第6讲分布式数据库中的事务管理和恢复

1.分布式事务概述

2.分布式事务的执行和恢复

3.两阶段提交协议

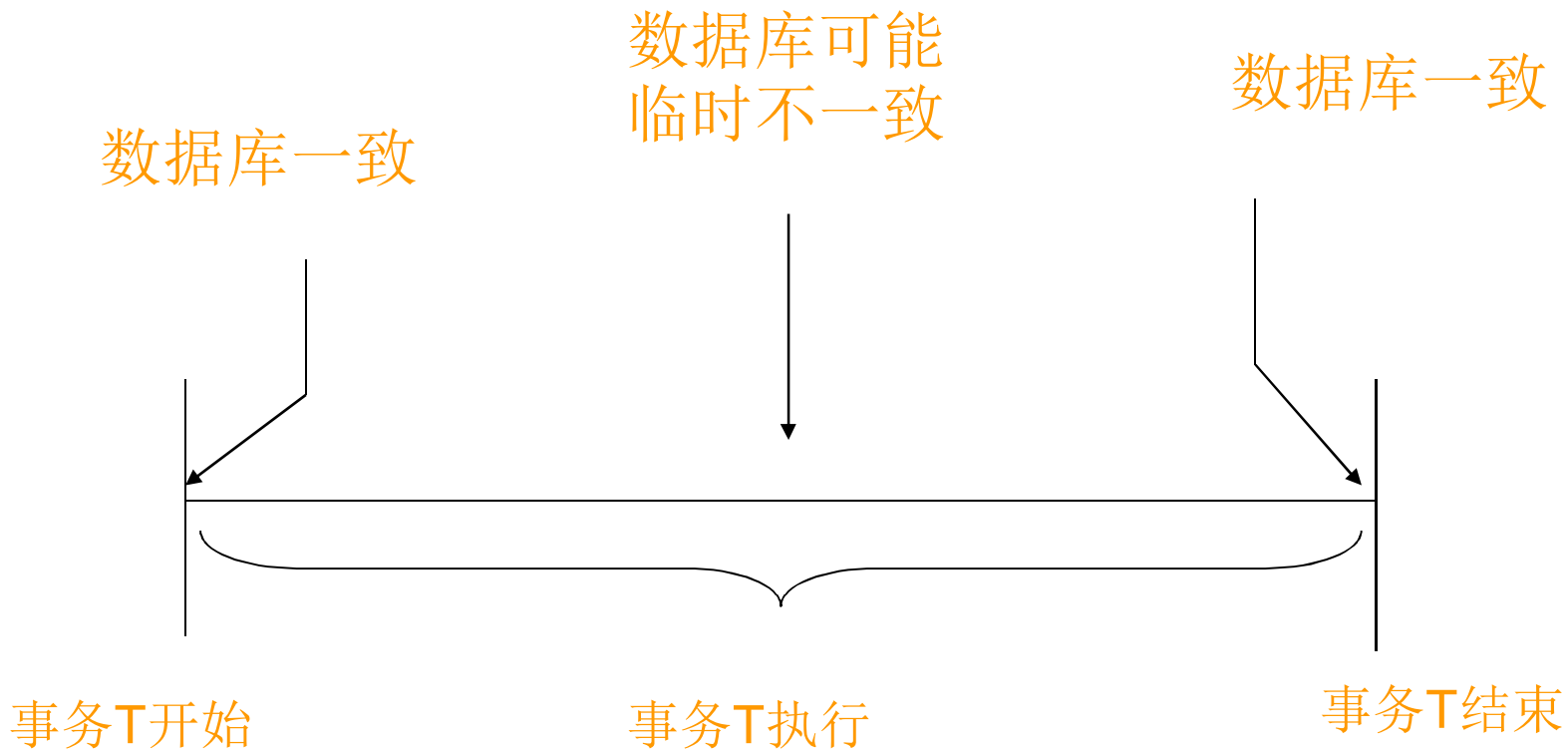
4.分布式数据库中的数据更新

5.总结

1 分布式事务概述

1.1 分布式事务定义和特性

- 事务概念：事务是访问或更新各种数据项的最小逻辑工作单位，它是一个操作序列。
- 当事务提交（commit）时数据库必须是一致的



1 分布式事务概述

1.1 分布式事务定义和特性

分布式事务

- 集中式
  - 事务和操作数据在一个站点上
  - 不存在传输费用
- 分布式
  - 操作数据分布在不同的站点上
  - 事务也在多个站点上执行
  - 站点和通信链路故障都可能导致错误发生
  - 分布式事务的恢复要比集中式事务复杂的多

1 分布式事务概述

1.1 分布式事务定义和特性

分布式数据库中的事务

事务分类:

- 全局事务
  - 通常由一个主事务和在不同站点上执行的子事务组成
  - 主事务：负责事务的开始、提交和异常终止
  - 子事务：完成对相应站点上的数据库的访问操作
- 局部事务
  - 仅访问或更新一个站点上的数据的事务

1 分布式事务概述

1.1 分布式事务定义和特性

分布式事务特性

- **ACID特性**
  - 原子性（**Atomicity**）

    事务的操作要么全部执行, 要么全部不执行 ,保证数据库一致性状态
  - 一致性（**Consistency**）

    事务的正确性， 串行性。并发执行的多个事务,其操作的结果应与以某种顺序串行执行这几个事务所得的结果相同.
  - 持久性（**Durability**）

    当事务提交后, 其操作的结果将永久化, 而与提交后发生的故障无关

1 分布式事务概述

1.1 分布式事务定义和特性

分布式事务特性

- 隔离性（Isolation）
  - 虽然可以有多个事务同时执行，但是单个事务的执行不应该感知其他事务的存在，因此事务执行的中间结果应该对其他并发事务隐藏

- 全局事务的主事务和子事务全部成功提交，才能改变数据库状态，有一个失败，其他子事务操作都要撤销。

1.1 分布式事务定义和特性

分布式事务特性举例

- 从账号A向账号B转账 $50:

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)

1 分布式事务概述

1.1 分布式事务定义和特性

分布式事务的一般结构

Begin Transaction原语：开始一个事务

T1[]
T2[]
:
:
Tn[]

子事务或操作序列

Commit原语：事务成功完成的结束

Rollback或Abort原语：事务失败的结束

1 分布式事务概述

1.1 分布式事务定义和特性

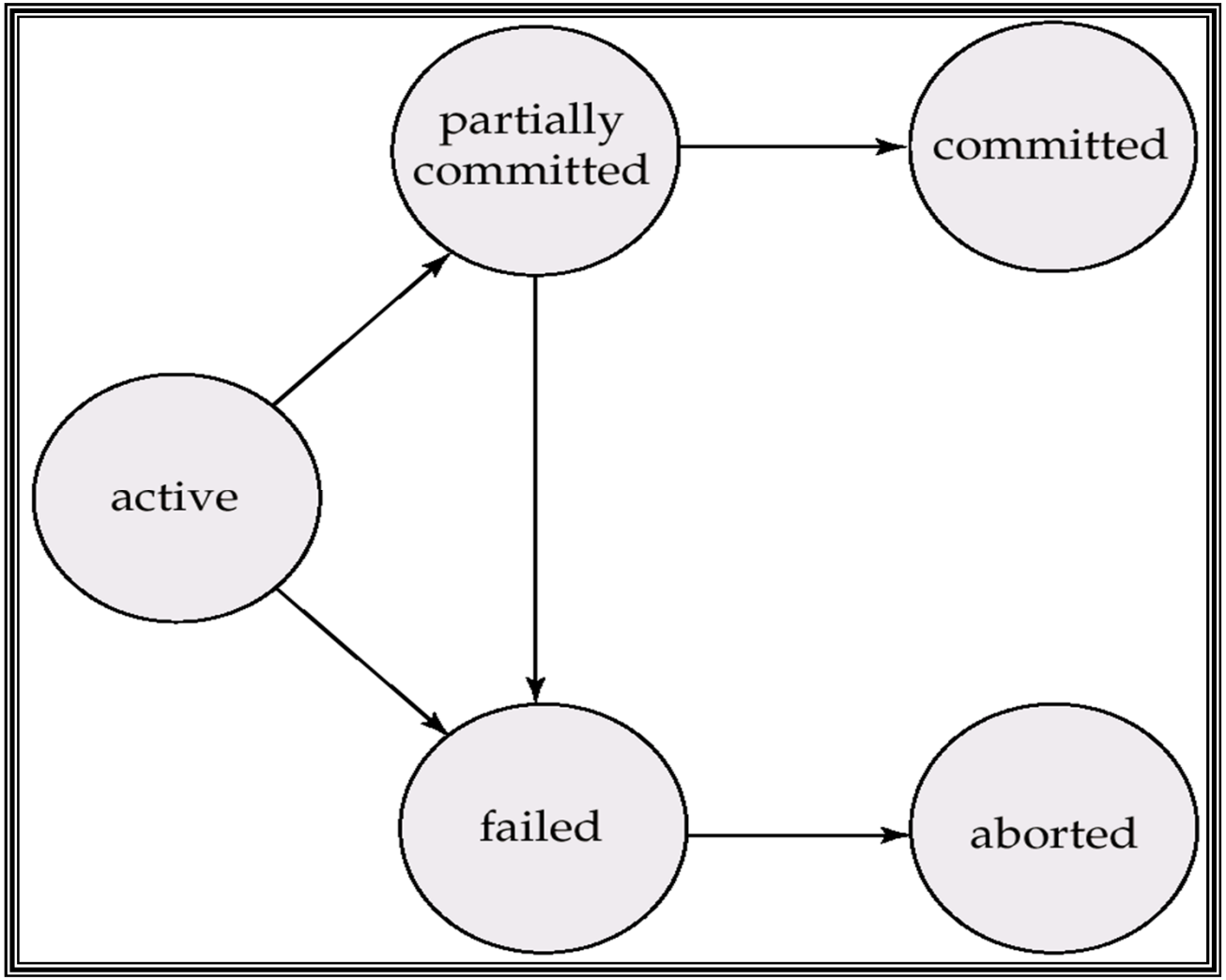
分布式事务的状态

- 活动　从事务开始执行的初始状态始， 事务执行中保持该状态
- 部分提交　事务的最后一个语句执行后进入该状态.
- 失败　一旦发现事务不能正常执行时进入该状态
- 夭折　当事务被回滚后，数据库恢复到事务开始执行前的状态　事务夭折后有两种选择
  - 重启动　仅当没有内部逻辑错误时
  - 杀死
- 提交　当事务成功执行后.

