

# 数据库管理系统实现技术

# 数据库事务处理技术

## （故障恢复）

# 本讲学习什么？

## 基本内容

1. 数据库故障恢复的宏观思路
2. 运行日志及其检查点
3. 三种类型的运行日志
4. 利用运行日志进行故障恢复

## 重点与难点

- 理解三种类型的故障：事务故障、系统故障和介质故障
- 三种类型故障的恢复手段：运行日志和副本
- 理解检查点的作用
- 理解三种类型的运行日志及其故障恢复的操作方法：**Undo**型日志，**Redo**型日志，**Undo/Redo**型日志

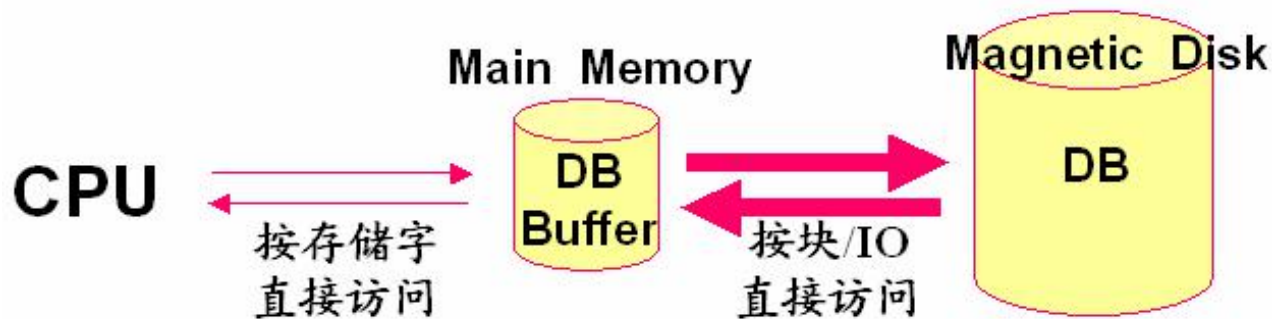
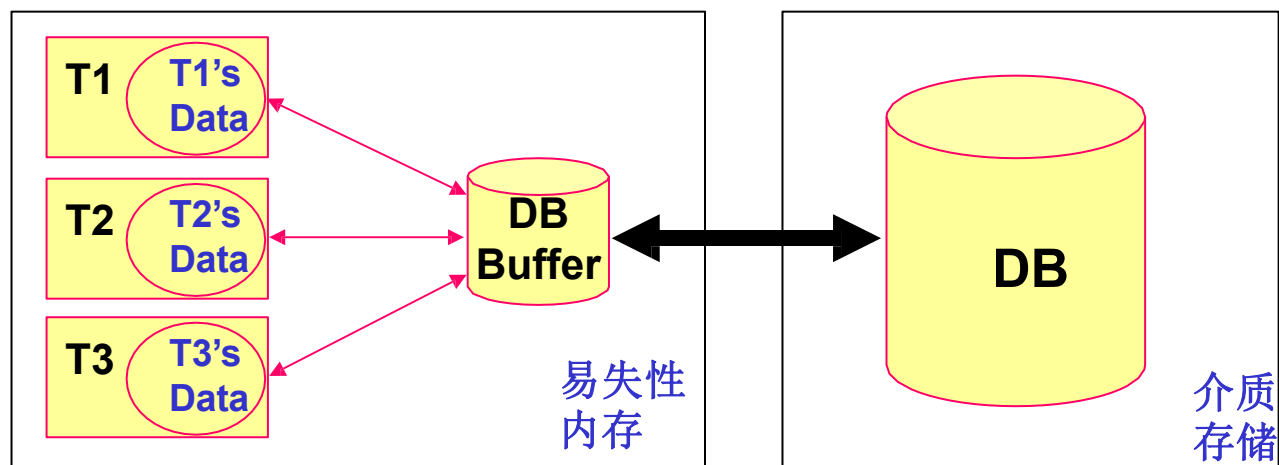
# 数据库的故障类型及其影响

# 数据库的故障类型及其影响

## (1)你要知道的

### DBMS的运行方式

- DBMS利用内存(主存)和外存(辅存)这样的存储体系来进行数据库管理
- 在内存中, 又将其分为程序数据(事务数据)和系统数据



# 数据库的故障类型及其影响

## (1)你要知道的

### 事务

- 事务是**DBMS**对数据库进行控制的基本逻辑单元。
- 事务：宏观上是由程序员设置的一条或多条**SQL**语句的一次执行；微观上是对数据元素的一系列基本操作，如读写等。需要**提交**和**撤销**。
- 数据元素：
  - 通常 **1 数据元素 = 1 磁盘块/内存页**
  - 也可以更小 (**=1 记录**)或更大 (**=1 关系**)
- 事务具有**四个**特性：**ACID**特性
  - 原子性**Atomicity**
  - 一致性**Consistency**
  - 隔离性**Isolation**
  - 持久性**Durability**
- 故障恢复涉及到如何保证**原子性**和**持久性**

# 数据库的故障类型及其影响

## (2)数据库故障类型

### 数据库的故障及其影响

#### □事务故障

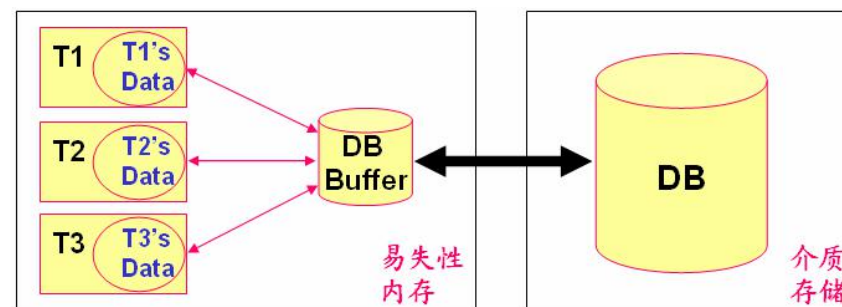
- ✓ 某一个程序(事务)自身运行错误所引起的故障
- ✓ 影响该程序(事务)本身

