高级数据库系统及其应用

第2部分　关系数据库系统实现

第9章 数据库恢复技术

xshxie@ustc.edu.cn

LOGO

第9章 数据库恢复技术

9.1 DB恢复问题及其处理概述

9.1.1 故障类型

9.1.2 故障恢复策略

9.1.3 数据存取的有关概念

9.1.4 事务写操作相关问题

9.1.1 DB故障类型

- 事务故障
- 系统崩溃
- 磁盘失败（损坏）
- 灾难性失败

9.1.2 DB故障恢复策略

1. 灾难性或磁盘失败恢复

- 可用的保障性技术
  - 各种磁盘校验技术、RAID技术；
  - 周期性数据转储备份，包括远程或异地备份。

2. 系统故障的恢复

- 系统故障会造成DB不一致状态
- 恢复方法
  - 撤销(UNDO) 故障发生时未完成事务对DB的所有影响，确保事务的原子性。
  - 重做(REDO)已成功提交的已完成事务，实现事务持久性

一般由系统在重新启动时自动完成，不需要用户干预。

恢复管理器(Recovery Manager)

负责提供事务的以下两个特性:

原子性 (Atomicity) :

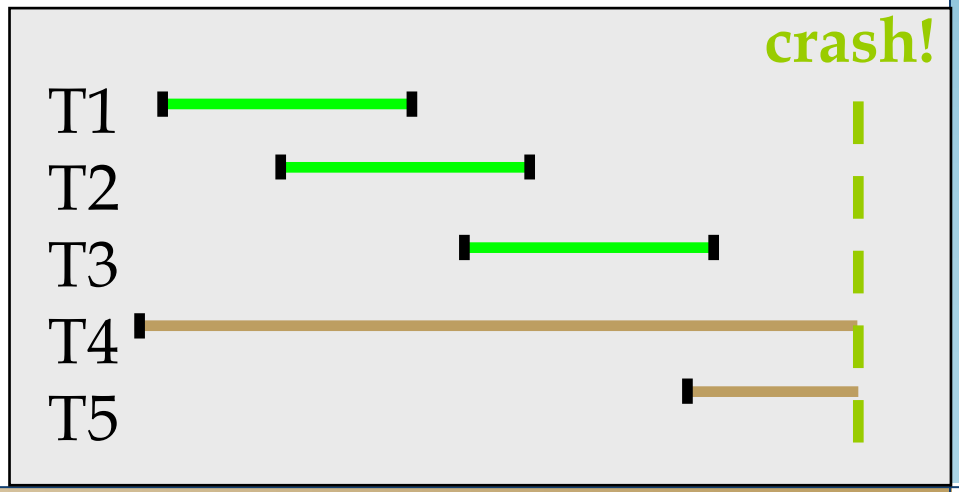
- 当事务因执行出错或死锁，被中止(abort)时，事务必须回滚(rollback)到执行起始点或一个保存点(savepoint).

持久性 (Durability) :

