事务相继执行.
- 一致性调度
  - 调度可以使得数据库从一个一致性状态转变为另一个一致性状态，则称调度为一致性调度

## 1.2 事务可串行化理论

- 可串行化调度
  - 如果一个调度等价于某个串行调度，则该调度称为可串行化调度。
  - 也就是说，该调度可以通过一系列非冲突动作的交换操作使其成为串行调度

## 1.3 并发控制机制的常用方法及其分类

- 保证只产生可串行化调度的机制
- 并发控制机制分类
  - 建立在相互排斥地访问共享数据基础上的算法
  - 通过一些准则(协议)对事务进行排序的算法
  - 悲观并发控制法（事务是相互冲突的），乐观并发控制法（没有太多的事务相互冲突）

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.1 基于封锁的并发控制方法概述

### 基本思想和概念

- 基本思想
  - 事务访问数据项之前要对该数据项封锁，如果已经被其他事务锁定，就要等待，直到哪个事务释放该锁为止
- 锁的粒度
  - 锁定数据项的范围
  - 数据项层次
    - 一条数据库记录
    - 数据库记录中的一个字段值
    - 一个磁盘块（页面）
    - 一个完整的文件
    - 整个数据库

## 2.1 基于封锁的并发控制方法概述

### 基本思想和概念

- 粒度对并发控制的影响
  - 大多数DBMS缺省设置为记录锁或页面锁
  - 粒度小，并发度高，锁开销大
    - 数据项比较多，锁也多，解锁和封锁操作多，锁表存储空间大
  - 粒度大，并发度低，锁开销小
    - 如果是磁盘块，封锁磁盘块中的一条记录B的事务T必须封锁整个磁盘块
    - 而另外一个事务S如果要封锁记录C,而C也在磁盘块中，由于磁盘块正在封锁中，S只能等待
    - 如果是封锁粒度是一条记录的话，就不用等待了

## 2.1 基于封锁的并发控制方法概述

### 基本思想和概念

- 如何来确定粒度
  - 取决于参与事务的类型
  - 如果参与事务都访问少量的记录，那么选择一个记录作为粒度较好
  - 如果参与事务都访问同一文件中大量的记录，则最好采用块或者文件作为粒度

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.1 基于封锁的并发控制方法概述

### 基本思想和概念

- **锁的类型：**更 新 (U) 锁可以防止通常形式的死锁。一般更新模式由一个事务组成，此事务读取记录，获取资源（页或行）的共享 (S) 锁，然后修改行，此操作要求锁转换为排它 (X) 锁。如果两个事务获得了资源上的共享模式锁，然后试图同时更新数据，则一个事务尝试将锁转换为排它 (X) 锁。共享模式到排它锁的转换必须等待一段时间，因为一个事务的排它锁与其它事务的共享模式锁不兼容；发生锁等待。第二个事务试图获取排它 (X) 锁以进行更新。由**于两个事务都要转换为排它 (X) 锁，并且每个事务都等待另一个事务释放共享模式锁，因此发生死锁。**

- **若要避免这种潜 在的死锁问题，请使用更新 (U) 锁。一次只有一个事务可以获得资源的更新 (U) 锁。如果事务修改资源，则更新 (U) 锁转换为排它 (X) 锁。否则，锁转换为共享锁。**

  - 共享锁：Share锁，S锁或者读锁
  - 排它锁：eXclusive锁，X锁，拒绝锁或写锁。
  - 更新锁：Update锁，U锁

- 锁的选择：
  - 数据项既可以读也可以写.则要用X锁
  - 如果数据项只可以读.则要用 S锁.

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.1 基于封锁的并发控制方法概述

### 基本思想和概念

- 锁的操作
  - **Read_lock(x):**读锁
  - **Write_lock(x)：**写锁
  - **unlock(x)：**解锁
- 数据项的状态
  - **read_locked:** 读锁
  - **Write_locked:**写锁
- 具体操作方法
  - 在系统锁表中记录关于锁的信息
  - 锁表中每条记录有四个字段：<数据项名称，锁状态，读锁的数目，正在封锁该数据项的事务>

## 2.1 基于封锁的并发控制方法概述

- 锁的转换

1. 特定条件下，一个已经在数据项x上持有锁的事务T，允许将某种封锁状态转换为另外一种封锁状态
2. 比如，一个事务先执行了read_lock（x）操作，然后他可以通过执行write_lock(x)操作来升级该锁
3. 同样，一个事务先执行了write_lock(x) 操作，然后他可以通过执行read_lock（x） 操作来降级该锁

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.1 基于封锁的并发控制方法概述

满足封锁规则不能保证产生串行化调度

| T1 | T2 |
|---|---|
| read_lock(Y); | read_lock(X); |
| read_item(Y); | read_item(X); |
| unlock(Y); | unlock(X); |
| write_lock(X); | write_lock(Y); |
| read_item(X); | read_item(Y); |
| X:= X+Y; | Y := Y + X; |
| write_item(X); | write_item(Y); |
| unlock(X); | unlock(Y); |

（a）两个事务T1和T2

初始值：X=20，Y=30

串行调度T1，T2的结果： X=50，Y=80

串行调度T2，T1的结果： X=70，Y=50

（b）T1和T2可能的串行调度的结果

| T1 | T2 |
|---|---|
| read_lock(Y); | |
| read_item(Y); | |
| unlock(Y); | |
| | read_lock(X); |
| | read_item(X); |
| | unlock(X); |
| | write_lock(Y); |
| | read_item(Y); |
| | Y := Y + X; |
| | write_item(Y); |
| | unlock(Y); |
| write_lock(X); | |
| read_item(X); | |
| X:= X+Y; | |
| write_item(X); | |
| unlock(X); | |

这个调度S的结果：

X=50，Y=50

(不可串行化)

（c）使用锁的一个不可串行化调度的结果

## 2.1 基于封锁的并发控制方法概述

### 分布式数据库基本封锁算法

- 简单的分布式封锁方法
  - 类似集中式，将同一数据的全部副本封锁，然后更新，之后解除全部封锁
  - 缺点是各站点间进行相当大的数据传输，如果有N个站点，就有：
    - N个请求封锁的消息　　N个封锁授权的消息
    - N个更新数据的消息　　N个更新执行了的消息
    - N个解除封锁的消息

## 2.1 基于封锁的并发控制方法概述

### 分布式数据库基本封锁算法

- 主站点封锁法
  - 定义一个站点为主站点，负责系统全部封锁管理
  - 所有站点都向主站点提出封锁和解锁请求，由它去处理
  - 缺点有：
    - 所有封锁请求都送往单个站点，容易由于超负荷造成“瓶颈”
    - 主站点故障造成会使系统瘫痪，封锁消息都在这里，制约了系统可用性和可靠性