1 分布式事务概述

1.1 分布式事务定义和特性

分布式事务的状态



1 分布式事务概述

1.1 分布式事务定义和特性

转账应用与

事务在两个账户之间执行“基金汇兑”操作。

如果汇兑的金额小于转出帐号现有金额，就撤销

如果大于等于就提交

全局关系

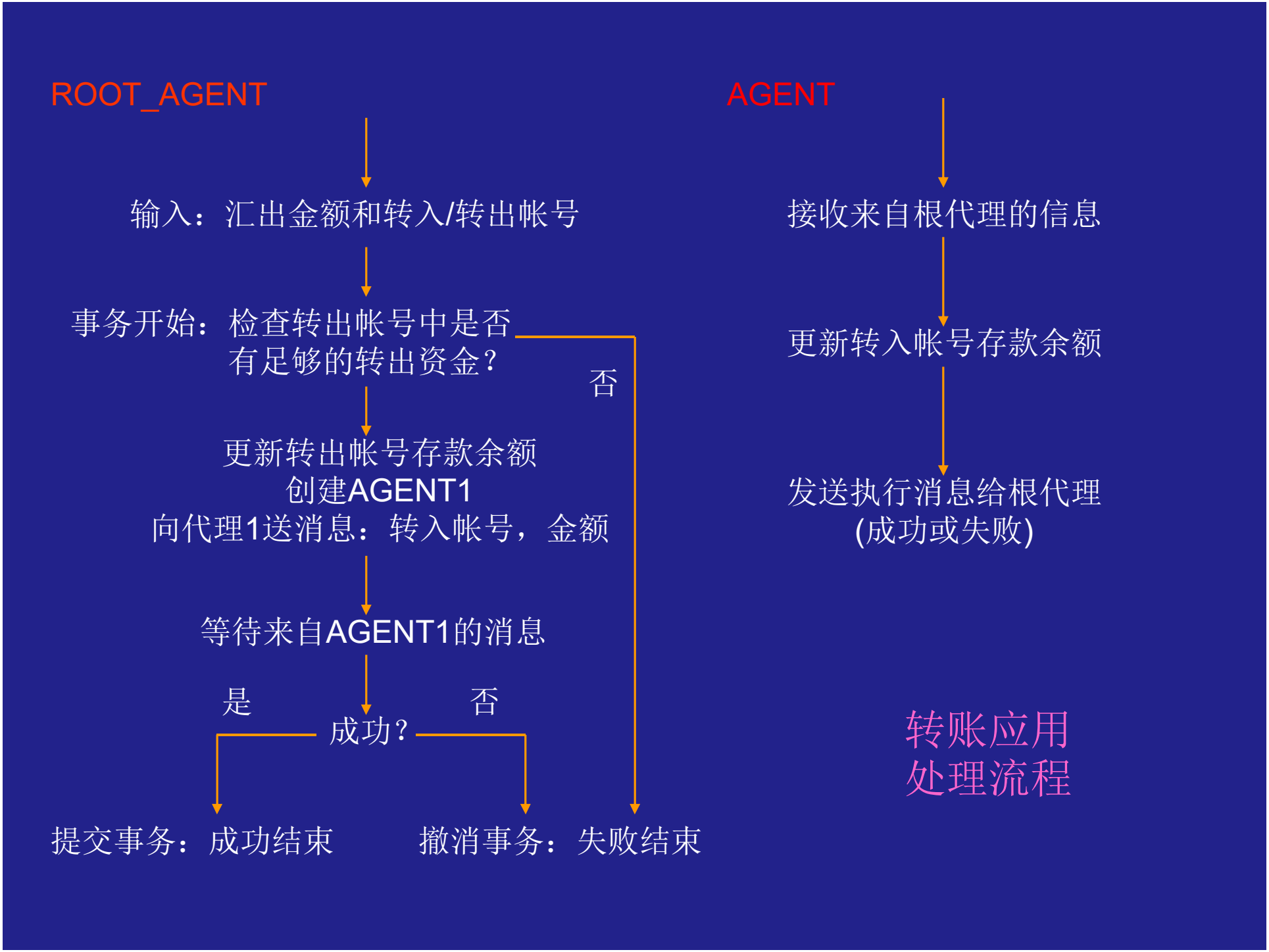
Account (Account-number, Amount)

假设账户分布在网络的不同站点上。

全局级转帐事务

FUND_TRANSFER:

```
read (terminal,$AMOUNT,$FROM_ACC,$TO_ACC);
begin_transaction;
select AMOUNT into $FROM_AMOUNT from ACCOUNT
      where ACCOUNT_NUMBER=$FROM_ACC;
if $FROM_AMOUNT-$AMOUNT<0 then abort
else begin
        update ACCOUNT
              set AMOUNT = AMOUNT-$AMOUNT
              where ACCOUNT_NUMBER = $FROM_ACC;
        update ACCOUNT
              set AMOUNT = AMOUNT+$AMOUNT
              where ACCOUNT_NUMBER = $TO_ACC;
        commit
end
```


ROOT_AGENT
输入：汇出金额和转入/转出帐号
事务开始：检查转出帐号中是否有足够的转出资金？
否
更新转出帐号存款余额
创建AGENT1
向代理1送消息：转入帐号，金额
等待来自AGENT1的消息
是
否
成功？
提交事务：成功结束
撤消事务：失败结束
AGENT
接收来自根代理的信息
更新转入帐号存款余额
发送执行消息给根代理
(成功或失败)
转账应用
处理流程

转账事务的两个代理

```
ROOT_AGENT:
        read(terminal,$AMOUNT,$FROM_ACC,$TO_ACC);
        begin_transaction
                select AMOUNT into $FROM_AMOUNT from ACCOUNT
                        where ACCOUNT_NUMBER=$FROM_ACC;
                if $FROM_AMOUNT-$AMOUNT<0 then abort
                else begin
                     update ACCOUNT
                     set AMOUNT = AMOUNT-$AMOUNT
                      where ACCOUNT_NUMBER = $FROM_ACC;
        create AGENT;
        send to AGENT($AMOUNT,$TO_ACC);
         wait()
        commit
    end
AGENT:
    receive from ROOT_AGENT($AMOUNT,$TO_ACC);
    update ACCOUNT set AMOUNT=AMOUNT+$AMOUNT where
          ACCOUNT=$TO_ACC;
    send to ROOT_AGENT('SUCCESS'/'FALL')
```

1.2 分布式事务管理的问题和目标

分布式事务管理问题

- 处理数据项的多个副本
  - 分布式事务处理负责保持同一数据的多个副本之间的一致性。
- 单个站点的故障
  - 一个站点或多个站点故障时，**DDBMS**继续与其他正常运行的站点一起继续工作
  - 当故障站点恢复时， **DDBMS**协同故障站点的**DBMS**，使该站点与系统连接时，局部数据库与其他站点同步
- 通信网络的故障
  - 必须能够处理两个或者多个站点间的通信网络故障
- 分布式提交
  - 如果提交分布式事务过程中有一个站点发生故障，提交就会产生问题
  - 两阶段提交协议用于解决这一问题

分布式事务管理目标

- 目标：事务能有效、可靠、并发的执行
  - 维护事务的**ACID**性质
  - 获得最小的主存和**CPU**开销，降低报文数目，加快响应时间
  - 获得最大限度的可靠性和可用性