#### □系统故障

- ✓ 由于掉电、非正常关机所引起的故障
- ✓ 影响正在运行的事务以及数据库缓冲区, 数据库缓冲区将涉及正在运行和已经运行的事务

#### □介质故障

- ✓ 由于介质损坏等所引起的故障
- ✓ 影响是全面的, 既影响内存中的数据, 又影响介质中存储的数据



把DB由当前不正确状态恢复到已知为正确的某一状态

DBMS中故障恢复程序约占10%

# 数据库故障恢复的宏观思路



# 数据库故障恢复的宏观思路

## (1)故障恢复与事务故障恢复

### 数据库故障恢复

➤把**DB**由当前不正确状态恢复到已知为正确的某一状态。

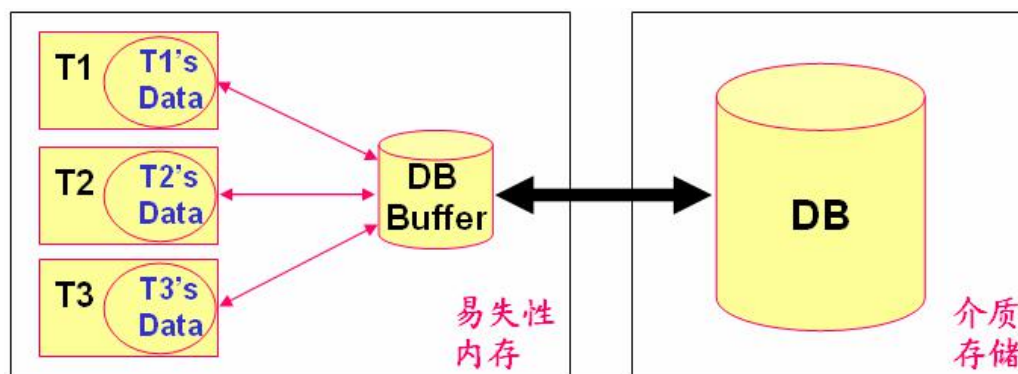
➤需要保证**事务**的：

✓**原子性**：事务的所有操作，要么全都执行，要么全都不执行。

✓**持久性**：已提交的事务对数据库产生的影响是持久的，未提交的事务对数据库不应有影响。

### 事务故障的恢复

➤事务故障可通过**重做事务(Redo)**和**撤销事务(Undo)**来恢复。重做事务可保证已提交事务的持久性，而撤销事务则消除未提交事务的影响



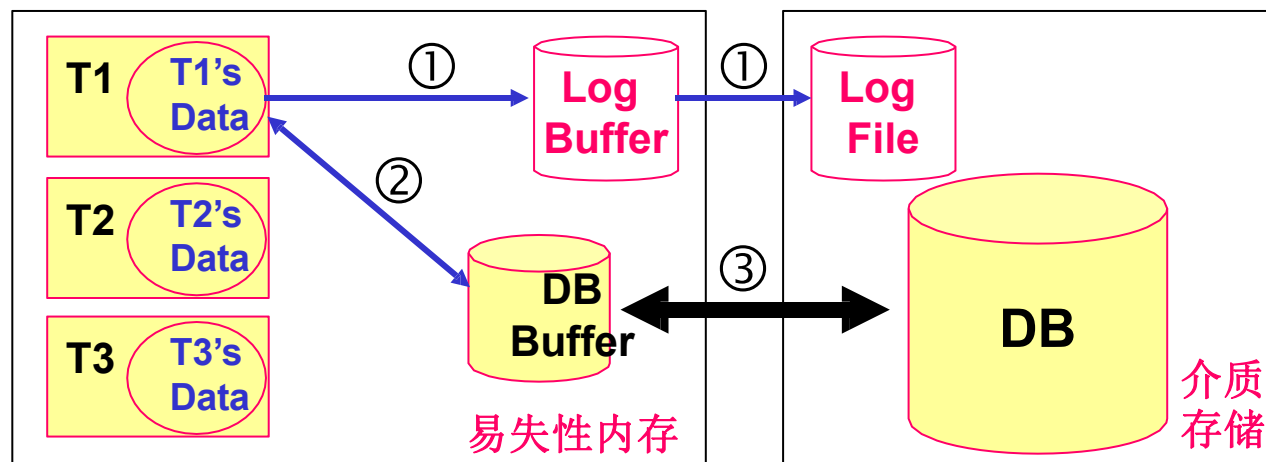
# 数据库故障恢复的宏观思路

## (2)系统故障恢复

### 系统故障恢复

#### ➤运行日志(System Log)

- 运行日志是**DBMS**维护的一个文件，该文件以流水方式记录了每一个事务对数据库的每一次操作及操作顺序
- 运行日志直接写入介质存储上，会保持正确性
- 当事务对数据库进行操作时：先写运行日志①；写成功后，再与数据库缓冲区进行信息交换②



# 数据库故障恢复的宏观思路

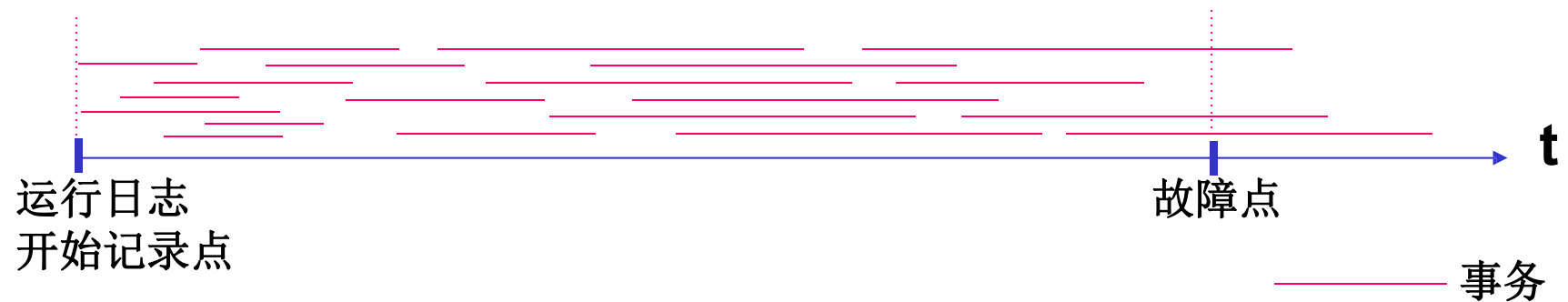
## (2)系统故障恢复

### ➤ 系统故障可通过运行日志来恢复

□ 按照运行日志记录的事务操作顺序重做事务(当事务在发生故障时已正确结束)  
或撤消事务(当事务在发生故障时未结束)