## 2.1 基于封锁的并发控制方法概述

### 分布式数据库基本封锁算法

- 主副本封锁法
  - 不指定主站点，指定数据项的主副本
  - 事务对某个数据项进行操作时，先对其主副本进行封锁，再进行操作
  - 主副本封锁，意味着所有的副本都被封锁
  - 主副本按使用情况，尽量就近分布
  - 可减少站点的负荷,使得各站点比较均衡
  - 可减少传输量

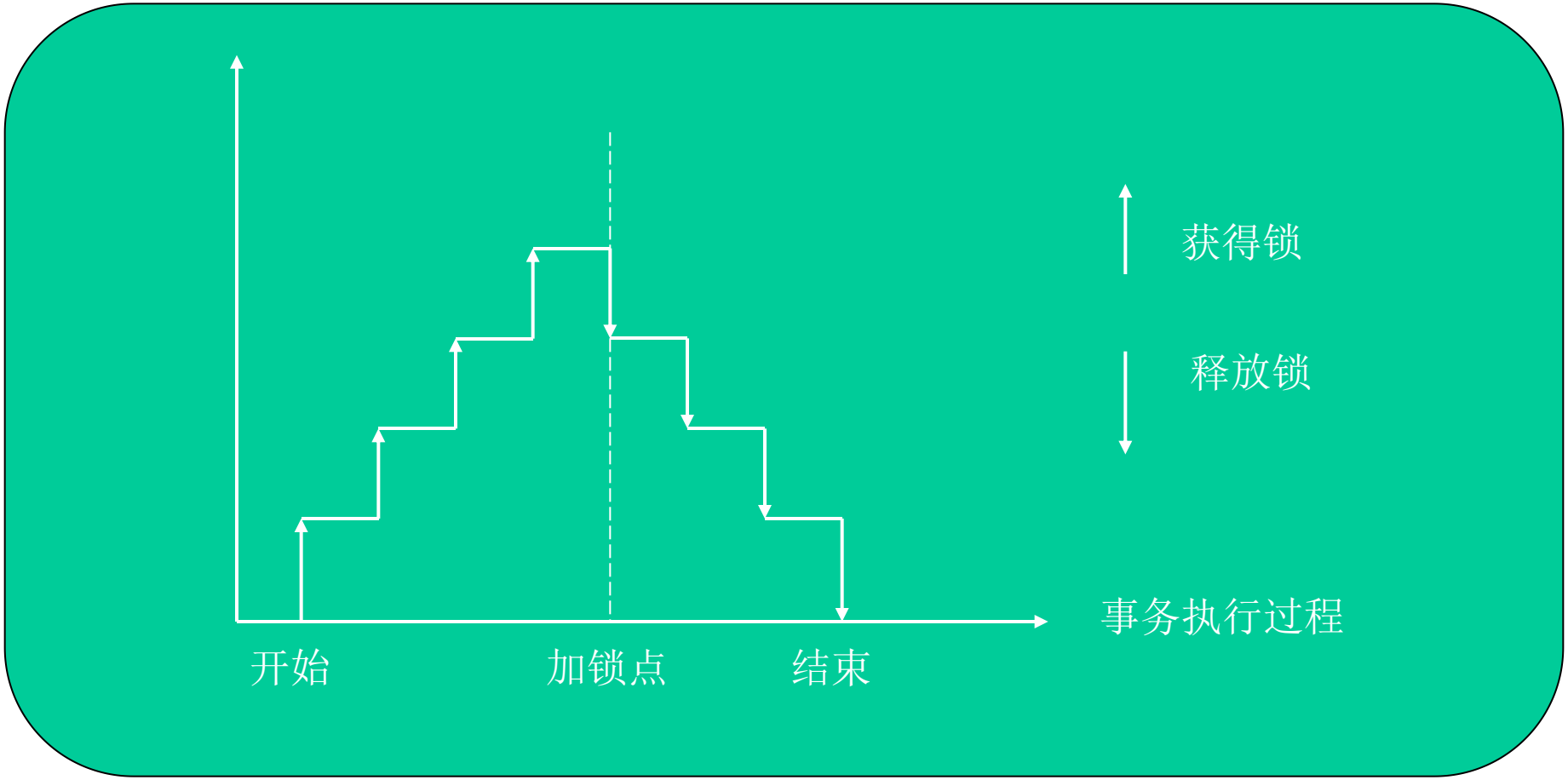
## 2.2 2PL协议（两阶段封锁协议）

### 基本2PL协议 （2PL可以保证事务执行的可串行性）

- 如果一个事务所有的封锁操作（读写）都在第一个解锁操作之前，那么它就遵守2PL协议
- 事务的执行中Lock的管理分成两个阶段
  - 上升阶段(成长阶段)：获取Lock阶段
  - 收缩阶段(衰退阶段）：释放Lock阶段
- 封锁点是指事务获得了它所要求的所有锁，并且还没有开始释放任何锁的时刻
- 如果允许锁的转换，锁的升级必须在成长阶段进行，锁的降级必须在锁的衰退阶段进行。

