南京大学 NANJING UNIVERSITY

奋进行动 团结奋斗 争先进位 坐言起行 应势而动

数 据 管 理 基 础

第10章 事务与数据库恢复技术

智能软件与工程学院

第10章 数据库恢复技术

事务的基本概念

故障和数据库恢复

数据转储和日志文件

恢复策略

具有检查点的恢复技术

数据库镜像

数据管理基础

事务的基本概念

智能软件与工程学院

10.1 事务的基本概念

1. 事务
2. 定义事务
3. 有关事务的SQL语句
4. 事务的ACID特性
5. 事务活动

事务（1）

- 事务(Transaction)是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位。

- 事务和程序是两个概念
  - 在关系数据库中，一个事务可以是一条SQL语句、一组SQL语句或整个程序
  - 一个程序通常包含多个事务

- 事务是恢复和并发控制的基本单位

事务（2）

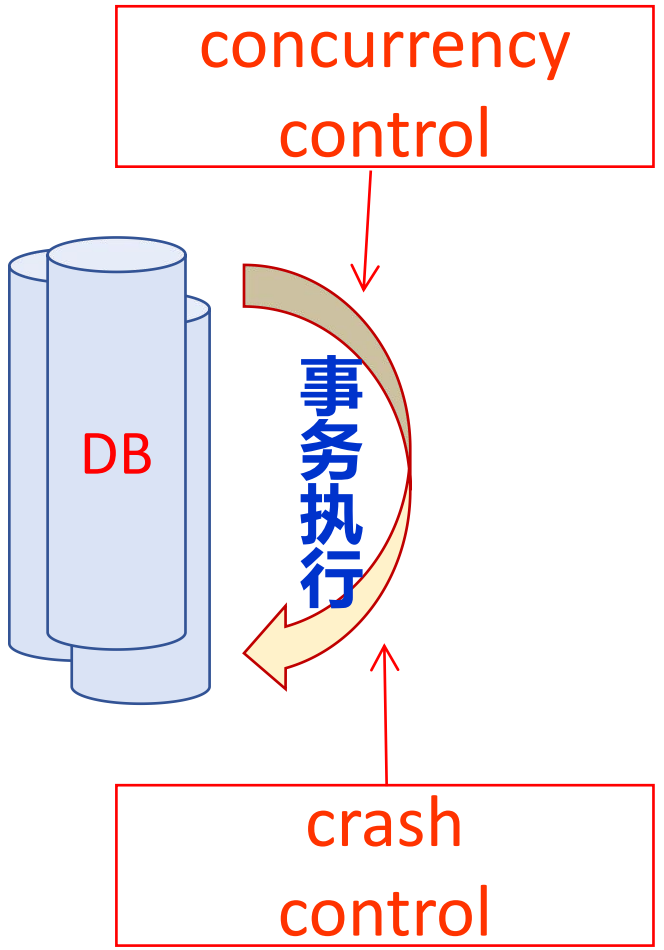
- ‘事务’(Transaction)是一个非常古老的思想，一般是指要做的一件事情；在计算机术语中，‘事务’是指访问并可能更新数据库中数据的一个程序执行单元(unit)；在传统语义学上，‘事务’可以被看成是一种用于实现数据库系统状态转换的行为。

- 三个层面上的‘事务’概念
  - 现实世界：一件‘事情’
  - 计算机世界：一个‘程序执行单元’
  - 数据库管理系统：一个用于访问数据库的‘执行单元’

- 在关系数据库系统中
  - 一个事务是指由“一条SQL语句”或者“一组SQL语句”所构成的一个执行单元，并具有ACID四个特性。
  - ‘事务’是实现用户访问、并发控制、数据库恢复的基本单位。

事务（3）

- 为什么要引入 ‘事务’ 概念?
  - 为了确保实现数据库的数据完整性 (integrity of the database)
  - 在一个数据库系统中，如果所有数据都满足数据完整性约束的定义，那么就称数据库处于一个一致性状态(consistent state)
  - 影响数据完整性的三个因素
    - concurrency
    - abort
    - crash
  - 在数据库管理系统中引入事务处理功能，可以确保在任何情况下，都能保证数据库中数据的正确性。



事务的例子

- [例]银行的转账业务：根据输入的两个银行存款账号A1和A2，以及转账金额X=200，从账号A1转X元到账号A2，即：账号A1的金额减去X，账号A2的金额增加X。在应用程序及DBMS中的执行处理过程如下（*其中：$READ(X,t)$表示将账号X的金额读入内存变量t，$WRITE(X,t)$表示将内存变量t的值作为账号X的金额写入数据库*）

应用程序：Process P1

```
update A
set balance = balance – 200.00
where A.aid = 'A1';
.....
update A
set balance =  balance + 200.00
where A.aid = 'A2';
```

在DBMS中对DB的访问执行过程

$READ(A1,t)$;
$t := t - 200$;
$WRITE(A1,t)$;
$READ(A2,t)$;
$t := t + 200$;
$WRITE(A2,t)$;

- 对该应用程序而言，数据库中数据的一致性就是指：账号A1和账号A2的金额之和不变！

Example 1: 单事务执行（无并发）

起始状态

Process P1:

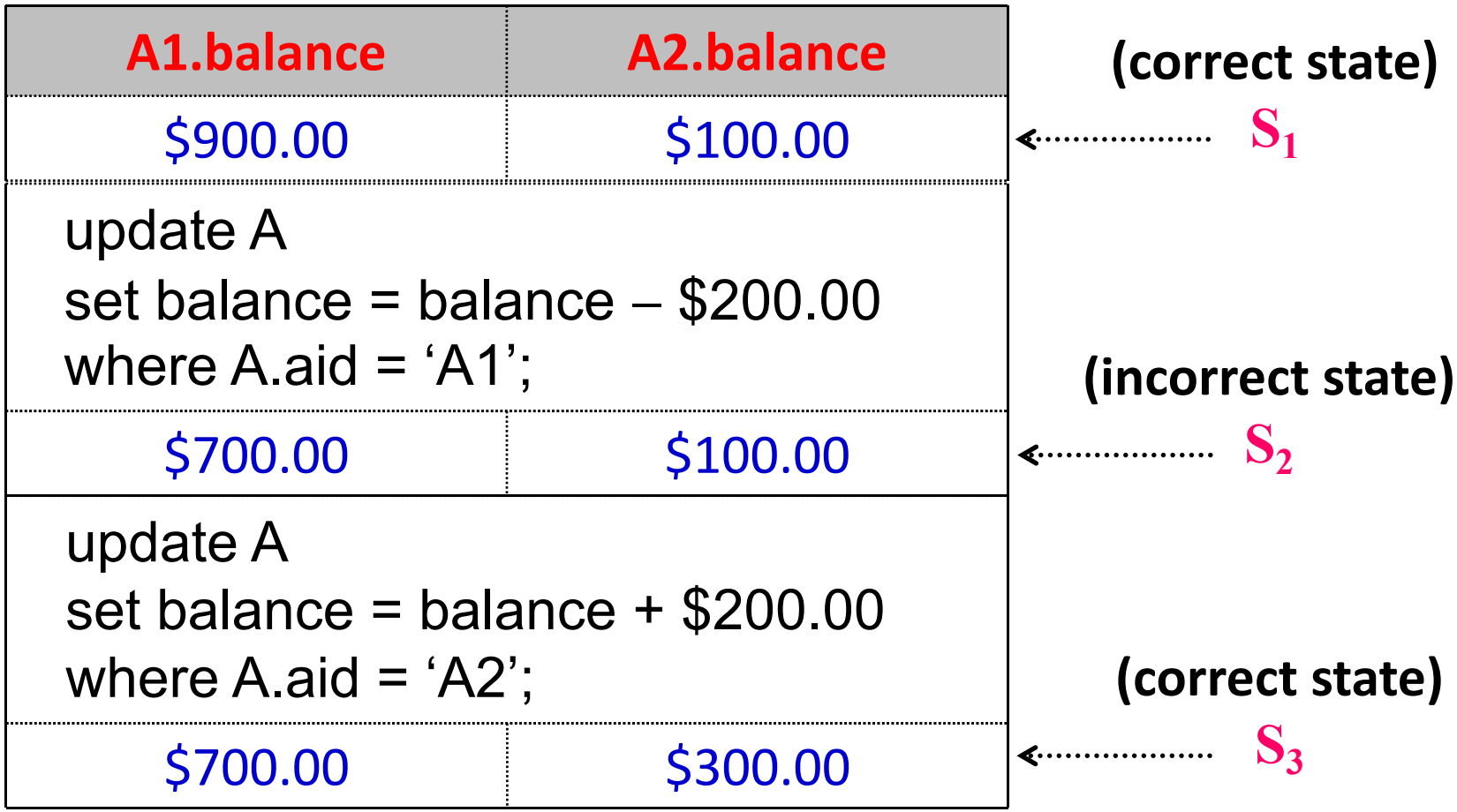
```
update A
set balance = balance – $200.00
where A.aid = 'A1';

update A
set balance =  balance + $200.00
where A.aid = 'A2';
```

A1.balance	A2.balance
$900.00	$100.00
..... Executing P1	
$700.00	$300.00

结束状态

- 在单事务无并发执行方式下，一个事务的执行结果总是正确的！但是.....