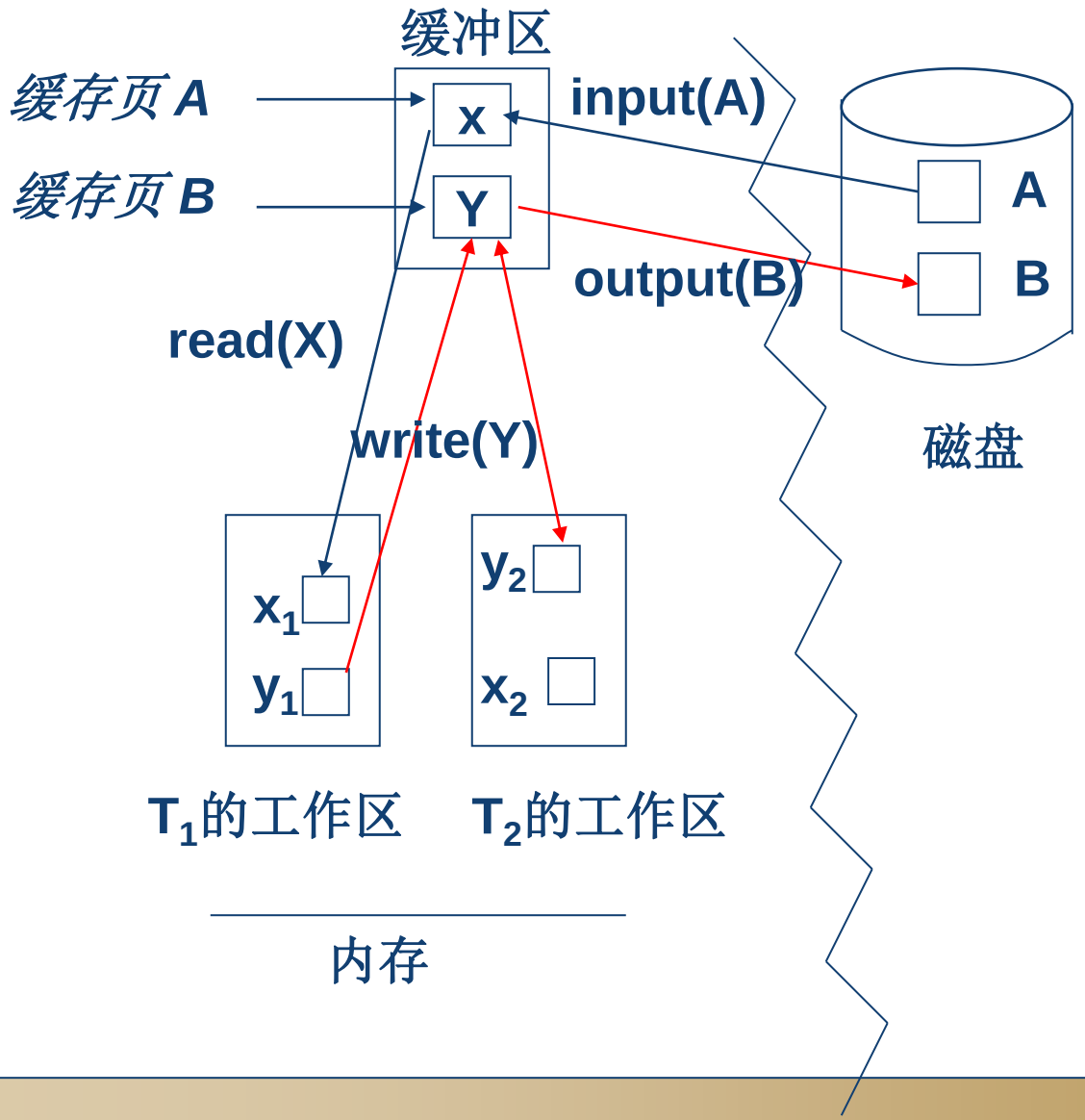
- 已提交事务对DB造成的影响，必须具有持久性。

系统崩溃重启后，我们希望的正确行为:

- 已提交事务T_1，T_2 & T_3 应是持久的
- 中止事务T_4 & T_5 的影响，应被撤销



9.1.3 数据存取的有关概念



9.1.4 事务写操作相关问题

1. 写缓存页到磁盘问题

- 只有事务写操作才存在需要恢复问题。出于简单性，本章讨论时，我们采用如下两点假设：
  - 写一个缓存页到磁盘的动作是原子的，即DBMS不会写不完整页到磁盘 .
  - 在恢复机制中，总使用基于封锁的调度协议。特别地，在页面级封锁时采用Strict 2PL封锁协议。
- 当一个页输出到磁盘时，其上不能有正在进行的更新。下列方法可以确保这一点：
  - 修改数据元素之前，事务在包含数据元素的缓存页上获得排它锁；一旦完成写，锁即可释放。这种持有时间很短的锁称为闩(latch)。

