

第十二章 故障恢复

章成源

湖南大学-信息科学与工程学院-计算机科学系

办公室：院楼403

Email: cyzhangcse@hnu.edu.cn

第12章 故障恢复

- 12.1 故障恢复概述
- 12.2 恢复的基本实现技术
- 12.3 恢复的基本原理

为什么需要故障恢复

- 数据库管理系统在正常运行的过程中可能会出现故障，这些故障有些是可以提前预知的，有些是不能提前预知的。
- 数据库管理系统的故障恢复子系统可以确保，无论出现哪种故障，都能将数据库中的数据恢复到故障发生前的正确的状态。
- 没有故障恢复，数据库中的数据会 **出错或丢失**

故障恢复的内涵

- 抽象成“三问”

- 问题一

□故障发生前，数据库的一致性状态是什么

- 问题二

□故障发生后，数据库有哪些一致性状态被破坏了

- 问题三

□故障恢复时，如何将破坏的数据库一致性状态恢复到故障发生前的一致性状态

第12章 故障恢复

- 12.1 故障恢复概述
- 12.2 恢复的基本实现技术
- 12.3 恢复的基本原理

12.1 故障恢复概述

- 12.1.1 故障的分类
- 12.1.2 事务读写的访问模式
- 12.1.3 故障下数据一致性的破坏

故障恢复概述

- 一、数据库故障产生的原因及影响

- 故障产生的原因

- 计算机硬件故障
 - 软件的错误
 - 操作员的失误
 - 恶意的破坏

- 故障的影响

- 运行事务非正常中断，影响数据库中数据的正确性（破坏了事务的**原子性 Atomicity**）
 - 破坏数据库，全部或部分丢失数据（破坏了事务的**持续性 Durability** 和 **原子性**）

故障恢复概述

- 数据库的恢复

- 数据库管理系统必须具有把数据库从**错误状态**恢复到**某一已知的正确状态**(亦称为一致状态或完整状态)的功能, 这就是数据库的恢复管理系统对故障的对策

- 恢复子系统是数据库管理系统的一个重要组成部分
- 恢复技术是衡量系统优劣的重要指标

故障的分类

- 数据库管理系统可能会发生各种各样的故障，每种故障需要使用不同的方法来处理。数据库系统主要会遇到下面四种故障
 - 事务故障
 - 系统故障
 - 介质故障
 - 用户错误（有时不计为故障分类）

事务故障

- 事务故障是指事务的运行没有到达预期的终点（COMMIT或者显式的ROLLBACK）就被终止，有两种错误可能造成事务执行失败
 - ❑ **逻辑错误**：事务由于某些内部情况而无法继续其正常执行，例如非法输入、溢出、完整性检查未通过或超出资源限制等。
 - ❑ **系统错误**：系统进入一种不良状态（例如死锁）导致无法继续其正常执行。

事务故障举例

- 事务T1：从账户x转出50元到账户y中
- 从账户x中转出50元之后，之后因为死锁等原因，造成事务T1回滚。
- x转出50元，而y尚未转入50元，从而造成数据的不一致



事务故障的恢复

- 事务故障意味着
 - ❑ 事务没有达到预期的终点(COMMIT或者显式的ROLLBACK)
 - ❑ 数据库可能处于不正确状态。
- 事务故障的恢复：**事务撤消 (UNDO)**
 - ❑ 强行回滚 (ROLLBACK) 该事务
 - ❑ 撤销该事务已经作出的任何对数据库的修改，使得该事务象根本没有启动一样

系统故障

- 又称软故障，是指造成系统停止运转的任何事件，使得系统要重新启动。
 - 特定类型的硬件错误（如CPU故障）
 - 操作系统故障
 - 数据库管理系统代码错误
 - 系统断电

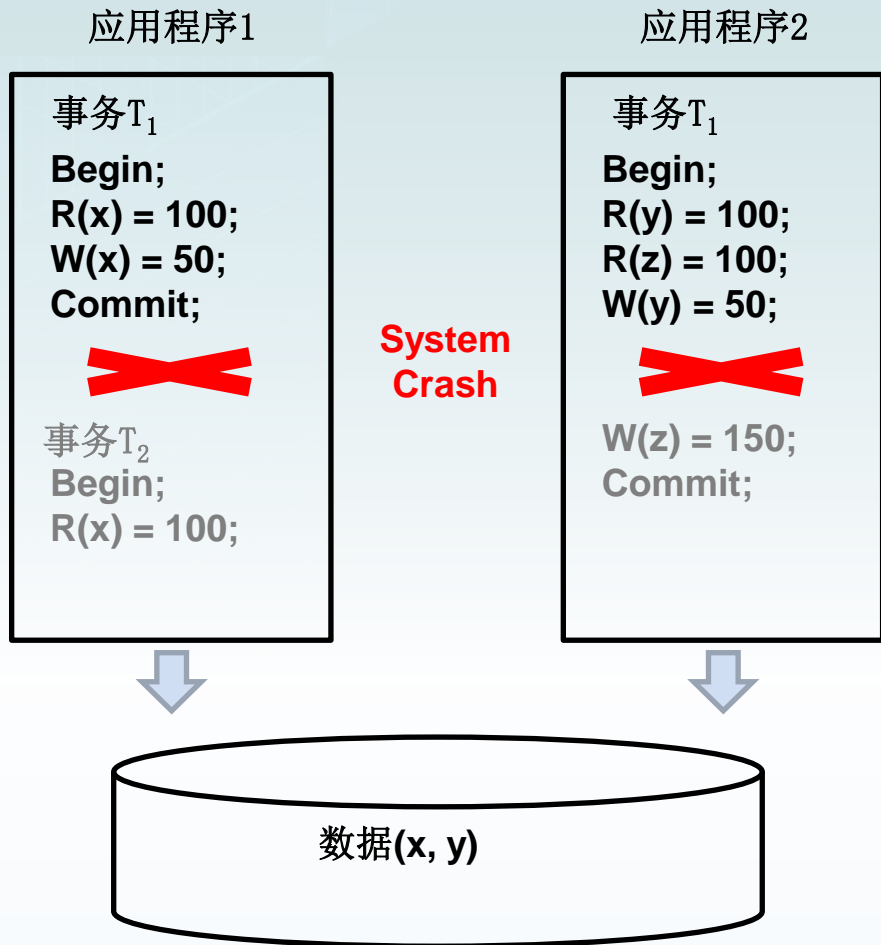
系统故障的影响

- 整个系统的正常运行突然被破坏
- 所有正在运行的事务都非正常终止
- 内存中数据库缓冲区的信息全部丢失
- 不破坏数据库

所有活跃事务都只运行了一部分，没有全部完成。

部分已完成事务更新后的数据还在缓冲区中，没有来得及刷到硬盘上，这些更新就丢失了。

系统故障



- 持续性:
 - 一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。
- 场景:
 - 事务 T_1 从 x 账号扣除50元，事务提交后系统发生故障， x 账号的修改还未写入磁盘；
 - 系统重启后事务 T_2 错误地查询到了账户有100元。
- 系统故障会破坏数据库的持久性和原子性

系统故障的恢复

- 发生系统故障时，一些尚未完成的事务的结果可能已送入物理数据库，造成数据库可能处于不正确状态。
- 恢复策略：系统重新启动时，恢复程序让所有非正常终止的事务回滚，强行撤销（**UNDO**）所有未完成事务

系统故障的恢复（续）

- 发生系统故障时，有些已完成的事务可能有一部分甚至全部留在缓冲区，尚未写回到磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失
- 恢复策略：系统重新启动时，恢复程序需要重做（**REDO**）所有已提交的事务

系统故障的恢复需要做两件事情：

1. 撤销所有未完成的事务
2. 重做所有已提交的事务

介质故障

- 也称为硬故障，指外存故障，与系统故障（软故障）相对
 - 磁盘损坏
 - 磁头碰撞
 - 瞬时强磁场干扰
- 介质故障破坏数据库或部分数据库，并影响正在存取这部分数据的所有事务
- 介质故障比前两类故障的可能性小得多，但破坏性大得多

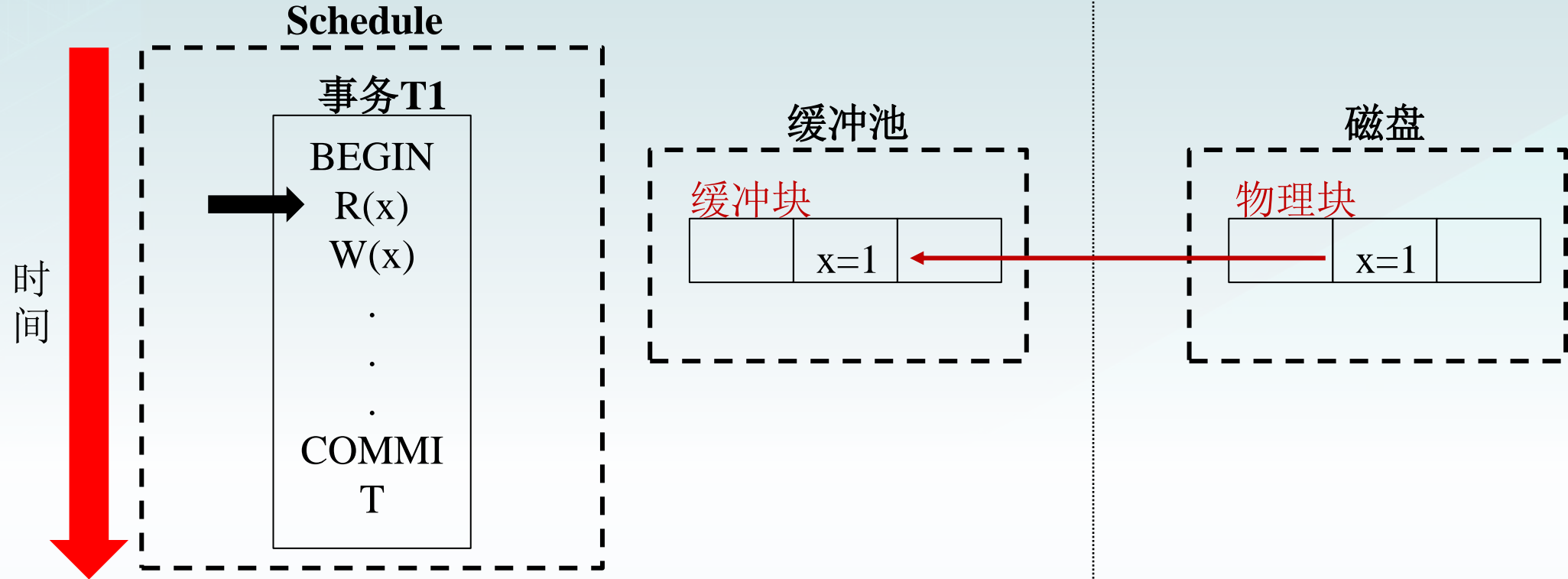
用户错误

- 用户在使用数据库的过程中可能会出现的一些误操作
 - 误删了表中的数据行
 - 误删除系统中的表
 - 用户提交了错误数据
- 可以提供一些手段帮助用户找回误操作带来的数据损失
- Oracle 的闪回技术可以用来帮助恢复用户错误