Example 1: 单事务执行（无并发）



- 在单事务无并发执行方式下，数据库的初始状态S_1和结束状态S_3都是正确的；但在事务执行过程中，中间状态S_2却有可能是不正确的！

Example 2: two process P1 & P2

Process P1	Process P2
update A set balance = balance – $200.00 where A.aid='A1'; //Now A1.bal=700	int b, sum=0;
	select A.balance into :b from A where A.aid='A1'; sum=sum+b; //Now sum=700

	select A.balance into :b
	from A where A.aid='A2';
	sum=sum+b; //Now sum=800
	//P2 return incorrect result
	//Credit card issuance refused

update A
set balance = balance+$200.00
where A.aid='A2'; //Now A2.bal=300
//P1 transfer complete	

多用户并发访问（无并发控制）：P2访问到的是错误数据

Example 3: two process P1 & P2 (有并发控制)

Process P1	Process P2

update A
set balance = balance – $200.00
where A.aid='A1'; //Now A1.bal=700
 	
update A
set balance = balance+$200.00
where A.aid='A2'; //Now A2.bal=300
//P1 transfer complete
	int b, sum=0;
	
	select A.balance into :b
	from A where A.aid='A1';
	//Can't execute, waiting .....
	
	// retry to execute .....
	select A.balance into :b
	from A where A.aid='A1';
	sum=sum+b; //Now sum=700
	select A.balance into :b
	from A where A.aid='A2';
	sum=sum+b; //Now sum=1000
	//P2 checked OK

多用户并发访问（有并发控制）：P2访问到的是正确数据！

定义事务（1）

❑显式定义方式

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    ……
COMMIT
```

- 事务正常结束
- 提交事务的所有操作（读+更新）
- 事务中所有对数据库的更新写回到磁盘上的物理数据库中

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    ……
ROLLBACK
```

- 事务异常终止
- 事务运行的过程中发生了障，不能继续执行
- 系统将事务中对数据库的所有已完成的操作全部撤销
- 事务滚回到开始时的状态

定义事务（2）

- 隐式方式
  - 当用户没有显式地定义事务时，
  - 数据库管理系统按缺省规定自动划分事务

事务结束

COMMIT

- 事务正常结束
- 提交事务的所有操作（读+更新）
- 事务中所有对数据库的更新写回到磁盘上的物理数据库中

ROLLBACK

- 事务异常终止
- 事务运行的过程中发生了故障，不能继续执行
- 系统将事务中对数据库的所有已完成的操作全部撤销
- 事务滚回到开始时的状态

有关事务的SQL语句（1）

- ‘事务’除了由一组对于数据库的访问操作构成以外，通常还应该包括少量的事务控制语句，用于定义一个事务的开始和结束(包括正常结束和非正常结束)。因此，与事务有关的控制语句主要有三条：
  - 事务的开始 (begin transaction)
  - 事务的结束
    - ① 正常结束
      - 提交事务 (commit transaction)
    - ② 非正常结束
      - 回退事务 (rollback transaction)

有关事务的SQL语句（2）

- **Begin transaction**
  - 现有的关系数据库管理系统并没有提供一个用于定义从什么时候开始一个事务的控制语句，事务的启动是隐式的
  - 可以通过三种方式来启动一个新的事务：
    1. 数据定义命令(DDL)
    2. 将系统设为自动提交方式（打开自动提交标志）
    3. 数据操纵命令(DML)

有关事务的SQL语句（3）

❑ 事务的启动方式

① 数据定义命令(DDL)

- 每一条数据定义命令都将被作为一个单独的事务来执行
- 在此之前，用户的当前active事务将被自动提交

② 将系统设为自动提交方式（打开自动提交标志）

- 每一条数据库访问命令都将被作为一个单独的事务来执行，并根据执行结果由DBMS自动完成提交或回退

③ 数据操纵命令(DML)

- 当DBMS接收到来自某个用户的数据操纵请求时，首先检查该用户当前有没有正在运行的active事务：
- 如果有，该请求被作为当前active事务的组成部分去执行；
- 如果没有，DBMS将自动为该用户启动一个新的事务，该请求就被作为新事务的第一条数据访问请求去执行。

有关事务的SQL语句（4）

- **Commit transaction**
  - 用于提交用户当前的active事务。如果能够成功完成提交，该事务在执行过程中对于数据库的所有修改操作结果都将永久地反应到数据库中，并且不可被取消。
  - 事务的提交操作也可能失败，其原因包括：
    - 发生系统故障
    - 在提交阶段执行的数据完整性检查不满足要求
  - 在事务提交失败后，用户可以通过回退(Rollback)操作来取消当前事务
    - 由系统自动提交的事务，如果提交失败，系统将自动执行事务的回退操作

有关事务的SQL语句（5）

❑ Rollback transaction

- 取消在该事务执行过程中的所有操作结果，将数据库回滚至该事务开始前的状态，以便重新执行或放弃（abort）该事务
- 事务的执行过程不可能被‘回滚’，执行rollback操作是为了撤销该事务在之前的执行过程中对数据库中的数据所作的修改

❑ 保存点 (savepoint)

- 在事务的执行过程中，用户可以为该事务设置若干个保存点
- 用户事务可以使用 Rollback 命令将当前事务回退到前面的某个保存点sp，放弃“在保存点sp之后，回退操作之前”执行过的对数据库的所有访问操作，并继续执行当前事务
- 不带保存点的回退操作将结束并放弃整个事务

有关事务的SQL语句（6）

- 除了上述三条事务控制语句外，数据库管理系统通常还会提供下述几条与事务有关的控制命令(DCL)

① 设置事务的自动提交命令

```
SET AUTOCOMMIT ON | OFF ;
```

② 设置事务的类型

```
SET TRANSACTION READONLY | READWRITE ;
```

③ 设置事务的隔离级别

```
SET TRANSACTION ISOLATION LEVEL
      READUNCOMMITTED |
      READCOMMITTED |
      READREPEATABLE |
      SERIALIZABLE ;
```

- 用户可以使用这三条命令来设置自己事务的运行方式！

有关事务的SQL语句（7）

- **SET TRANSACTION READONLY | READWRITE ;**
  - 用于定义，在这之后启动的所有事务在执行过程中对数据库的访问方式
    - **READONLY：只读型事务**
      - 在事务的运行过程中只能执行对数据库的‘读’操作，而不能执行‘写’类型的操作
      - 直到定义新的事务类型
    - **READWRITE：读/写型事务**
      - 在事务运行过程中可以执行对数据库的‘读/写’操作
      - 这是事务的缺省类型定义

有关事务的SQL语句（8）

```
SET TRANSACTION ISOLATION LEVEL
        READUNCOMMITTED |
        READCOMMITTED |
        READREPEATABLE |
        SERIALIZABLE ;
```

- 定义当前用户的事务与其它并发执行事务之间的隔离级别；
- 有四种不同的隔离级别可供选择；
- 事务的‘隔离级别’与事务并发所采用的封锁策略紧密相关。选择不同的隔离级别，系统所采用的封锁策略则不同！

- 基于封锁的并发控制（基本思想）：事务T，数据对象O
  - 当事务T持有数据O上的封锁时，允许 T 以‘读’方式访问 O
  - 当事务T持有数据O上的‘写’封锁时，允许 T 以‘写’方式访问 O
  - 一个事务在一个数据对象上，最多只能持有1把一种类型的‘锁’

有关事务的SQL语句（9）

① **READUNCOMMITTED**：未提交读
- 在该方式下，当前事务不需要申请任何类型的封锁，因而可能会‘读’到其他并发事务未提交的修改结果
- 禁止一个事务以该方式去执行对数据的‘写’操作，以避免与其它并发事务的‘写’冲突现象

② **READCOMMITTED**：提交读
- 在‘读’一个数据对象A之前，需要先申请并获得数据对象A上的‘共享性’封锁，在‘读’操作执行结束之后立即释放该封锁
- 以避免读取到其它并发事务未提交的修改结果

③ **READREPEATABLE**：可重复读
- 在‘读’数据对象A之前需要先申请并获得数据对象A上的‘共享性’封锁，并将该封锁维持到当前事务的结束
- 可以避免其它的并发事务对当前事务正在使用的数据对象的修改

④ **SERIALIZABLE**：可序列化(可串行化)
- 并发事务以一种可串行化的调度策略实现其并发执行，以避免它们相互之间的干扰现象

有关事务的SQL语句（10）

- 不管采用何种隔离级别，在一个事务以‘写’方式访问数据对象A之前，需要先申请并获得数据对象A上的‘排它性’封锁，并将该封锁维持到当前事务的结束。

	Read 操作		Write 操作	
	锁类型	封锁时间	锁类型	封锁时间
READUNCOMMITTED 未提交读	No Lock	—	不允许执行 Write 操作	
READCOMMITTED 提交读	共享锁	读操作	排它锁	事务
READREPEATABLE 可重复读	共享锁	事务		
SERIALIZABLE 可串行化	共享锁	事务		

事务的隔离级别与封锁策略之间的关系

事务的特性（ACID特性）

- 事务的ACID特性：
  - 原子性（Atomicity）
    - 事务是数据库的逻辑工作单位
      - 事务中包括的诸操作要么都做，要么都不做
  - 一致性（Consistency）
  - 隔离性（Isolation）
  - 持续性（Durability）

一致性

- 事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态

- 一致性状态
  - 数据库中只包含成功事务提交的结果

- 不一致状态
  - 数据库系统运行中发生故障，有些事务尚未完成就被迫中断；
  - 这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这时数据库就处于一种不正确的状态

一致性与原子性

- 银行转帐：从帐号A中取出一万元，存入帐号B。
  - 定义一个事务，该事务包括两个操作

A	B
$A = A - 10000$	
	$B = B + 10000$

  - 这两个操作要么全做，要么全不做
    - 全做或者全不做，数据库都处于一致性状态。
    - 如果只做一个操作，用户逻辑上就会发生错误，少了一 万元，数据库就处于不一致性状态。

隔离性、持续性

- 隔离性
  - 一个事务的执行不能被其他事务干扰
  - 一个事务内部的操作及使用的数据对其他并发事务是隔离的
  - 并发执行的各个事务之间不能互相干扰

- 持续性也称永久性（Permanence）
  - 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。
  - 接下来的其他操作或故障不应该对其执行结果有任何影响。