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.2 基本2PL协议




两阶段封锁协议

## 2.2 2PL协议

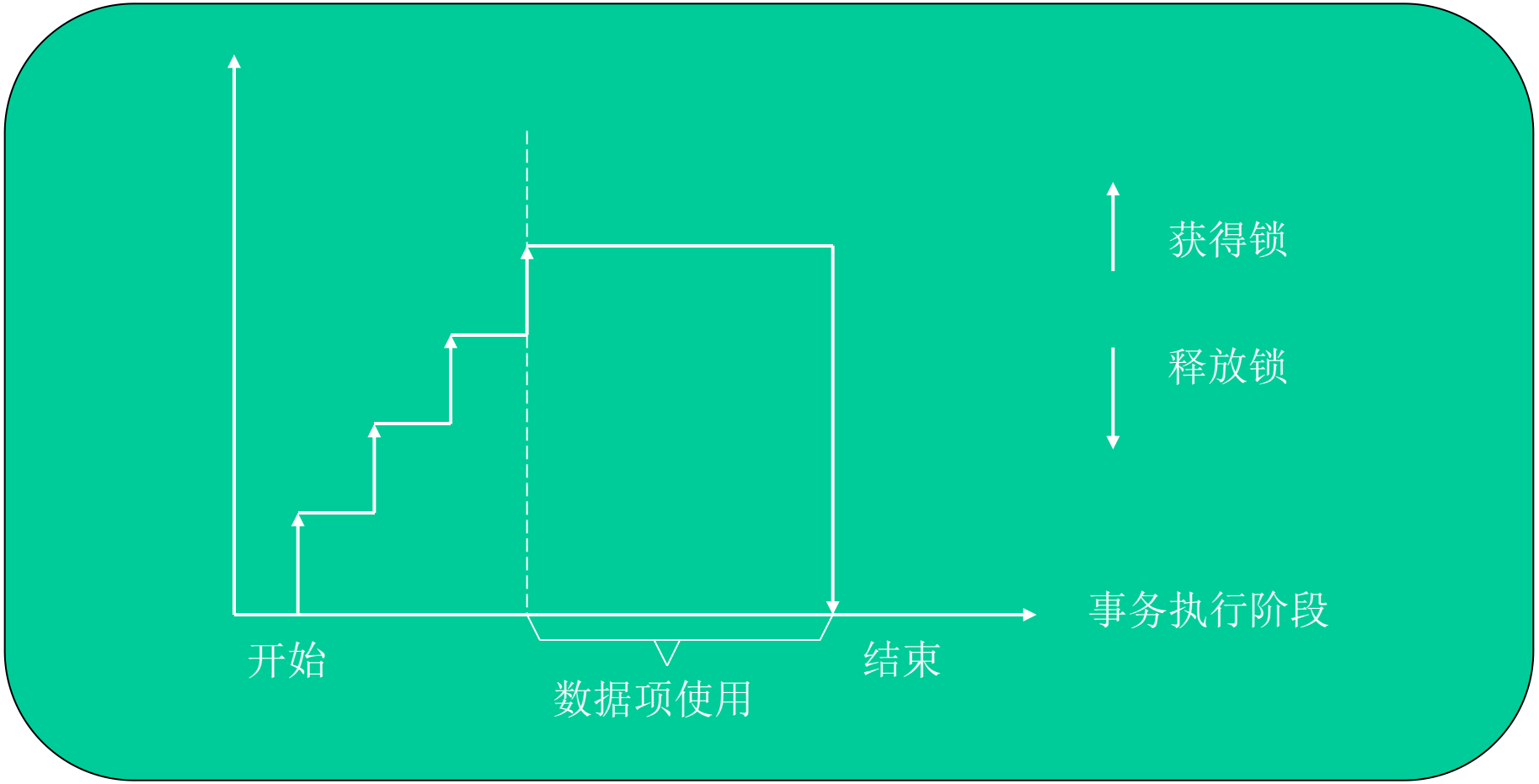
- 保守2PL
  - 要求事务在开始执行之前就持有所有它要访问的数据项上的锁
  - 事务要预先声明它的读集和写集
- 大多数2PL调度器实现的是严格2PL(S2PL)
  - 事务在提交或者撤销之前，绝对不释放任何一个写锁
  - 事务结束时（提交或者撤销），同时释放所有的锁
- 严酷2PL
  - 事务T在提交或撤销之前，不能释放任何一个锁（写锁或者读锁），因此它比严格2PL更容易实现

## 2.2 2PL协议

- 保守2PL与严酷2PL之间的区别
  - 前者，事务必须在开始之前封锁它所需要的所有数据项，因此，一旦事务开始就处在收缩阶段
  - 后者,直到事务结束（提交或者撤销）后才开始解锁，因此，事务一直处于扩张阶段，直到结束

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.2 2PL协议




严酷2PL协议

## 2.3 2PL协议的实现方法

### 集中式2PL的实现方法

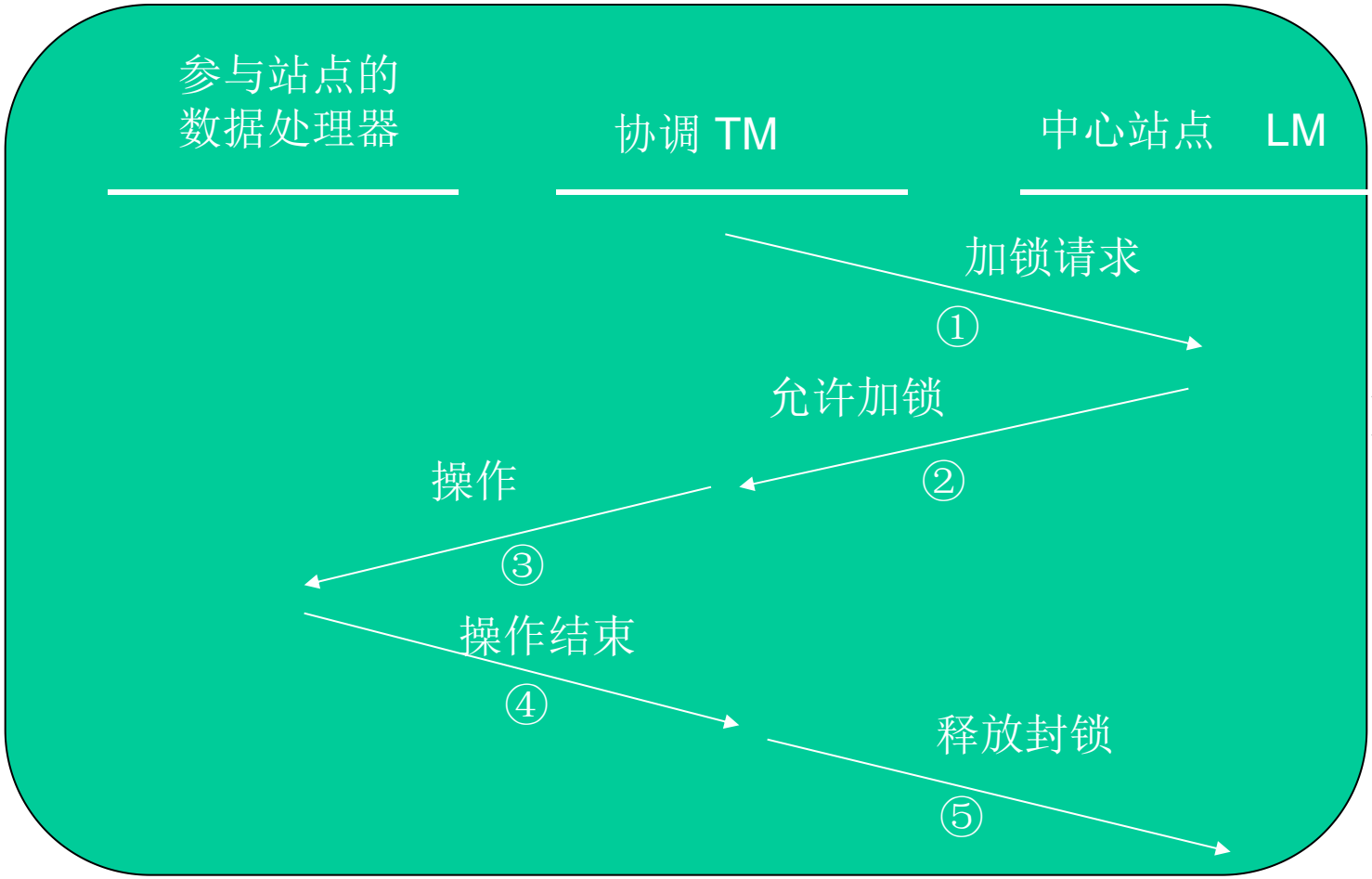
- 2PL很容易扩展到分布式DBMS(无论复制或无复制DDB),
- 其最简单的方法是选择一个站点(主站点)做Lock管理器, 其他站点上的事务管理器都需要与该选出的站点Lock管理器通信, 而不是与本站点Lock管理器通信. 这就是集中式2PL方法