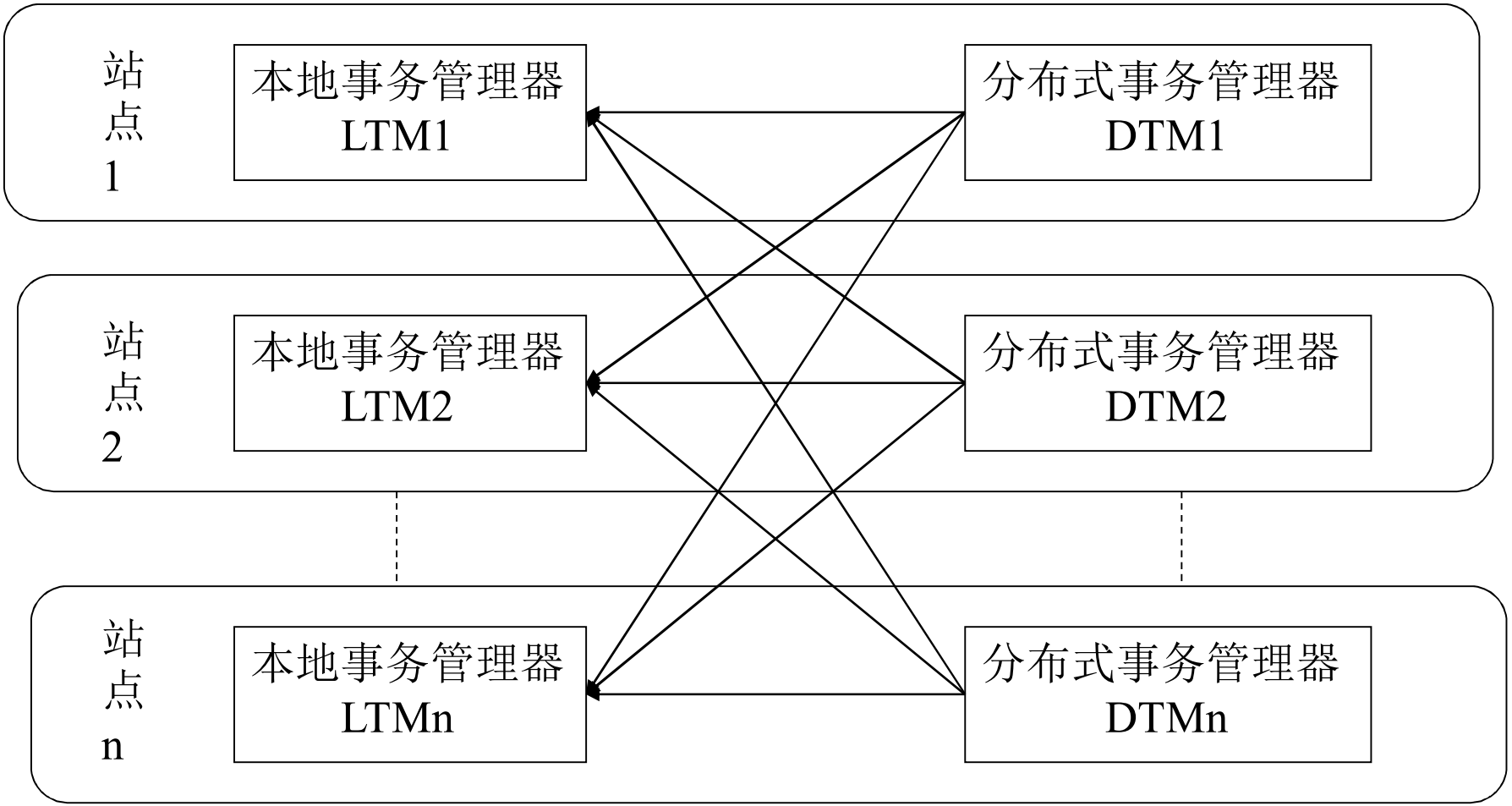
2 分布式事务的执行与恢复

2.1 分布式事务管理的抽象模型

LTM: Local Transaction Manager

DTM: Distributed Transaction Manager

抽象模型



2 分布式事务的执行与恢复

2.1 分布式事务管理的抽象模型

事务管理

- **DTM功能**
  - 保证分布式事务**ACID**特性，
    - 特别是原子性，使每一站点的子事务都成功执行，或者都不执行。
    - 通过向各站点发**begin-transaction,commit**或者**abort,create**原语来实现的
  - 负责协调由该站点发出的所有分布式事务的执行
    - 启动分布式事务的执行
    - 将分布式事务分解为子事务，并将其分派到恰当的站点上执行
    - 决定分布式事务的终止（子事务都提交或者都撤销）
  - 支持分布式事务执行位置透明性
    - 实现了对网络上各站点的各子事务的监督和管理
    - 完成对整个分布式事务执行过程的调度和管理
    - 保证分布式数据库系统的高效率

2 分布式事务的执行与恢复

2.1 分布式事务管理的抽象模型

事务管理

- **LTM功能**
  - 保证本地事务的**ACID**特性
  - 维护一个用于恢复的日志，代替**DTM**把分布事务的执行与恢复信息记入日志
  - 参与适当的并发控制模式，以协调在该站点上执行的事务的并发执行。

2.2 分布式数据库系统中的故障

- 故障类型
  - 事务故障
    - 由非预期的、不正常的程序结束所造成的故障，如：计算溢出、完整性破坏、操作员干预、输入输出错误、死循环等）
    - 处理方法：内存、磁盘上信息没有损失，使用**Log**做**Rollback**
  - 系统故障
    - 造成系统停止运行的任何事件，要求系统重启动，如**CPU**出错、缓冲区满、系统崩溃等
    - 处理方法：内存、**I/O Buffer**内容皆丢失，**DB**没有破坏，恢复时，搜索**Log,** 确定**Rollback**的事务。

2.2 分布式数据库系统中的故障

– 介质故障：

- 辅助存储器介质遭破坏
- 处理方法：如数据丢失, 日志无损失，从某个Dump状态开始执行已提交事务；数据与日志都丢失 不可能完全恢复

以上三种可以统称为站点故障.

2 分布式事务的执行与恢复

2.2 分布式数据库系统中的故障

通讯故障

- 报文故障
  - 报文错，报文失序、丢失、延迟
- 网络分割故障（网络断连）

通讯发生, 某个报文Message从Site x 发往Site y, 正常情况:

(a) 在某时间段Dmax 之前, x 站点收到y发回的应答信息 (Ack)

(b) y收到的Message是一个合适的次序

(c) Message本身的信息是正确的

但是当某个Dmax之后, x还没收到y的Ack, 则可能发生:

(a) Message 或 Ack 信息丢失

(b) 网络分割, 即网络不通

2.3 事务故障恢复的基本概念

- 事务恢复
  - 当发生故障时，保证事务原子性的措施称为事务故障恢复，简称事务恢复
  - 主要依靠日志来实现
- 事务状态转移跟踪（操作）
  - **Begin_transaction**：标记事务开始执行
  - **Read & write**：表示事务对某个数据项进行读写
  - **End_transaction**：表示读写操作已完成，标记事务执行结束
  - **Commit_transaction**：表示事务已经成功结束，任何改变已不可更改
  - **Rollback (abort)**：表示事务没有成功结束，撤销事务对数据库所作的任何改变

2.3 事务故障恢复的基本概念

- 事务的提交点
  - 当事务T所有的站点数据库存取操作都已成功执行；
  - 所有操作对数据库的影响都已记录在日志中。到达提交点
  - 提交点后事务就成为已提交的事务，并假定其结果以永久记录在数据库中
  - 事务在日志中写入提交记录[commit,T]
  - 在系统发生故障时，需要扫描日志，检查日志中写入[start_transaction,T],但没有写入[commit,T]的所有事务T
  - 恢复时必须回滚这些事务以取消他们对数据库的影响
  - 此外，还必须对日志中记录的已提交子事务的所有写操作进行恢复。

2.3 事务故障恢复的基本概念

- 事务的提交点相关操作
  - 日志文件保存到磁盘上
  - 一般先将文件的相关块，从磁盘拷贝到主存的缓冲区，然后更新，再写回磁盘
  - 缓冲区中会经常存在一个或多个日志文件块，写满后一次性写回磁盘
  - 系统崩溃时，主存中的信息会丢失，这些信息无法利用
  - 因此，事务到达提交点之前，未写到磁盘的日志必须写入，称为事务提交前强制写日志。

    （强制写入机制的效率如何？）

2.3 事务故障恢复的基本概念

- 日志
  - **Log：记录所有对DB的操作**
  - 事务标识：每个事务给定一个具有惟一性的标识符
  - **Log记录项：**
    - **[start_transaction, T],**
    - **[write_item, T, x, 旧值, 新值]**
    - **[read_item, T, x]**
    - **[commit, T]**
    - **[abort, T]**
  - 写动作：写Log比写数据优先
  - **Log存储：一般存在盘上，还会定期备份到磁带上**

2.3 事务故障恢复的基本概念

Log举例

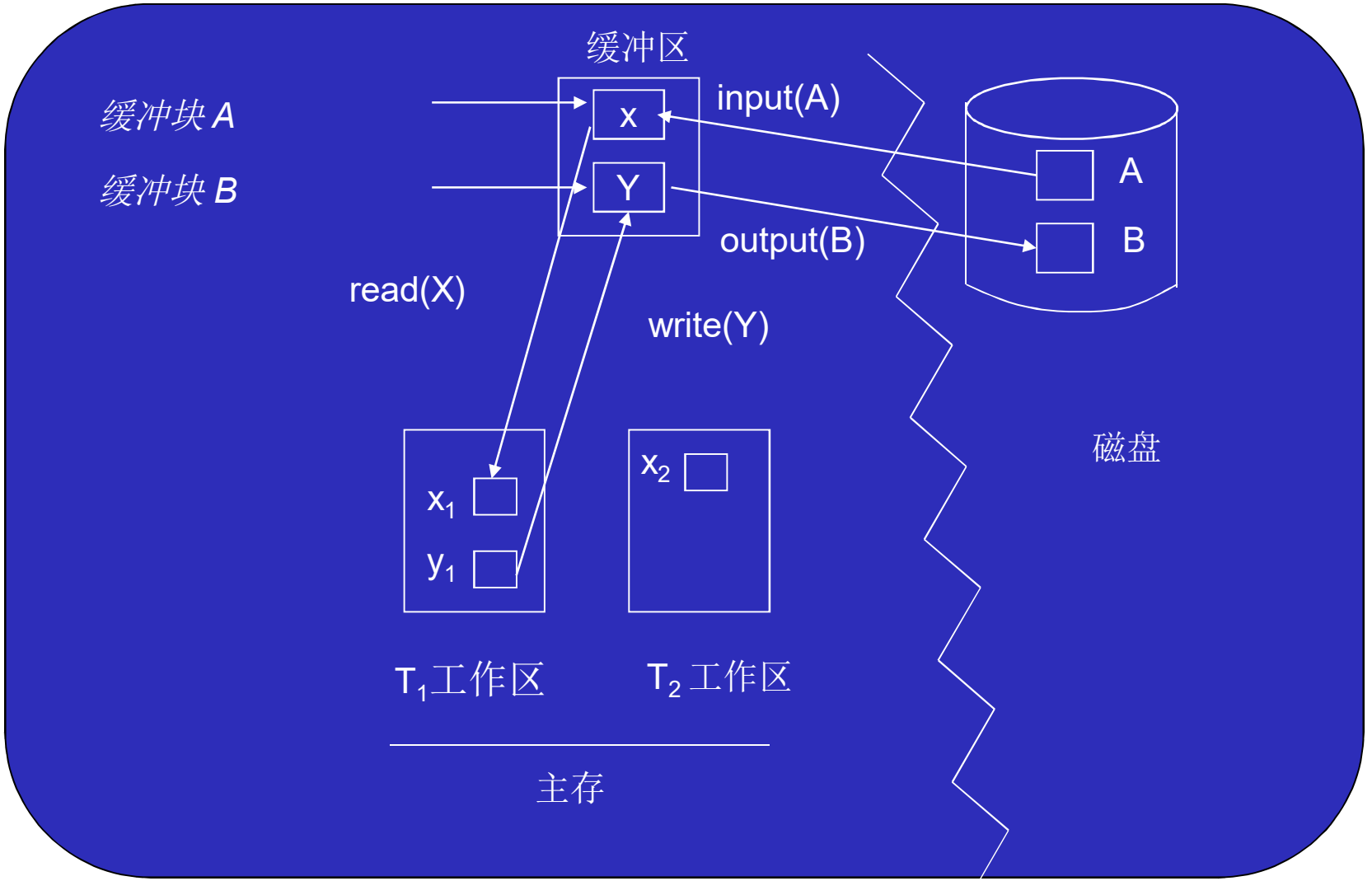
Log	Write	Output
<**start**,T_0>		
<write,T_0, A, 1000, 950>		
<write,T_0, B, 2000, 2050>		
	$A = 950$	
	$B = 2050$	
<**commit**,T_0 >		
<**start**,T_1 >		
<write,T_1, C, 700, 600>		
	$C = 600$	
		B_B, B_A
<**commit**, T_1 >		
		B_C