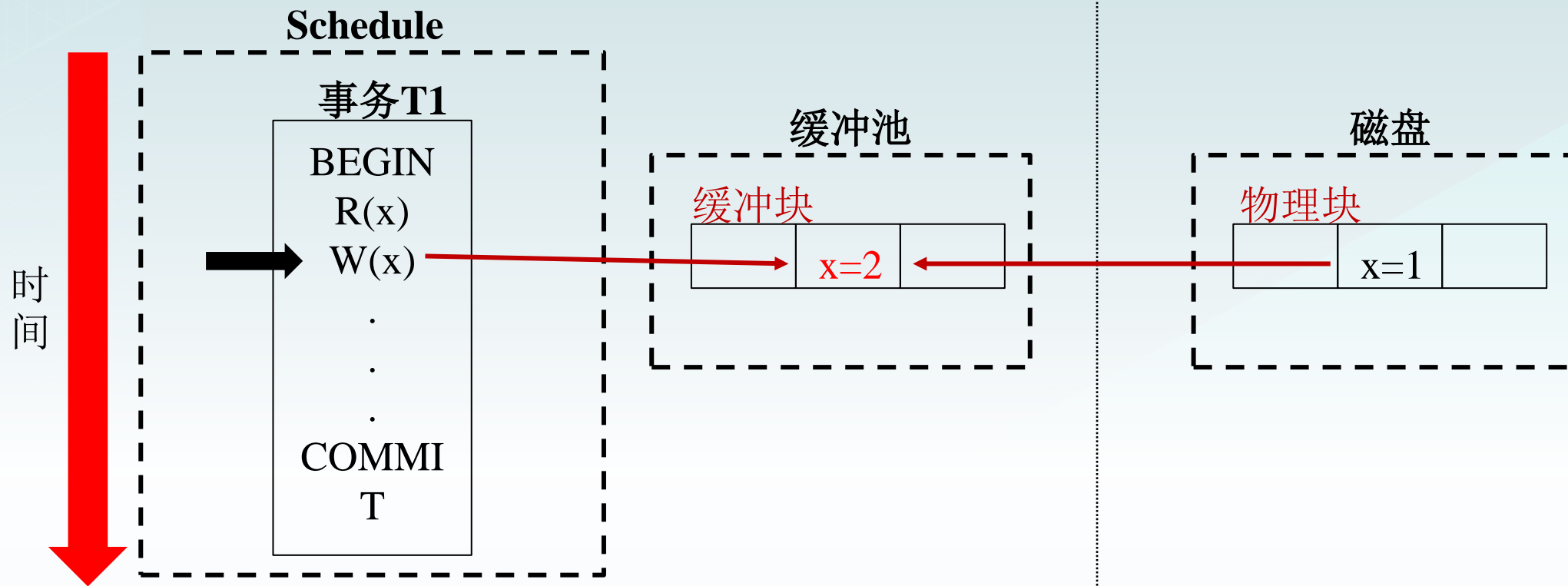
12.1 故障恢复概述

- 12.1.1 故障的分类
- 12.1.2 事务读写的访问模式
- 12.1.3 故障下数据一致性的破坏

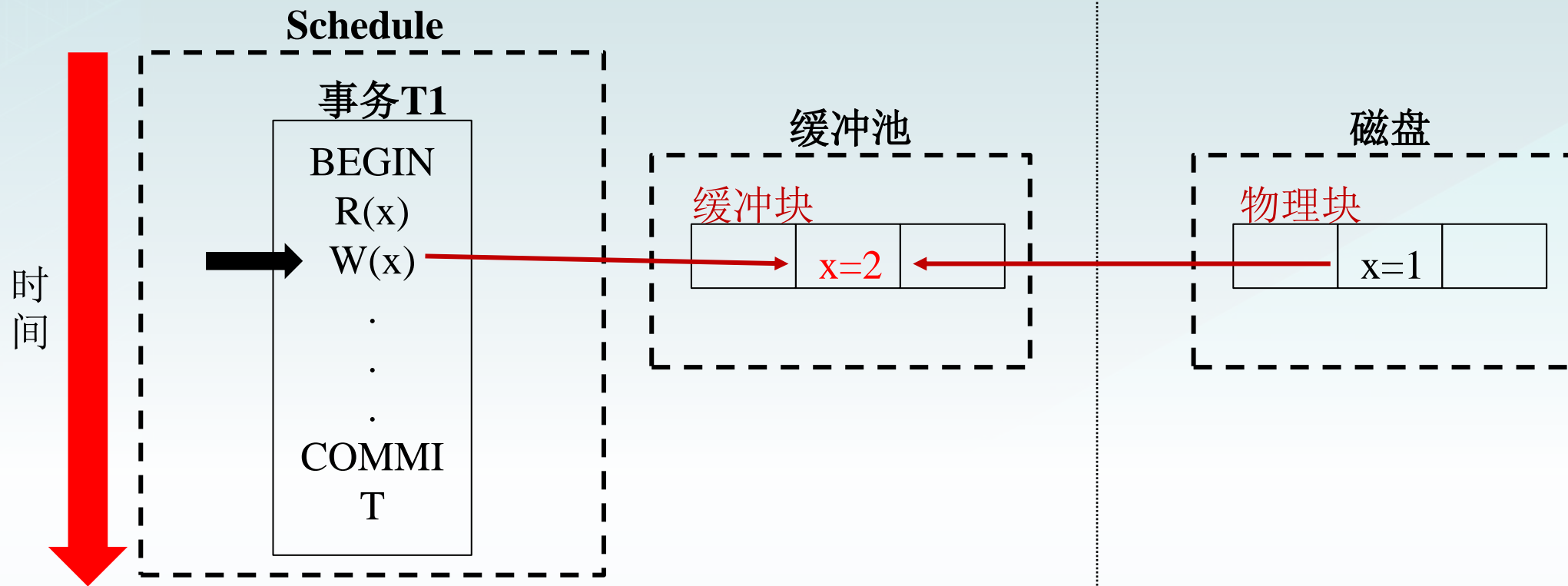
事务的访问模式



事务的访问模式



事务的访问模式



事务的访问模式

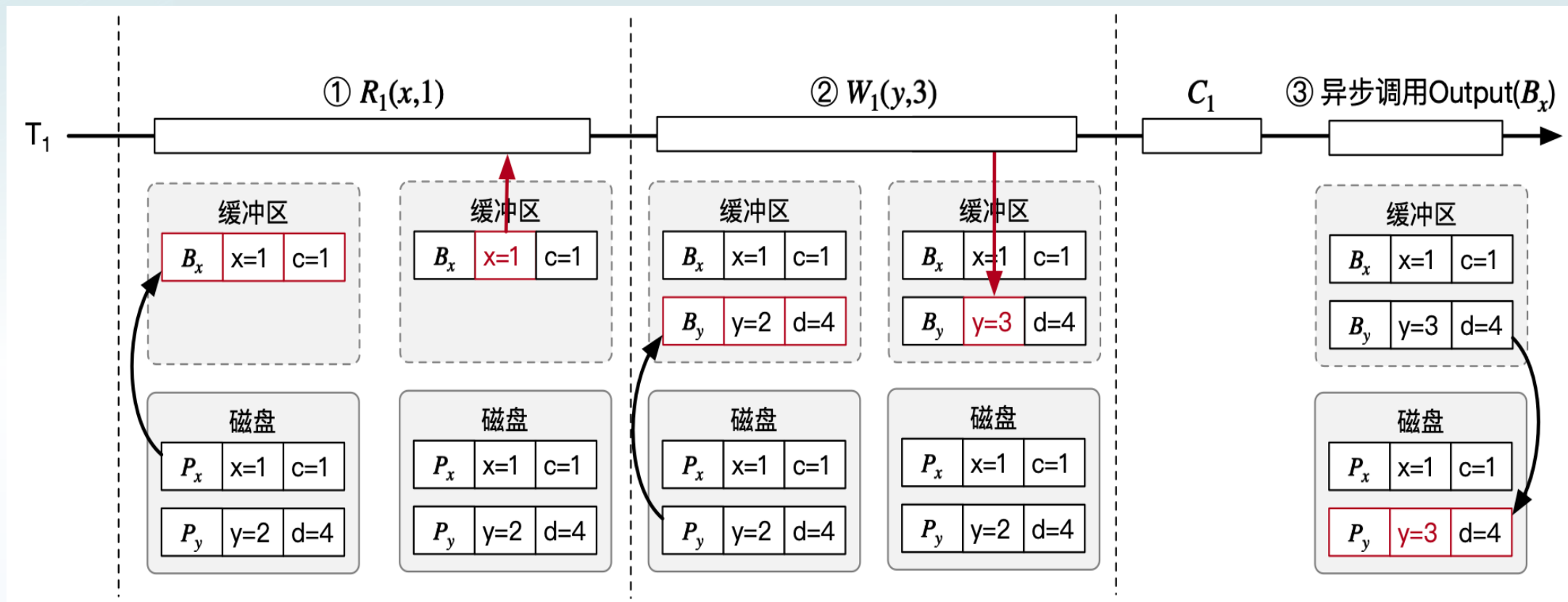
- 由于缓冲区的大小有限，无法将所有的物理块缓存到缓冲区当中，因此块会在磁盘和内存之间进行频繁换入换出
 - **Input(A)**。当事务读取存储在磁盘中的物理块A时，而这个物理块并没有缓存在缓冲区中，这个事务首先发出Input(A)请求将磁盘中的物理块 A 加载到缓冲区中
 - **Output(A)**。当事务对缓冲区中的缓冲块A进行了修改或者由于缓冲区空间不够时，数据库系统会发起 Output(A)请求将缓冲块 A 写到磁盘中，并替换原有的物理块。

事务的访问模式

- 给定数据项 x ，令 P_x 为对应的磁盘块； B_x 为对应的缓冲块
- 事务读写的执行过程
 - **Read(x)**: 如果 B_x 不存在，加载 P_x 到 B_x ；从 B_x 中读取 x
 - **Write(x)**: 如果 B_x 不存在，加载 P_x 到 B_x ；将 x 写入到 B_x

事务的访问模式

- 事务的访问模式示例

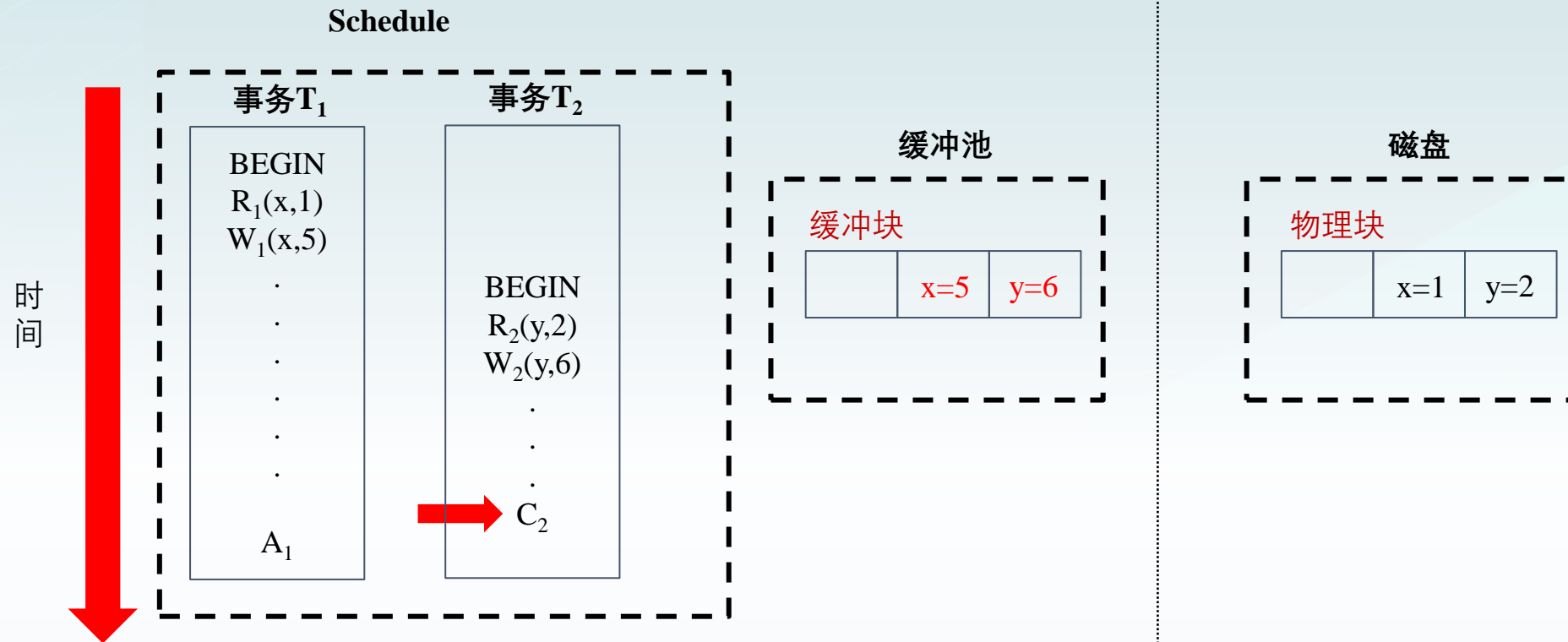


事务的缓冲区策略

- 从未提交事务的角度来看
 - ❑ STEAL: **允许**未提交事务的写落盘
 - ❑ NO-STEAL: **不允许**未提交事务的写落盘
- 从已提交事务的角度来看
 - ❑ FORCE: 事务一旦提交, **强制同步**该事务的写落盘
 - ❑ NO-FORCE: 事务一旦提交, **不强制同步**该事务的写落盘