## 2.3 2PL协议的实现方法

- 术语
  - 协调事务管理器(coordinating TM)：事务原发站点
  - 数据处理器(data processor，DP)：其他参与站点
  - 中心站点LM：主站点锁管理器

## 2.3 2PL协议的实现方法

# 集中式2PL的实现方法




中心站点LM不需要向DP发送操作

集中式2PL的通信结构

## 2.3 2PL协议的实现方法

### 主副本2PL的实现方法

- 是主站点2PL的直接扩展
  - 选择一组站点做Lock管理器
  - 每个Lock管理器管理一组数据(即每个数据选择一个站点作自己的Lock管理器)
  - 事务管理器根据Lock申请的数据对象分别向这些数据的LM发出锁申请
  - 必须先为每一个数据项确定一个主副本站点

## 2.3 2PL协议的实现方法

### 分布式2PL的实现方法

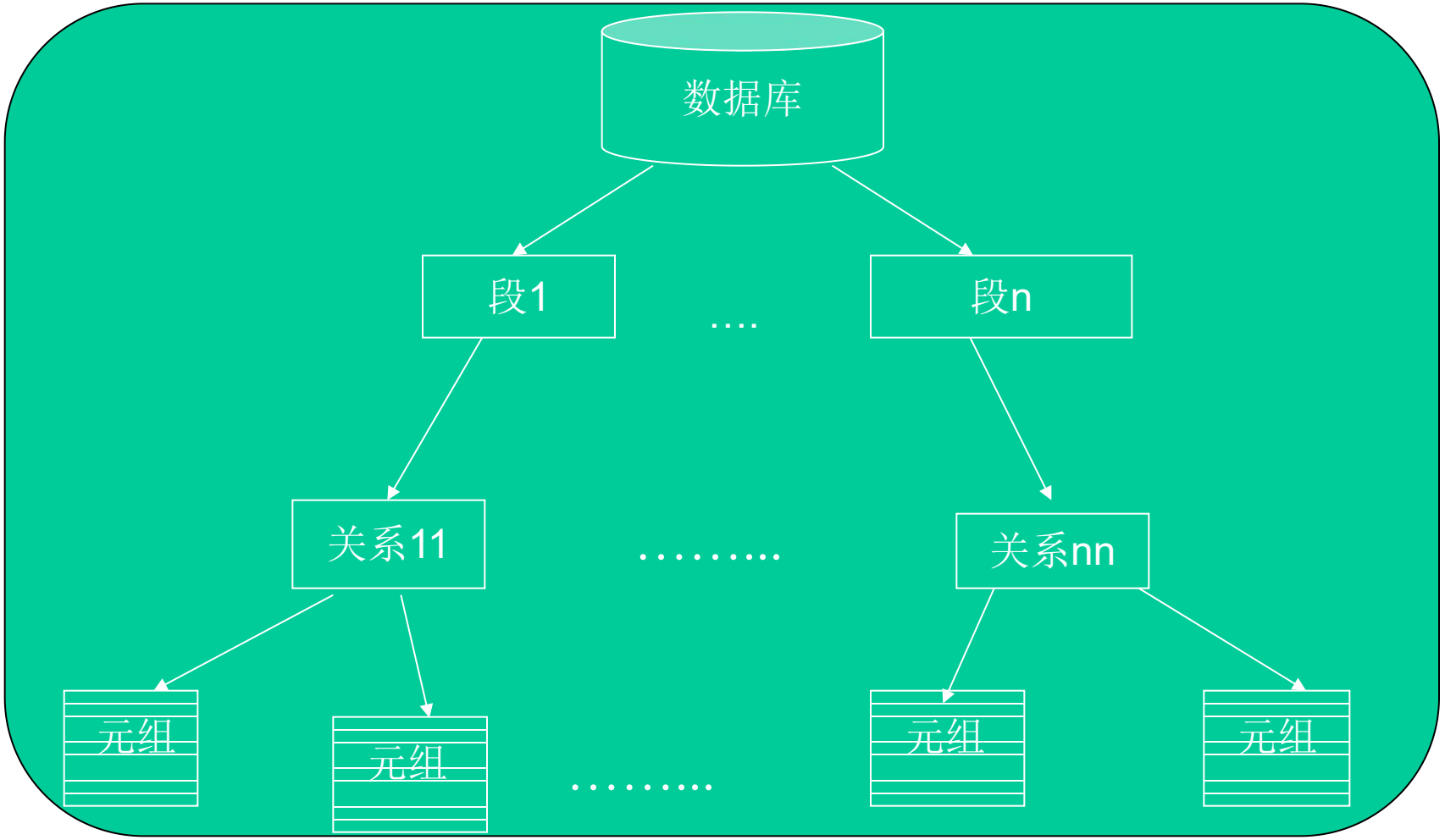
- 特点
  - 每个站点都有LM
  - 无副本DDB上如同主副本2PL
  - 有冗余副本DDB，等待一定数量副本加锁成功
- 与集中式相似，但有不同
  - 集中式中向中心站点封锁管理程序发送的信息，在分布式中发送给所有参与站点的封锁管理程序
  - 分布式通过参与者的封锁管理程序
  - 参与者的数据处理器向协调者的事务管理程序发送“操作结束”信息，协调者事务管理器向参与者事务管理器发送解锁指令