注: B_X 表示含有X的存储块.

2 分布式事务的执行与恢复

2.3 事务故障恢复的基本概念

数据访问



2.3 事务故障恢复的基本概念

- 检查点（Checkpoint）
  - 设置一个周期性（时间/容量）操作点

  a) Log Buffer内容写入Log数据集

  b) 写检查点Log信息：当前活动事务表, 每个事务最近一次Log记录在Log文件中的位置

  c) DB Buffer内容写入DB

  d) 将本次检查点Log项在Log文件中的地址记入“重启动文件”

2.4 事务故障的恢复

- 事务本身也会发生故障，也是主要通过日志来实现恢复
- 恢复原则
  - 孤立和逐步退出事务的原则

    undo 事务已对**DB**的修改 **(** 不影响其他事务的可排除性局部故障，如事务操作的删除、超时、违反完整性原则、资源限制和死锁等**)**
  - 成功结束事务原则

    Redo 已成功事务的操作
  - 夭折事务原则

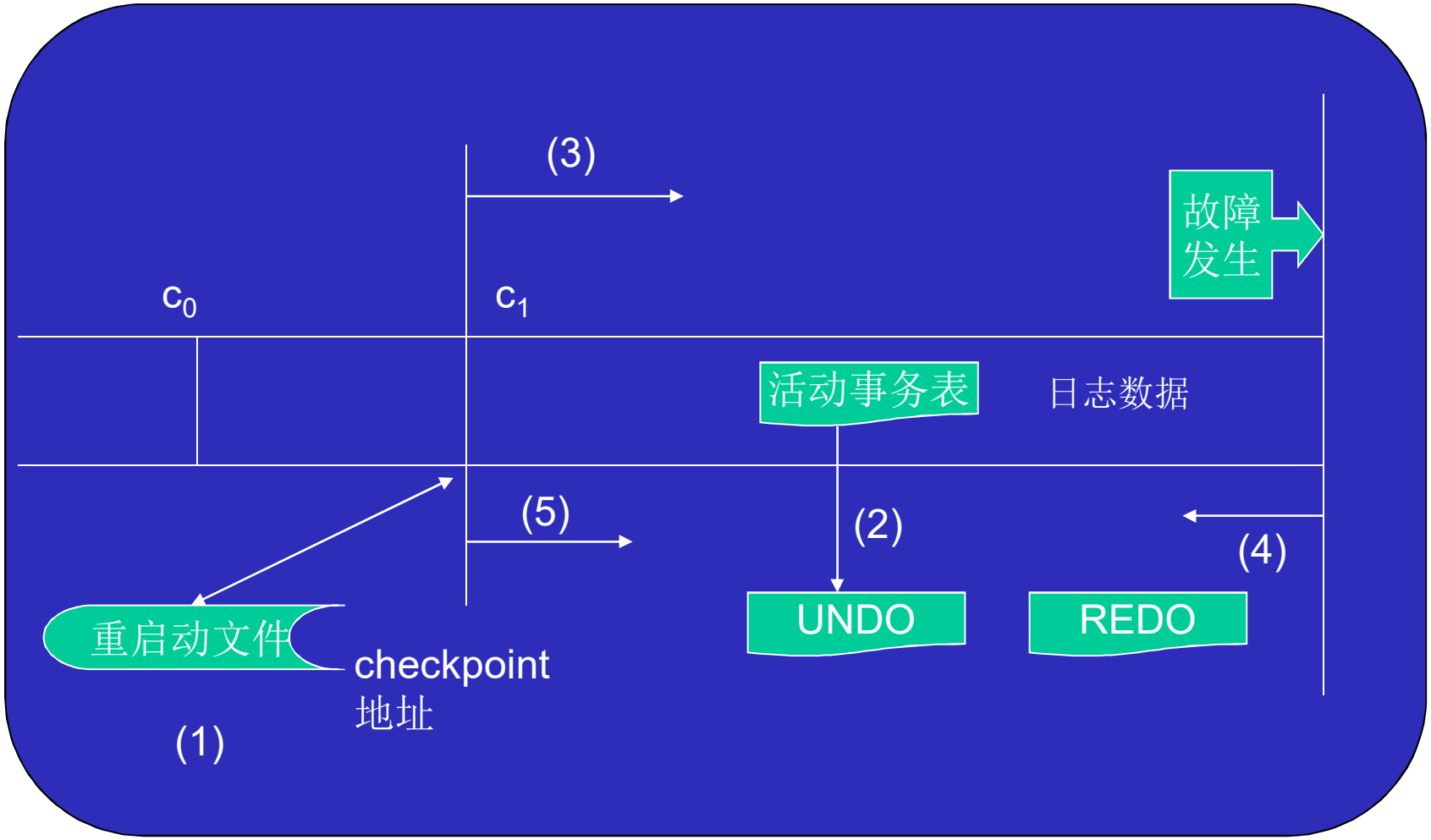
    撤销全部事务**,** 恢复到初态，两种做法：利用数据备份和**Undo**

2.4 事务故障的恢复

- 本地事务恢复 (与集中式恢复相同)
  - 从“重启动文件” 读出最近Checkpoint的地址, 并定出Checkpoint在Log文件中的位置
  - 创建Redo表（初态为空）, Undo表(即Checkpoint相应内容中的活动事务表)
  - 检查得出Undo事务（向前扫描，遇到begin transaction的log记录，其对应的事务）与Redo事务（向前扫描，遇到commit的log记录，其对应事务）
  - 反向扫描Log, 将Undo表中事务回滚, 直到遇到对应的Begin Trans
  - 正向扫描Redo事务的Log记录, 并执行之, 直到对应的Commit记录