2. 与缓冲池管理有关的两个额外问题

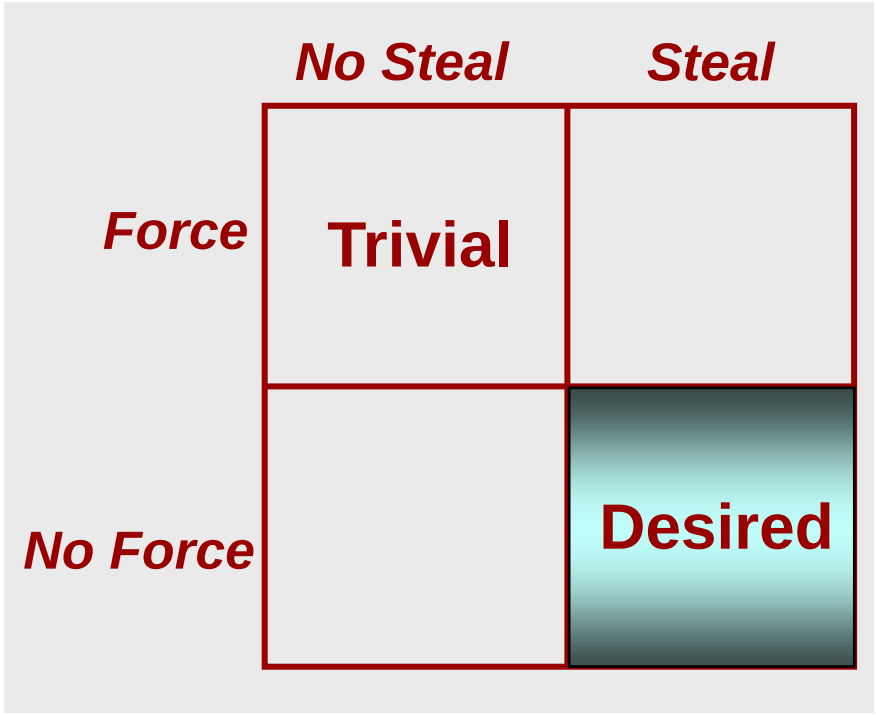
9.1.4 事务写操作相关问题

1. 写缓存页到磁盘问题

2. 与缓冲池管理有关的两个额外问题

- 当需要新页时，如果缓冲池已满，则需要置换一个缓存页，将一个现有页移出缓冲池。
- 如果所选的移出页被更新过，则必须将它输出到磁盘。

- 是否允许‘窃用’未提交事务修改页占用的缓存页框？
  - If not, poor throughput
  - If so, how can we ensure atomicity?
- 强制(Force)写已提交事务所有修改缓存页到磁盘?
  - Provides durability
  - But poor response time



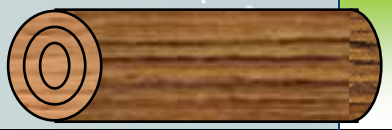
9.2 数据库日志

9.2.1 日志技术概述

9.2.2 基于WAL规则的日志

9.2.3 检查点技术

9.2 DB日志(LOG)



- 是一种记录**DBMS**已执行动作历史的技术，负责为每个“更新”动作做记录，为可能的重做/撤销准备必要的信息
  - 是一种记录型文件，记录有唯一的标识号（LSN, log sequence number，单调增数）。采用APPEND模式进行顺序写，但也允许随机指定LSN来读记录。
  - 除了日志记录集外，日志中还通常有一条主记录（文件头），存储一些全局性信息。
  - 物理上，日志必须是一种“稳定存储”的记录型电子文档。通常有多个位于不同磁盘位置的副本。
- 常规**LOGREC**内容:
  - <XID, pageID, offset, length, old data, new data>

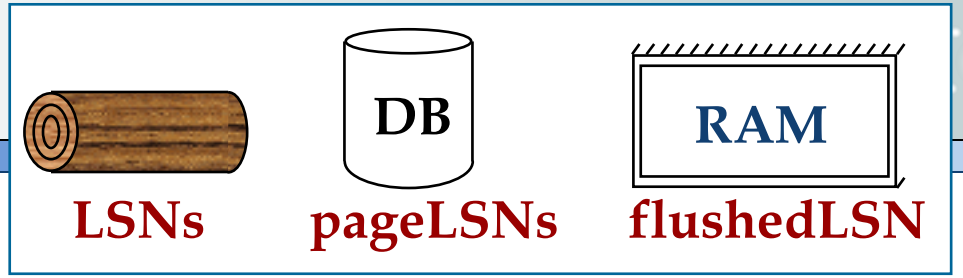
9.2.2 基于WAL的日志

- **WAL规则(The Write-Ahead Logging Protocol)**

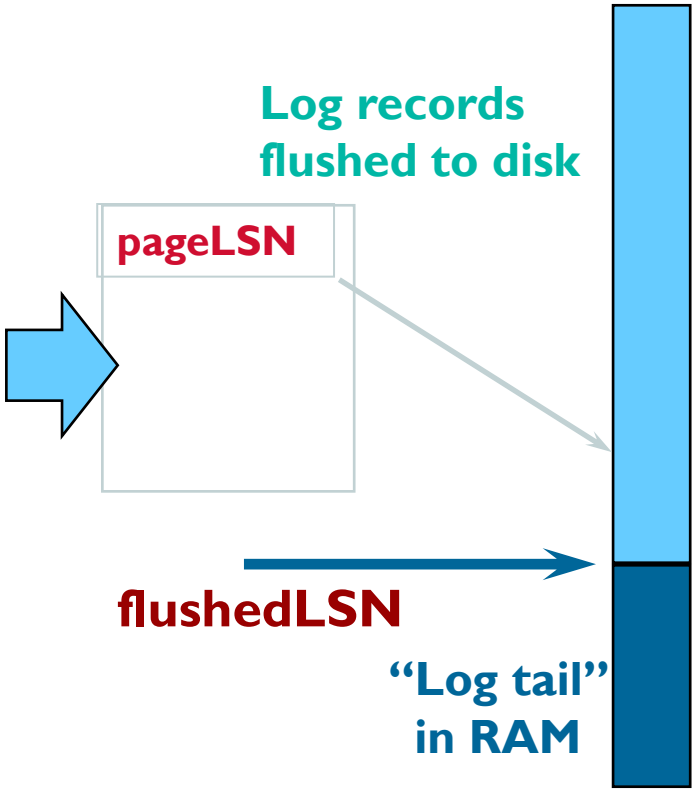
1. 先写日志（Force the log record for an update *before* the corresponding data page gets to disk）
   - **保证原子性的主要原理**Guarantees Atomicity
2. 在事务提交前，写事务相关的所有日志记录到稳存