## 2.4 多粒度封锁与意向锁

- 多粒度封锁
  - 封锁的粒度不是单一的一种粒度，而是有多种粒度
  - 可以定义多粒度树，根节点是整个数据库，叶节点表示最小的封锁粒度
- 直接封锁
  - 事务对要进行读/写的数据对象直接申请加锁
- 分层封锁
  - DB中各数据对象从大到小存在一种层次关系, 例如划分为DB, 段, 关系, 元组, 字段等
  - 当封锁了外层数据对象时, 蕴含着也同时封锁了它的所有内层数据对象
  - 数据项的显式封锁和隐式封锁

# 2 分布式数据库系统并发控制机制的封锁技术

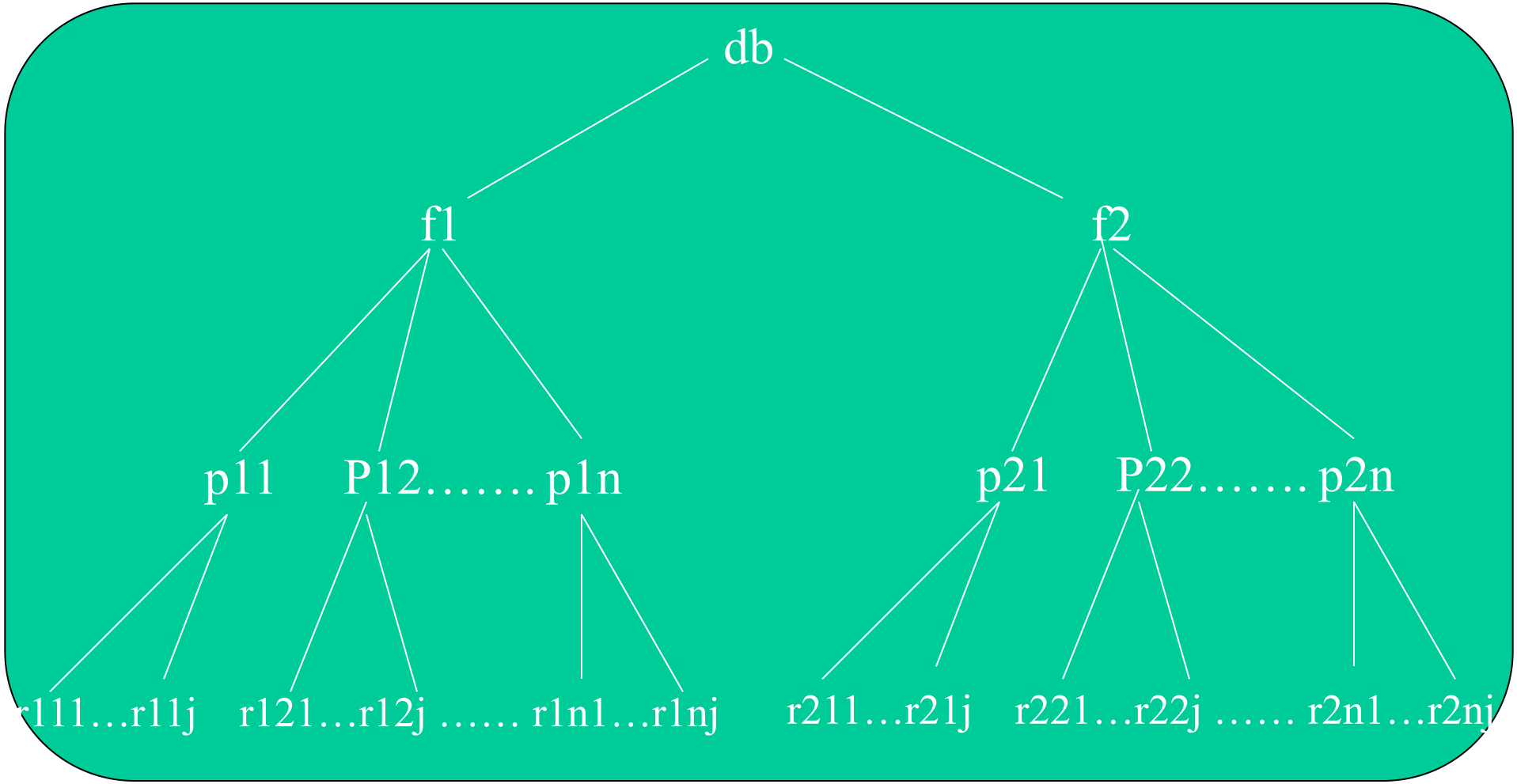
## 2.4 多粒度封锁与意向锁




多级粒度树

# 2 分布式数据库系统并发控制机制的封锁技术

## 2.4 多粒度封锁与意向锁




用来说明多粒度级别封锁的粒度层次结构

## 2.4 多粒度封锁与意向锁

- 例子
  - 假定事务T1要更新文件f1中的所有记录，T1请求并获得了f1上的一个写锁
  - 那么f1下面的页面和记录就获得了隐式写锁
  - 如果这时候，事务T2想从f1中的某个页面中读某个记录，那么T2就要申请该记录上的读锁
  - 但是要确认这个读锁和已经存在锁的相容性，确认的方法就是要遍历该树：从记录到页，到文件最后到数据库，如果在任意时刻，在这些项中的任意一个上存在冲突锁，那么对记录的封锁请求就被拒绝，T2被阻止，要等待

## 2.4 多粒度封锁与意向锁

- 意向锁
  - 如果对一个节点加意向锁，则说明该节点的下层节点正在被封锁
  - 对任一节点封锁时，必须先对它的上层节点加意向锁
- 意向锁的类型
  - 意向共享锁(IS)：指示在其后代节点上将会请求共享锁
  - 意向排它锁(IX)：指示在其后代节点上将会请求排他锁
  - 共享意向排它锁(SIX)：指示当前节点处在共享方式的封锁中，但是在它的某些后代节点中将会请求排他锁。
  - 例如：对某个表加SIX锁，则表示该事务要读整个表（加S锁），同时会更新个别元组（加IX锁）

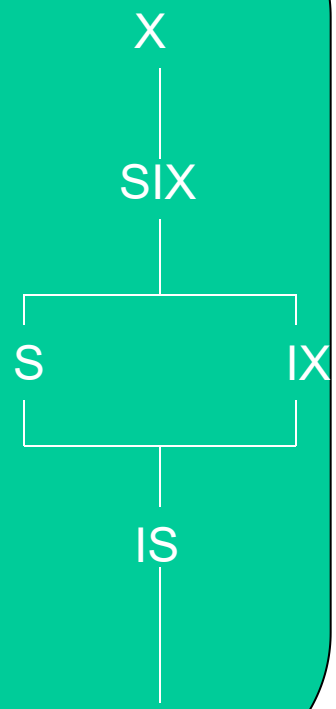
# 2 分布式数据库系统并发控制机制的封锁技术

## 2.4 多粒度封锁与意向锁

锁的强度：对其它锁的排斥程度

### 锁的相容矩阵

| $T_1$ \ $T_2$ | S | X | IS | IX | SIX | - |
|---|---|---|---|---|---|---|
| S | Y | N | Y | N | N | Y |
| X | N | N | N | N | N | Y |
| IS | Y | N | Y | Y | Y | Y |
| IX | N | N | Y | Y | N | Y |
| SIX | N | N | Y | N | N | Y |
| - | Y | Y | Y | Y | Y | Y |