事务的缓冲区策略

NO-STEAL + FORCE

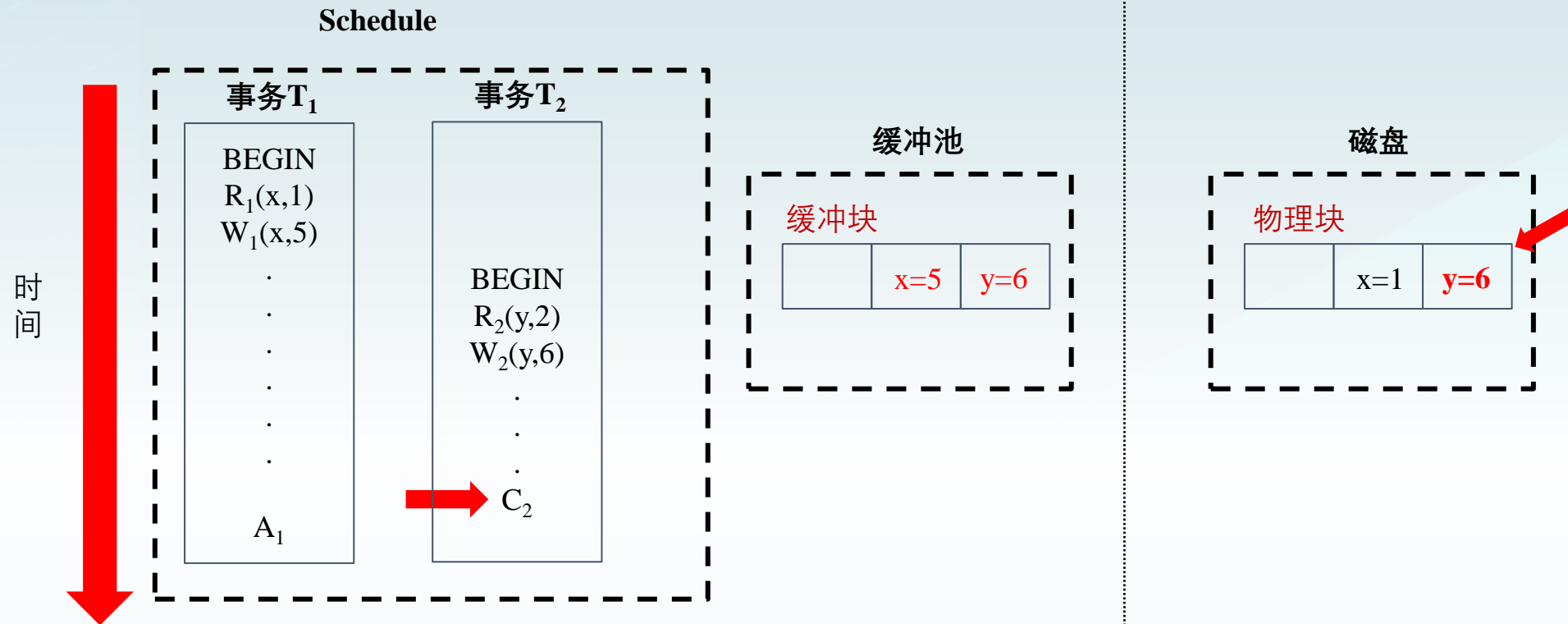


FORCE: 事务一旦提交, 强制同步该事务的写落盘

NO-STEAL: 不允许未提交事务的写落盘

事务的缓冲区策略

NO-STEAL + FORCE

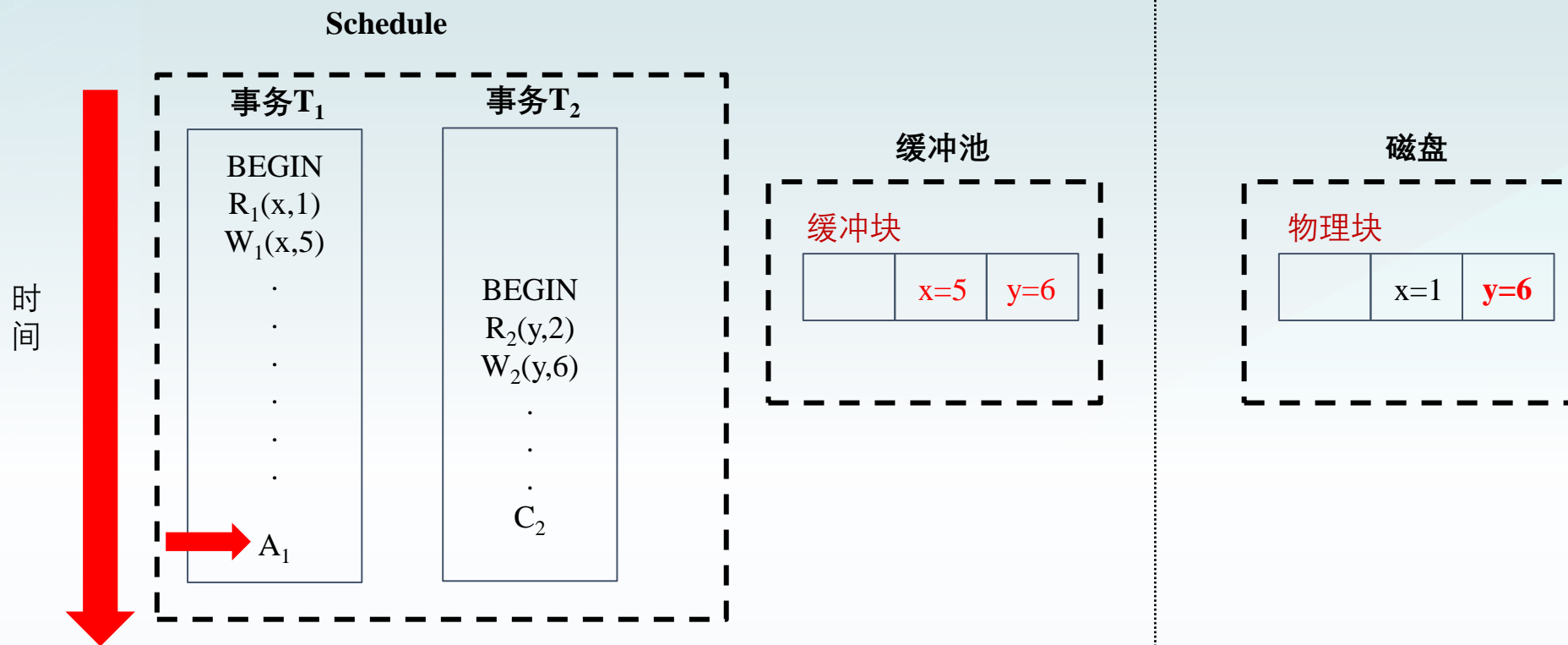


FORCE: 事务一旦提交, 强制同步该事务的写落盘

NO-STEAL: 不允许未提交事务的写落盘

事务的缓冲区策略

NO-STEAL + FORCE

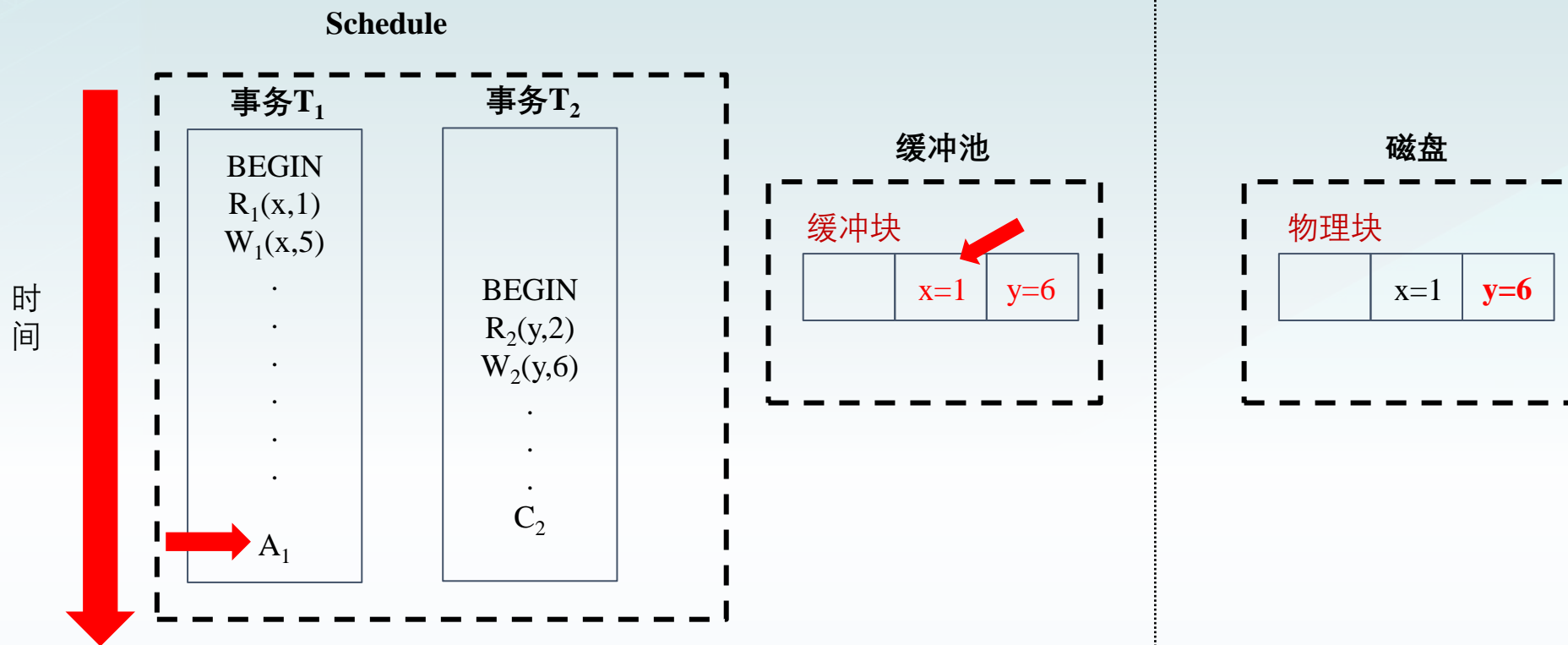


FORCE: 事务一旦提交, 强制同步该事务的写落盘

NO-STEAL: 不允许未提交事务的写落盘

事务的缓冲区策略

NO-STEAL + FORCE



FORCE: 事务一旦提交, 强制同步该事务的写落盘

NO-STEAL: 不允许未提交事务的写落盘

12.1 故障恢复概述

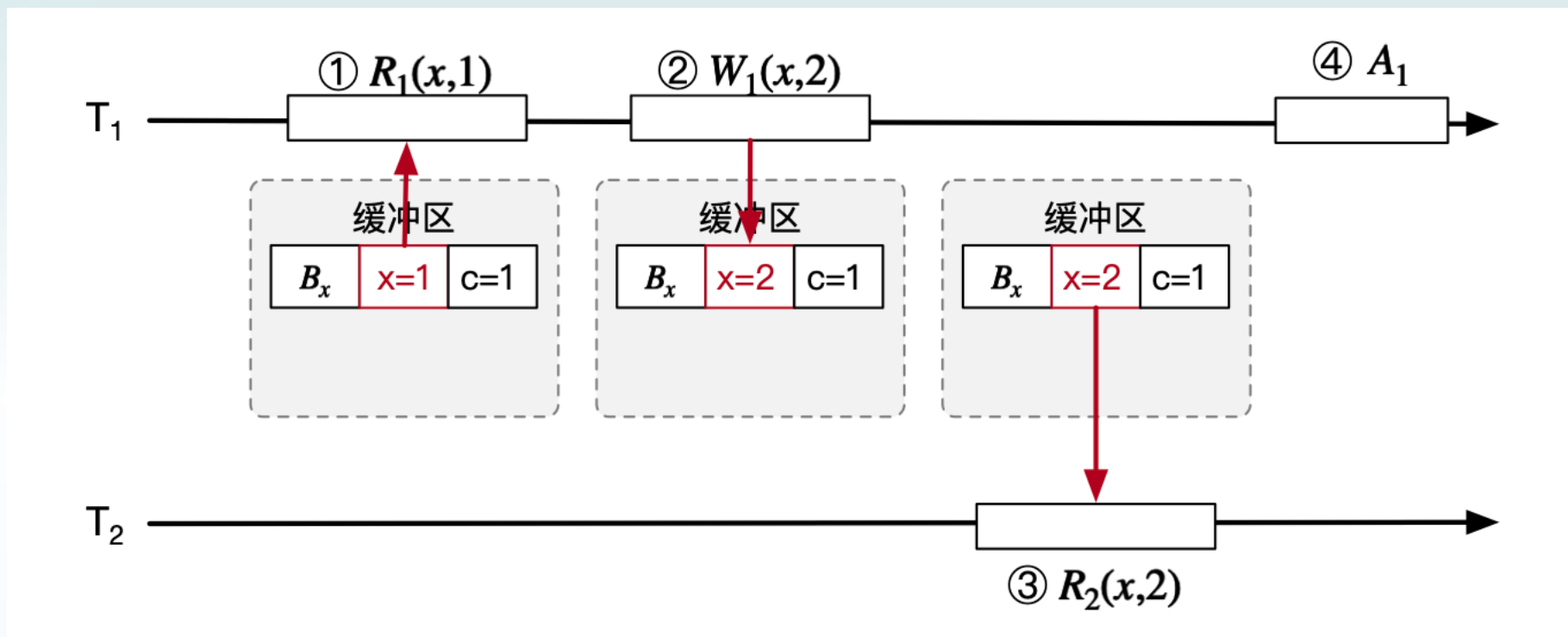
- 12.1.1 故障的分类
- 12.1.2 事务读写的访问模式
- 12.1.3 故障下数据一致性的破坏

事务故障造成的破坏

- 事务故障发生前，数据库的一致性状态是什么？
 - 指的是这个事务的所有操作都未发生时，数据库的一致性状态。
- 事务故障发生后，数据库有哪些一致性状态被破坏？
 - 事务将**修改的数据项写入**到缓冲区中对应的**缓冲块**，且**该缓冲块**被系统**异步写入**到**磁盘的物理块**中，导致数据库中存储了脏数据。
 - 事务将**修改的数据项写入**到缓冲区中对应的**缓冲块**，导致**并发事务读取**了该事务写入的**脏数据**。这一问题是通过**并发控制**来保证。

事务故障造成数据页的破坏

- 事务故障发生前，数据库的一致性状态是什么？
- 事务故障发生后，数据库有哪些一致性状态被破坏？



事务故障造成数据页的破坏

- 故障恢复时，如何将破坏的数据库一致性状态恢复到故障发生前的一致性状态？
 - ❑ 发生事务故障后，故障事务修改的页面需要被撤销（UNDO），将未完成事务的修改进行撤销，以恢复到故障发生前的一致性状态；
 - ❑ 在事务撤销操作完成之前，需要设计合理的并发控制算法，保证没有其他事务能够读取到故障事务写入的脏数据。

系统故障造成数据页的破坏

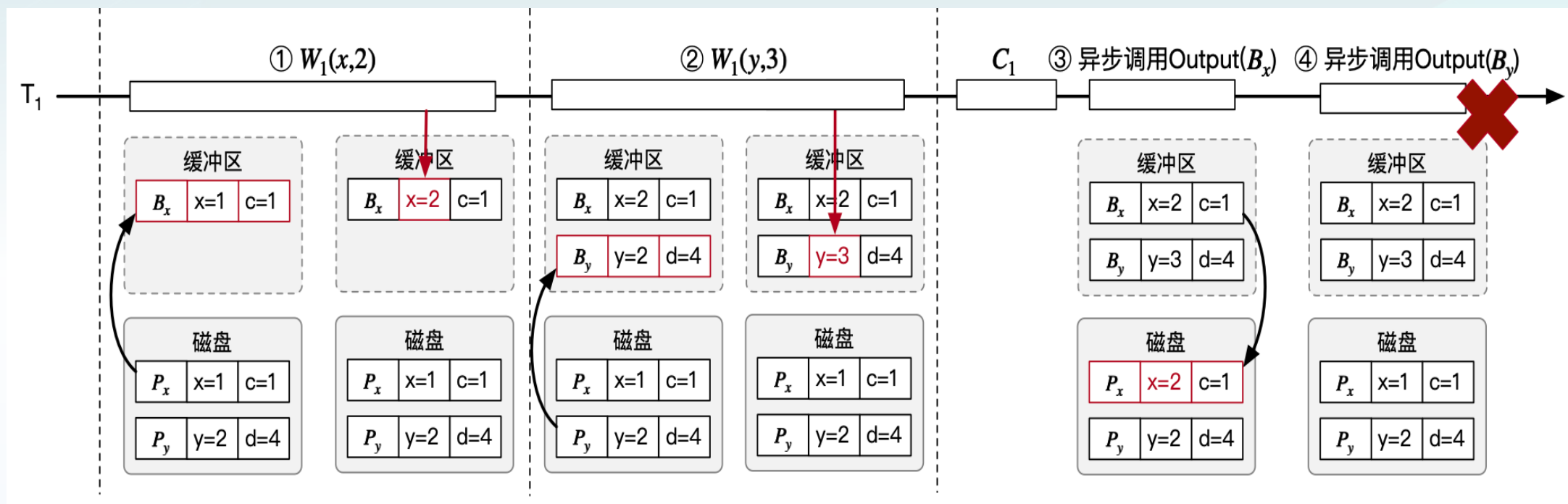
- 系统故障发生前，数据库的一致性状态是什么？
 - 系统故障发生前数据库的一致性状态，指的是**系统中未提交事务的所有操作都未发生，且已提交事务的写操作都已经应用在缓冲区和磁盘上时，数据库的一致性状态**