   Write all log records for a Xact *before* commit
   - **保证持久性的主要原理（确保我们能基于日志重建提交事务影响）**

WAL日志组织



- 每条日志记录有唯一的、顺序递增的**LSN**
- 每个数据页(*data page*)包含一个 **pageLSN**域:
  - 存储最后修改该页的日志记录之LSN
- 系统跟踪 **flushedLSN**:
  - 到目前为止已刷新到稳存的最大日志LSN
- **WAL规则要求:**
  - pageLSN ≤flushedLSN



9.2.3 检查点技术

1. 问题的提出

- 由于日志记录了所有修改动作的必要信息，使得DBMS可以重复历史，但代价可能非常昂贵。
  - 撤销（UNDO）处理需要搜索**整个**日志
  - REDO处理：**从头重新执行**，浪费了大量时间
- **DBMS通过采用：周期性的创建一些检查点(checkpoint)**技术，可大幅减小或降低恢复所需时间或工作量。

9.2.3 检查点技术

1. 问题的提出

2. 检查点实现的基本思想

- 在日志文件中增加检查点标志记录，并在检查点标志记录写到稳存之前，完成一些必要的、到现在为止的‘阶段性’归总工作；
- 当系统崩溃后重启时，重做扫描开始点和撤销回溯终点，就不再总是日志的首条记录。
  - 它们都可通过分析最近可用检查点记录中的归总信息获得。

静态检查点和模糊检查点

- 系统建立一个检查点需要一定的时间。
- 根据建立检查点期间，是否允许继续执行事务，或启动新事务，可将检查点分为静态检查点和动态检查点。
  - 静态检查点管理相对简单，但它会影响系统性能。
  - 动态检查点也称模糊检查点(fuzzy checkpoint)，它对系统系统影响不大，但管理相对复杂。

9.3 ARIES日志的系统崩溃恢复

9.3.1 ARIES简介

9.3.2 ARIES的检查点记录

ARIES

- 它试图以概念上相对简单且系统化的方式，提供一套能确保事务原子性和持久性的、具有良好性能的的恢复管理算法。
- 它能与绝大多数并发控制机制很好协调工作的。

- **ARIES（Algorithm for Recovery and Isolation Exploiting Semantics）**

- 使用**并发**封**锁**控制假**设**

本章**讨论时**，默**认时**都假定使用基于**strict-2PL协议**的封**锁调**度器，**并**假**设**主要基于**页级封锁**。少**数场**合，如**逻辑**日志中，也可能涉及元**组级**封**锁**。

ARIES恢复管理算法综述

- 采用基于“允许窃用页框且非强制页” 工作模式。
- 当系统崩溃后重启时，恢复管理器将被激活，并按以下三个阶段进行处理：
  - **分析**（Analysis）：鉴别崩溃发生时，缓冲区中的脏页和当时仍活跃的事务。
  - **重做**（Redo）：重做从日志的适当起点（比如：被修改的最早脏页对应日志记录）开始的所有动作，恢复系统到崩溃时的DB状态。
  - **撤销**（Undo）：撤销上次崩溃时所有未提交事务的动作效果，使DB只反映已提交事务的影响。

一个简单的ARIES日志片段示例

- 分析阶段，分析识别出
  - 崩溃时仍活跃事务：T_1, T_3；
  - 已提交事务T_2
  - 崩溃时的脏页P_1、P_3和P_5。

    注意，扫描LSN60不会再修改脏页表！ recLSN记录致该页变脏的最早日志记录序号
- 在重做阶段，提交事务T_2的所有动作，必须按日志顺序重新应用一次。
- 在撤销阶段，T_1和T_3的所有动作，必须按日志记录的相反顺序，依次逐个撤销。