2 分布式事务的执行与恢复

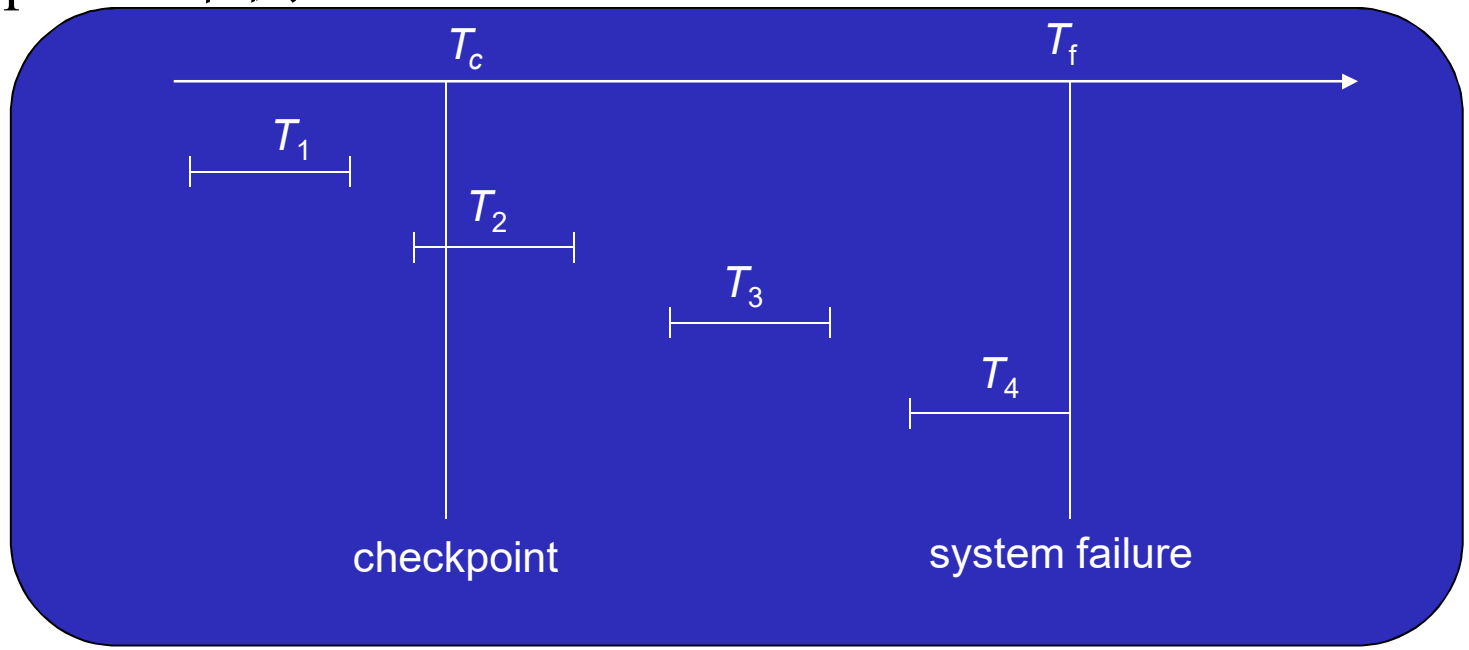
2.4 事务故障的恢复



利用日志进行事务恢复的过程

2.4 事务故障的恢复

Checkpoints 举例



- T_1 可以忽略 (因为有检查点，更新已经被写入磁盘)
- T_2 和 T_3 redo.
- T_4 undo

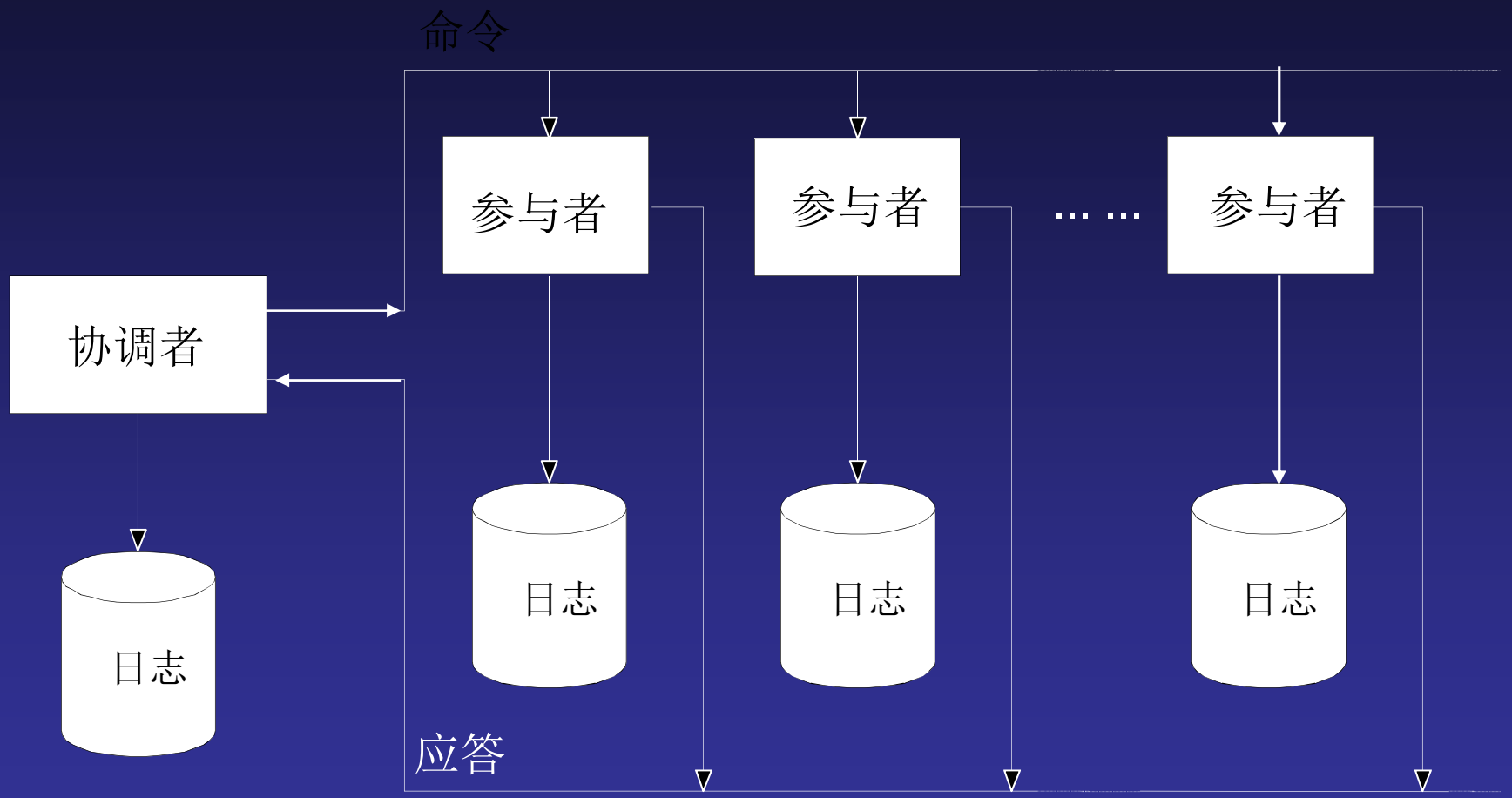
3 两阶段提交协议

3.1 基本思想和内容

- 基本思想
  - 将本地原子性提交行为的效果扩展到分布式事务, 保证了分布式事务提交的原子性, 并在不损坏Log的情况下, 实现快速故障恢复, 提高DDB系统的可靠性.
  - 第一阶段：表决阶段
  - 第二阶段：执行阶段
- 两类代理
  - 协调者(Coordinator)：提交和撤销事务的决定权，一般是总代理
  - 参与者(Participants)：负责在本地数据库中执行写操作，并且向协调者提出提交和撤销子事务的意向

3 两阶段提交协议

3.1 基本思想和内容

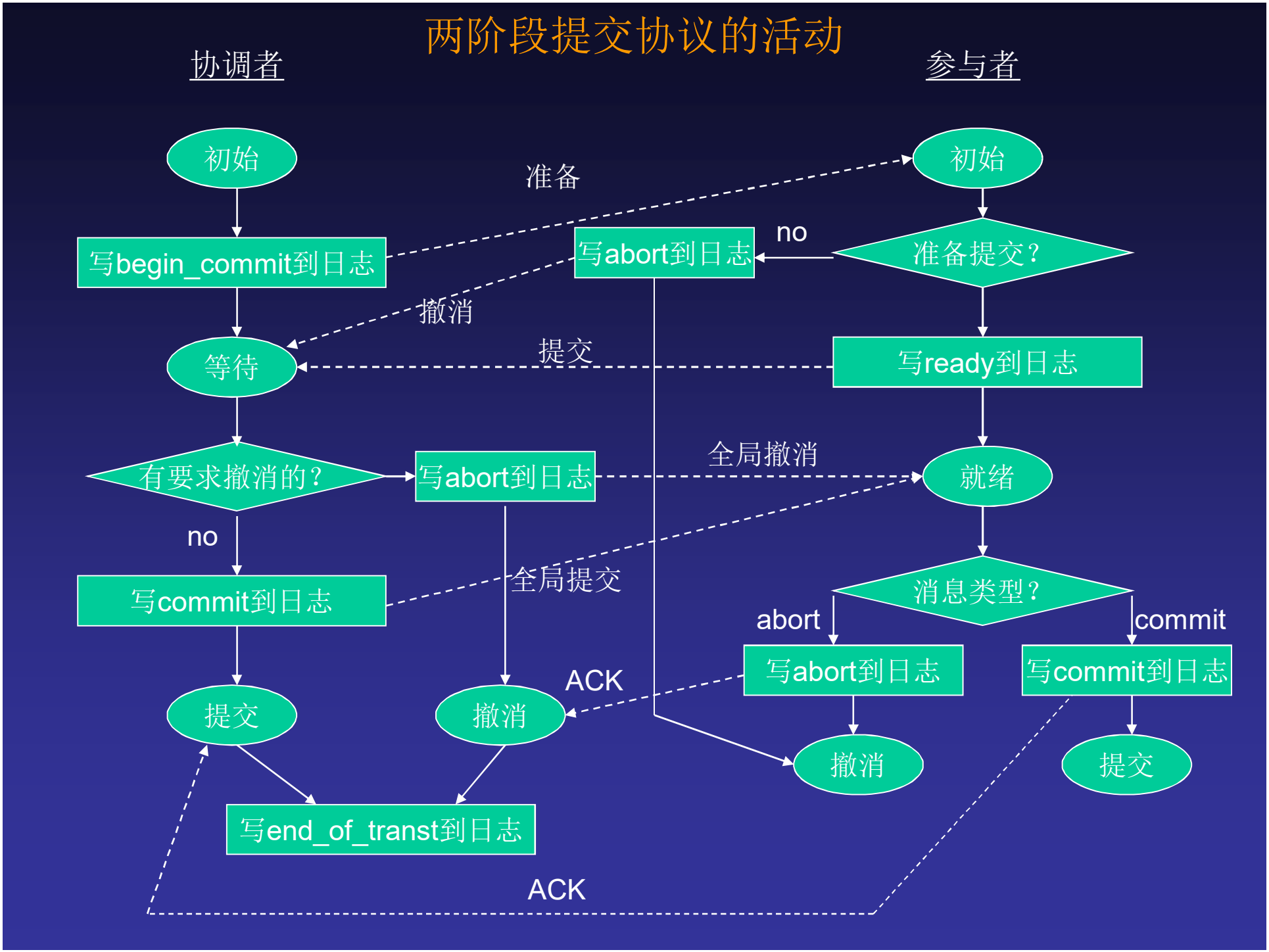


2PC中协调者和参与者的关系

3 两阶段提交协议

3.1 基本思想和内容

- 表决阶段
  - 目的是形成一个共同的决定
  - 首先，协调者给所有参与者发送“准备”消息，进入等待状态
  - 其次，参与者收到“准备”消息后，检查是否能够提交本地事务
    - 如能，给协调者发送“建议提交”消息,进入就绪状态
    - 如不能，给协调者发送“建议撤销”消息，可以单方面撤销
  - 第三，协调者收到所有参与者的消息后，他就做出是否提交事务的决定，
    - 只要有一个参与者投了反对票，就决定撤销整个事务，发送“全局撤销”消息给所有参与者，进入撤销状态
    - 否则，就决定提交整个事务，发送“全局提交”消息给所有参与者，进入提交状态
- 执行阶段
  - 实现表决阶段的决定，提交或者撤销