### ➤ 但故障恢复是需要时间的

□ 运行日志保留了若干天的记录，当发生系统故障时应从哪一个点开始恢复呢？



# 数据库故障恢复的宏观思路

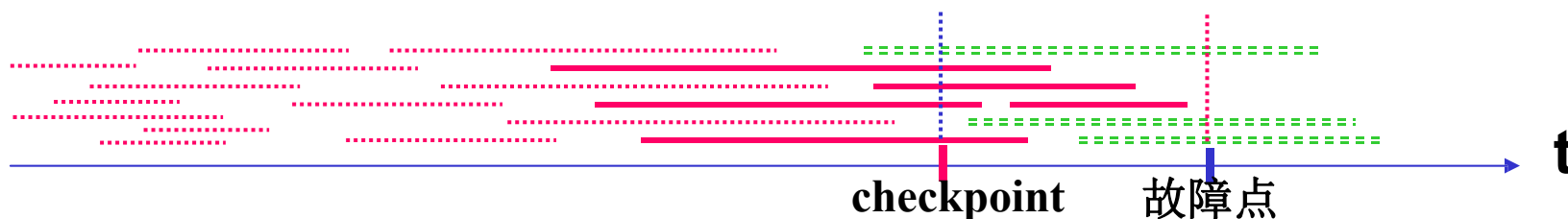
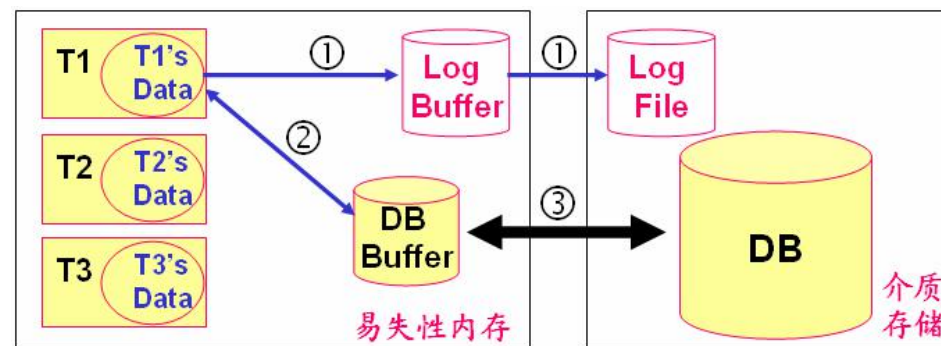
## (2)系统故障恢复

### ➤ DBMS在运行日志中定期的设置和更新**检查点(checkpoint)**

- 检查点是这样的时刻: 在该时刻, **DBMS**强制使内存**DB Buffer**中的内容与介质**DB**中的内容保持一致, 即将**DB Buffer**更新的所有内容写回**DB**中
- 检查点表征了: 在检查点之前内存中数据与介质中数据是保持一致的

### ➤ 系统故障的恢复

- 检查点之前结束的事务不需要恢复(已经写回**DB**)
- 检查点之后结束或发生的事务需 要依据运行日志进行恢复(不能确定 是否写回**DB**): 故障点前结束的重做, 故障点时刻未结束的撤消