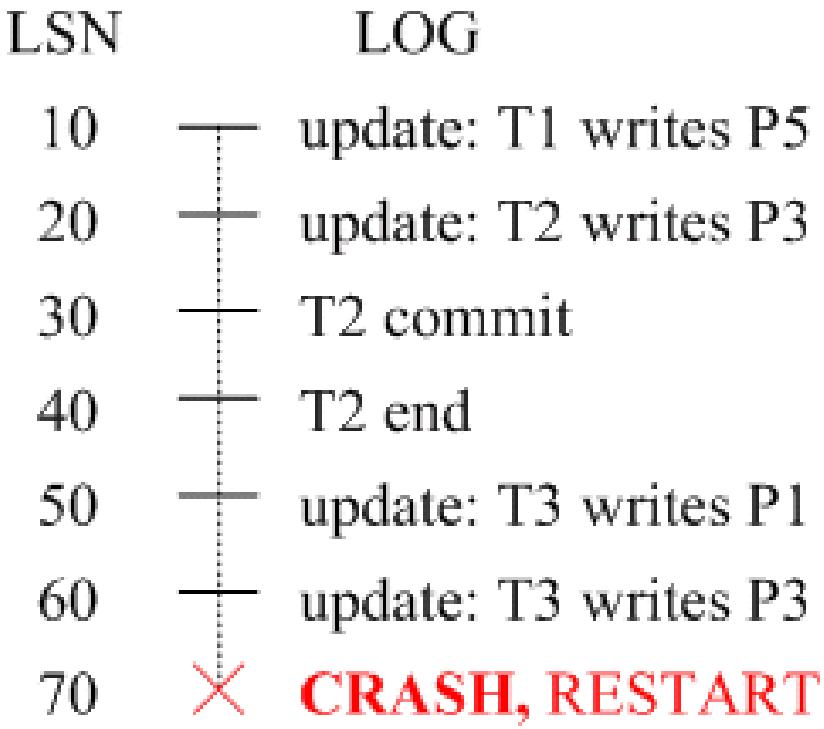


图 9.3 一个简单的 ARIES 日志片段

AREIS数据结构：日志记录

ARIES日志记录类型

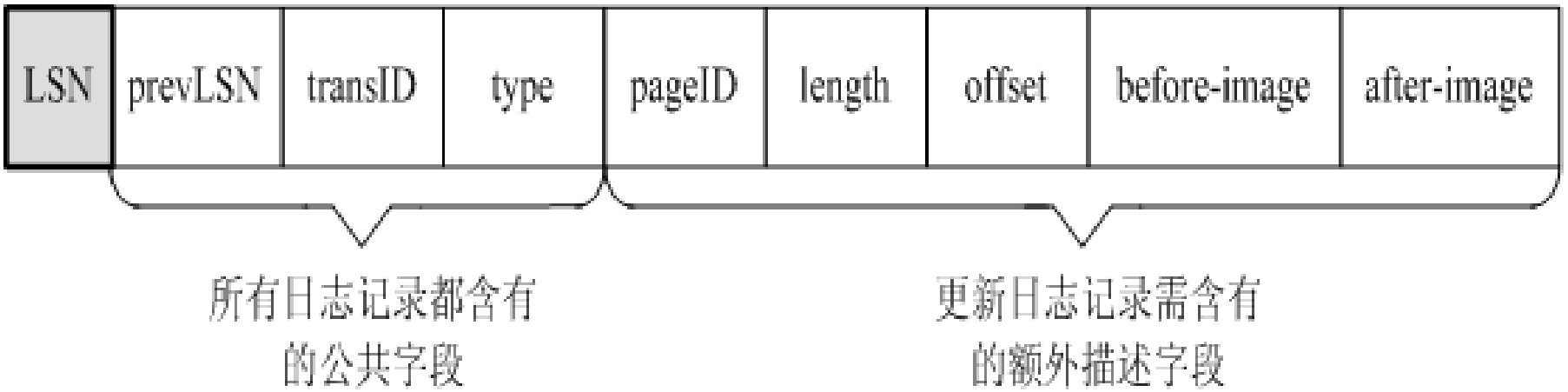
LogRecord 字段:

- prevLSN
- XID
- type
- only in **update**
  - pageID
  - length
  - offset
  - before-image
  - after-image

The log record types:

- Update
- Commit
- Abort
- End **(signifies end of commit or abort)**
- CLRs **(Compensation Log Records)**
  - Log of UNDO actions
  - “Cancel out” an update step

ARIES常规日志记录的结构模式



COMMIT、ABORT --LOGREC

Commit LOGREC

- 当事务决定提交，它将先强制写一条类型为commit(内含该事务id)的日志记录到日志尾，并执行一次刷新当前日志尾到稳存的动作。
- 当事务的commit LOGREC写到稳存后，就可认为该事务已经提交。