两阶段提交协议的活动
协调者
参与者
初始
初始
准备
写begin_commit到日志
写abort到日志
no
准备提交？
撤消
提交
等待
写ready到日志
有要求撤消的？
写abort到日志
全局撤消
就绪
no
写commit到日志
全局提交
消息类型？
abort
commit
ACK
写abort到日志
写commit到日志
提交
撤消
撤消
提交
写end_of_transt到日志
ACK

3 两阶段提交协议

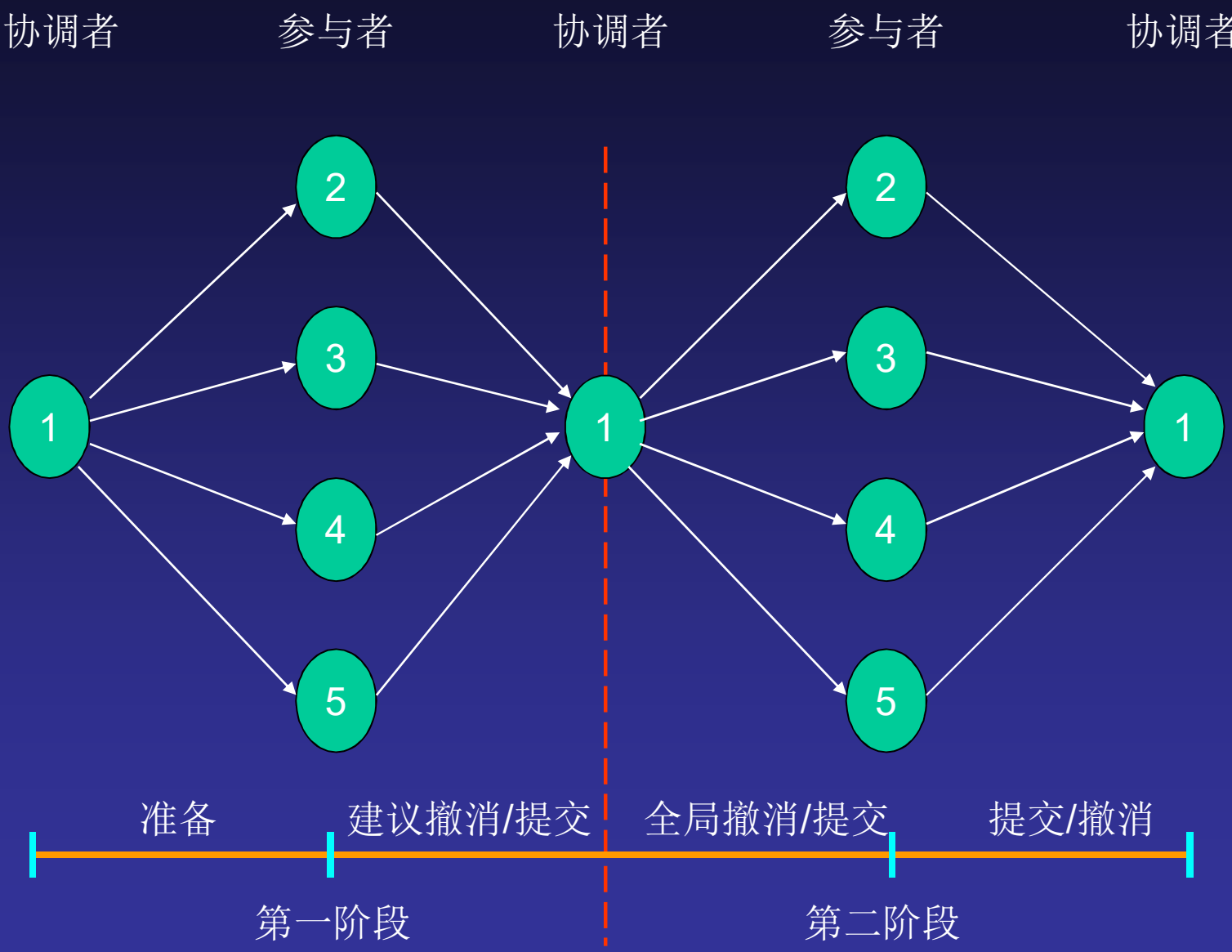
3.1 基本思想和内容

- **2PC协议的重要特点**
  - 允许参与者单方面撤销事务
  - 一旦参与者确定了提交或撤销协议，它就不能再更改它的提议
  - 当参与者处于就绪状态时，根据协调者发出的消息种类，它可以转换为提交状态或者撤销状态
  - 协调者根据全局提交规则做出全局终止决定
  - 协调者和参与者可能进入互相等待对方消息的状态，使用定时器，保证退出消息等待状态

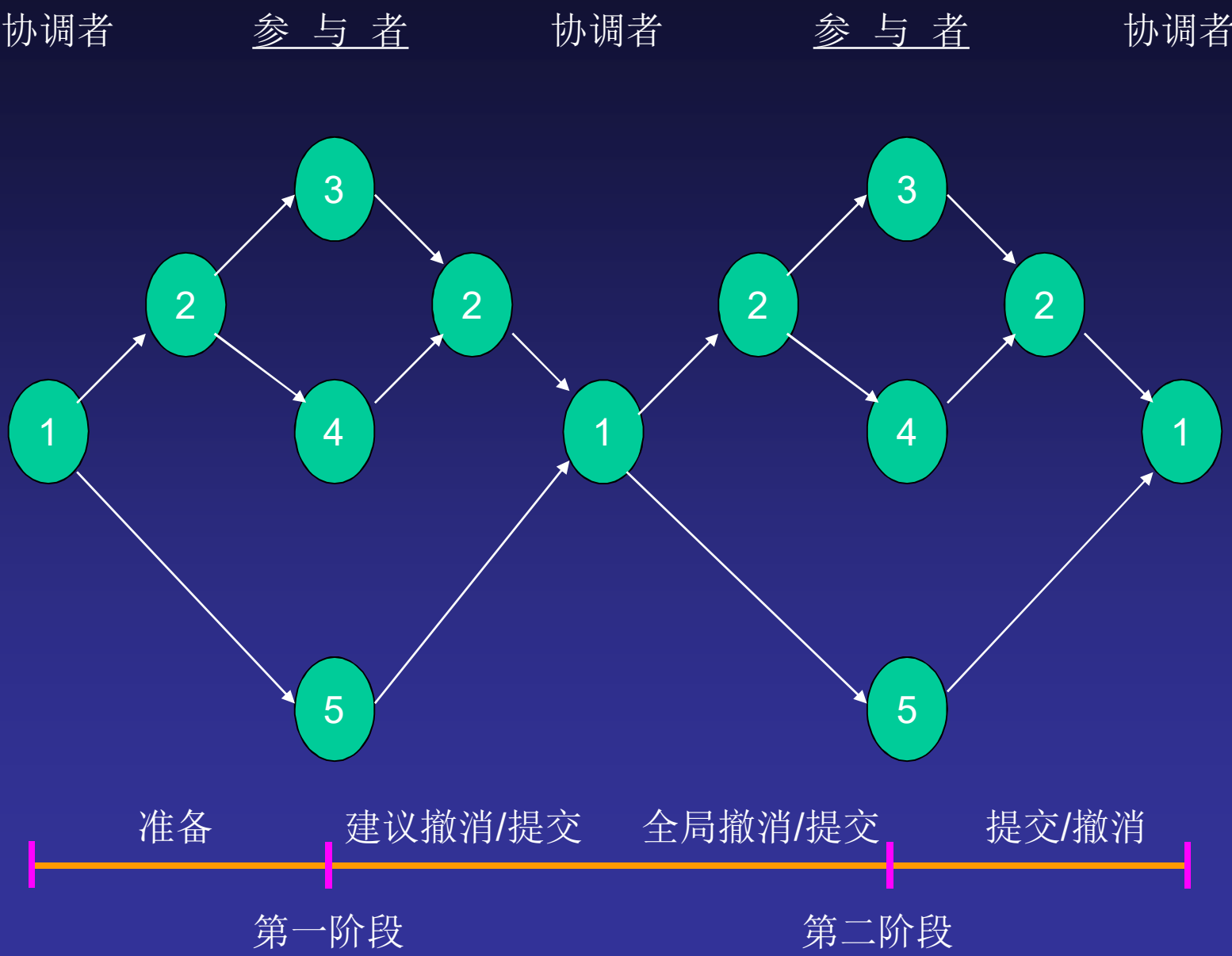
3.2 通信结构

- 集中式
  - 通讯只发生在协调者和参与者之间，参与者之间不交换信息
- 分层式
  - 协调者是在树根的**DTM**代理者，协调者与参与者之间的通讯不用直接广播的方法进行，而是使报文在树中上下传播。每个**DTM**代理是通信树的一个内部节点，它从下层节点处收集报文或向它们广播报文。
- 线性
  - 参与者之间可以互相通信。系统中的站点间要排序，消息串行传递。支持没有广播功能的网络
- 分布式
  - 允许所有参与者在第一阶段相互通信，从而可以独立做出事务终止决定。