**Y=yes，表示相容的请求　N=no，表示不相容的请求**

**(a)数据锁的相容矩阵**

**(b)锁的强度的偏序关系**

## 2.4 多粒度封锁与意向锁

- 多粒度封锁协议的规则
  - 必须遵守锁的相容性规则
  - 必须首先封锁树的根节点，可以用任何一种方式的锁
  - 只有当节点N的父节点已经被事务T以IS或IX方式封锁后，节点N才可以被T以S或者IS方式封锁
  - 只有当节点N的父节点已经被事务T以IX或SIX方式封锁后，节点N才可以被T以X,IX或者SIX方式封锁
  - 只有当事务T还没有释放任何节点时，T才可以封锁一个节点
  - 只有当事务T当前没有封锁节点N的任何子节点时，T才可以为节点N解锁。

## 2.4 多粒度封锁与意向锁

- 总结
  - 具有意向锁的多粒度加锁方法中，任意事务T要对一个数据对象加锁，必须先对它的上层节点加意向锁
  - 申请封锁时应该按自上而下的次序进行
  - 释放锁时则应该按自下而上的次序进行
  - 具有意向锁的多粒度加锁方法提高了系统的并发度, 减少了加锁和释放锁的开销
  - 它已经在实际的DBMS系统中广泛应用，例如Oracle中

# 3 分布式数据库系统并发控制的时标技术

## 3.1 基于时标的并发控制方法

- 基本概念
  - 不通过互斥来支持串行性，而是通过在事务启动时赋给时标（时间戳）来实现
  - 时标是用来唯一识别每个事务并允许排序的标识
  - 如果 $ts(T_1) < ts(T_2) \ldots < ts(T_n)$, 则调度器产生的序是: $T_1, T_2, \ldots T_n$
- 规则
  - 如果T1的操作$O_1(x)$和T2的操作$O_2(x)$是冲突操作, 那么, $O_1$在$O_2$之前执行, 当且仅当$ts(T_1)<ts(T_2)$

# 3 分布式数据库系统并发控制的时标技术

## 3.1 基于时标的并发控制方法

- 时标分配方法
  - 全局时标
    - 使用全局的单调递增的计数器
    - 全局的计数器维护是个难题
  - 局部时标
    - 每个站点基于其本地计数器自治地指定一个时标
    - 标识符由两部分组成：〈本地计数器值，站点标识符〉
  - 站点标识符是次要的，主要是本地计数器值
  - 可以使用站点系统时钟来代替计数器值

# 3 分布式数据库系统并发控制的时标技术

## 3.1 基于时标的并发控制方法

- 时标法思想
  - 每个事务赋一个唯一的时标，事务的执行等效于按时标次序串行执行
  - 如果发生冲突，是通过撤销并重新启动一个事务来解决的
  - 事务重新启动时，则赋予新的时标
  - 优点是没有死锁，不必设置锁
  - 封锁和死锁检测引起的通信开销也避免了
  - 但要求时标在全系统中是唯一的