中止(abort) LOGREC

- 当一个事务中止时，一条类型为abort(内含事务id)的日志记录将被写到日志尾。
- 还会启动一个撤消该事务的 Undo 过程。

end -LOGREC

- 当一个事务提交或中止后，除了稳定写**commit**或**abort**型日志记录外，**DBMS**还将执行一些额外的动作步骤。
  - 当这些额外动作步都完成后，将会写一条(包含事务id、类型为)end日志记录到稳存。
    - 额外动作，比如执行删除该事务在事务表中记录行的清理工作。
  - 稳定写end日志记录，也不能确保被修改的相关数据页已更新到磁盘。
    - 对于commit事务，end记录只能保证该事务已被从事务表中移除。

CLR-LOGREC

- 当由一条更新日志记录U所记录的改变被撤销时，将会有一条类型为CLR的补偿日志记录C被写到日志尾。
  - 与更新记录 U 的 prevLSN 相对应，补偿日志记录 C 中有一个称为 undoNextLSN 的字段，用来指示同一事务将被撤消的下一条更新记录LSN。
  - 一般情况下，C 的 undoNextLSN 将被设为与 U 的 prevLSN 同一值。

与日志相关的其它数据结构

事务表Transaction Table(T.T):

- 每个活跃事务占一个表项
- 每个表项内容：〈XID, status, lastLSN 〉

脏页表Dirty Page Table(D.P.T):

- 缓冲池中每个脏页占一个表项
- 每个表项内容：〈PageID, recLSN〉

磁盘中页面结构

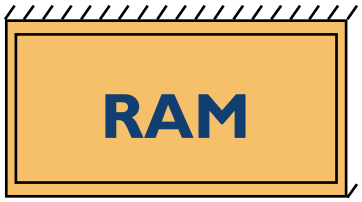
DB的每个页都有 page_LSN 域。

ARIES相关数据结构的存储位置

LogRecords
- prevLSN
- XID
- type
- pageID
- length
- offset
- before-image
- after-image

master record

Data pages
- 每页内
- 含有一个
- pageLSN

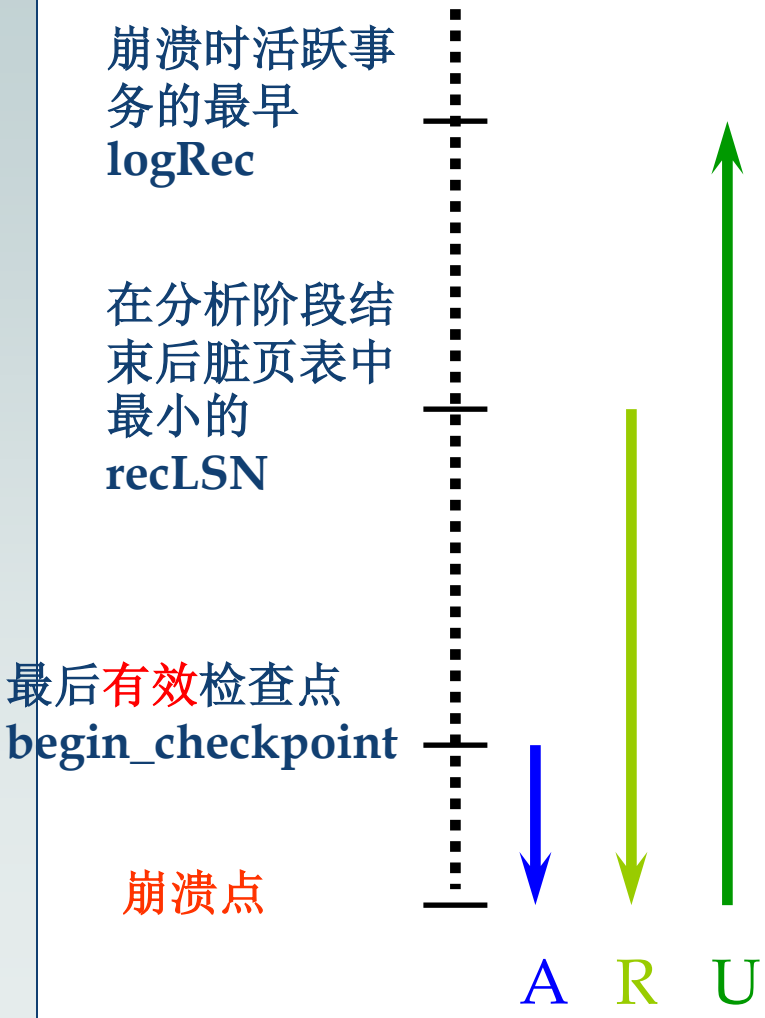


Xact Table
- XID
- lastLSN
- status

Dirty Page Table
- PageID
- recLSN

flushedLSN

ARIES崩溃恢复



- 从一个最后检查点开始扫描 (通过查主记录可获得该**LSN**)
- 算法三个阶段:
  1. **Figure out which Xacts committed since checkpoint, which failed (Analysis)**
  2. **REDO** 所有动作(重复历史)
  3. **UNDO** 所有失败事务的影响