事务的特性

- 保证事务ACID特性是事务处理的任务

- 破坏事务ACID特性的因素

  （1）多个事务并行运行时，不同事务的操作交叉执行

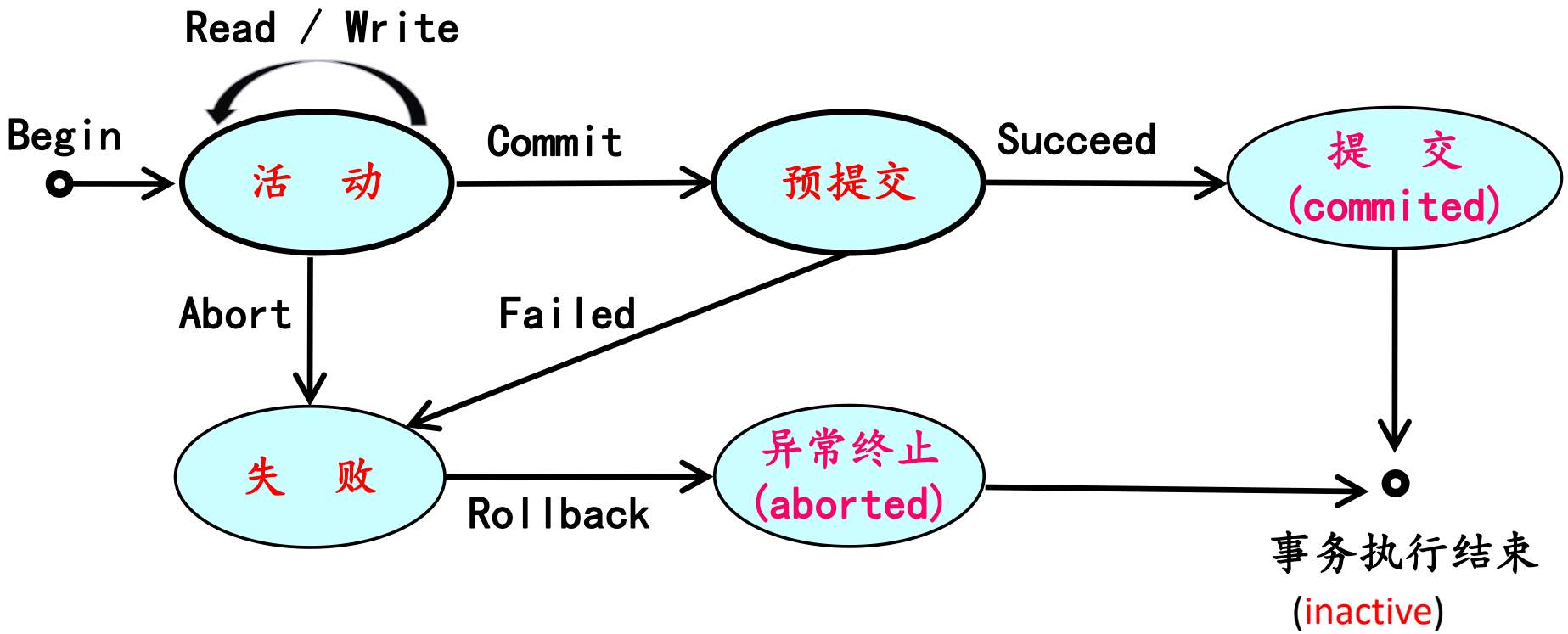
  - 数据库管理系统必须保证多个事务的交叉运行不影响这些事务的隔离性

  （2）事务在运行过程中被强行停止

  - 数据库管理系统必须保证被强行终止的事务对数据库和其他事务没有任何影响

事务活动（1）

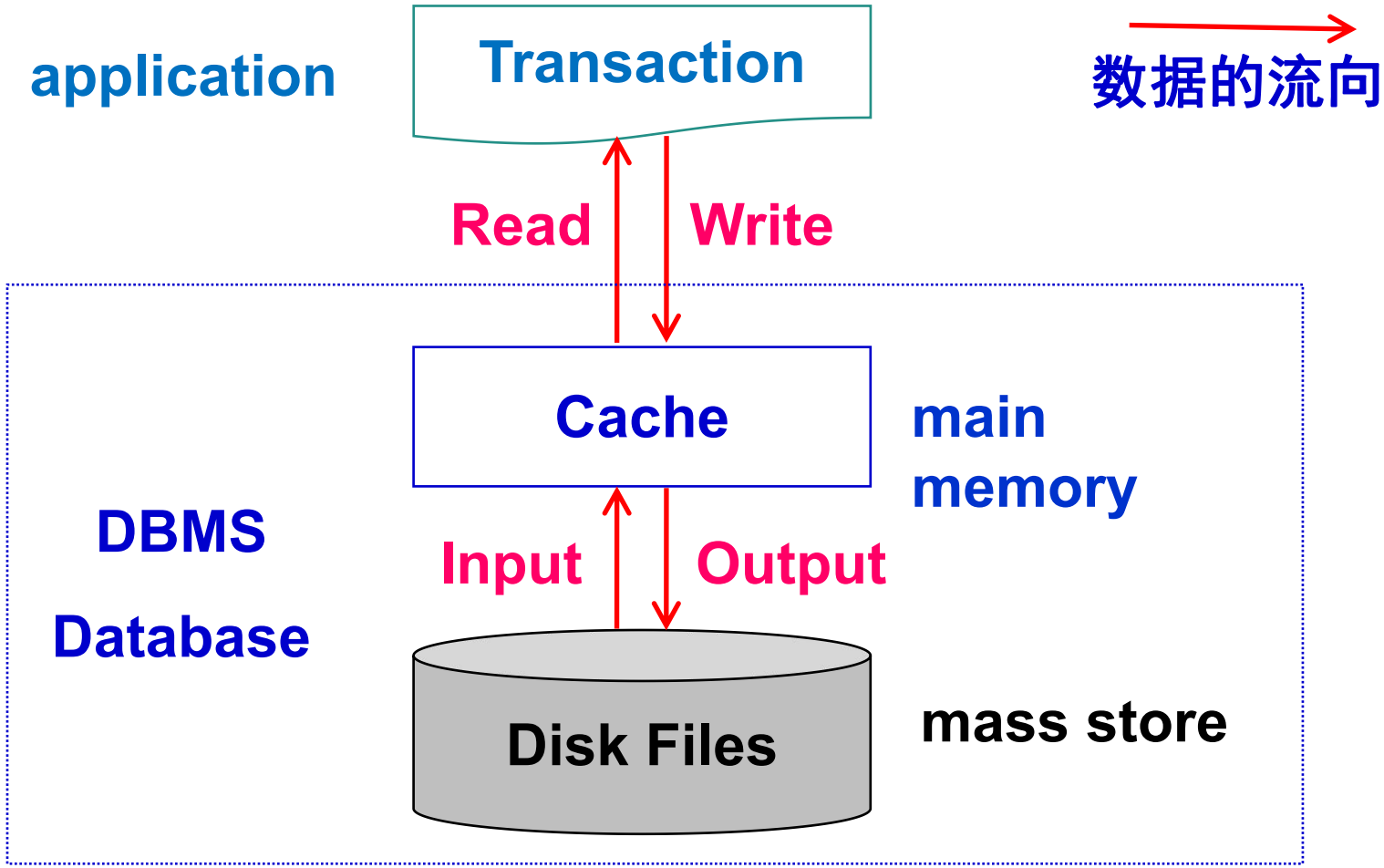
- 为了精确地描述一个事务的工作过程，我们建立了一个抽象的事务模型，用于描述事务的状态变迁情况。



事务的状态变迁图

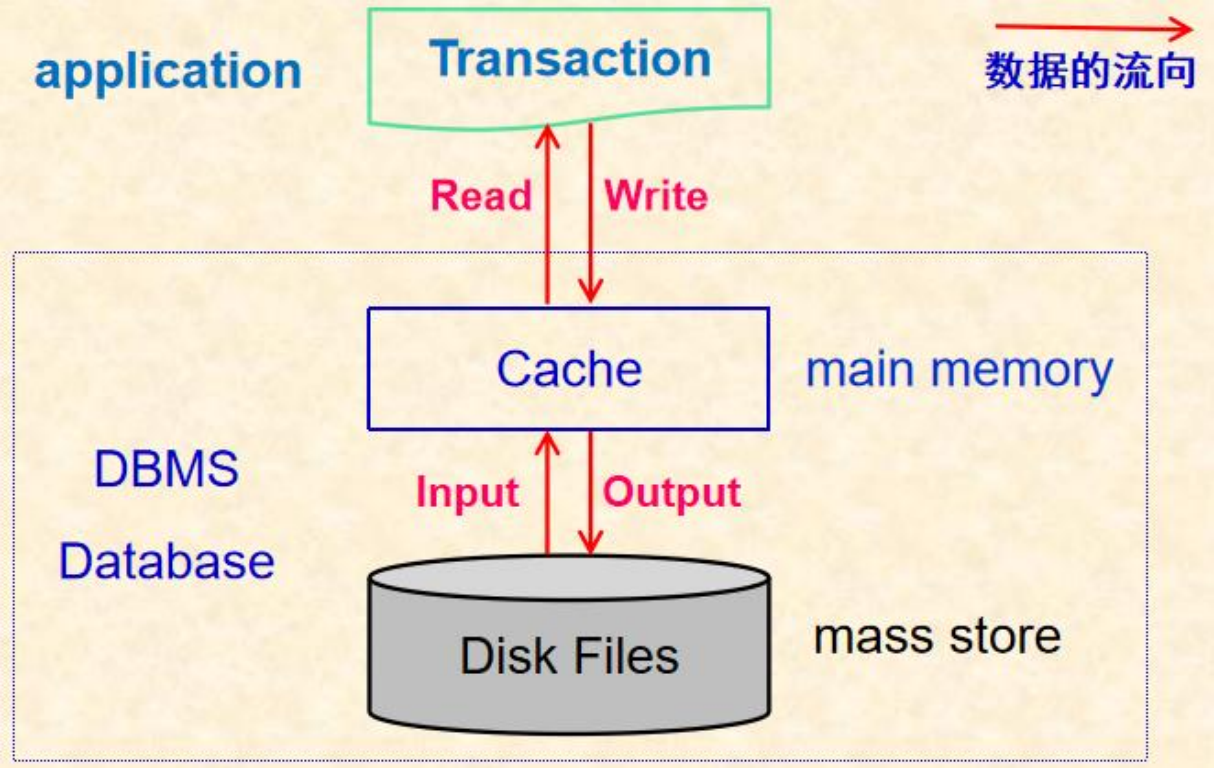
事务活动（2）

- 在一个事务访问数据库的过程中，存在两组类型的‘读/写’操作（如下图所示）



事务活动（3）

- 数据库管理系统在执行来自于用户事务的READ/WRITE操作时，首先访问的是数据库内存缓冲区中的数据；



- 如果用户事务希望访问的数据不在当前的内存缓冲区中，DBMS将调用一个INPUT操作，将需要访问的数据从计算机的磁盘中读入DBMS的内存缓冲区，然后再在内存缓冲区中执行用户事务的READ/WRITE操作；