系统故障造成数据页的破坏

- 故障发生后，数据库有哪些一致性状态被破坏？
 - 系统故障发生后，系统需要重启，导致缓存区中的所有数据丢失，所有运行事务都被中止
 - **未提交事务的写**：故障发生时，未提交事务的执行被中断，而这些事务中可能有一些事务的写已经写入到磁盘。
 - **已提交事务的写**：已提交事务写入的数据可能部分或全部留在缓冲区中，系统故障导致缓冲区中的数据丢失，而这些数据尚未来得及写到磁盘

系统故障造成数据页的破坏

- 系统故障发生前，数据库的一致性状态是什么？
- 系统故障发生后，数据库有哪些一致性状态被破坏？



系统故障造成数据页的破坏

- 系统故障发生前，数据库的一致性状态是什么？
- 系统故障发生后，数据库有哪些一致性状态被破坏？
- 故障恢复时，如何将破坏的数据库一致性状态恢复到故障发生前的一致性状态
 - ❑ 对于未提交事务持久化到磁盘中的修改，系统需要撤销这些事务的修改（UNDO）；
 - ❑ 对于已提交事务尚未持久化到磁盘中的修改，系统需要重做这些事务的修改（REDO）

第12章 故障恢复

- 12.1 故障恢复概述
- 12.2 恢复的基本实现技术
- 12.3 恢复的基本原理

恢复的实现技术

恢复机制涉及的关键问题

1. 如何建立冗余数据

- 数据转储（备份backup）
- 登记日志文件（logging）

2. 如何利用这些冗余数据实施数据库恢复

12.2 恢复的基本实现技术

- **12.2.1 日志文件**
- **12.2.2 WAL日志**
- **12.2.3 数据转储**
(数据库系统概论)

日志文件

- 日志文件：用来记录事务对数据库的更新操作的文件

- 日志文件的格式

- 以记录为单位的日志文件
- 以数据块为单位的日志文件

→ Transaction Id	事务ID
→ Object Id	对象ID
→ Before Value (UNDO)	旧值
→ After Value (REDO)	新值

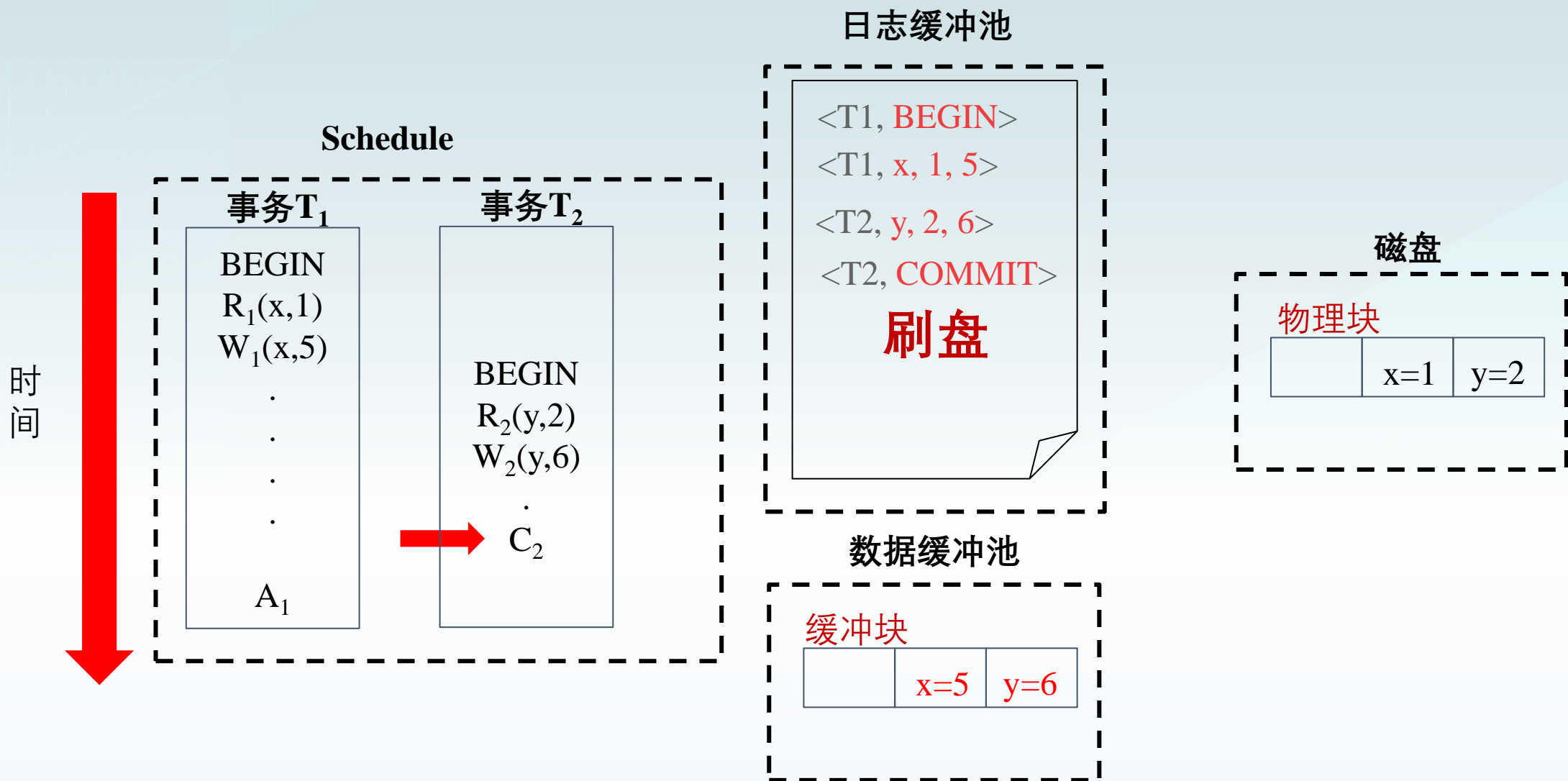
- 日志文件内容

- 各个事务的开始标记(**BEGIN TRANSACTION**)
- 各个事务的结束标记(**COMMIT**或**ROLLBACK**)
- 各个事务的所有更新操作：
- 事务标识、操作类型(插入、删除或修改)、操作对象（记录项或数据块）、更新前数据的旧值、更新后数据的新值等

日志文件分类

- 物理日志：
 - 记录对数据库中特定元组所做的更改。
 - 例如：物理日志记录了属性的原始值和更改后的值。
- 逻辑日志：
 - 记录由事务执行的更高层面的操作。
 - 例如：由事务调用的 **UPDATE**、**DELETE** 和 **INSERT** 查询。

日志举例



日志文件的分类-Undo日志

- 格式: $\langle T, X, v \rangle$

□ 事务T改变了数据库元素的X, X原来的值是v

→ Transaction Id

事务ID

→ Object Id

对象ID

→ Before Value (UNDO)

旧值

- 规则

□ $\langle T, X, v \rangle$ 必须先写到磁盘, 然后把X的新值写到磁盘上

□ 如果事务提交, 则其COMMIT日志记录必须在事务提交前把更新的数据库元素写到日志记录所在磁盘后再写到数据库所在磁盘。

□ 与事务相关的内容必须按如下顺序写磁盘

➤ 1.日志记录 (更新数据元素的日志记录)

➤ 2.更新的数据库元素

➤ 3. $\langle \text{Commit } T \rangle$

日志文件的分类-Redo日志

- 格式: $\langle T, X, w \rangle$

→ Transaction Id	事务ID
→ Object Id	对象ID
→ After Value (REDO)	新值

□ 事务T为数据库元素X写入新值w

- 规则

□ 日志记录 $\langle T, X, w \rangle$ 及 $\langle \text{COMMIT } T \rangle$

□ 数据元素X更新后的值w

日志文件的分类-Undo/Redo日志

- 格式: $\langle T, X, v, w \rangle$

→ Transaction Id	事务ID
→ Object Id	对象ID
→ Before Value (UNDO)	旧值
→ After Value (REDO)	新值

- $\langle T, X, v, w \rangle$ 事务T改变了数据库元素X的值，老值是v，新值是w
- 规则
 - 先写日志记录 $\langle T, X, v, w \rangle$ ，后写数据元素X更新后的值w
 - 事务提交时，必须将日志记录 $\langle \text{COMMIT } T \rangle$ 刷到磁盘

12.2 恢复的基本实现技术

- 12.2.1 日志文件
 - 12.2.2 WAL日志
 - 12.2.3 数据转储
- (数据库系统概论)

登记日志文件的原则

- 为了保证事务的原子性和持久性，数据库系统必须遵循预写式日志（Write-ahead logging, WAL）机制，该机制包括下面三项内容：
 - 数据页的每个修改都应该生成日志记录，并且在数据页面刷到磁盘之前，其对应的日志记录必须先刷到持久化的日志设备上；
 - 在事务的提交日志刷到磁盘之前，所有和该事务相关的日志都应该被刷到磁盘；（数据库的日志记录的顺序严格按并发事务执行的时间顺序）
 - 事务在进入提交阶段前必须先将提交日志刷到磁盘上；

登记日志文件的原则

- 为什么要先写日志文件

- 如果先写了数据库修改，而在日志文件中没有登记下这个修改，则以后就无法恢复这个修改了
- 如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的UNDO操作，并不会影响数据库的正确性

登记日志文件的原则

- 根据预写式日志的规定，可以得到日志记录被刷到磁盘上的日志文件中的三种情况：
 - 缓冲区的日志大小达到一定的阈值，将缓冲区刷盘。
 - 数据缓冲区中的数据页面刷盘时，对应的日志记录也要刷盘。
 - 事务提交时，该事务以及和该事务相关的日志都要刷盘。

事务的缓冲区策略

- 几乎所有的商用数据库采用的是**NO-FORCE&STEAL**策略

	NO-STEAL	STEAL
NO-FORCE	Redo日志	Redo/Undo日志
FORCE		Undo日志

事务的缓冲区策略

- 几乎所有的商用数据库采用的是**NO-FORCE&STEAL**策略

正常执行时的性能

	NO-STEAL	STEAL
NO-FORCE	Redo日志	Redo/Undo 日志 最快
FORCE	最慢	Undo日志

故障恢复时的性能

	NO-STEAL	STEAL
NO-FORCE	Redo日志	Redo/Undo 日志 最慢
FORCE	最快	Undo日志

事务的缓冲区策略

- 几乎所有的商用数据库采用的是**NO-FORCE&STEAL**策略

正常执行时的性能

	NO-STEAL	STEAL
NO-FORCE	Redo日志	Redo/Undo 日志 最快
FORCE	最慢	Undo日志

故障恢复时的性能

	NO-STEAL	STEAL	
NO-FORCE	Redo日志	Redo/Undo 日志 最慢	Undo Redo
FORCE	最快	Undo日志	

不用Undo
不用Redo

12.2 恢复的基本实现技术

- 12.2.1 日志文件
- 12.2.2 WAL日志
- 12.2.3 数据转储
(数据库系统概论)

什么是数据转储?

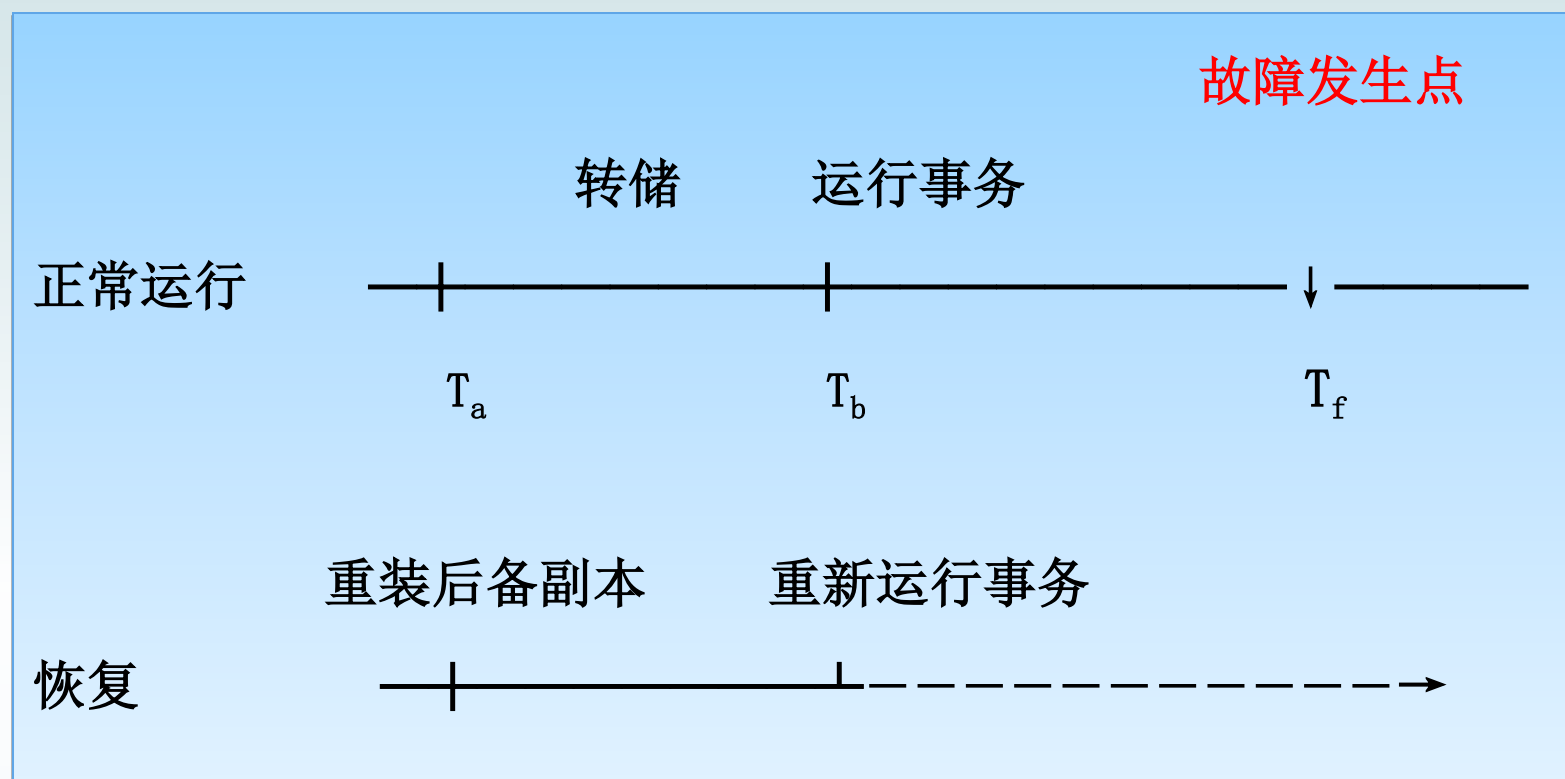
- 转储是指数据库管理员定期地将整个数据库复制到磁带、磁盘或其他存储介质上保存起来的过程
- 备用的数据文本称为**后备副本(backup)**或**后援副本**
- 备份的定义
 - 是指DBA定期地将整个数据库复制到磁带、磁盘或其他存储介质上保存起来的过程。
- 备份的作用
 - 数据库遭到破坏后可以将后备副本重新装入，从而恢复到备份时的状态。