9.4 基于ARIE日志的系统崩溃恢复

9.4.1 分析阶段

9.4.2 Redo阶段

9.4.3 Undo阶段

9.4.4 重启时再次崩溃处理

9.4.5 其他相关算法以及ARIES的并发控制特性

算法 9.1　ARIES 分析阶段扫描处理算法描述

```
begin
    读取最近的 begin_checkpoint 记录的 LSN 作为扫描起点;
    并从下一个 end_checkpoint 日志记录项内容中初始化脏页表和事务表;
    do loop 向前扫描日志记录直到日志结束:
        if 遇到一个某事务 T 的 end 型日志记录 then
            将 T 从事务表中移除（因为 T 已不再是活跃事务）;
        else
            if T 不在事务表中 then
                将 T 加入到事务表中，其中，
                    lastLSN 字段置为当前日志记录的 LSN;
                    if 日志记录是 commit 记录 then
                        状态字段置为 C
                    else
                        状态字段置为 U;
                    end if
            else
                将事务 T 对应表项的 lastLSN 置为当前日志的 LSN;
            end if
            if 遇到了一个影响页 P 的需重做日志记录 then
                if 页 P 不在脏页表中 then
                    将 项<P 的页 id，本日志记录的 LSN> 插入到脏页表中;
                end if
            end if
        end if
    end loop;
    删除事务表中状态为'C'的已完成事务;
end begin;
```

例题 9.2

脏页表

pageID	recLSN
P500	
P600	
P505	

事务表

transID	status	lastLSN
T1000	U	
T2000	U	

LSN	prevLSN	transID	type	pageID	length	offset	before..	after..
…	…	…	…	…	…	…	…	…
210	-	T1000	Update	P500	3	21	ABC	DEF
230	-	T2000	Update	P600	2	41	HI	LM
250	230	T2000	Update	P500	3	20	gde	GDE
270	210	T1000	Update	P505	4	21	ABCD	DEFG

图9.5 日志记录

- 事务T1000改变了页p500的第21-23字节值('ABC'→'DEF');
- 事务T2000改变了页p600的第41-22字节值('HI'→'LM');
- 事务T2000改变了页p500的第20-22字节值('gde'→'GDE');
- 事务T1000改变了页p505的第21-24字节值('ABCD'→'DEFG');

REDO阶段

- 主要任务：重复历史(重构崩溃时的状态)
- 具体算法描述

算法 9.2 ARIES Redo 阶段处理算法描述

```
begin
    从脏页表中最小的 recLSN 开始向前扫描日志;
    for each CLR or update LOGREC LSN do
        if  NOT ( (被影响的页不在脏页表中) 或
                  (虽然被影响页在脏页表中，但最后修改脏页的 recLSN > LSN) 或
                  (pageLSN≥LSN ))  then
            重应用日志动作(Reapply logged action)
            Set pageLSN =LSN; 但不需为此写日志记录!
        endif
    end for;
end begin
```