- 一个事务的修改结果（用WRITE操作写入数据库的数据），如何被持久化地写入数据库（由DBMS调用OUTPUT操作，将修改后的内存缓冲区页面写入计算机的磁盘文件），取决于系统采用的并发控制及事务提交的实现方式。

1、‘活动’状态

- 事务在开始执行后，立即进入'活动'状态。在'活动'状态中，DBMS将根据事务的请求执行对数据库的访问操作；
- 在DBMS的事务管理子系统看来，用户事务对数据库的访问操作就是对数据库中数据的读/写操作。

读操作	• 将数据读入用户事务的私有工作区间 • 如果该数据当前不在DBMS的系统缓冲区中，那么DBMS首先将该数据从磁盘读入系统缓冲区，然后再将其拷贝到用户事务的私有工作区
写操作	• 将修改后的数据‘写入’数据库 • 这里的‘写’操作并不是立即将数据永久性地写入磁盘，很可能暂时存放在DBMS的系统缓冲区中

2、‘预提交’状态

- 当事务的最后一条访问语句执行结束之后，事务进入‘预提交’状态。此时事务对于数据库的访问操作虽然已经执行结束，但其对于数据的修改结果可能还在内存的系统缓冲区中，必须将其真正写入数据库的磁盘；

- 事务在‘预提交’阶段，必须确保将当前事务的所有修改操作的执行结果被真正写入到数据库的磁盘中去；
  - 在所有‘写’磁盘操作执行结束后，事务就进入‘提交’状态

- 在‘预提交’阶段，虽然事务本身的操作命令已经执行结束，但是在“完成提交任务”的过程中仍然会发生系统故障，从而导致当前事务的执行失败。
  - 在‘预提交’失败后，当前事务也将被放弃(abort)，事务转而进入‘失败’状态

3、‘失败’状态

- 处于‘活动’状态的事务在顺利到达并执行完最后一条语句之前就中止执行，或者在‘预提交’状态下因发生系统故障而中止执行时，我们称事务进入‘失败’状态

  - 事务从‘活动’状态转变为‘失败’状态的原因可能是
    - 应用程序(或用户)主动放弃(abort)当前事务
    - 因并发控制的原因而被放弃的事务，如：
      - 封锁申请的等待超时
      - 死锁
    - 发生系统故障

  - 事务从‘预提交’状态转变为‘失败’状态的原因
    - 发生系统故障

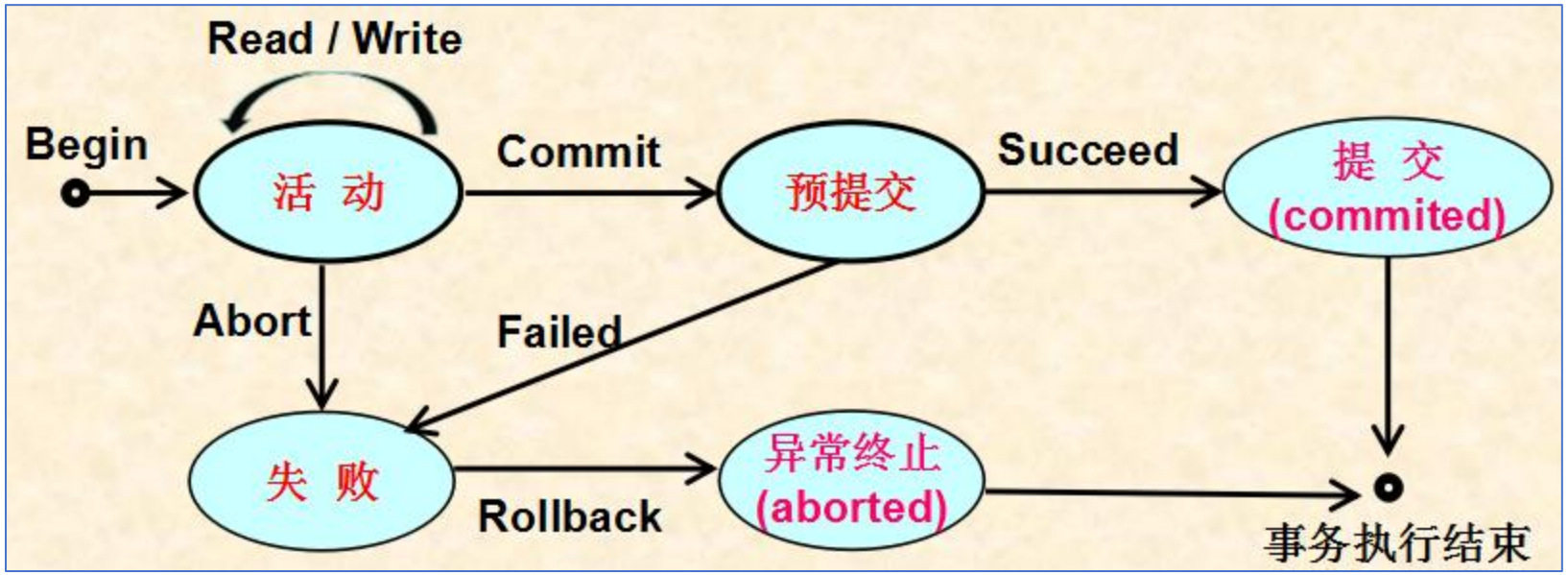
4、‘异常中止’状态

- 处于‘失败’状态的事务，很可能已对磁盘中的数据进行了一部分修改。为了保证事务的原子性，系统应该撤消(undo操作)该事务对数据库已作的修改。在撤消操作完成以后，事务将被打上一个放弃的标志(aborted)，转而进入‘异常中止’状态
  - 回退(rollback)
    - 对事务的撤消操作也称为事务的‘回退’或‘回滚’
    - 事务的‘回退’由DBMS的‘恢复子系统’实现
  - 在事务进入‘异常中止’状态后，系统有两种选择：
    - 作为一个新的事务，重新启动
    - 作为一个被放弃的事务，结束运行

5、‘提交’状态

- 事务进入‘预提交’状态后，‘并发控制子系统’将检查该事务与并发执行的其它事务之间是否发生干扰现象；（检查是否满足并发运行的‘隔离性’）
- 在检查通过以后，系统执行提交(commit)操作，把对数据库的修改全部写到磁盘上（实现事务执行结果的‘持久化’），并通知系统，事务已成功地结束；
- 为事务打上一个提交标志(committed)，事务就进入‘提交’状态。

事务活动（4）



- 不论是‘提交’状态，还是‘异常中止’状态，都意味着一个事务的执行结束；
- 当一个事务开始运行后，该事务就进入‘活跃’状态，我们称其是一个‘**active transaction**’。在一个事务执行结束后，其状态就自动转为‘不活跃’，被称为‘**inactive transaction**’；
- 并发控制仅仅是针对当前的所有‘**active transaction**’。

数据管理基础

故障和数据库恢复

智能软件与工程学院

故障和数据库恢复

- 故障是不可避免的
  - 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
  - 破坏数据库，全部或部分丢失数据

故障和数据库恢复

- 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

- 故障的种类
  - 事务内部的故障
  - 系统故障
  - 介质故障
  - 计算机病毒

- 各类故障，对数据库的影响有两种可能性
  - 一是数据库本身被破坏
  - 二是数据库没有被破坏，但数据可能不正确，这是由于事务的运行被非正常终止造成的。

事务内部的故障 1

- 例如，银行转账事务，这个事务把一笔金额从一个账户甲转给另一个账户乙。

```
BEGIN TRANSACTION
    读账户甲的余额BALANCE;
BALANCE=BALANCE-AMOUNT;   /*AMOUNT 为转账金额*/
IF(BALANCE < 0 ) THEN {
    打印‘金额不足，不能转账’; /*事务内部可能造成事务被回滚的情况*/
     ROLLBACK;                  /*撤销刚才的修改，恢复事务*/
}
ELSE {
     读账户乙的余额BALANCE1;
      BALANCE1=BALANCE1+AMOUNT;
      写回BALANCE1;
      COMMIT;
}
```

事务内部的故障 2

- 这个例子所包括的两个更新操作要么全部完成要么全部不做。否则就会使数据库处于不一致状态，例如只把账户甲的余额减少了而没有把账户乙的余额增加。
- 在这段程序中若产生账户甲余额不足的情况，应用程序可以发现并让事务滚回，撤销已作的修改，恢复数据库到正确状态。
- 事务内部更多的故障是非预期的，是不能由应用程序处理的。
  - 运算溢出
  - 并发事务发生死锁而被选中撤销该事务
  - 违反了某些完整性限制而被终止等
- 事务故障仅指这类非预期的故障