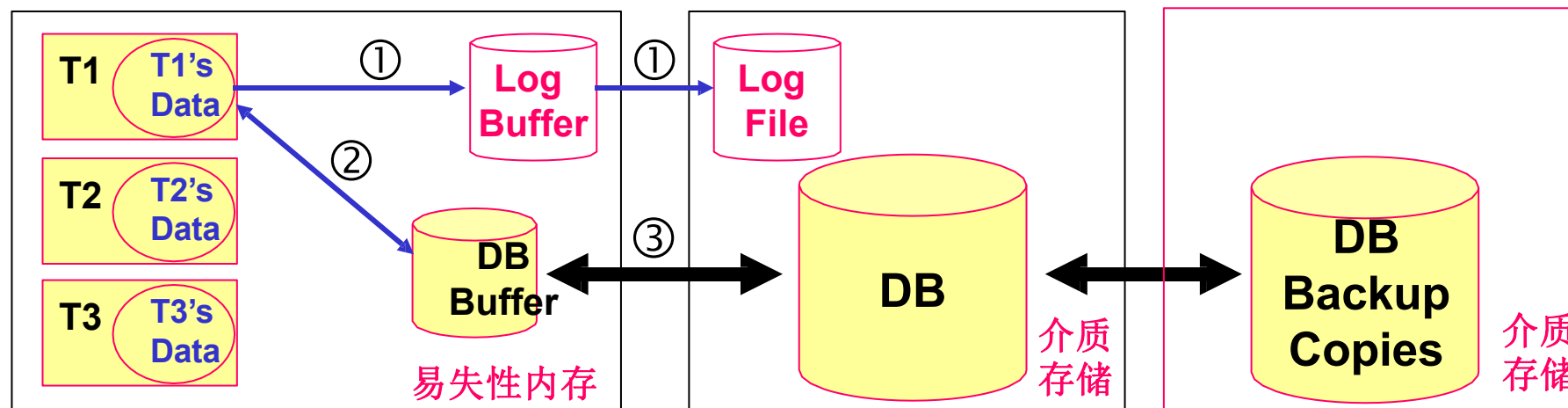
# 数据库故障恢复的宏观思路

## (3)介质故障恢复

### 介质故障恢复

#### ➤ 副本(Copy)

- 在某一时刻，对数据库在其他介质存储上产生的另一份等同记录
- 用副本替换被损坏的数据库



# 数据库故障恢复的宏观思路

## (3)介质故障恢复

### ➤ 介质故障的恢复

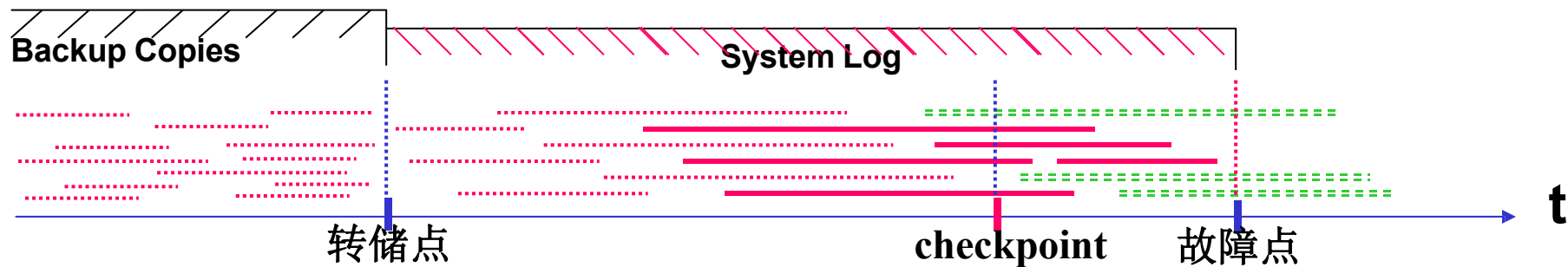
- 用副本替换被破坏的数据库

- 由于介质故障影响全面, 在用副本恢复后还需要依据运行日志进行恢复

### ➤ 如何确定备份的时刻: 转储点

- 过频, 影响系统工作效率; 过疏, 会造成运行日志过大, 也影响系统运行性能

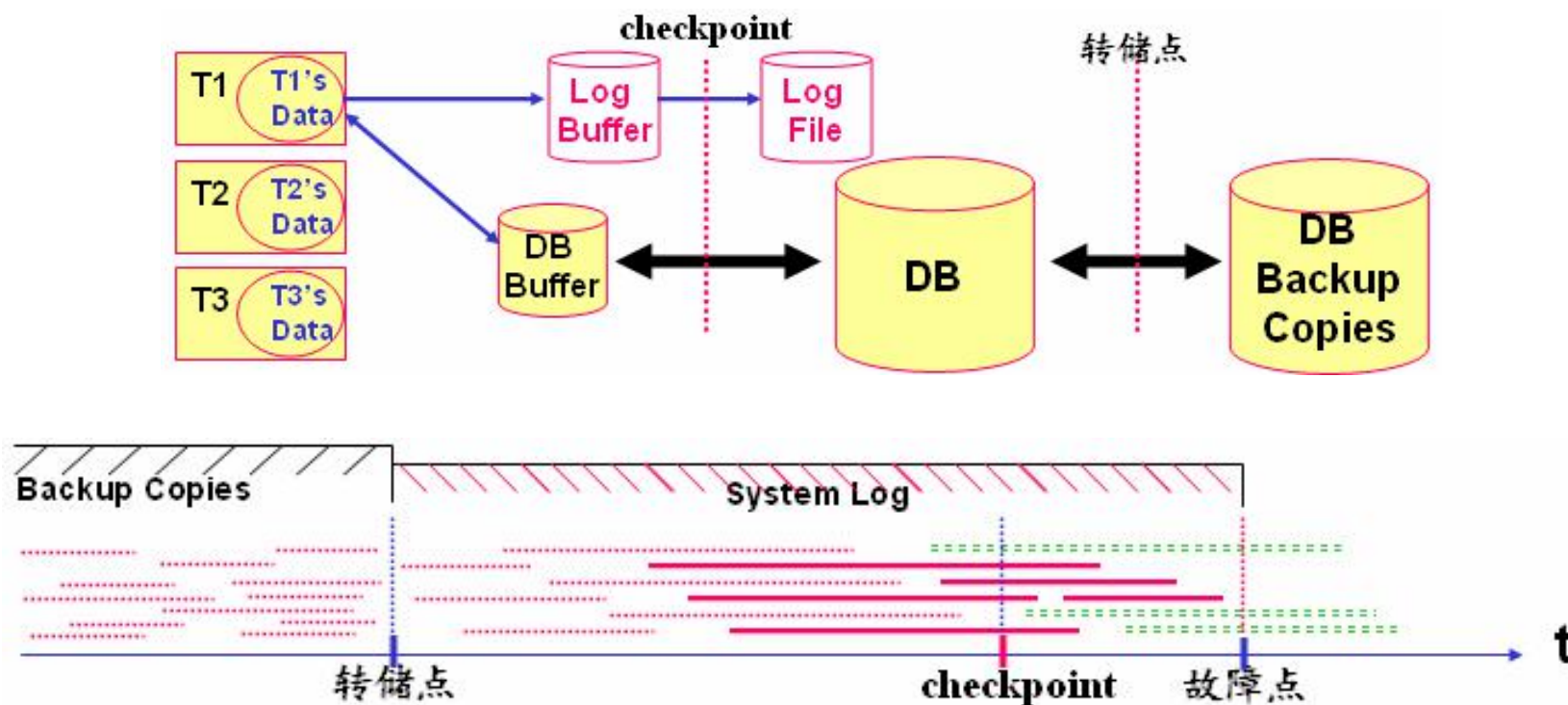
- 备份转储周期与运行日志的大小密切相关, 应注意防止衔接不畅而引起的漏洞



# 数据库故障恢复的宏观思路

## (4)小结

- 三种类型故障：事务故障、系统故障和介质故障
- 三种恢复手段：事务的撤消与重做, 运行日志和备份
- 两个重要时刻：检查点和转储点



什么是日志？



# 什么是日志?

## (1)事务涉及到的

### 数据库通常由元素构成

✓通常, **1 元素 = 1 磁盘块 = 1 内存页/块**

✓可以更小, **=1 记录** 或更大 **=1 关系**

### ➤每个事务都会读/写某些元素

✓**READ(X,t)**: 将元素X读到事务的局部变量t中

✓**WRITE(X,t)**: 将事务局部变量t写回元素X

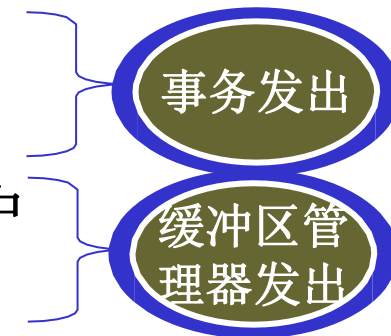
✓**INPUT(X)**: 将元素X从磁盘读入到内存缓冲区中

✓**OUTPUT(X)**: 将元素X写回到磁盘中

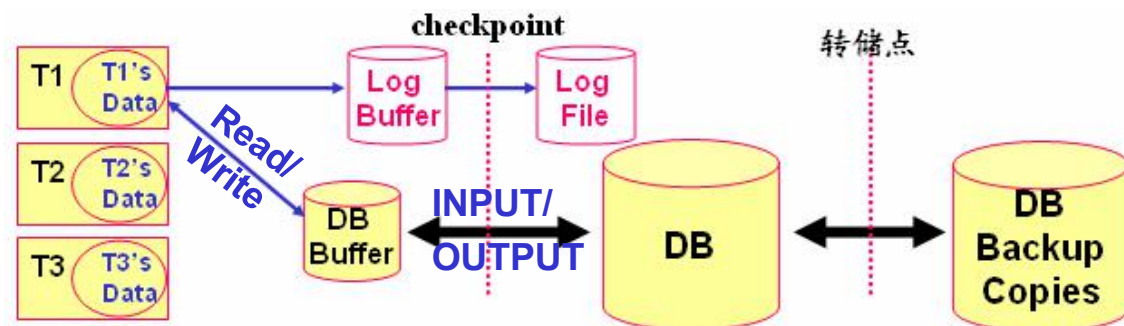
### ➤每个事务都以提交或者撤销结束

✓**COMMIT**: 事务提交

✓**ABORT**: 事务撤销



**Output(X)**是强制进行输出, 将缓冲区内容写回磁盘

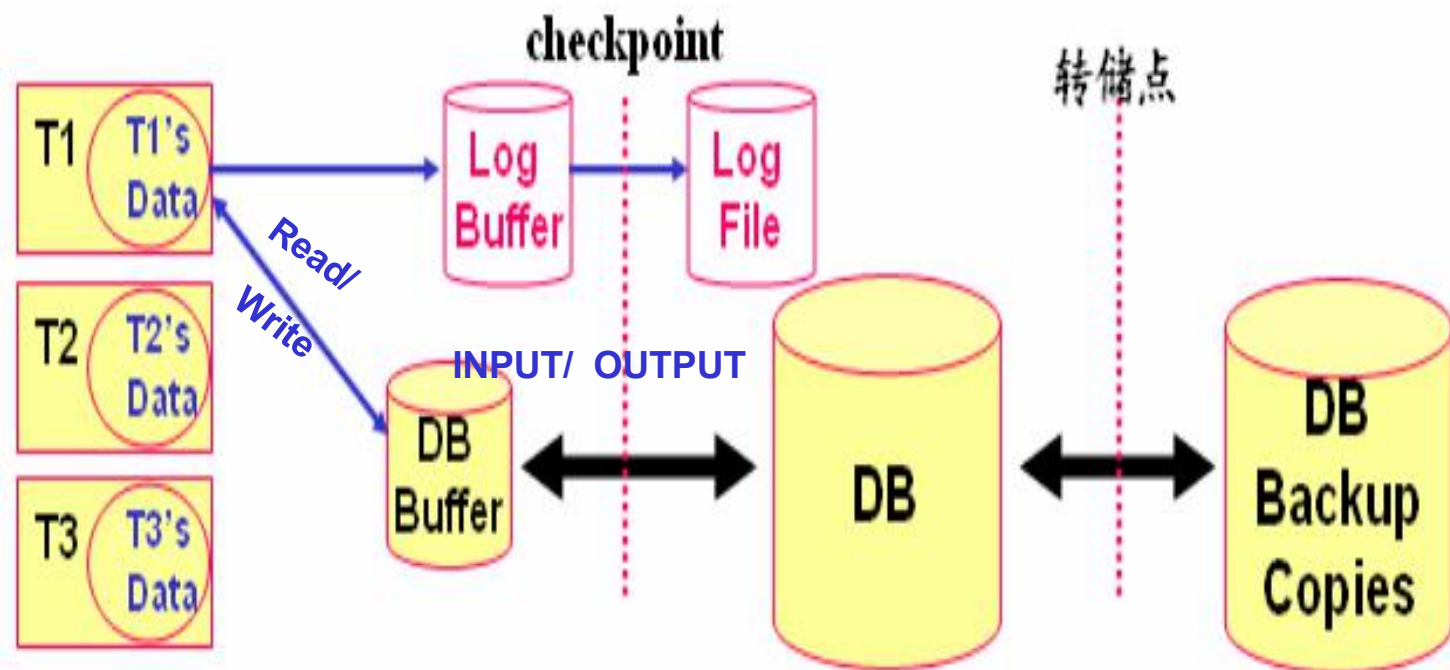


# 什么是日志？

## (1)事务涉及到的

### DBMS需要保证事务的：

- ✓**持久性**：已提交的事务对数据库产生的影响是持久的，未提交的事务对数据库不应有影响。
- ✓**原子性**：事务的所有操作，要么全都执行，要么全都不执行。



持久性：已提交事务--缓冲区内内容保证写回磁盘  
未提交事务--缓冲区内内容不能影响磁盘

什么是日志？

(2)不同的缓冲区策略会影响事务的持久性

## 缓冲区处理策略

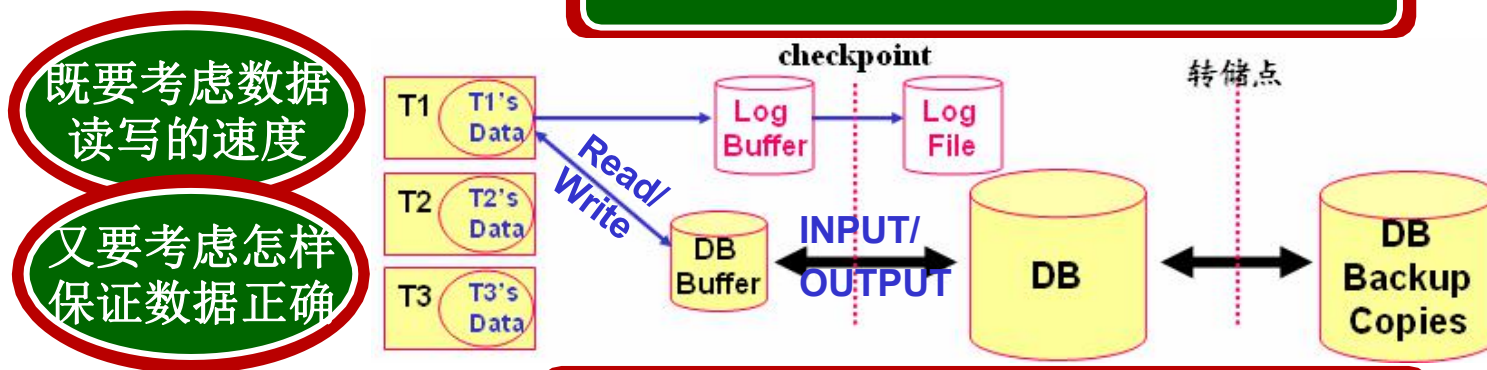
**Force:** 内存中的数据最晚在**commit**的时候写入磁盘。

**No steal:** 不允许在事务**commit**之前把内存中的数据写入磁盘。

**No force:** 内存中的数据可以一直保留，在**commit**之后过一段时间再写入磁盘。(此时在系统崩溃的时候可能还没写入到磁盘，需要**Redo**)。--灵活

**Steal:** 允许在事务**commit**之前把内存中的数据写入磁盘。(此时若系统在**commit**之前崩溃时，已经有数据写入到磁盘了，要恢复到崩溃前的状态，需要**Undo**)。--灵活

当前最常用的: **Steal+No force**