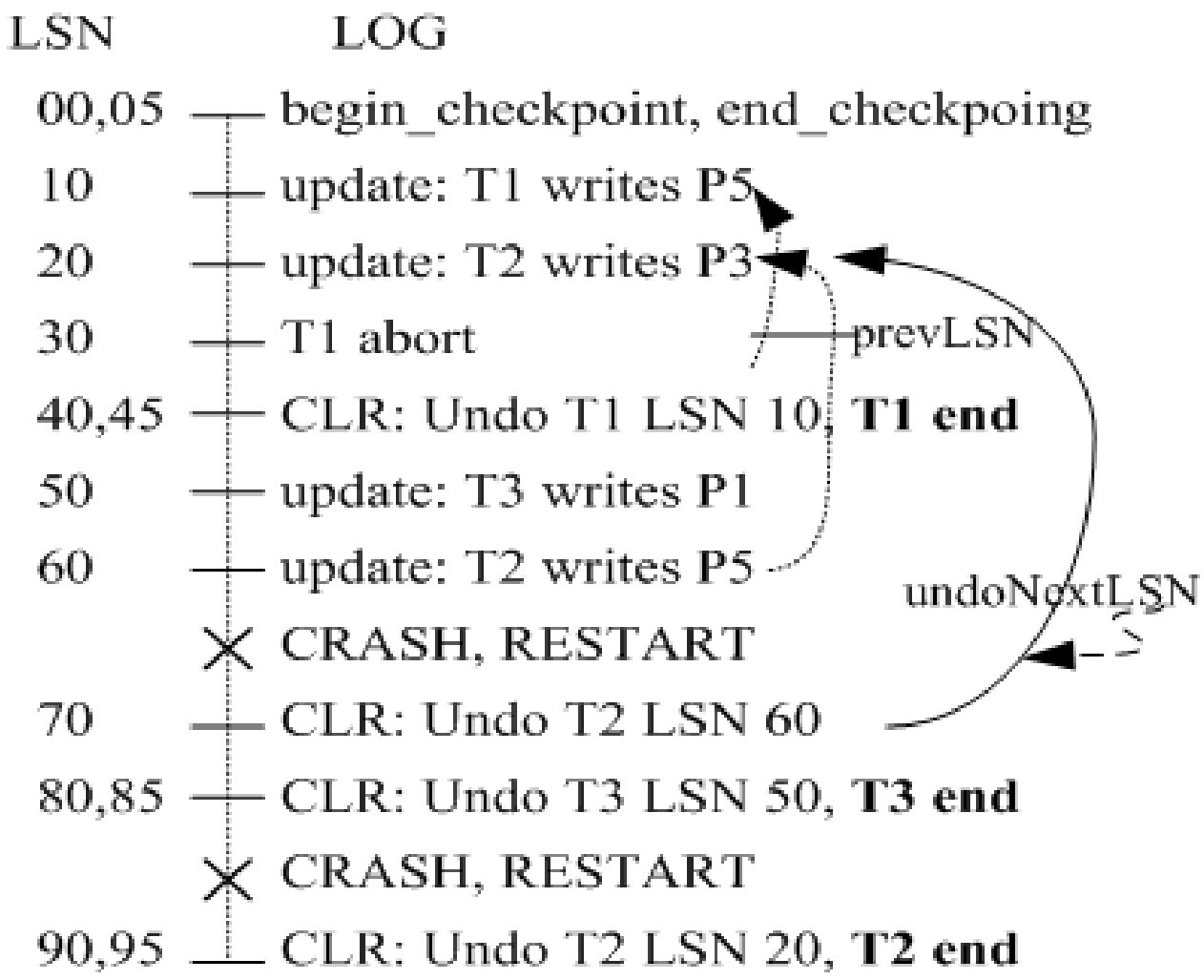
算法9.3 UNDO阶段处理描述

```
begin
    将事务表中所有事务的 lastLSN 加入到 ToUndo 集;
    repeat
        从当前 ToUndo 集中摘取最大的 LSN 值➔curLastLSN;
        Set curLogRec= <curLastLSN 对应的日志记录>;
        if   curLogRec.type = update then
            一个对应的 CLR 记录被写入日志尾;
            实际执行补偿撤消动作（修改数据页）;
            if   curLogRec.prevLSN 非空 then
                将 curLogRec.prevLSN 加入 ToUndo 集;
            end if
        else if curLogRec.type = CLR then
            if   curLogRec.undoNextLSN 非空 then
                将 curLogRec.undoNextLSN 加入 ToUndo 集; //这里不修改数据页!
            else   //curLogRec.undoNextLSN 为空
                写 end 日志记录 < curLogRec.LSN, curLogRec.transID, 'end' >到稳存;
            end if
        end if
    until ToUndo 成为空集;
end;
```

如需要修改，REDO时肯定已经重执行过

重启时再次崩溃处理

- 图9.7 在Undo期间再次发生崩溃



9.5 转储备份与恢复

9.5.1 静态转储与动态转储

9.5.2 利用检查点的备份恢复

- **什么是数据转储**
  - 转储是指DBA将整个数据库复制到磁带或另一个磁盘上保存起来的过程。
- **备用的数据文本被称为后备副本或后援副本**
- **转储的主要类型**
  - 静态转储与动态转储
  - 海量转储与增量转储

9.5.1 静态转储

- 在系统中无运行事务时进行转储
  - 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
- 优点：实现简单
- 缺点：降低了数据库的可用性
  - 转储必须等用户事务结束
  - 新的事务必须等转储结束

9.5.2 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
- 利用动态转储得到的副本进行故障恢复
  - 利用后备副本加上日志文件，把数据库恢复到某一时刻的正确状态。

9.5.3 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

	增量	海量
备份实施	更快	
恢复实施		更方便、快捷
占用空间	少	多
小数据库		用海量
大数据库	间隔使用两种方式	

Thanks!

The end.

LOGO