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

- 系统故障，称为软故障，是指造成系统停止运转的任何事件（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），使得系统要重新启动。
  - 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
  - 不破坏数据库
  - 内存中数据库缓冲区的信息全部丢失

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，强行撤消（UNDO）所有未完成事务

- 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

介质故障

- 介质故障，称为硬故障，指外存故障
  - 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

计算机病毒

- 计算机病毒
  - 一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序
  - 可以繁殖和传播，造成对计算机系统包括数据库的危害

- 计算机病毒已成为计算机系统的主要威胁，自然也是数据库系统的主要威胁

- 数据库一旦被破坏仍要用恢复技术把数据库加以恢复

恢复

- 恢复操作的基本原理：冗余
  - 利用存储在系统别处的冗余数据来重建数据库中已被破坏或不正确的那部分数据
- 恢复的实现技术：复杂
  - 一个大型数据库产品，恢复子系统的代码要占全部代码的10%以上
- 恢复机制涉及的关键问题
  - 如何建立冗余数据（数据转储，登记日志文件）
  - 如何利用这些冗余数据实施数据库恢复

数据管理基础

数据转储和日志文件

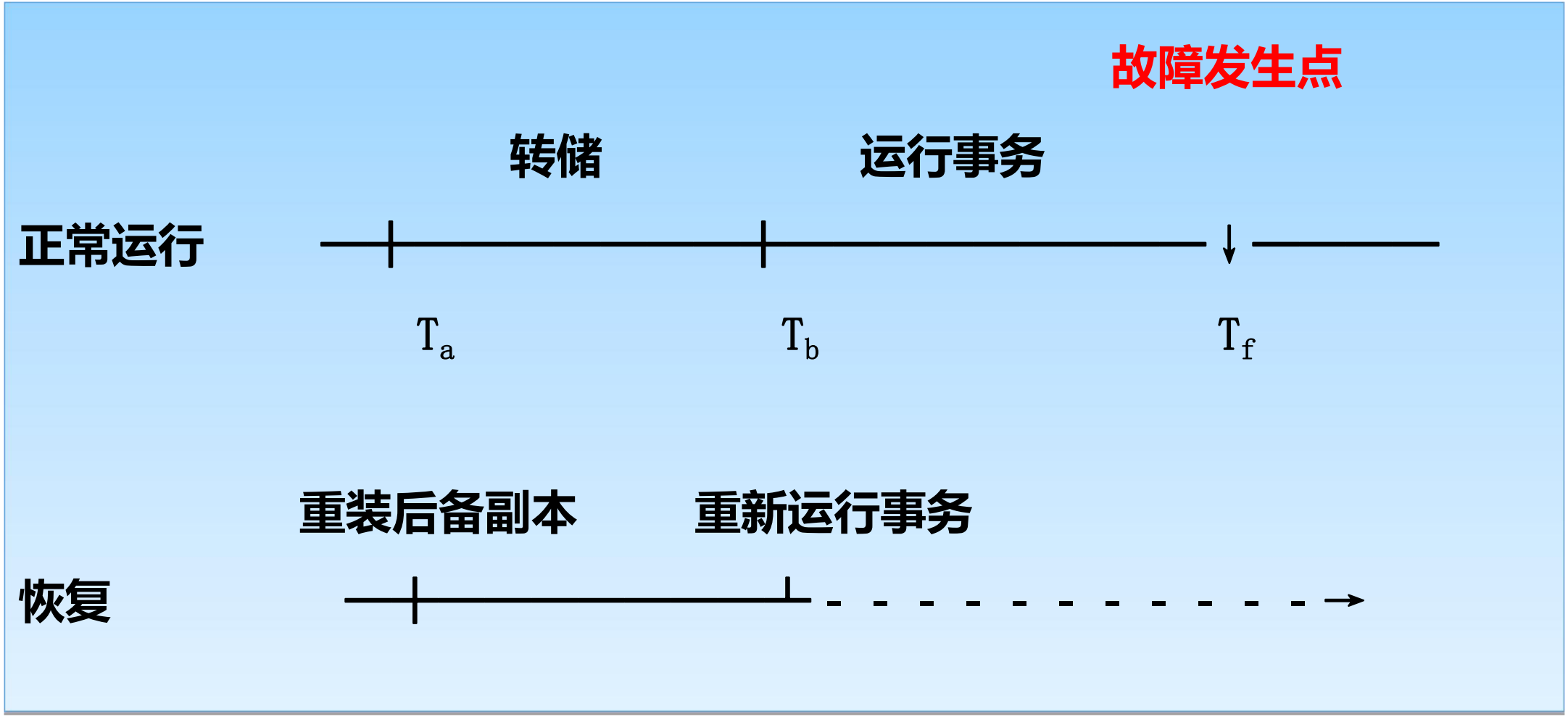
智能软件与工程学院

数据转储 1

- 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
- 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 2

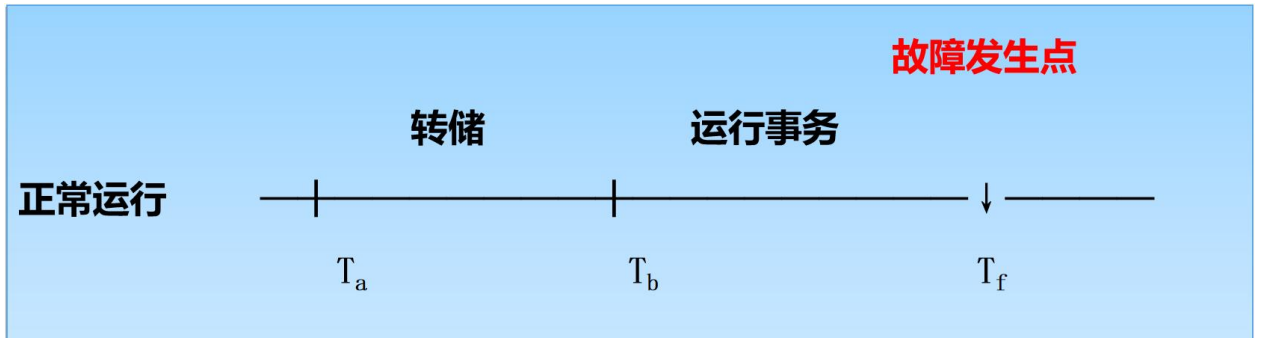
[例]

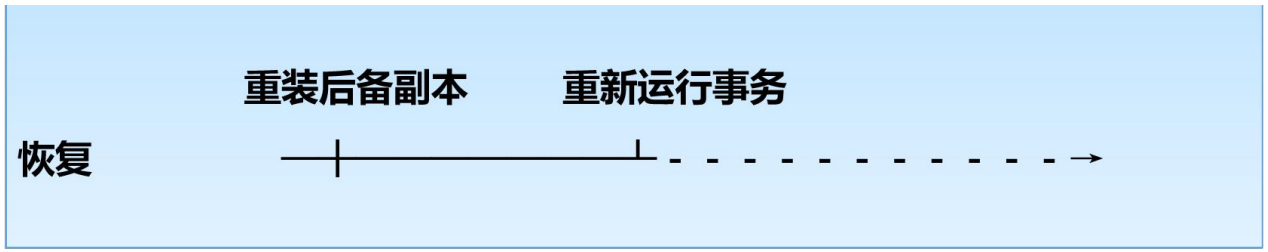


转储和恢复

数据转储 3

- 上图中（如右所示）
  - 系统在T_a时刻停止运行事务，进行数据库转储 （静态转储）
  - 在T_b时刻转储完毕，得到T_b时刻的数据库一致性副本
  - 系统运行到T_f时刻发生故障
  - 为恢复数据库，首先由数据库管理员重装数据库后备副本，将数据库恢复至T_b时刻的状态
  - 重新运行自T_b～T_f时刻的所有更新事务，把数据库恢复到故障发生前的一致状态





转储方法-静态转储

- **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

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
  - 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=100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
- **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

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

日志文件

- 日志文件(log file)是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

- 用途
  - 进行事务故障恢复
  - 进行系统故障恢复
  - 协助后备副本进行介质故障恢复

以记录为单位的日志文件 1

- 以记录为单位的日志文件内容
  - 日志文件中的一个日志记录 (log record)包含
    - 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

以记录为单位的日志文件 2

- 以记录为单位的日志文件，每条日志记录的内容
  - 事务标识（标明是哪个事务）
  - 操作类型（插入、删除或修改）
  - 操作对象（记录内部标识）
  - 更新前数据的旧值（对插入操作而言，此项为空值）
  - 更新后数据的新值（对删除操作而言，此项为空值）