缓冲区内内容不一定和磁盘内容一致哟

什么是日志?

(3)事务故障会影响事务的原子性

**Begin TRANSACTION**

READ(A,t);

$t := t * 2;$

WRITE(A,t);

READ(B,t);

$t := t * 2;$

WRITE(B,t)

COMMIT;

**End TRANSACTION**

原子性:  
A和B同时乘以2

DBMS如何  
保证呢?

什么是日志？

(3)事务故障会影响事务的原子性

Action	事务	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A)	8	8		8	8
<del>t=t*2</del>	16	8		8	8
WRITE(A)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B)	8	16	8	8	8
<del>t=t*2</del>	16	16	8	8	8
WRITE(B)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

什么是日志?

(3)事务故障会影响事务的原子性

原子性: A和B同时乘以2, 是否受影响?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)	16	16	16	8	8
OUTPUT(A)	16	16	16	16	故障发生于 Output(A)后 Output(B)前
OUTPUT(B)	16	16	16	16	

# 什么是日志?

## (4)怎样记录日志?

### 日志

- 一个包含日志记录的只能追加的顺序文件, 不同事务的日志记录交错存储, 按发生时间存储
- 发生系统故障时, 使用日志进行恢复:
  - 故障时已提交的事务, 重做(Redo)
  - 故障时未提交的事务, 撤销(Undo)
- 日志记录的信息
  - ✓**<Start T>**, 表示事务T已经开始
  - ✓**<Commit T>**, 表示事务T成功完成
  - ✓**<Abort T>**, 事务T未成功, 被中止
  - ✓**<T, X, v<sub>1</sub>>** 或者 **<T, X, v<sub>2</sub>>** 或者 **<T, X, v<sub>1</sub>, v<sub>2</sub>>**  
表示事务T改变了数据库元素X, X原来的值为v<sub>1</sub>(X的旧值), X新的值为v<sub>2</sub>.
- 三种日志: **Undo型日志**, **Redo型日志**, **Undo/Redo型日志**

记录内容和记录次序不同, 恢复策略也不同

什么是日志?

(4)怎样记录日志?

## 缓冲区处理策略与日志/恢复策略的关系

	No Steal	Steal
No Force		最快
Force	最慢	

读写性能

	No Steal	Steal
No Force	只需Redo 无需Undo	需要Redo 需要Undo
Force	无需Redo 无需Undo	无需Redo 只需Undo

日志/恢复策略



什么是日志?