数据转储

- 数据库遭到破坏后可以将后备副本重新装入
- 重装后备副本只能将数据库恢复到转储时的状态
- 要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务

数据转储

[例]



转储和恢复

2. 转储方法

- (1) 静态转储与动态转储
- (2) 海量转储与增量转储

(1) 静态转储与动态转储

- 静态转储
 - 在系统中无运行事务时进行的转储操作
 - 转储开始时数据库处于一致性状态
 - 转储期间不允许对数据库的任何存取、修改活动
 - 得到的一定是一个数据一致性的副本
 - 优点：实现简单
 - 缺点：降低了数据库的可用性
 - 转储必须等待正运行的用户事务结束
 - 新的事务必须等转储结束

静态转储与动态转储

- 动态转储
 - 转储操作与用户事务并发进行
 - 转储期间允许对数据库进行存取或修改
 - 优点
 - 不用等待正在运行的用户事务结束
 - 不会影响新事务的运行
 - 动态转储的缺点
 - 不能保证副本中的数据正确有效
 - 例在转储期间的某时刻 T_c ，系统把数据 $A=110$ 转储到磁带上，而在下一时刻 T_d ，某一事务将 A 改为 200。
后备副本上的 A 过时了

例

转储开始

数据库

A	B	C	D
1	2	3	4
1	2	3	4

例

转储开始
备份数据库

A	B	C	D
1	2	3	4
1	2	3	4

T1	T2	备份
		副本A=1

转储过程

例

转储开

始 备份

数据库

A	B	C	D
1	2	3	4
1			
1	7	3	4

T1	T2	备份
B:=7		副本A=1

转储过程

例

转储开始
备份数据库

A	B	C	D
1	2	3	4
1	7		
1	7	3	4

T1	T2	备份
B:=7		副本A=1 副本B=7

转储过程

例

转储开始
备份数据库

A	B	C	D
1	2	3	4
1	7		
1	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=7

转储过程

例

转储开
始 备份
数据库

A	B	C	D
1	2	3	4
1	7	6	
1	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=7
		副本C=6

转储过程

例

转储开
始 备份
数据库

A	B	C	D
1	2	3	4
1	7	6	
5	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=7
A:=5		副本C=6

转储过程

例

转储开始

备份

数据库

	A	B	C	D
	1	2	3	4
	1	7	6	4
	5	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=2
A:=5		副本C=6
		副本D=4

转储过程

静态转储与动态转储

- 利用动态转储得到的副本进行故障恢复
 - 需要把动态转储期间各事务对数据库的修改活动登记下来，建立日志文件
 - 后备副本加上日志文件就能把数据库恢复到某一时刻的正确状态

例

转储开始
备份
数据库

A	B	C	D
1	2	3	4
1	7	6	4
5	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=7
A:=5		副本C=6
ROLL BACK	commit	副本D=4
		转储结束

转储过程

例

转储开

始 备份

A	B	C	D
1	2	3	4
1	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=2
A:=5		副本C=6
ROLL BACK	Commit	副本D=4
		转储结束

转储过程

例

转储开始
备份
数据库

A	B	C	D
1	2	3	4
1	7	6	4
1	7	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=7
A:=5		副本C=6
ROLL BACK		副本D=4
	commit	转储结束

转储过程

例

转储开始

备份

数据库

T1 UNDO

T2 REDO

A	B	C	D
1	2	3	4
1	7	6	4
1	2	6	4

T1	T2	备份
B:=7		副本A=1
	C:=6	副本B=7
A:=5		副本C=6
ROLL BACK		副本D=4
	Commit	转储结束

转储过程

(2) 海量转储与增量转储

- 海量转储: 每次转储全部数据库
- 增量转储: 只转储上次转储后更新过的数据
- 海量转储与增量转储比较
 - 从恢复角度看, 使用海量转储得到的后备副本进行恢复往往更方便
 - 如果数据库很大, 事务处理又十分频繁, 则增量转储方式更实用更有效

(3) 转储方法小结

- 转储方法分类

转储方式	转储状态	
	动态转储	静态转储
海量转储	动态海量转储	静态海量转储
增量转储	动态增量转储	静态增量转储

- 在数据转储效率、数据库运行效率、故障恢复效率三个方法各有利弊
- DBA通常会根据数据库的使用情况，确定一个合适的转储周期，并配合使用这四类方法

第12章 故障恢复

- 12.1 故障恢复概述
- 12.2 恢复的基本实现技术
- 12.3 恢复的基本原理

事务故障的恢复

- 事务故障的恢复策略为

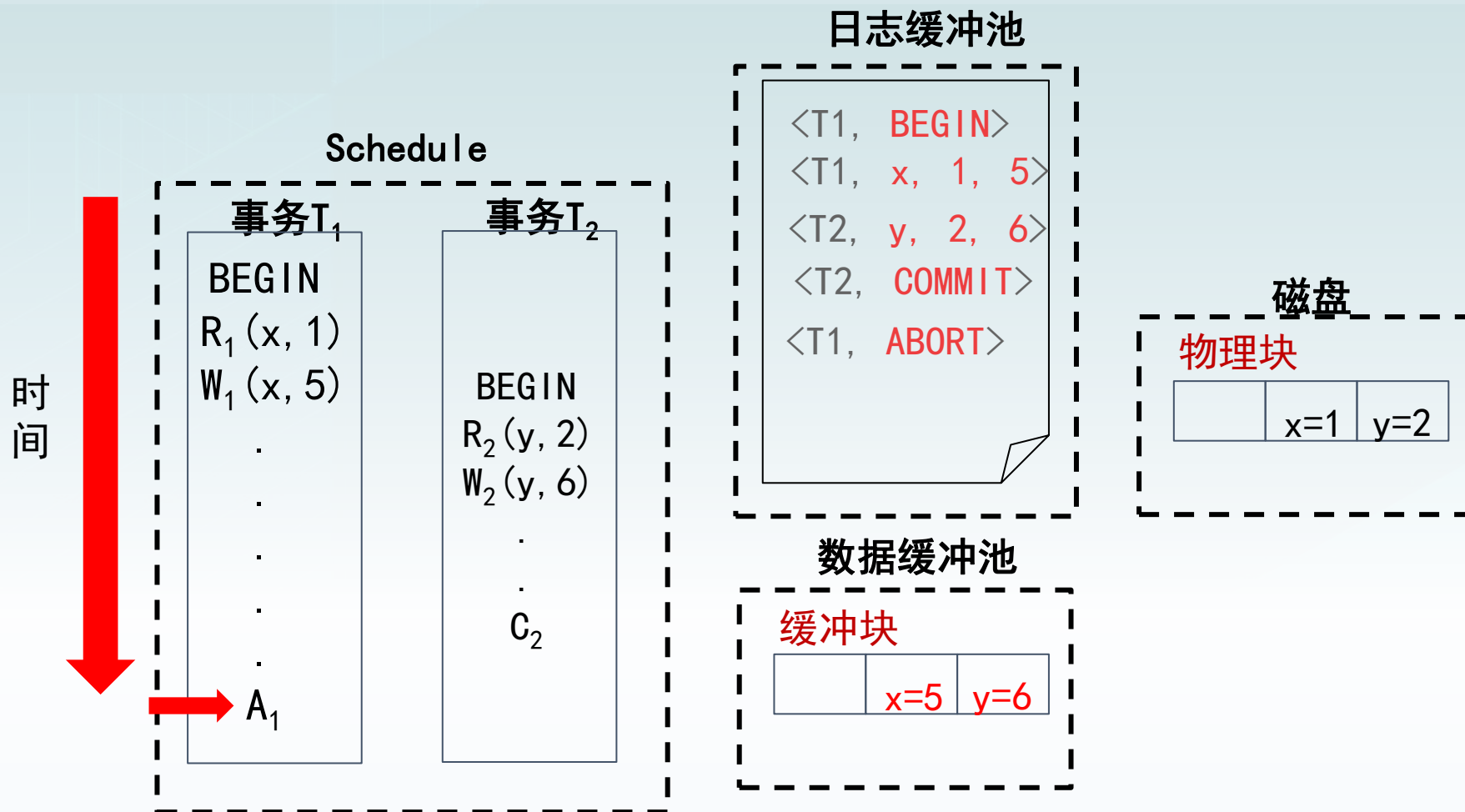
- 执行Undo操作

- 具体为

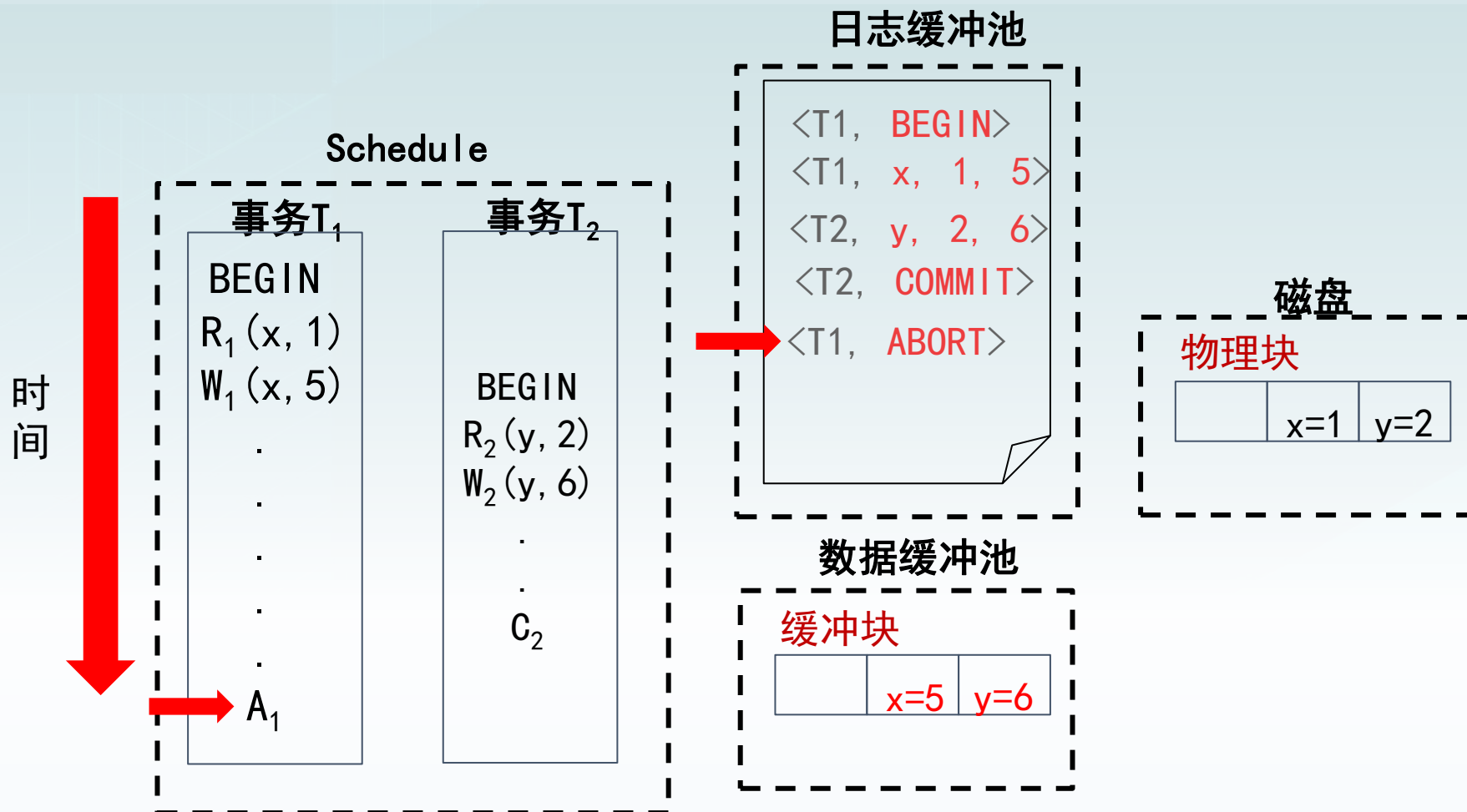
1. 反向扫描文件日志，查找该事务的更新操作
2. 对该事务的更新操作执行逆操作
3. 继续反向扫描日志文件，查找该事务的其他更新操作
4. 如此处理下去，直至读到此事务的开始标记