以数据块为单位的日志文件

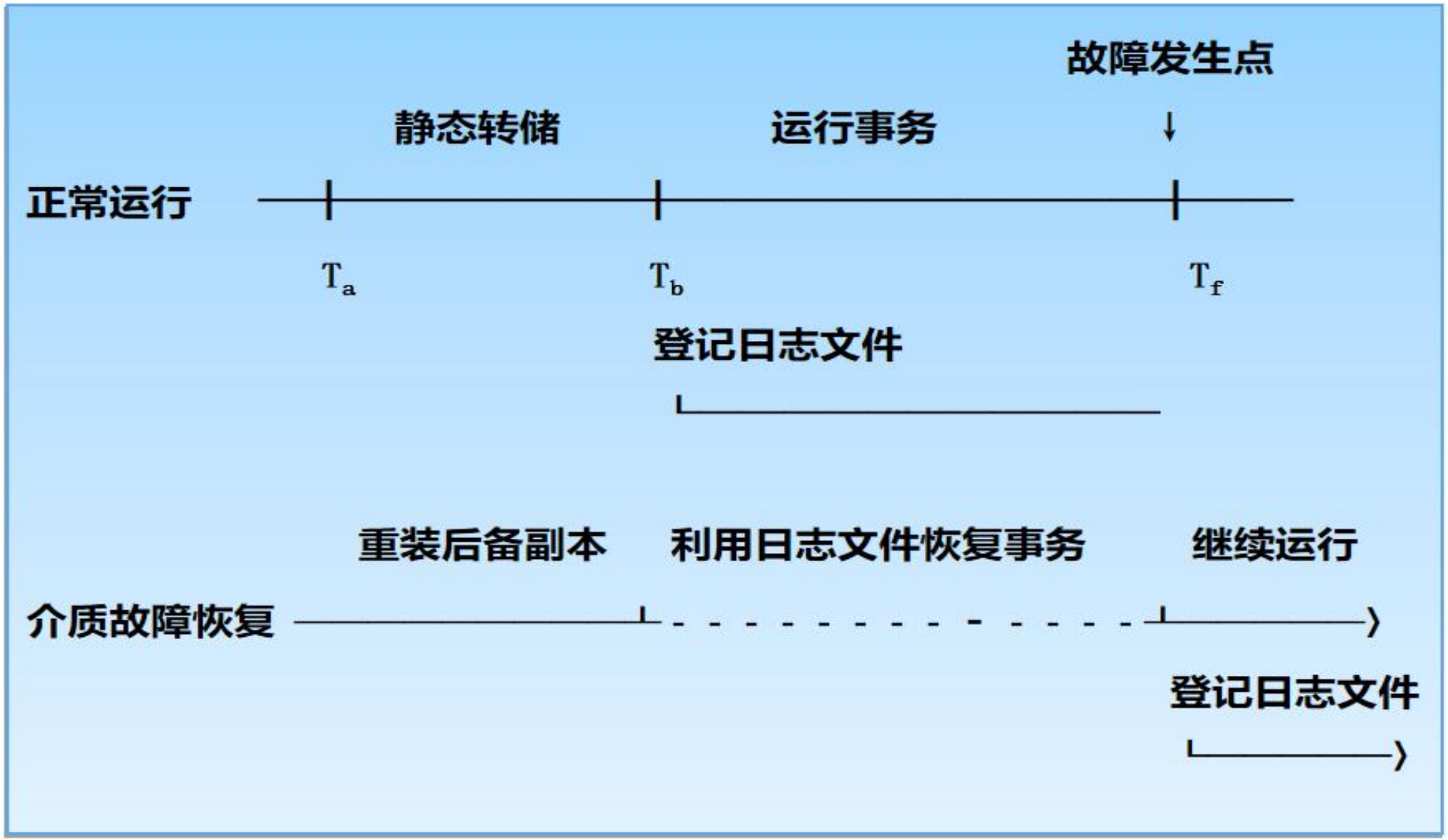
- 以数据块为单位的日志文件，每条日志记录的内容
  - 事务标识
  - 被更新的数据块

日志文件的作用 1

- 具体作用
  - 事务故障恢复和系统故障恢复必须用日志文件。
  - 在动态转储方式中必须建立日志文件，后备副本和日志文件结合起来才能有效地恢复数据库。

  - 在静态转储方式中，也可以建立日志文件。
    - 当数据库毁坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态
    - 利用日志文件，把已完成的事务进行重做处理
    - 对故障发生时尚未完成的事务进行撤销处理
    - 不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态

日志文件的作用 2



利用日志文件恢复

登记日志文件 1

- 为保证数据库是可恢复的，登记日志文件时必须遵循两条原则
  - 登记的次序严格按并发事务执行的时间次序
  - 必须先写日志文件，后写数据库
    - 写日志文件操作：把表示这个修改的日志记录写到日志文件中
    - 写数据库操作：把对数据的修改写到数据库中

登记日志文件 2

- 为什么要先写日志文件
  - 写数据库和写日志文件是两个不同的操作
  - 在这两个操作之间可能发生故障
  - 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

数据管理基础

恢复策略

智能软件与工程学院

事务故障的恢复

- 事务故障：事务在运行至正常终止点前被终止

- 恢复方法
  - 由恢复子系统利用日志文件撤消（UNDO）此事务已对数据库进行的修改

- 事务故障的恢复由系统自动完成，对用户是透明的，不需要用户干预

事务故障的恢复步骤

- 反向扫描文件日志（即从最后向前扫描日志文件），查找该事务的更新操作。

- 对该事务的每一条更新操作执行逆操作，即将日志记录中“更新前的值”写入数据库。具体处理方式如下：
  - 插入操作，“更新前的值”为空，则相当于做删除操作
  - 删除操作，“更新后的值”为空，则相当于做插入操作
  - 若是修改操作，则相当于用修改前值代替修改后值

- 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理。
- 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了。

系统故障的恢复

- 系统故障造成数据库不一致状态的原因
  - 未完成事务对数据库的更新可能已写入数据库
  - 已提交事务对数据库的更新可能还留在缓冲区没来得及写入数据库

- 恢复方法
  - **Undo** 故障发生时未完成的事务
  - **Redo** 已完成的事务

- 系统故障的恢复由系统在重新启动时自动完成，不需要用户干预

系统故障的恢复步骤

- 正向扫描日志文件（即从头扫描日志文件）
  - 重做(REDO) 队列: 在故障发生前已经提交的事务
    - 这些事务既有BEGIN TRANSACTION记录，也有COMMIT记录
  - 撤销 (UNDO)队列:故障发生时尚未完成的事务
    - 这些事务只有BEGIN TRANSACTION记录，无相应的COMMIT记录

- 对撤销(UNDO)队列事务进行撤销(UNDO)处理
  - 反向扫描日志文件，对每个撤销事务的更新操作执行逆操作
  - 即将日志记录中“更新前的值”写入数据库
- 对重做(REDO)队列事务进行重做(REDO)处理
  - 正向扫描日志文件，对每个重做事务重新执行登记的操作
  - 即将日志记录中“更新后的值”写入数据库

介质故障的恢复

- 介质故障的恢复的工作
  - 重装数据库
  - 重做已完成的事务
- 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

介质故障的恢复步骤

- 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。

- 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。

数据管理基础

具有检查点的恢复技术

智能软件与工程学院

具有检查点的恢复技术

- 恢复的两个问题
  - 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

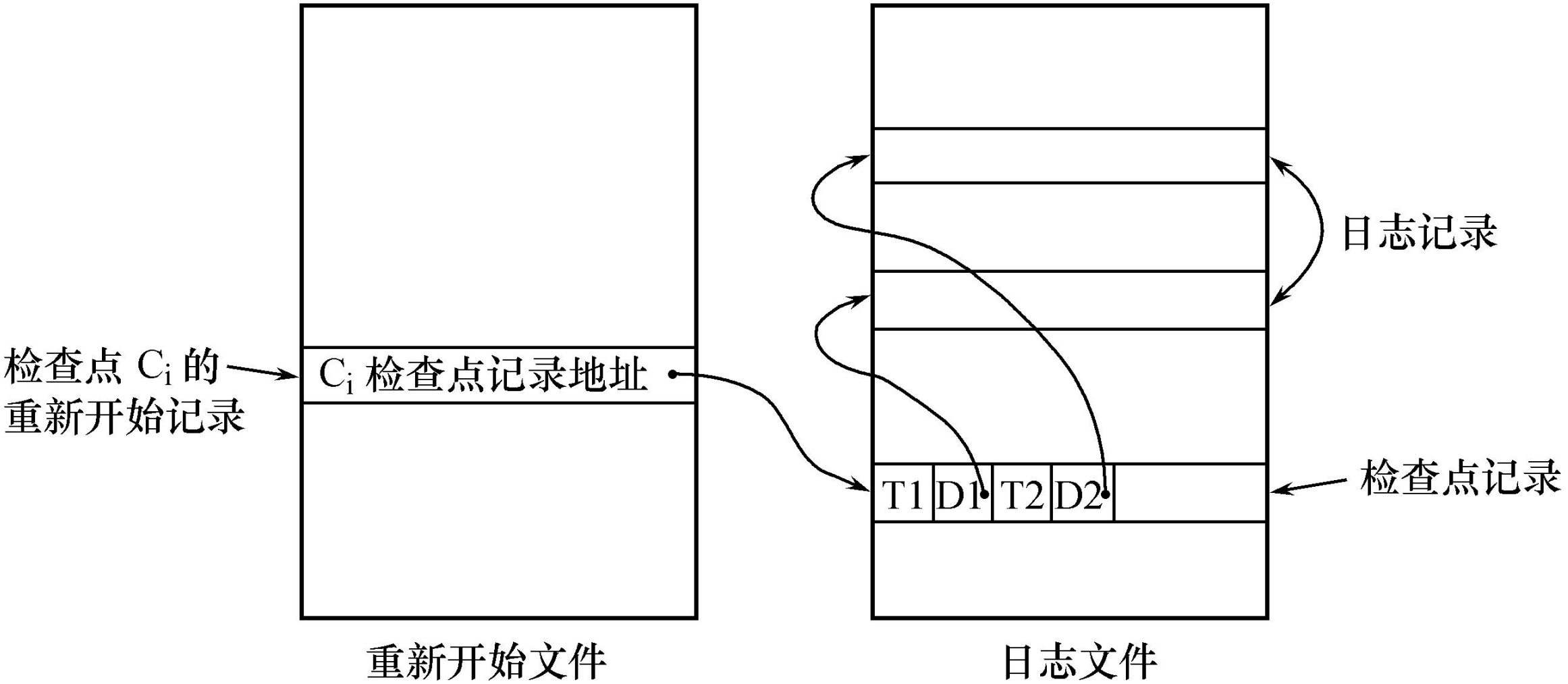
- 具有检查点（checkpoint）的恢复技术
  - 在日志文件中增加检查点记录（checkpoint）
  - 增加重新开始文件
  - 恢复子系统在登录日志文件期间动态地维护日志