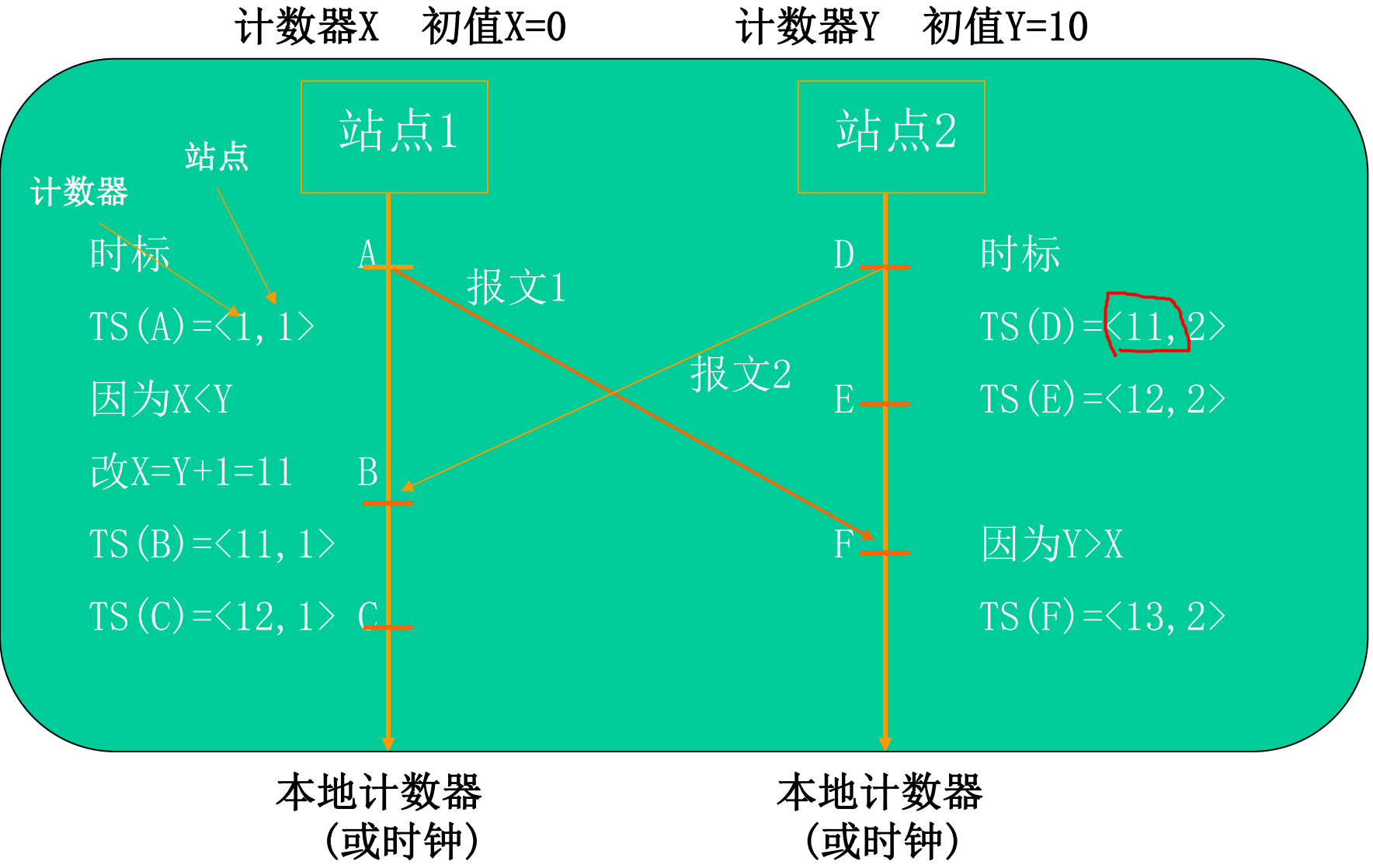
# 3 分布式数据库系统并发控制的时标技术

## 3.1 基于时标的并发控制方法

- 时标性质
  - 唯一性
  - 单调性
- 全局唯一时间的形成与调整
  - 每个站点设置一个计数器, 每发生一个事务, 计数器加一
  - 发送报文时包含本地计数器值, 近似同步各站点计数器

# 3 分布式数据库系统并发控制的时标技术

## 3.1 基于时标的并发控制方法



## 基本时标法

- 规则
  - 每个事务在本站点开始时赋予一个全局唯一时标
  - 在事务结束前，不对数据库进行物理更新
  - 事务的每个读操作或写操作都具有该事务的时标
  - 对每个数据项x, 记下写和读操作的最大时标，记为WTM(x)和RTM(x)
  - 如果事务被重新启动，则被赋予新的时标

## 3.2 基本时标法

- 基本时标法执行过程
  - 令read_TS是事务对x进行读操作时的时标
    - 若read_TS<WTM(x)，则拒绝该操作, 事务重新启动
    - 否则，执行，令RTM(x)=max{RTM(x), read_TS}
  - 令write_TS是事务对x进行写操作时的时标
    - 若write_TS < RTM(x) 或 write_TS < WTM(x) ,则拒绝该操作，事务重新启动
    - 否则, 执行，令WTM(x) = max{WTM(x), write_TS}
- 缺点是重启动多

## 3.3 保守时标法

- 基本思想
  - 一种消除重启动的方法
  - 通过缓冲年轻的操作，直至年长的操作执行完成，因此操作不会被拒绝，事务也绝不被重启动
- 规则
  - 1 每个事务只在一个站点执行, 它不能激活远程的程序, 但是可以向远程站点发读/写请求
  - 2 站点i接收到来自不同站点j的读/写请求必须按时标顺序，即每个站点必须按时标顺序发送读/写数据请求，在传输中也不会改变这个顺序
  - 3 每个站点都为其它站点发来的读/写操作开辟一个缓冲区, 分别保存接收到的读/写申请

## 3.3 保守时标

- 假定某个站点k上,各个缓冲区队列都已不为空，即每个站点都已向它至少发送了一个读和一个写操作，就停止接收，处理在缓冲区中的操作
- 假定站点i至少有一个缓冲的读和缓冲的写来自网中其它站点, 根据规则2, Site i 知道没有年老的请求来自其它Site(因为按序接收, 所以不可能有比此更年老的请求到来, 年老的比年轻的先到)

## 3.3 保守时标法

# 例子

已知站点i的缓冲区队列中有来自所有站点的读/写请求如下所示:

| 站点1 | 站点2 | 站点3 | …… | 站点n |
|---|---|---|---|---|
| R11 | R21 | R31 | | Rn1 |
| R12 | R22 | R32 | | |
| R13 | R23 | | | |
| | R24 | | | |
| W11 | W21 | W31 | …… | Wn1 |
| | W22 | W32 | … | Wn2 |
| | W23 | | | |

## 3.3 保守时标法

执行步骤:

(1) 设RT=min(Rij), WT= min(Wij)

(2) 按下法处理缓冲区中的Rij和Wij

a. 若队列中有

(Rij) <= WT的Rij , 则顺序执行这些Rij，执行完删掉

b. 若队列中有

(Wij) <= RT的Wij, 则顺序执行这些Wij，执行完删掉

(3) 修改 RT=min(Rij), WT=min(Wij) ，此时的Rij和Wij是队列中剩余的

(4) 重复上述(2)和(3), 直到没有满足条件的操作, 或者:

a. 若某个或某些R队列为空时, RT=0;

b. 若某个或某些W队列为空时, WT=0

## 3.3 保守时标法

### 存在问题和解决方法

- 如果一个站点从来不向某个站点发送操作的话，那么执行过程中的假定就不符合，操作就无法进行。解决办法是，周期性的发送带有时标的空操作
- 此方法要求网络上所有站点都连通，这在大系统中很难办到。为避免不必要的通信，可对无读写操作请求的站点，发送一个时标很大的空操作
- 此方法过分保守，一律按照时序来进行，其中包括了不冲突的操作

# 4 分布式数据库系统并发控制的多版本技术

## 4.1 多版本概念和思想

- 基本思想
  - 保存已更新数据项的旧值，维护一个数据项的多个版本
  - 通过读取数据项的较老版本来维护可串行性，使得系统可以接受在其他技术中被拒绝的一些读操作
  - 写数据项时，写入一个新版本，老版本依然保存
- 缺点
  - 需要更多的存储来维持数据库数据项的多个版本
- 分类
  - 基于时标排序
  - 基于两阶段封锁

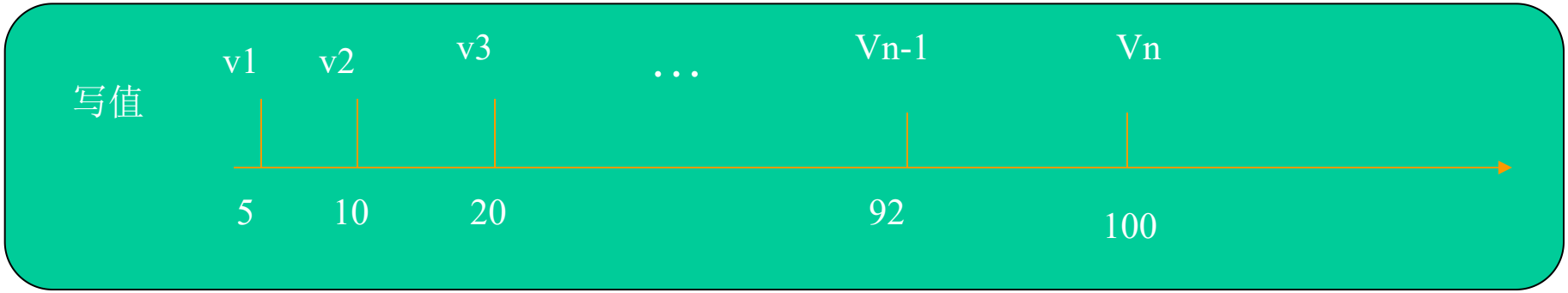
## 4.2 基于时标的多版本技术

- 数据项X的多版本
  - X1, X2, X3,…, Xk
- 系统保存的值
  - Xi的值
  - 两种时标
    - Read_TS(Xi): 读时标，成功读取版本Xi的事务的时标，最大的一个
    - Write_TS(Xi): 写时标，写入版本Xi的值的事务的时标