- 事务故障发生前，数据库的一致性状态是什么？
 - 指的是这个事务的所有操作都未发生时，数据库的一致性状态。
- 事务故障发生后，数据库有哪些一致性状态被破坏？
 - 事务将修改的数据项写入到缓冲区中对应的缓冲块，且该缓冲块被系统异步写入到磁盘的物理块中，导致数据库中存储了脏数据。
 - 事务将修改的数据项写入到缓冲区中对应的缓冲块，导致并发事务读取了该事务写入的脏数据。这一问题是通过并发控制来保证。

事务故障的恢复

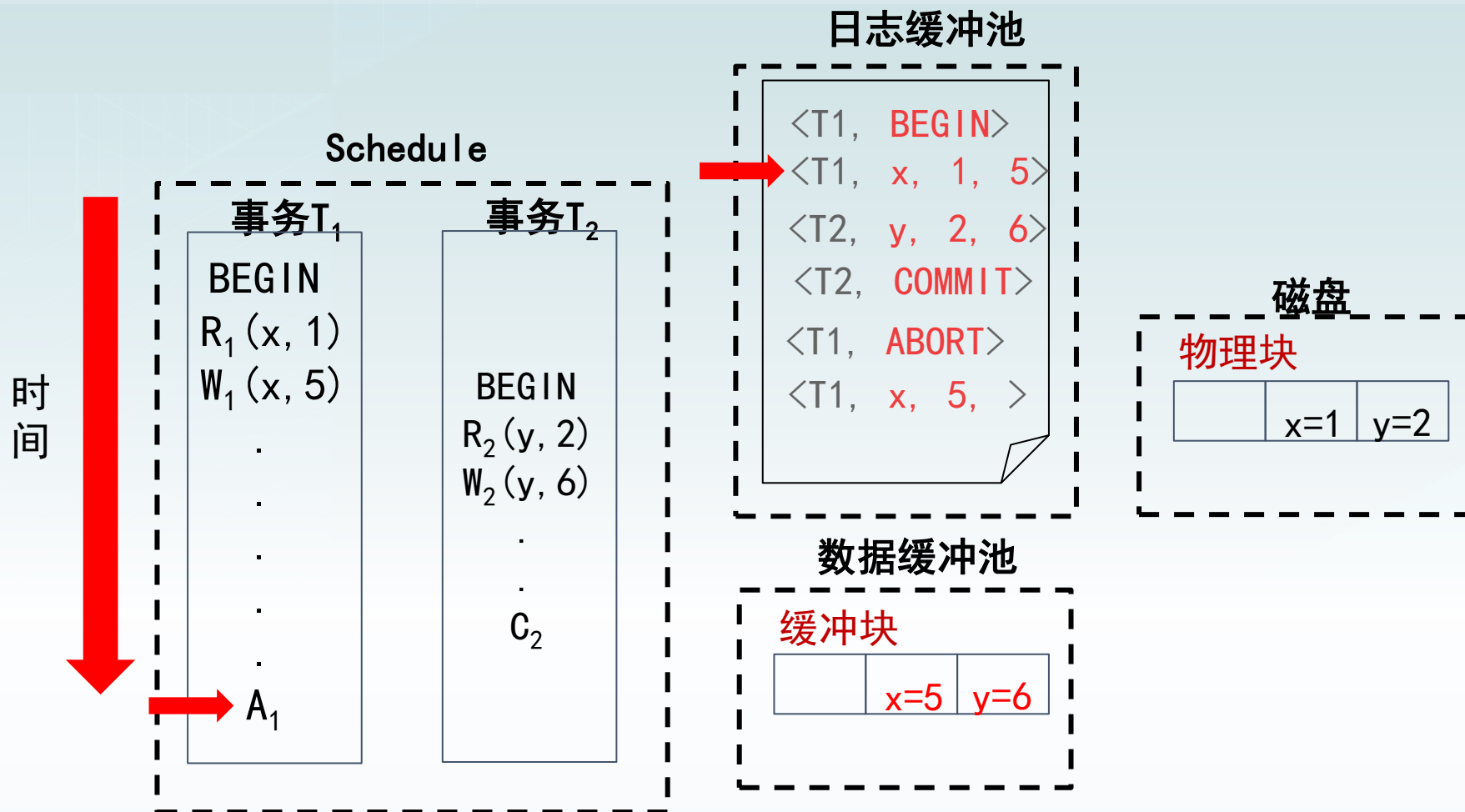


事务故障的恢复



1. 反向扫描文件日志，查找该事务的更新操作
2. 对该事务的更新操作执行逆操作
3. 继续反向扫描日志文件，查找该事务的其他更新操作
4. 如此处理下去，直至读到此事务的开始标记

事务故障的恢复



1. 反向扫描文件日志，查找该事务的更新操作
2. 对该事务的更新操作执行逆操作
3. 继续反向扫描日志文件，查找该事务的其他更新操作
4. 如此处理下去，直至读到此事务的开始标记

系统故障的恢复

- 系统故障的恢复策略为

- 执行Undo和Redo操作

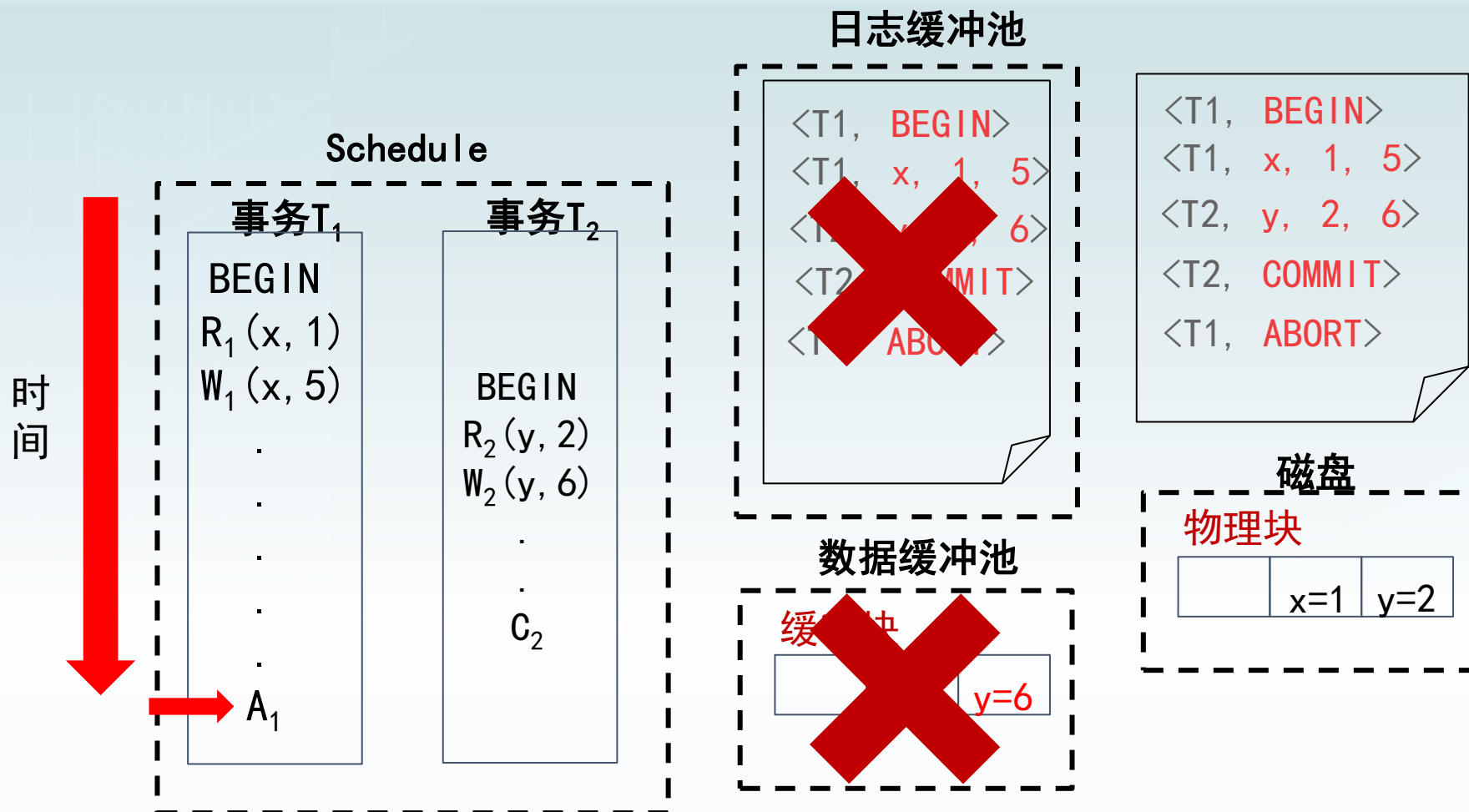
- 具体的

1. 正向扫描日志文件，记录重做(REDO) 队列和撤销(Undo)队列
2. 反向扫描日志文件，对撤销(Undo)队列事务进行撤销(UNDO)处理
3. 正向扫描日志文件，对重做(Redo)队列事务进行重做(REDO)处理

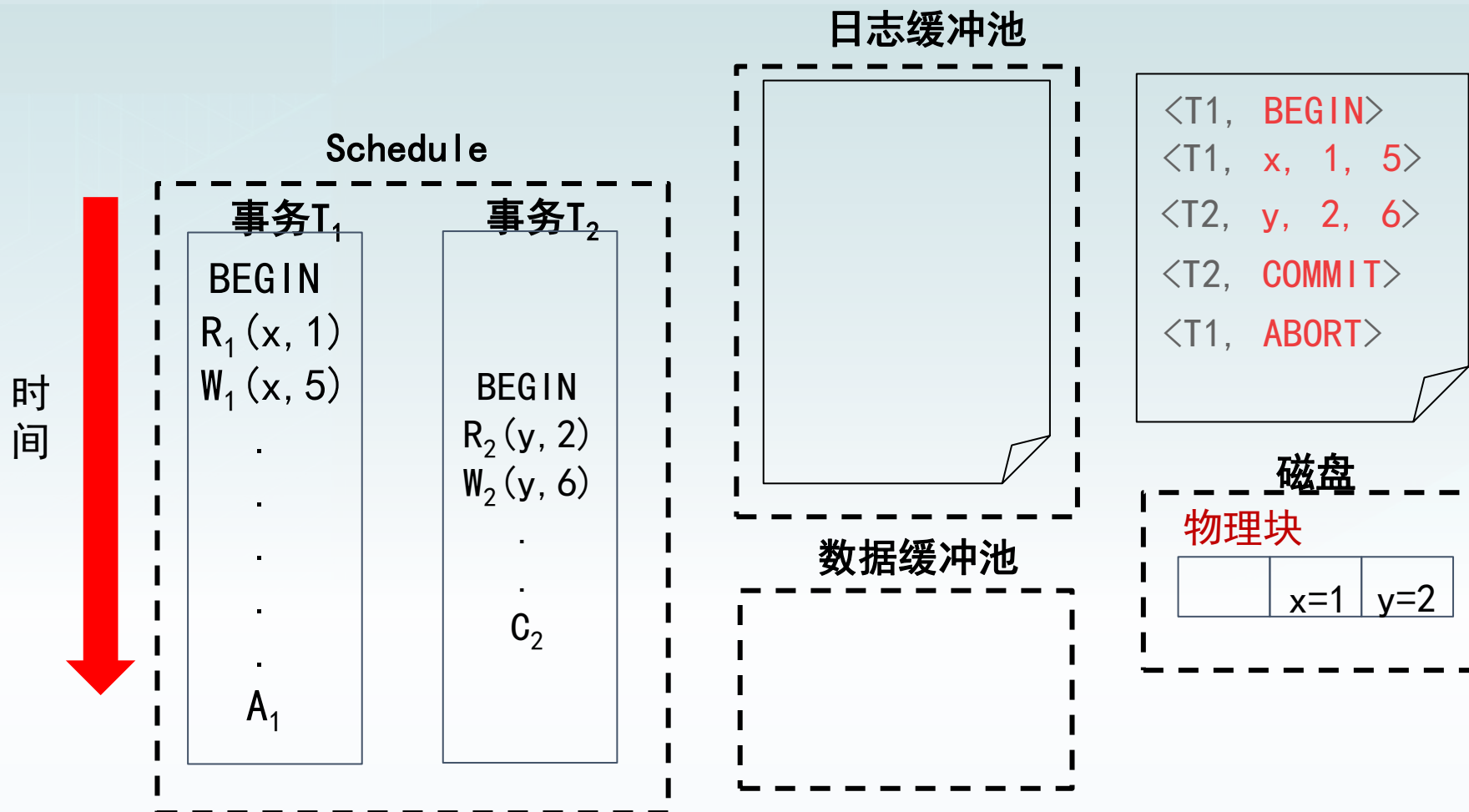
- 故障发生后，数据库有哪些一致性状态被破坏？

- 系统故障发生后，系统需要重启，导致缓存区中的所有数据丢失，所有运行事务都被中止
 - **未提交事务的写**：故障发生时，未提交事务的执行被中断，而这些事务中可能有一些事务的写已经写入到磁盘。
 - **已提交事务的写**：已提交事务写入的数据可能部分或全部留在缓冲区中，系统故障导致缓冲区中的数据丢失，而这些数据尚未来得及写到磁盘