检查点技术 1

- 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 2



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：
  - 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
  - 在日志文件中写入一个检查点记录
  - 将当前数据缓冲区的所有数据记录写入磁盘的数据库中
  - 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

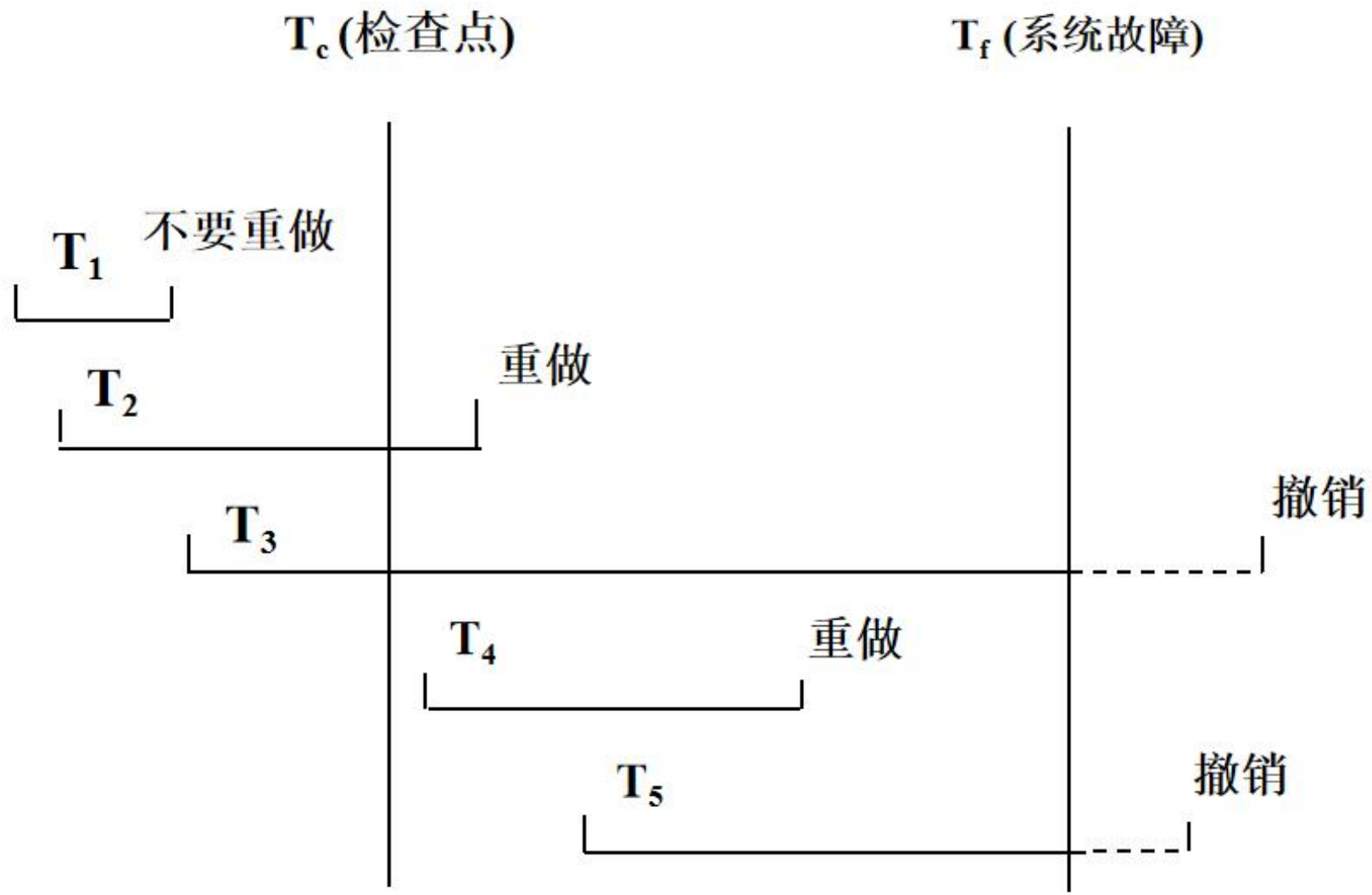
- 恢复子系统可以定期或不定期地建立检查点,保存数据库状态
  - 定期
    - 按照预定的一个时间间隔，如每隔一小时建立一个检查点
  - 不定期
    - 按照某种规则，如日志文件已写满一半建立一个检查点

利用检查点的恢复策略 1

- 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

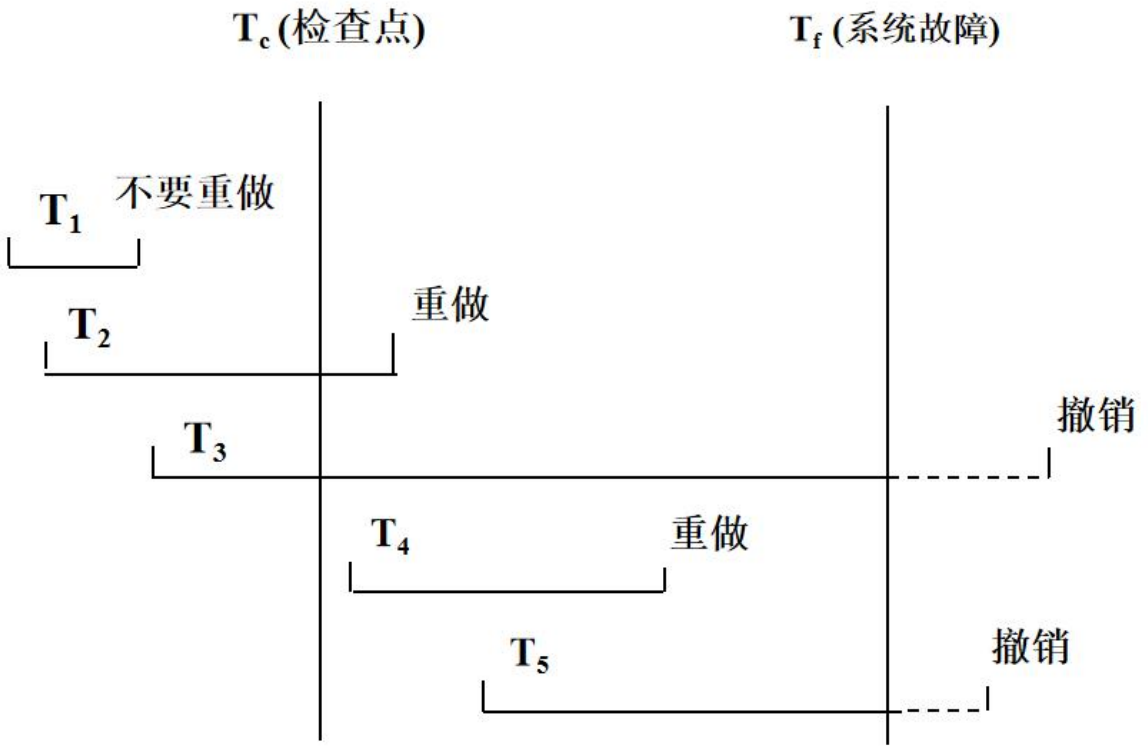
利用检查点的恢复策略 2

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



利用检查点的恢复策略 3

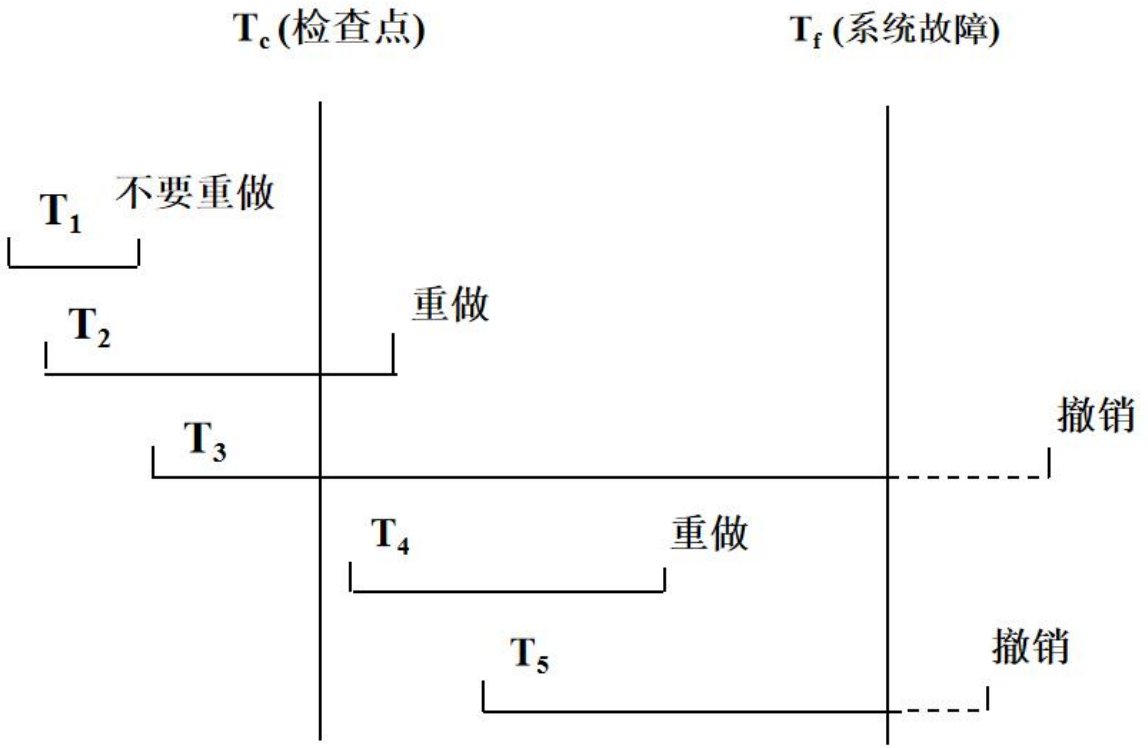
- 故障状态
  - T_1：在检查点之前提交
  - T_2：在检查点之前开始执行，在检查点之后故障点之前提交
  - T_3：在检查点之前开始执行，在故障点时还未完成
  - T_4：在检查点之后开始执行，在故障点之前提交
  - T_5：在检查点之后开始执行，在故障点时还未完成