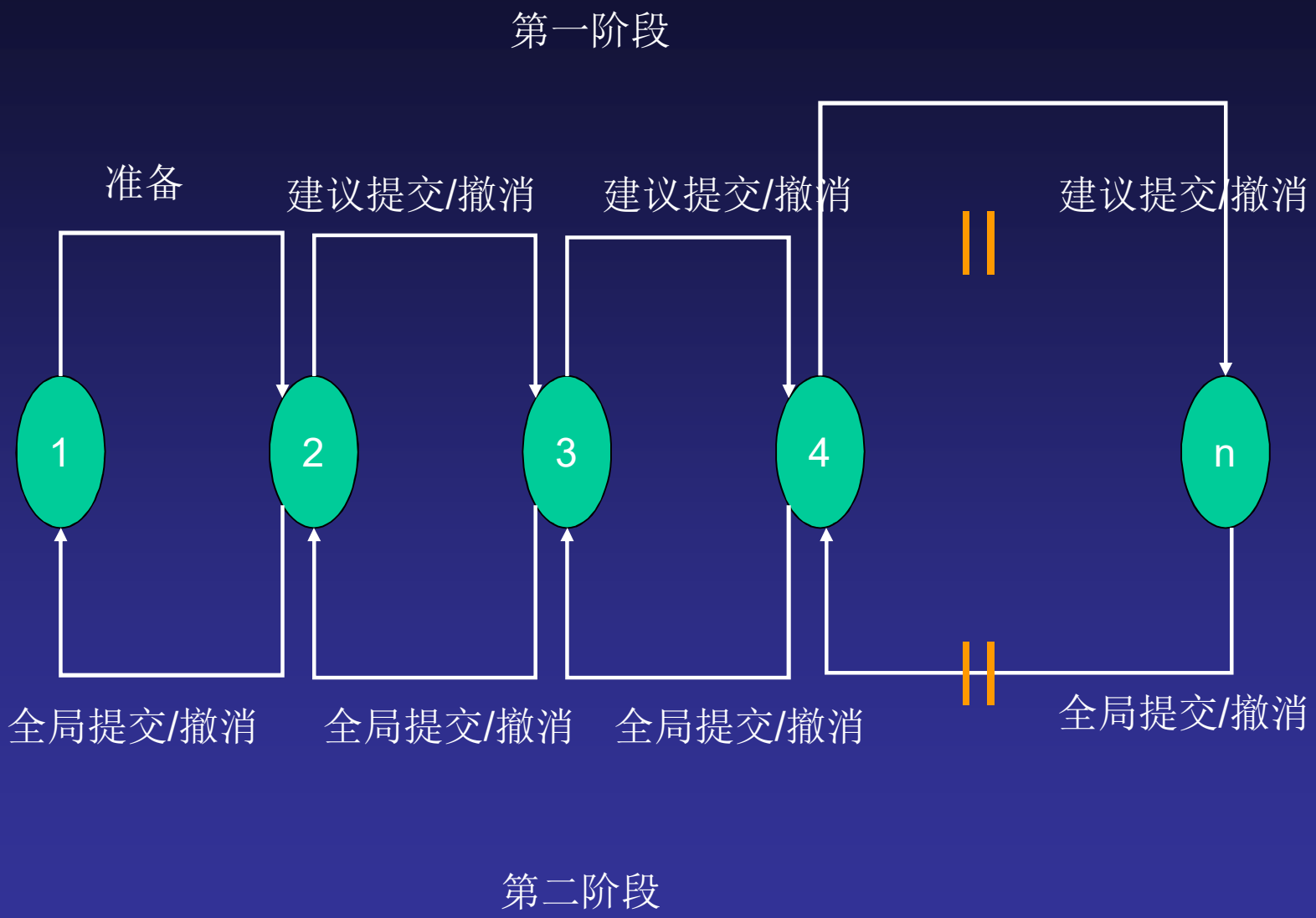
集中式


协调者
参与者
协调者
参与者
协调者
1
2
3
4
5
1
2
3
4
5
1
准备
建议撤消/提交
全局撤消/提交
提交/撤消
第一阶段
第二阶段

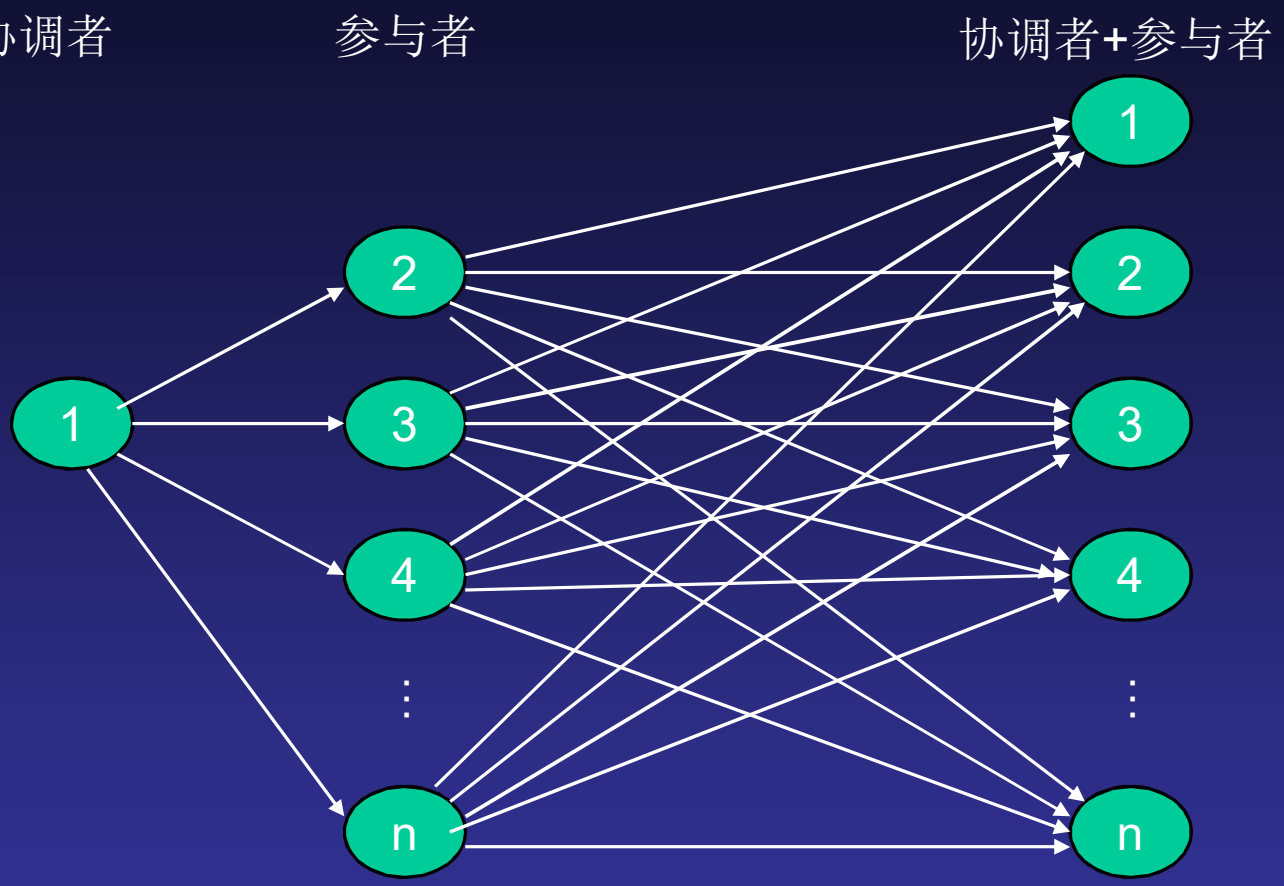
分层式


协调者
参 与 者
协调者
参 与 者
协调者
3
2
2
1
4
1
5
3
2
2
1
4
1
5
准备
建议撤消/提交
全局撤消/提交
提交/撤消
第一阶段
第二阶段

线性式


第一阶段
准备
建议提交/撤消
建议提交/撤消
建议提交/撤消
1
2
3
4
n
全局提交/撤消
全局提交/撤消
全局提交/撤消
全局提交/撤消
第二阶段

分布式



3 两阶段提交协议

3.2 两阶段提交协议和故障恢复

1. 站点故障

a> 参与者在将“Ready”记录入Log之前故障

- 此时协调者(C)达到超时，Abort发生。站点（P）恢复时，重启动程序将执行Abort，不必从其他站点获取信息。

b> 当将“Ready”写入Log后，站点故障

- 此时所有运行的站点都将正常结束事务（Commit/Abort)。P恢复时，因为P已Ready，所以不可判定C的最终决定。因此恢复时，重启动程序要询问C或其他站点。