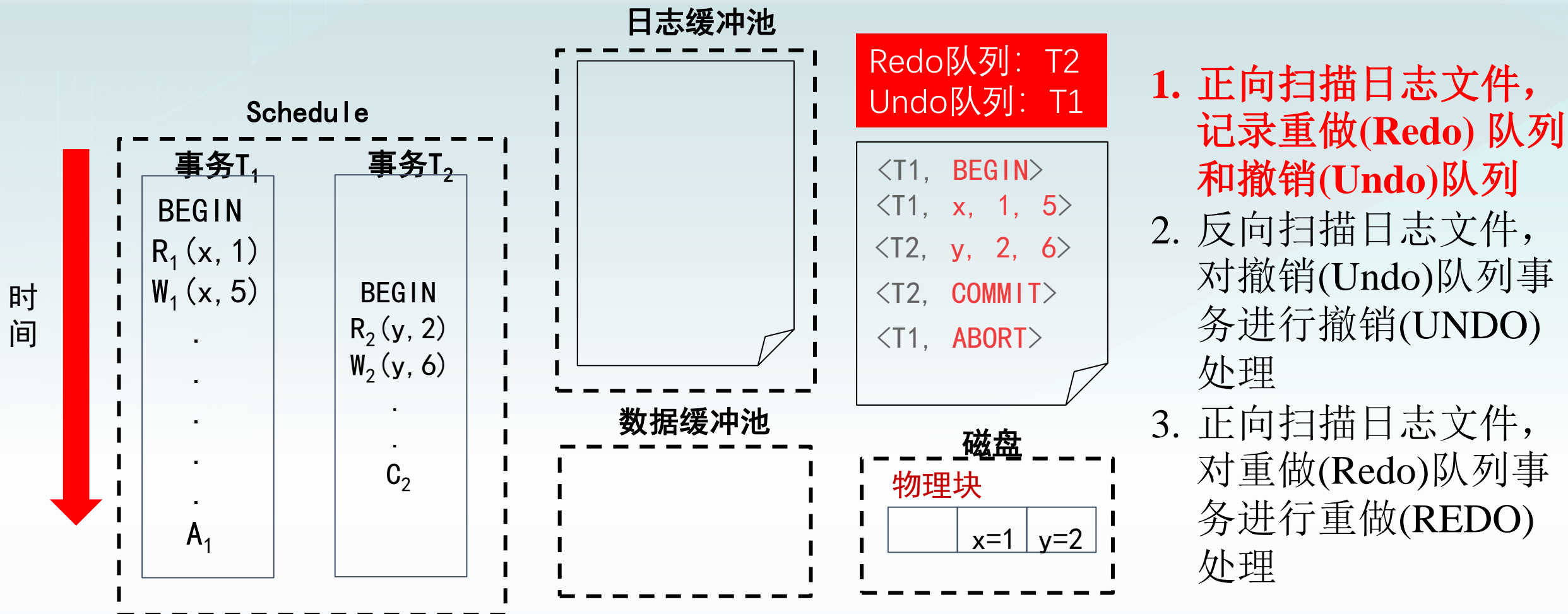
系统故障的恢复



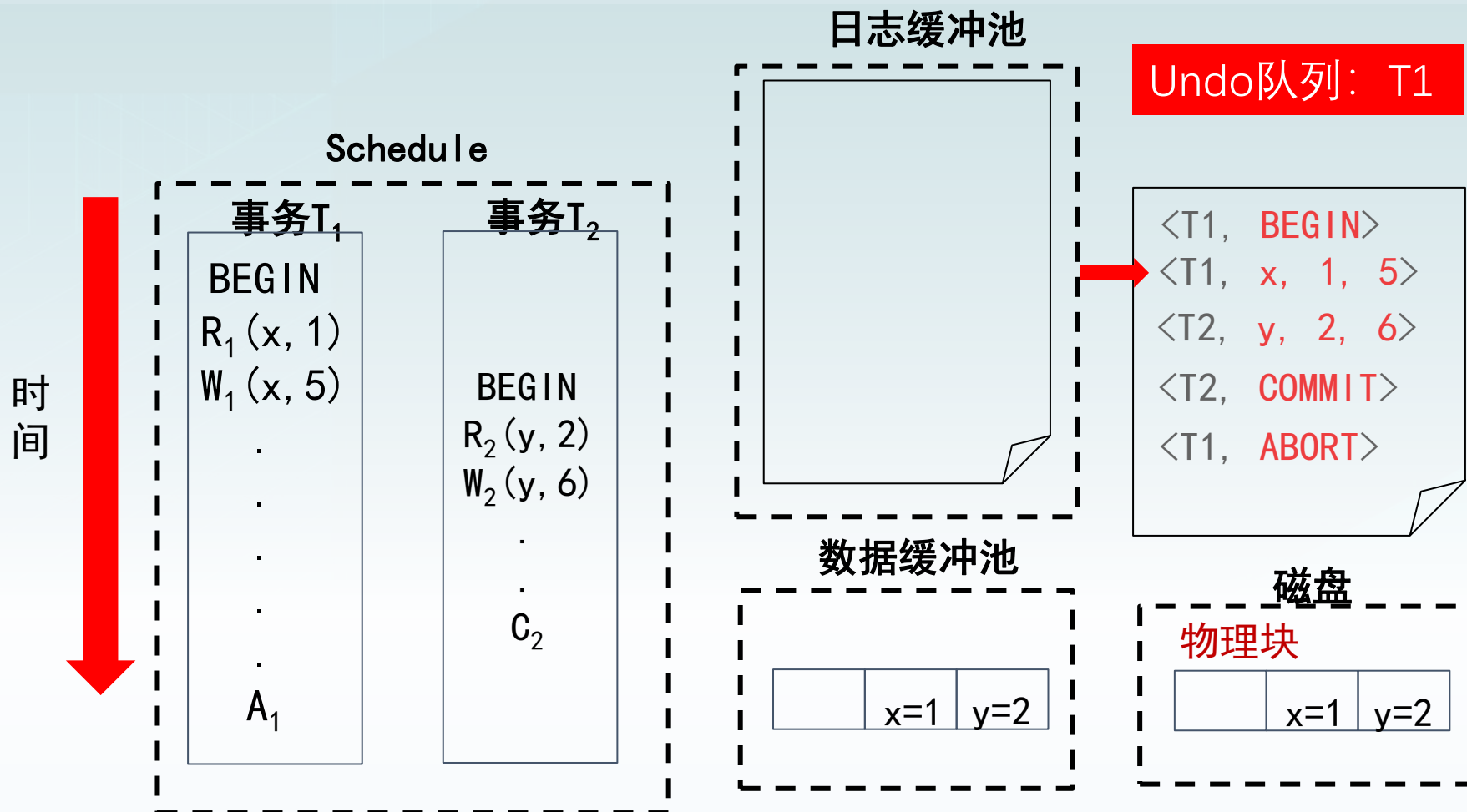
系统故障的恢复



系统故障的恢复

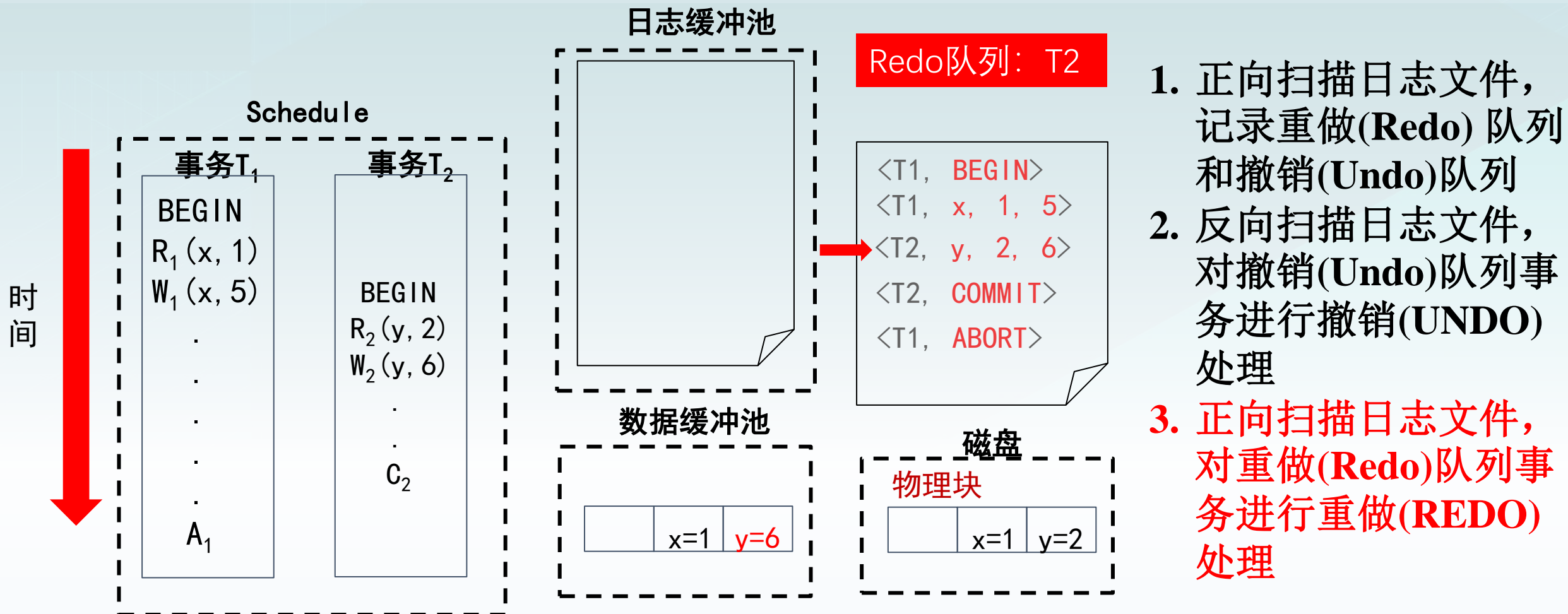


系统故障的恢复



1. 正向扫描日志文件, 记录重做(Redo) 队列和撤销(Undo)队列
2. 反向扫描日志文件, 对撤销(Undo)队列事务进行撤销(UNDO)处理
3. 正向扫描日志文件, 对重做(Redo)队列事务进行重做(REDO)处理

系统故障的恢复



介质故障的恢复

- 恢复步骤

(1) 装入最新的后备数据库副本(离故障发生时刻最近的转储副本)，使数据库恢复到最近一次转储时的一致性状态。

- 对于静态转储的数据库副本，装入后数据库即处于一致性状态
- 对于动态转储的数据库副本，还须同时装入转储时刻的日志文件副本，利用恢复系统故障的方法（即REDO+UNDO），才能将数据库恢复到一致性状态。

介质故障的恢复

(2) 装入有关的日志文件副本(转储结束时刻的日志文件副本)，重做已完成的事务。

- 首先扫描日志文件，找出故障发生时已提交的事务的标识，将其记入重做队列。
- 然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库。

介质故障的恢复

介质故障的恢复需要数据库管理员介入

- 数据库管理员的工作
 - 重装最近转储的数据库副本和有关的各日志文件副本
 - 执行系统提供的恢复命令
- 具体的恢复操作仍由数据库管理系统完成

具有检查点的恢复技术

- 1.问题的提出
- 2.检查点技术
- 3.利用检查点的恢复策略

1. 问题的提出

- 两个问题
 - 搜索整个日志将耗费大量的时间
 - 重做处理：重新执行，浪费了大量时间

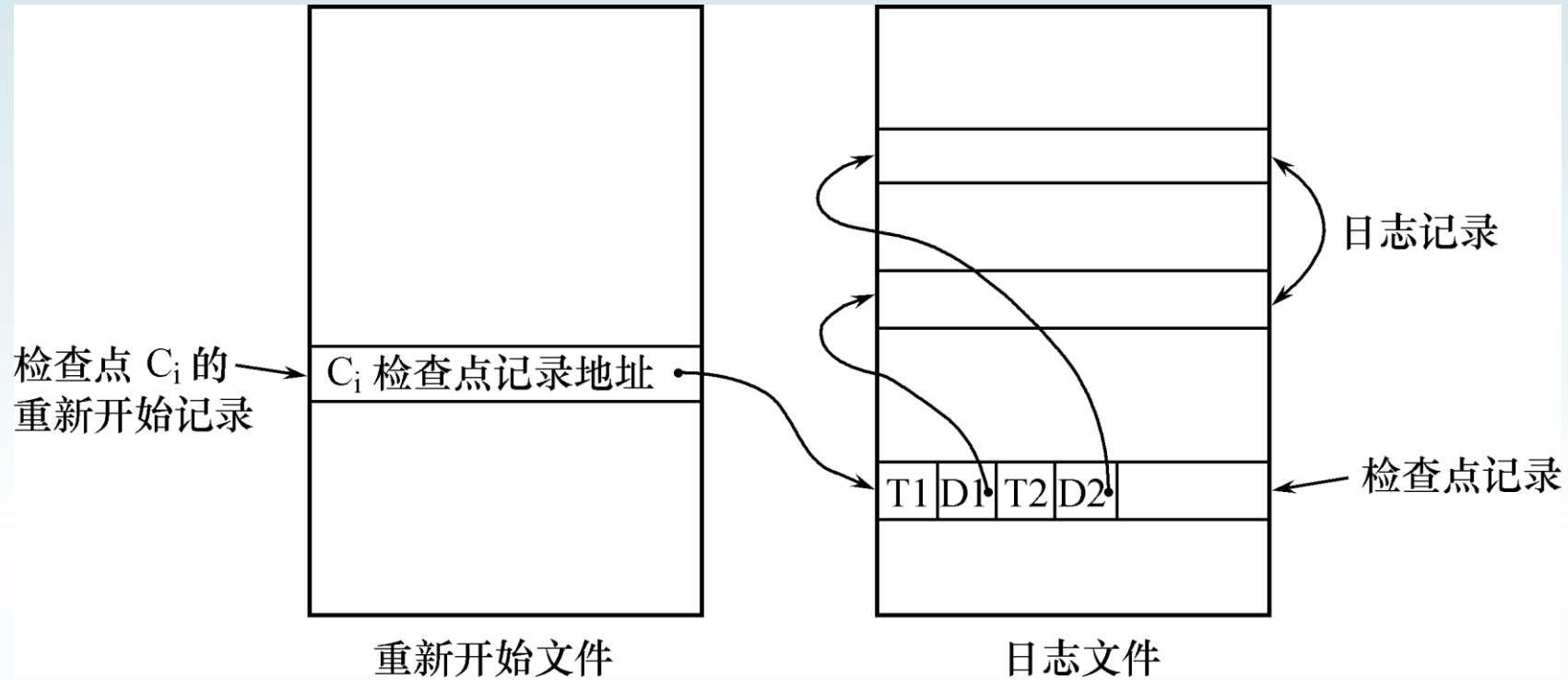
解决方案

- 具有检查点（checkpoint）的恢复技术
 - 在日志文件中增加检查点记录（checkpoint）
 - 增加重新开始文件
 - 恢复子系统在登录日志文件期间动态地维护日志