## 4.2 基于时标的多版本技术

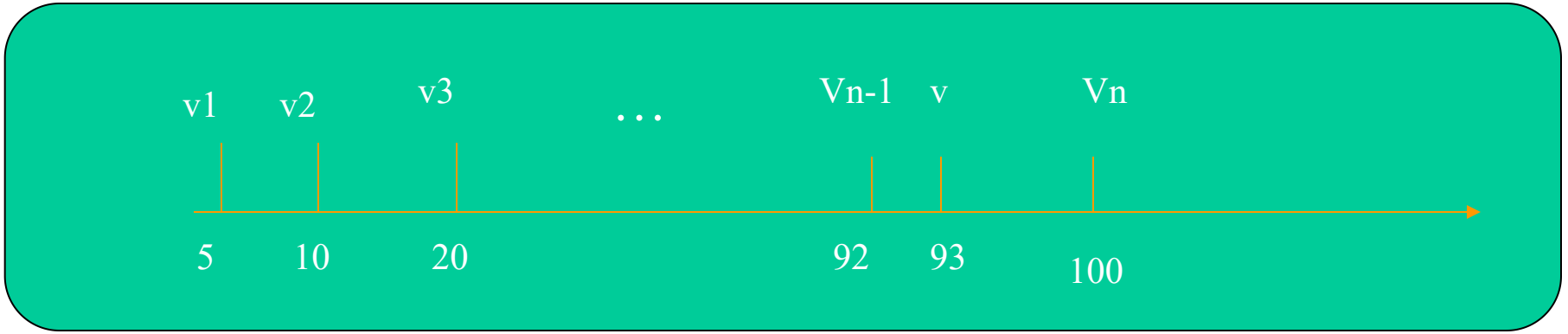
- 多版本规则
  - 如果事务T发布一个write_item(X)操作，并且X的版本Xi具有X所有版本中最高的write_TS(Xi),同时write_TS(Xi)<=TS(T)且read_TS(Xi)>TS(T)，那么撤销并回滚T；否则创建X的一个新版本，并且令read_TS(Xi)= write_TS(Xi)= TS(T)
  - 事务T发布一个read_item(X)操作，如果write_TS(Xi)<=TS(T)并且X的版本Xi具有X所有版本中最高的write_TS(Xi),则把Xi的值返回给事务T,并且将read_TS(Xi)的值置为TS(T)和当前read_TS(Xi)中较大的一个

图示




若读 TS(Ri)=95, 则读 <92, Vn-1> 的值

若写TS(Wk)=93, 则出现了矛盾




于是要拒绝TS(Wk), 否则 TS(Ri)=95读的就是Vn-1, 而不是v的值, 但是按规定 TS(Ri)=95应该读的是v值

## 4.3 采用验证锁的多版本两阶段封锁

- 三种锁方式
  - 读，写，验证
- 四种锁状态
  - 读封锁（read_locked）
  - 写封锁（write_locked）
  - 验证封锁（certify_locked，写提交前验证没有其他读）
  - 未封锁（unlocked）
- 锁相容性
  - 标准模式锁相容性（写锁和读锁）
  - 验证模式锁相容性（写锁、读锁和验证锁）

## 4.3 采用验证锁的多版本两阶段封锁

| | 读 | 写 |
|---|---|---|
| 读 | 是 | 否 |
| 写 | 否 | 否 |

(a) 读/写封锁模式的相容性表

| | 读 | 写 | 验证 |
|---|---|---|---|
| 读 | 是 | 是 | 否 |
| 写 | 是 | 否 | 否 |
| 验证 | 否 | 否 | 否 |

(b) 读/写/验证封锁模式的相容性表

## 4.3 采用验证锁的多版本两阶段封锁

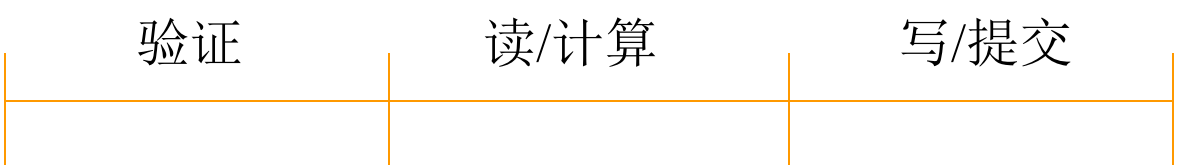
- 多版本2PL的思想
  - 当只有一个单独的事务T持有数据项上的写锁时，允许其他事务T'读该项X，这是通过给予每个项X的两个版本来实现的
    - 一个版本X是由一个已提交的事务写入的
    - 另一个版本X'是每个事务T获得该数据项上写锁时创建的、
  - 当事务T持有这个写锁时，其他事务可以继续读X的已提交版本
  - 事务T可以写X'的值，而不影响X已提交版本的值
  - 但是，一旦T准备提交，它必须在能够提交之前，得到持有写锁的数据项上的验证锁（要等所有读锁释放之后）
  - 一旦获得验证锁，数据项的已提交版本X被置为版本X'的值，版本X'被丢弃，验证锁被释放。

## 5.1 基本思想和假设

- 基本思想
  - 对于冲突操作不像悲观方法那样采取挂起或拒绝的方法，而是让一个事务执行直到完成
- 基于如下假设
  - 冲突的事务是少数（查询为主的系统中，冲突少于5%）
  - 大多数事务可以不受干扰地执行完毕

# 5 分布式数据库系统并发控制的乐观方法

## 5.2 执行阶段划分




悲观法事务执行的各个阶段


乐观法事务执行的各个阶段

# 总　结

- 并发控制的概念和理论
- 分布式数据库系统并发控制的封锁技术
- 分布式数据库系统并发控制的时标技术
- 分布式数据库系统并发控制的多版本技术
- 分布式数据库系统并发控制的乐观方法