(5)我们将继续学习...

## 日志

一个包含日志记录的、只能追加的顺序文件, 不同事务的日志记录交错存储, 按发生时间存储。

➤发生系统故障时, 使用日志进行恢复:

- 故障时已提交的事务, 重做(Redo)
- 故障时未提交的事务, 撤销(Undo)



**Undo型日志及其故障恢复？**

# Undo型日志及其故障恢复?

## (1)问题

### 日志

一个包含日志记录的、只能追加的顺序文件, 不同事务的日志记录交错存储, 按发生时间存储。

➤发生系统故障时, 使用日志进行恢复:

□故障时已提交的事务, 重做(Redo)

□故障时未提交的事务, 撤销(Undo)



- 如何记录日志文件, 记录什么?
- 如何设置检查点?
- 如何依据日志文件进行故障恢复

## Undo型日志及其故障恢复?

### (2)Undo型日志的日志记录规则

## Undo型日志

➤对于任一事务T，按下列**顺序**向磁盘输出T的日志信息：


□首先，**<T, X, v>**被写到日志中

□其次，**OUTPUT(X)**

□最后，**<COMMIT T>**或**<ABORT T>**被写到日志中

➤注意：Undo型日志仅保留旧值。**<T, X, v>**，v为X原来的值(X的旧值)

➤Undo型日志：“将事务改变的所有数据写到磁盘前不能提交该事务”



日志文件中  
记录什么，  
什么次序呢？

## Undo型日志及其故障恢复?

### (2)Undo型日志的日志记录规则

#### 示例: Undo型日志

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

## Undo型日志及其故障恢复?

### (3)利用Undo型日志进行故障恢复

## 利用undo型日志进行恢复

➤首先，确定每一个事务是否已完成?

□ <START T>....<COMMIT T>.... = yes

□ <START T>....<ABORT T>..... = no(已结束，但未完成)

□ <START T>..... = no

➤然后，从日志的尾部开始按日志记录的反序，处理每一日志记录，撤销未完成事务的所有修改

□ <COMMIT T>: 标记T已完成

□ <ABORT T>: 标记T已结束但未完成

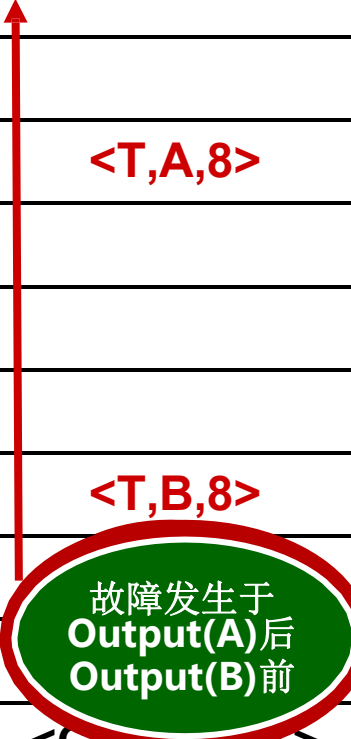
□ <T,X,v>: 如果T未完成，则将X=v写回磁盘；否则跳过；

□ <START T>: 跳过

## Undo型日志及其故障恢复?

### (3)利用Undo型日志进行故障恢复

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	--	8	
READ(B,t)	8	16	8	--	8	
t:=t*2	16	16	8	--	8	
WRITE(B,t)	16	16	16	--	8	<T,B,8>
OUTPUT(A)	16	16	16	--	--	
OUTPUT(B)	16	16	16	--	--	
COMMIT						<COMMIT T>



因事务未提交。通过日志恢复A=8, B=8, 保证了事务的原子性

## Undo型日志及其故障恢复?

### (3)利用Undo型日志进行故障恢复

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
INPUT(A)		8		8	8	
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8>
INPUT(B)	16	16	8	8	8	
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

因事务已提交。无需做任何事情，已保证了事务的原子性。

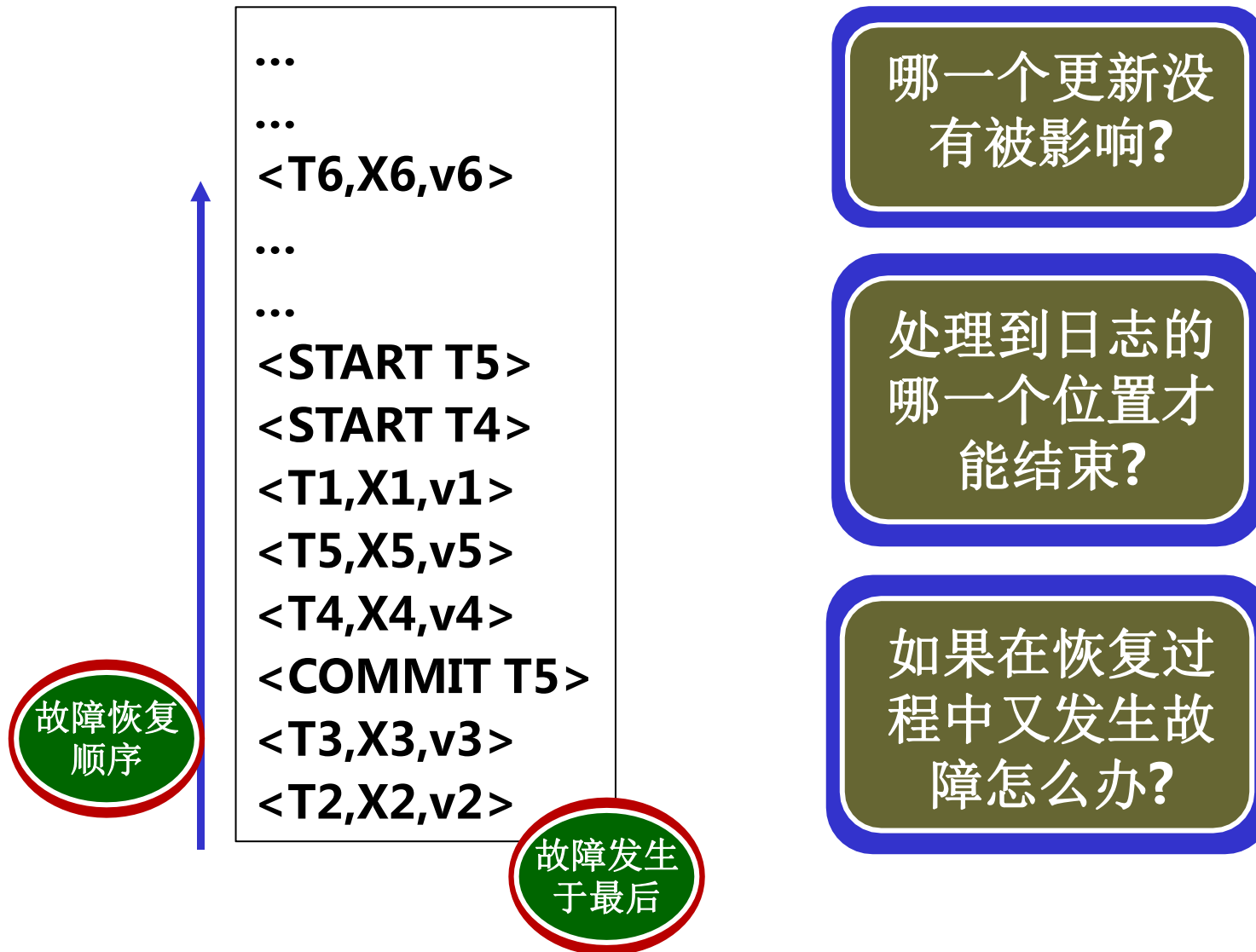
故障发生于  
COMMIT T后



# Undo型日志及其故障恢复?

## (4)检查点及其使用

为什么需要检查点?