c> 当C将“Prepare”写入Log，但“G-commit”/”G-abort”还没有写入前故障

- 所有回答“Ready”的P等待C恢复。C重启动时，将重开提交协议，重发“Prepare”，于是P要识别重发。

d> C在将“G-commit”/”G-abort”写入Log后，“end_of_transt”没有写入前故障

- 收到命令的P正常执行，C重启动程序必须再次向所有P重发命令。以前没有收到命令的P也必须等待C恢复，P要识别两次命令。

e> “end_of_transt”写入Log后故障

- 无任何动作发生

2. 报文丢失

a> 从P发出的“Ready”/“Abort”报文丢失

- C达到超时，整个事务执行“G-abort”。

b> “Prepare”报文丢失

- P等待，C得不到回答，结果同2.a>

c> “G-commit”/”G-abort”报文丢失

- P处于不确定状态。回答“Abort”的可以确定其工作，回答“Ready”的不行。此时可以修改加入计时器，超时则申请重发命令。

d> “Ack”报文丢失

- C超时，可重发“G-commit”/”G-abort”命令，P无论是否有活动，都重发“Ack”报文

3. 网络分割

站点假设分成两组：协调者组和参与者组。

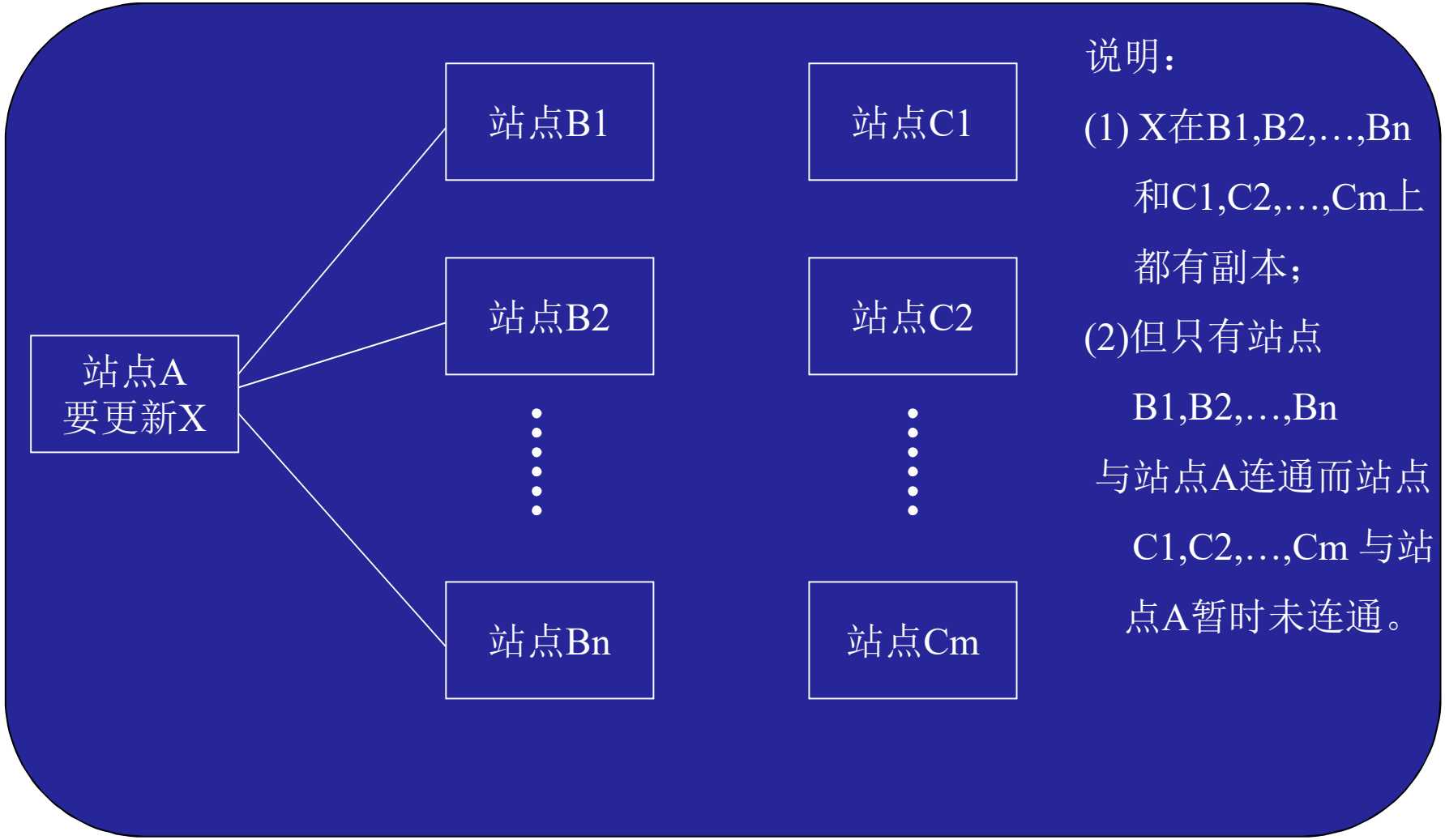
- 一组是协调者，一组是参与者。于是从协调者看是参与者组故障。 从参与者组看是协调者站点故障。其动作按产生网络分割时的状态，类似站点故障处理。

4.1 多站点数据更新

- 多站点数据更新
  - 方法：站点A上有事务T对X更新, X在B1,…Bn和C1,…Cm上有副本, 则也要对这些副本更新
  - 问题
    - 多个站点同时更新不现实(每一个站点某一时刻与站点A连通的概率小于1）
    - 对未连通站点上的副本更新时出错, 更新的顺序也不一定是连通顺序

4 分布式数据库中的数据更新

4.1 多站点数据更新

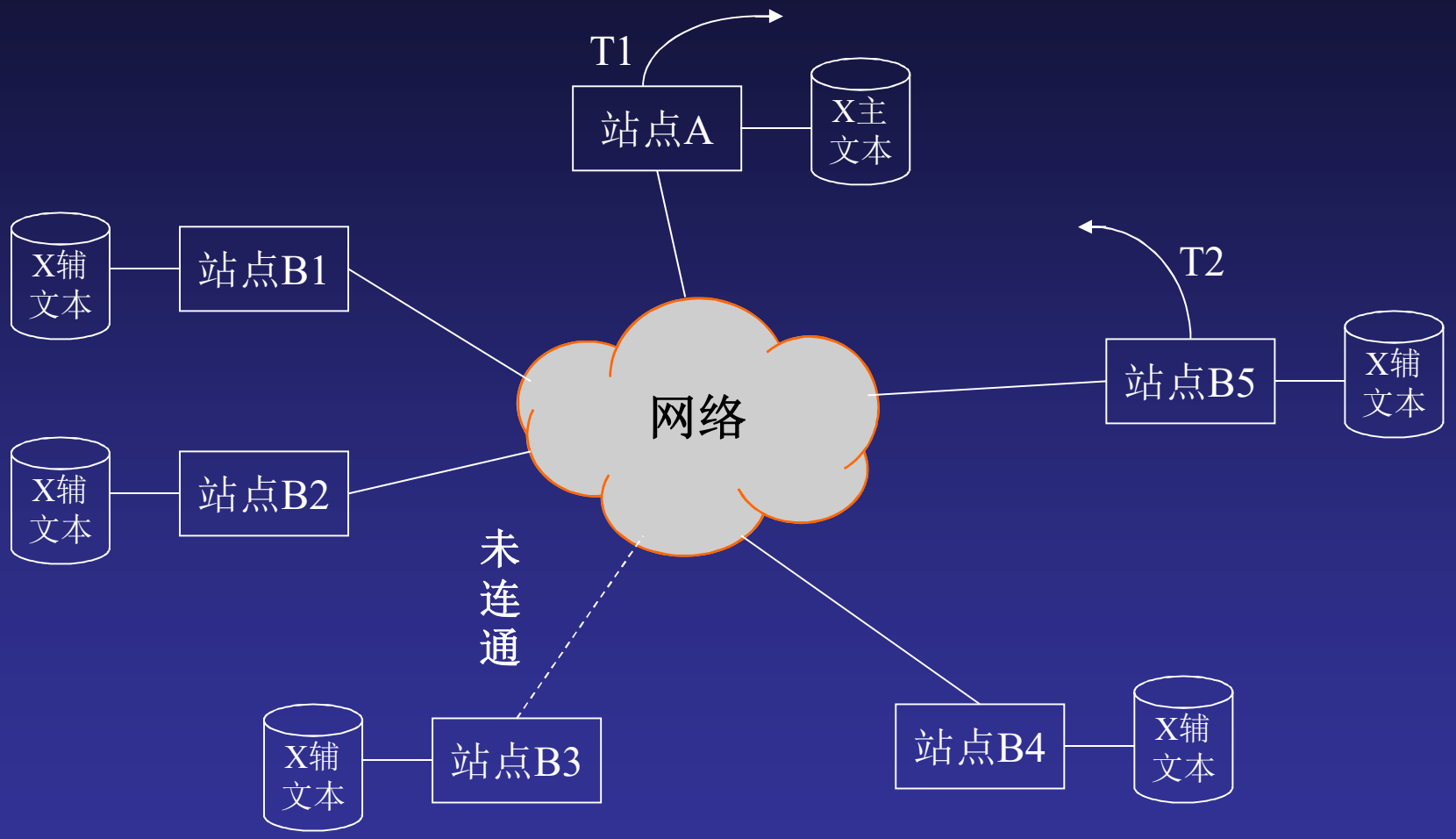


4.2 主文本更新

- 主文本更新
  - 思想：指定主副本，修改只对主副本进行。更新传播到辅助副本时，也按在主副本上执行的更新顺序执行
  - 问题
    - 修改传播必须在短时间内完成，否则将获得“过时”数据
    - 主副本不可用，引得其他副本也不可用

4 分布式数据库中的数据更新

4.2 主文本更新



4.2 主文本更新

- 移动主文本法
  - 如果主站点此时尚未连通，则另选一个辅站点中的辅文本为该数据新的主文本进行更新
  - 待原主文本站点连通后，系统将自动把它改为辅文本，并按记录要求执行更新
  - 如果更新在主文本上进行，但主文本站点与网络未接通，则此次更新操作失败，事务被撤销，因为无法传播更新
- 移动文本法的问题
  - 网络分割成很多部分时，更新处理会不一致
  - 网络分割成**W1,W2**，**W1**中**X**更新为**R, W2**中**X**更新为**S,**网络连通时，使用**R**还是**S**来恢复**X**呢？

4.3 快照及其更新

- **快照(Snapshot)**
  - 与视图相似, 是导出的关系
  - 快照的数据是实际存放在数据库中的，视图不是
  - 用于某些需要“冻结”数据的应用

4.3 快照方法

- 例子

Define Snapshot HP-Book as
SELECT * FROM Book WHERE Price>$100
REFRESH Every day

- 特点
  - 快照不考虑数据的辅助副本, 只关心主副本和这个主副本上定义的多个快照
  - 快照与视图一样可以定义为一个或多个主副本的部分或全部
  - 查询操作可使用快照，也可使用主副本，对更新操作还是在主副本上进行
  - 主文本更新时刷新快照

5 总　结

- 分布式事务概述
- 分布式事务的执行与恢复
  - 事务管理抽象模型
  - 事务恢复
- 2PC协议
- 分布式数据库中的数据更新