2. 检查点技术

- 检查点记录的内容
 - 建立检查点时刻所有正在执行的事务清单
 - 这些事务最近一个日志记录的地址
- 重新开始文件的内容
 - 记录各个检查点记录在日志文件中的地址

检查点技术



具有检查点的日志文件和重新开始文件

动态维护日志文件的方法

- 动态维护日志文件的方法

周期性地执行如下操作：建立检查点，保存数据库状态。

具体步骤是：

- (1) 将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上
- (2) 在日志文件中写入一个检查点记录
- (3) 将当前数据缓冲区的所有数据记录写入磁盘的数据库中
- (4) 把检查点记录在日志文件中的地址写入一个重新开始文件

建立检查点

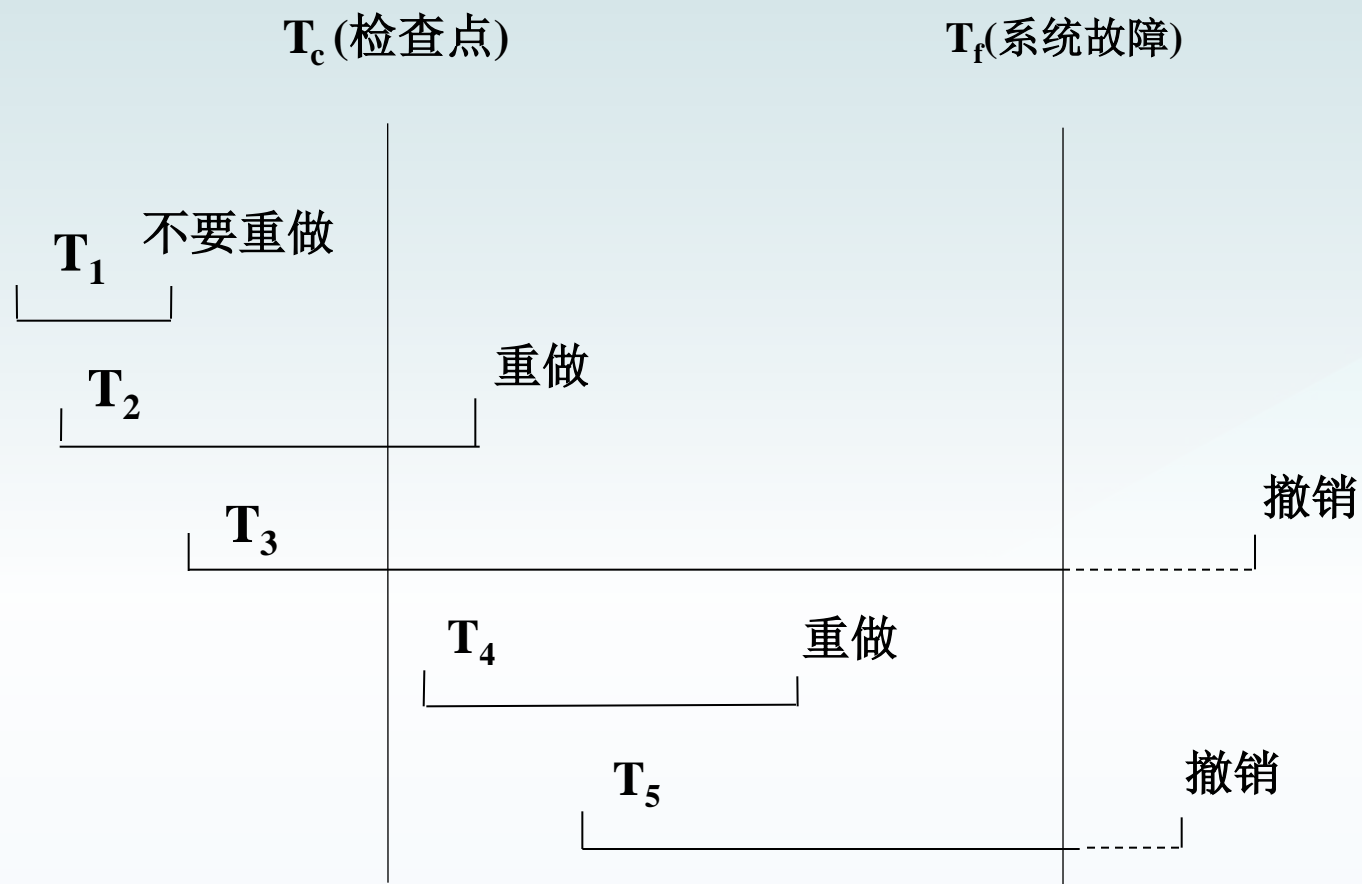
- 恢复子系统可以定期或不定期地建立检查点,保存数据库状态
 - 定期
 - 按照预定的一个时间间隔, 如每隔一小时建立一个检查点
 - 不定期
 - 按照某种规则, 如日志文件已写满一半建立一个检查点

3. 利用检查点的恢复策略

- 使用检查点方法可以改善恢复效率
 - 当事务T在一个检查点之前提交，T对数据库所做的修改已写入数据库
 - 写入时间是在这个检查点建立之前或在这个检查点建立之时
 - 在进行恢复处理时，没有必要对事务T执行重做操作

利用检查点的恢复策略

系统出现故障时，恢复子系统将根据事务的不同状态采取不同的恢复策略



利用检查点的恢复步骤

(1) 从重新开始文件中找到最后一个检查点记录
在日志文件中的地址，由该地址在日志文件中找
到最后一个检查点记录

利用检查点的恢复步骤

(2) 由该检查点记录得到检查点建立时刻所有正在执行的事务清单ACTIVE-LIST

- 建立两个事务队列
 - UNDO-LIST
 - REDO-LIST
- 把ACTIVE-LIST暂时放入UNDO-LIST队列，REDO队列暂为空。

利用检查点的恢复步骤

(3) 从检查点开始正向扫描日志文件，直到日志文件结束

- 如有新开始的事务 T_i ，把 T_i 暂时放入UNDO-LIST队列
- 如有提交的事务 T_j ，把 T_j 从UNDO-LIST队列移到REDO-LIST队列;直到日志文件结束

(4) 对UNDO-LIST中的每个事务执行UNDO操作

对REDO-LIST中的每个事务执行REDO操作

检查点技术

- **[例]**系统崩溃时日志文件记录内容如图所示
- , 试写出**系统重启后恢复处理的步骤及恢复操作**(指UNDO, REDO操作), 并指明**A, B, C, D**恢复后的值分别是多少?

- **分析阶段**: 从最后一次检查点开始**顺向扫描**日志, 确定**重做事务集**REDO-set和**撤销事务集**UNDO-set.
 - 将既有 $\langle T, \text{START} \rangle$ 又有 $\langle T, \text{COMMIT} \rangle$ 日志记录的事务 T 加入REDO-set;
 - 将只有 $\langle T, \text{START} \rangle$ 没有 $\langle T, \text{COMMIT} \rangle$ 日志记录的事务 T 加入UNDO-set。

```
<T0, START>
<T1, START>
<T0, A, 2, 12>
<T2, START>
<T0, COMMIT>
<T1, C, 6, 16>
<T2, B, 4, 14>
<Checkpoint {T1, T2}>
<T1, A, 12, 20>
<T3, START>
<T1, COMMIT>
<T2, B, 14, 40>
<T3, A, 20, 60>
<T4, START>
<T3, D, 8, 18>
<T3, COMMIT>
<T4, C, 16, 30>
```

图 日志文件

■ 分析过程如下：

	UNDO-set	REDO-set
➤ <Checkpoint { T_1, T_2 }>	{ T_1, T_2 }	{ }
➤ < T_3 , START>	{ T_1, T_2, T_3 }	{ }
➤ < T_1 , COMMIT>	{ T_2, T_3 }	{ T_1 }
➤ < T_4 , START>	{ T_2, T_3, T_4 }	{ T_1 }
➤ < T_3 , COMMIT>	{ T_2, T_4 }	{ T_1, T_3 }

- 分析阶段：从最后一次检查点开始顺向扫描日志，确定重做事务集REDO-set和撤销事务集UNDO-set。

- 将既有< T , START>又有< T , COMMIT>日志记录的事务 T 加入REDO-set;
- 将只有< T , START>没有< T , COMMIT>日志记录的事务 T 加入UNDO-set。

```

< $T_0$ , START>
< $T_1$ , START>
< $T_0$ , A, 2, 12>
< $T_2$ , START>
< $T_0$ , COMMIT>
< $T_1$ , C, 6, 16>
< $T_2$ , B, 4, 14>
<Checkpoint { $T_1, T_2$ }>
< $T_1$ , A, 12, 20>
< $T_3$ , START>
< $T_1$ , COMMIT>
< $T_2$ , B, 14, 40>
< $T_3$ , A, 20, 60>
< $T_4$ , START>
< $T_3$ , D, 8, 18>
< $T_3$ , COMMIT>
< $T_4$ , C, 16, 30>

```

图11-21 日志文件

■ 撤销过程如下：

<T₄, C, 16, 30>: C=16

<T₂, B, 14, 40>: B=14

<T₂, B, 4, 14>: B=4

■ 撤销后的结果为: B=4, C=16。

- **撤销阶段**：首先，从日志尾反向扫描日志文件至遇到最后一次检查点止，对每一条属于 UNDO-set={T₂, T₄} 中事务的更新操作日志依次执行 UNDO 操作。

●其次，对于同时出现在 UNDO-set={T₂, T₄} 与 Checkpoint {T₁, T₂} 列表中的事务集 {T₂}，从最后一次检查点开始继续反向扫描日志至遇到这些事务的 START 止，对属于 {T₂} 中事务的更新操作日志依次执行 UNDO 操作。

```
<T0, START>
<T1, START>
<T0, A, 2, 12>
<T2, START>
<T0, COMMIT>
<T1, C, 6, 16>
<T2, B, 4, 14>
<Checkpoint {T1, T2}>
<T1, A, 12, 20>
<T3, START>
<T1, COMMIT>
<T2, B, 14, 40>
<T3, A, 20, 60>
<T4, START>
<T3, D, 8, 18>
<T3, COMMIT>
<T4, C, 16, 30>
```

检查点技术

- [例] 系统崩溃时日志文件记录内容如图所示
- , 试写出系统重启后恢复处理的步骤及恢复操作(指UNDO, REDO操作), 并指明A, B, C, D恢复后的值分别是多少?

- 重做阶段: 从最后一次检查点开始顺向扫描

日志, 对每一条属于REDO-set={ T_1 , T_3 }中事务的更新操作日志依次执行REDO操作。

- 重做过程如下:

$\langle T_1, A, 12, 20 \rangle$: $A=20$

$\langle T_3, A, 20, 60 \rangle$: $A=60$

$\langle T_3, D, 8, 18 \rangle$: $D=18$

- 重做后的结果为: $A=60$, $D=18$ 。

```
<T0, START>
<T1, START>
<T0, A, 2, 12>
<T2, START>
<T0, COMMIT>
<T1, C, 6, 16>
<T2, B, 4, 14>
<Checkpoint {T1, T2}>
<T1, A, 12, 20>
<T3, START>
<T1, COMMIT>
<T2, B, 14, 40>
<T3, A, 20, 60>
<T4, START>
<T3, D, 8, 18>
<T3, COMMIT>
<T4, C, 16, 30>
```

图 日志文件

检查点技术

- [例] 系统崩溃时日志文件记录内容如图所示

恢复完成后的结果为： $A=60$ ， $B=4$ ， $C=16$ ， $D=18$ 。

D 恢复后的值分别是多少？

- **重做阶段**：从最后一次检查点开始**顺向扫描**日志，对每一条属于**REDO-set**= $\{T_1, T_3\}$ 中事务的**更新操作**日志依次执行REDO操作。

- **重做过程如下**：

$\langle T_1, A, 12, 20 \rangle$ ： $A=20$

$\langle T_3, A, 20, 60 \rangle$ ： $A=60$

$\langle T_3, D, 8, 18 \rangle$ ： $D=18$

- 重做后的结果为： $A=60$ ， $D=18$ 。

$\langle T_0, \text{START} \rangle$
 $\langle T_1, \text{START} \rangle$
 $\langle T_0, A, 2, 12 \rangle$

$\langle T_2, B, 4, 14 \rangle$
 $\langle \text{Checkpoint} \{T_1, T_2\} \rangle$
 $\langle T_1, A, 12, 20 \rangle$
 $\langle T_3, \text{START} \rangle$
 $\langle T_1, \text{COMMIT} \rangle$
 $\langle T_2, B, 14, 40 \rangle$
 $\langle T_3, A, 20, 60 \rangle$
 $\langle T_4, \text{START} \rangle$
 $\langle T_3, D, 8, 18 \rangle$
 $\langle T_3, \text{COMMIT} \rangle$
 $\langle T_4, C, 16, 30 \rangle$

图 日志文件

课堂测试

1.下面关于检查点相关操作描述正确的是（）

- A.故障发生时还未完成的事物要予以撤销
- B.检查点之后才提交的事物要重做
- C.检查点之后才提交的事物要予以撤销
- D.在检查点之前已提交的事物不需要重做
- E.在检查点之前已提交的事物要予以撤销



本章小结

- 数据库的故障恢复技术，本质上就是为了保证各类故障下，数据库管理系统至少不丢失数据，同时也不会让数据出错。数据库的故障恢复技术能够保证事务的原子性、一致性和持续性。
- 学习数据库管理系统中事务读写的访问模式、可能出现的各类故障。
- 学习“故障发生前，数据库的一致性状态是什么”、“故障发生后，数据库有哪些一致性状态被破坏”、“故障恢复时，如何将被破坏的数据库一致性状态恢复到故障发生前的一致性状态”这三个基本问题。

作业

□课后习题

教材11章2、3、5

教材12章1、5