利用检查点的恢复策略 4

❑ 恢复策略

- T_3和T_5在故障发生时还未完成，所以予以撤销
- T_2和T_4在检查点之后才提交，它们对数据库所做的修改在故障发生时可能还在缓冲区中，尚未写入数据库，所以要重做
- T_1在检查点之前已提交，所以不必执行重做操作



利用检查点的恢复步骤

- 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
- 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。
- 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
- 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

数据管理基础

数据库镜像

智能软件与工程学院

数据库镜像 1

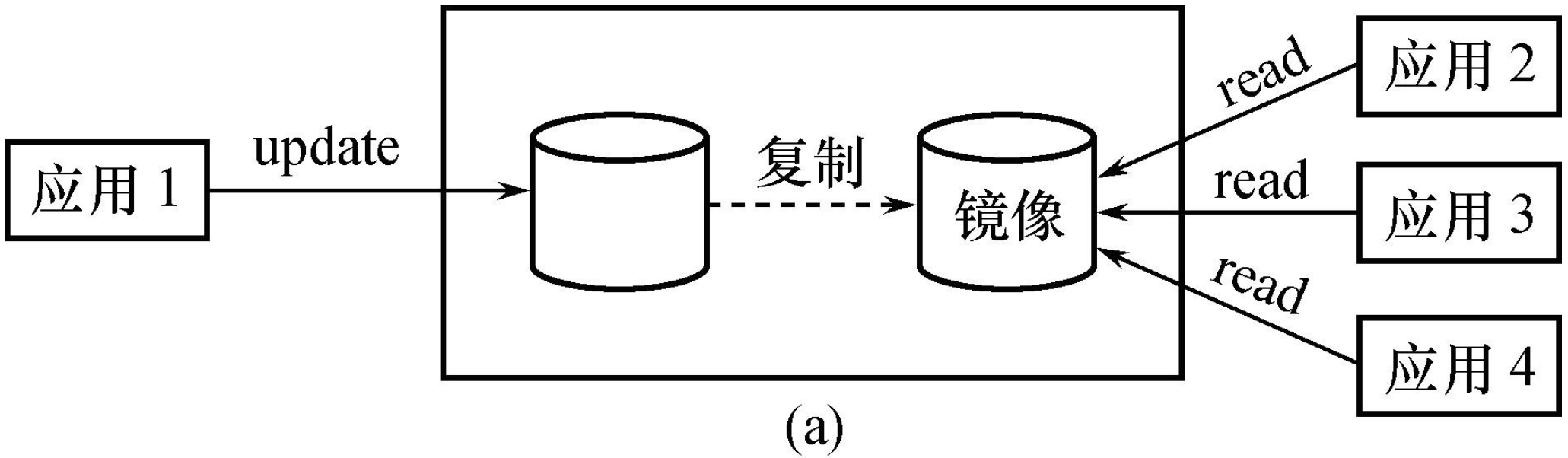
- 介质故障是对系统影响最为严重的一种故障，严重影响数据库的可用性
  - 介质故障恢复比较费时
  - 为预防介质故障，数据库管理员必须周期性地转储数据库

- 提高数据库可用性的解决方案
  - 数据库镜像（Mirror）

数据库镜像 2

❑ 数据库镜像

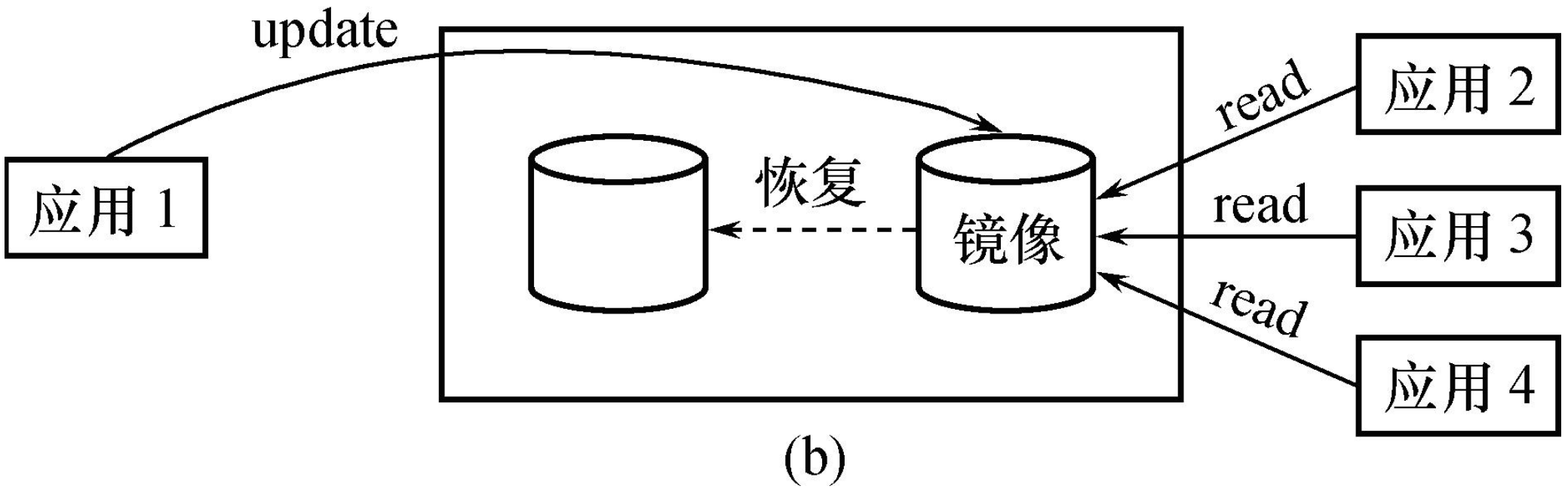
- 数据库管理系统自动把整个数据库或其中的关键数据复制到另一个磁盘上
- 数据库管理系统自动保证镜像数据与主数据的一致性
- 每当主数据库更新时，数据库管理系统自动把更新后的数据复制过去



(a)

数据库镜像的用途 1

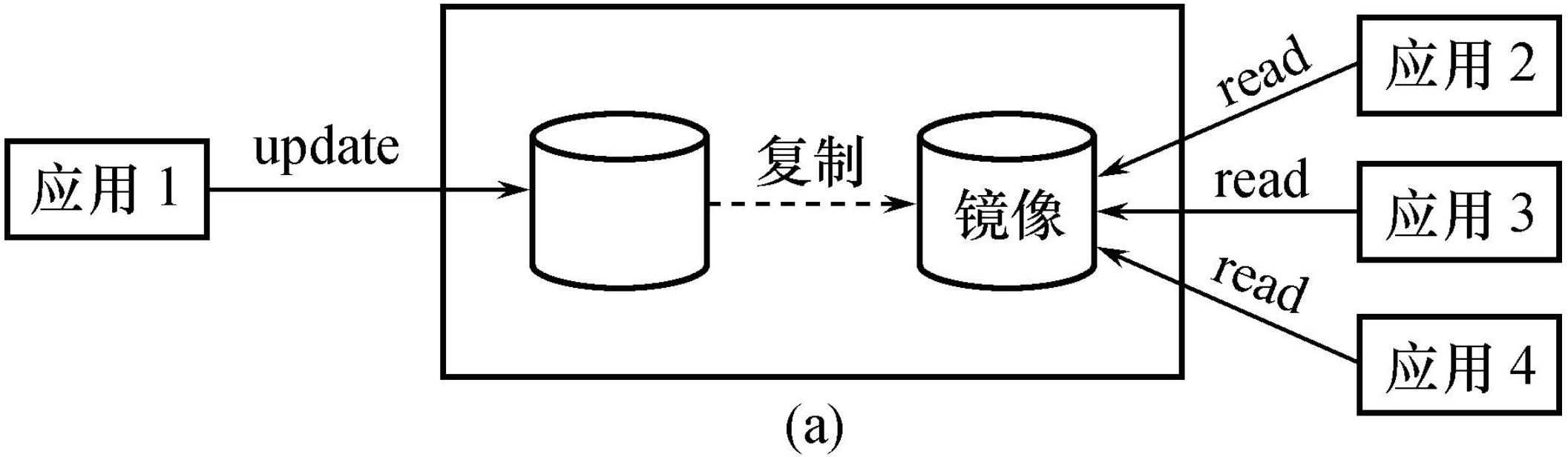
- 出现介质故障时
  - 可由镜像磁盘继续提供使用
  - 同时数据库管理系统自动利用镜像磁盘数据进行数据库的恢复
  - 不需要关闭系统和重装数据库副本



(b)

数据库镜像的用途 2

- 没有出现故障时
  - 可用于并发操作
  - 一个用户对数据加排他锁修改数据，其他用户可以读镜像数据库上的数据，而不必等待该用户释放锁



(a)

数据库镜像

- 频繁地复制数据自然会降低系统运行效率
  - 在实际应用中用户往往只选择对关键数据和日志文件镜像
  - 不是对整个数据库进行镜像

小结

- 事务的概念和性质
  - 事务是数据库的逻辑工作单位
  - 数据库管理系统保证系统中一切事务的原子性、一致性、隔离性和持续性，就保证了事务处于一致状态
- 故障的种类
  - 事务故障
  - 系统故障
  - 介质故障
- 恢复中最经常使用的技术
  - 数据库转储
  - 登记日志文件
- 恢复的基本原理
  - 利用存储在后备副本、日志文件和数据库镜像中的冗余数据来重建数据库
- 事务
  - 不仅是恢复的基本单位
  - 也是并发控制的基本单位