### 检查点

#### ➤ 静止检查点：周期性地对日志设置检查点

- 停止接受新的事务, 等到所有当前活跃事务提交或终止, 并在日志中写入了**COMMIT**或**ABORT**记录后
- 将日志刷新到磁盘, 写入日志记录**<CKPT>**, 并再次刷新日志

#### ➤ 非静止检查点

- 在设置检查点时不必关闭系统, 允许新事务进入
- 写入一条**<START CKPT(T1,...,Tk)>** 其中**T1,...,Tk** 是所有活跃的未结束的事务
- 继续正常的操作, 直到**T1,...,Tk**都完成时, 写入**<END CKPT>**

# Undo型日志及其故障恢复?

## (4)检查点及其使用

故障需恢复到所遇到的第一个检查点<CKPT>

...  
...  
<T9,X9,v9>  
...  
...  
**(all completed)**  
**<CKPT>**  
<START T2>  
<START T3>  
<START T5>  
<START T4>  
<T1,X1,v1>  
<T5,X5,v5>  
<T4,X4,v4>  
<COMMIT T5>  
<T3,X3,v3>  
<T2,X2,v2>

other transactions

transactions T2,T3,T4,T5

# Undo型日志及其故障恢复?

## (4)检查点及其使用

故障需恢复到所遇到的第一个检查点<CKPT>

...  
...  
...  
...  
...  
...

<START CKPT T4, T5, T6>

...  
...  
...  
...

<END CKPT>

...  
...  
...

earlier transactions  
Plus T4, T5, T6

T4, T5, T6, plus  
later transactions

later transactions

**Redo型日志及其故障恢复？**

# Redo型日志及其故障恢复?

## (1)问题

### 日志

一个包含日志记录的、只能追加的顺序文件, 不同事务的日志记录交错存储, 按发生时间存储。

➤发生系统故障时, 使用日志进行恢复:

- 故障时已提交的事务, 重做(Redo)
- 故障时未提交的事务, 撤销(Undo)



- 如何记录日志文件, 记录什么?
- 如何设置检查点?
- 如何依据日志文件进行故障恢复

## Redo型日志

➤Undo型日志的问题 “将事务改变的所有数据写到磁盘前不能提交该事务” —如何解决?

➤对于任一事务T，按下列顺序向磁盘输出T的日志信息:


□首先， $\langle T, X, v \rangle$ 被写到日志中

□其次， $\langle \text{COMMIT } T \rangle$ 被写到日志中

□最后，**OUTPUT(X)**

➤注意: redo型日志保留新值。 $\langle T, X, v \rangle$ ，v为X更新后的值(X的新值)

➤注意: 与undo型的差别，在后两步，先写提交记录后输出，还是先输出，再写提交记录。



日志文件中  
记录什么，  
什么次序呢?

## Redo型日志及其故障恢复?

### (2)Redo型日志的日志记录规则

#### 示例: Redo型日志

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	



## 利用redo日志进行恢复

### ➤确定每一个事务是否已完成?

- <START T>....<COMMIT T>.... = yes
- <START T>....<ABORT T>.....=no(已结束, 但未完成)
- <START T>..... = no

### ➤从日志的起始位置开始按日志记录的正序处理每一日志记录, 重做已提交事务的所有修改:

- <COMMIT T>: 标记T已完成
- <ABORT T>: 标记T已结束但未完成
- <T,X,v>: 如果T已完成, 则将X=v写回磁盘; 否则跳过;
- <START T>: 跳过

## Redo型日志及其故障恢复?

### (3)利用Redo型日志进行故障恢复

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,16>
						故障发生于 COMMIT T前
OUTPUT(A)	16	16	16	--	--	
OUTPUT(B)	16	16	16	--	--	

因事务未提交。原始的A=8, B=8, 并未被改动。<T,A,16>等被跳过

## Redo型日志及其故障恢复?

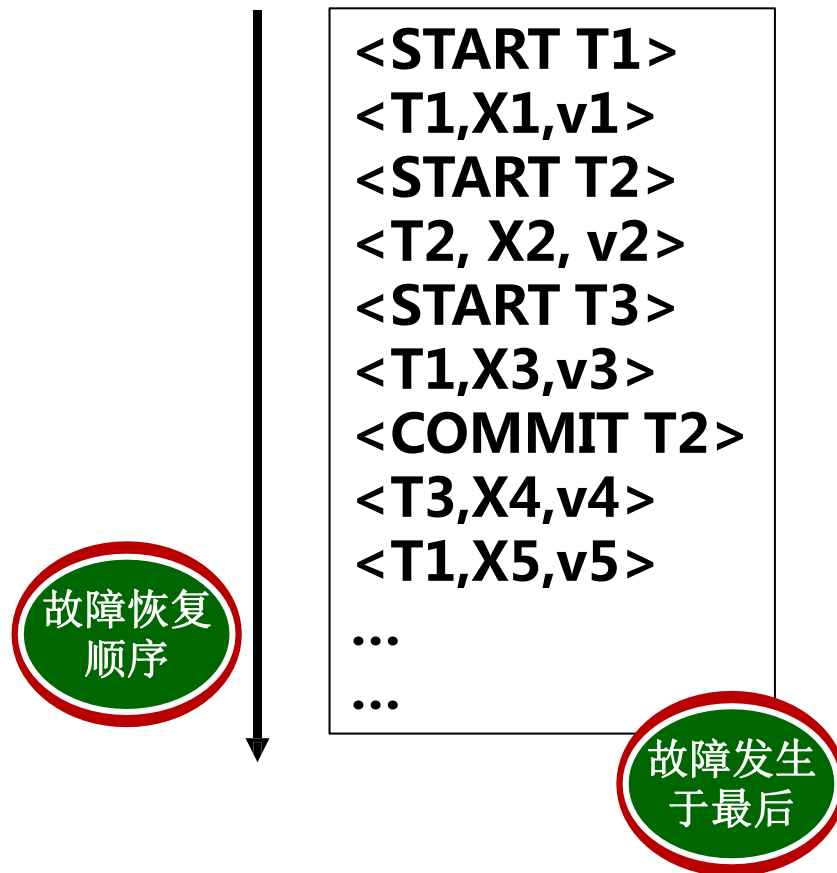
### (3)利用Redo型日志进行故障恢复

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
READ(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		16	8	<T,A,16>
READ(B,t)	8	16	8	16	8	
t:=t*2	16	16	8	16	8	
WRITE(B,t)	16	16	16	16	16	<T,B,16>
						<COMMIT T>
OUTPUT(A)	16	16	16	--	--	故障发生于 COMMIT T后
OUTPUT(B)	16	16	16	--	--	

因事务已提交。按次序用<T,A,16> <T,B,16>等的新值更新数据库

## Redo型日志及其故障恢复?

### (4)检查点及其运用



哪一个更新没有被影响?

都从日志开始处处理吗?

如果在恢复过程中又发生故障怎么办?

# 检查点

## ➤非静止检查点

- 在进行检查点设置时不必关闭系统，允许新事务进入
- 写入一条**<START CKPT(T1,...,Tk)>** 其中**T1,...,Tk** 是所有活跃的未结束的事务
- 将所有已提交的事务写回磁盘，
- 继续正常的操作，直到**T1,...,Tk**都完成时，写入**<END CKPT>**

## Redo型日志及其故障恢复?

### (4)检查点及其运用

**Step1:**寻找到最后的  
<END CKPT>

```
...  
<START T1>  
...  
<COMMIT T1>  
...  
<START T4>  
...  
<START CKPT T4, T5, T6>  
...  
...  
...  
...  
<END CKPT>  
...  
...  
...  
<START CKPT T9, T10>  
...
```

**Step2:**从T4,T5,T6的  
最早开始处恢复起。  
忽略更早提交的事务

**Undo/Redo结合型日志及其故障恢复？**

# Undo/Redo结合型日志及其故障恢复?

## (1)问题

### 日志

一个包含日志记录的、只能追加的顺序文件, 不同事务的日志记录交错存储, 按发生时间存储。

➤发生系统故障时, 使用日志进行恢复:

□故障时已提交的事务, 重做(Redo)

□故障时未提交的事务, 撤销(Undo)



- 如何记录日志文件, 记录什么?
- 如何设置检查点?
- 如何依据日志文件进行故障恢复



# Redo型日志与Undo型日志的比较

## ➤Undo型日志:

- OUTPUT必须先做。

- 如果<COMMIT T>可见, T确定地已将其数据写回磁盘, 因此不必重做 --- 但可能引起性能下降(因可能频繁地写磁盘)

## ➤Redo型日志:

- OUTPUT必须后做。

- 如果<COMMIT T>不可见, T确定地没有将其任何数据写回到磁盘, 因此无需撤销 --- 但灵活性差(数据必须在Commit后才可见)

## ➤如更喜欢灵活性 -- Undo/Redo型日志

## Undo/Redo型日志

➤对于任一事务T，按下列顺序向磁盘输出T的日志信息：


□第(1)步， $\langle T, X, u, v \rangle$ 被写到日志中

□第(2)or(3)步， $\langle \text{COMMIT } T \rangle$ 被写到日志中

□第(3)or(2)步， $\text{OUTPUT}(X)$

➤注意：undo/redo型日志既保留**新值v**，也保留**旧值u**。

➤注意：与undo型和redo型的差别，在后两步。**Redo**型是先写提交记录后输出；**undo**型是先输出，再写提交记录；**undo/redo**型则无所谓谁先谁后，只要保证 $\langle T, X, u, v \rangle$ 被先于**OUTPUT**写完即可。



日志文件中  
记录什么，  
什么次序呢?

## Undo/Redo结合型日志及其故障恢复?

### (2)Undo/Redo型日志的日志记录规则

示例: **Undo/Redo**型日志

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8,16>
READ(B,t)	8	16	8	8	8	
t:=t*2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
OUTPUT(A)	16	16	16	16	8	
						<COMMIT T>
OUTPUT(B)	16	16	16	16	16	

**OUTPUT**可以按需要在**COMMIT**之前或之后

## Undo/Redo结合型日志及其故障恢复?

### (3)利用Undo/Redo型日志进行故障恢复

## 利用undo/Redo型日志进行恢复

➤首先，确定每一个事务是否已完成?

- <START T>....<COMMIT T>.... = yes
- <START T>....<ABORT T>..... = no(已结束，但未完成)
- <START T>..... = no

➤自前向后地，按日志记录的正序，重做所有已提交的事务；自后向前，按日志记录的反序，撤销所有未完成事务的所有修改。

- <COMMIT T>: 标记T已完成
- <ABORT T>: 标记T已结束但未完成
- <T,X,u,v>: 如果T未完成，则将X=u写回磁盘；否则将x=v写回磁盘；
- <START T>: 跳过

# Undo/Redo结合型日志及其故障恢复?

## (3)利用Undo/Redo型日志进行故障恢复

战德臣 教授

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		8	8	
t:=t*2	16	8		8	8	
WRITE(A,t)	16	16		8	8	<T,A,8,16>
READ(B,t)	8	16	8	--	8	
t:=t*2	16	16	8	--	8	
WRITE(B,t)	16	16	16	--	8	<T,B,8,16>
OUTPUT(A)	16	16	16	--	--	
						故障发生于 COMMIT T前
OUTPUT(B)	16	16	16	--	--	

故障发生在COMMIT之前，按反序进行撤销修改。即写回旧值

## Undo/Redo结合型日志及其故障恢复?

### (3)利用Undo/Redo型日志进行故障恢复

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						<START T>
REAT(A,t)	8	8		--	--	
t:=t*2	16	8		--	--	
WRITE(A,t)	16	16		16	--	<T,A,8,16>
READ(B,t)	8	16	8	16	--	
t:=t*2	16	16	8	16	--	
WRITE(B,t)	16	16	16	16	16	<T,B,8,16>
OUTPUT(A)	16	16	16	16	16	
						<COMMIT T>
OUTPUT(B)	16	16	16	16	16	

故障发生于  
COMMIT 后

故障发生在COMMIT之后，按正序进行重做修改。即写回新值

## Undo/Redo结合型日志及其故障恢复?

### (3)利用Undo/Redo型日志进行故障恢复

**<START T1>**  
**<T1,X1,v0,v1>**  
**<START T2>**  
**<T2, X1, v1, v2>**  
**<T2, X2, k0, k1>**  
**<START T3>**  
**<T1,X3,m0,m1>**  
**<COMMIT T2>**  
**<COMMIT T1>**  
**<T3,X1,v2,v3>**  
**<T3,X3,m1,m2>**  
**<ABORT T3>**  
...  
...

自后向前地撤销所有未提交的事务；自前向后地重做所有已提交的事务；先做<撤销>，再做<重做>

**X 1 = v 2**  
**X 2 = k 1**  
**X 3 = m 1**

也需考虑检查点的设置，同学自我学习

回顾本讲学习了什么？



# 回顾本讲学习了什么？

