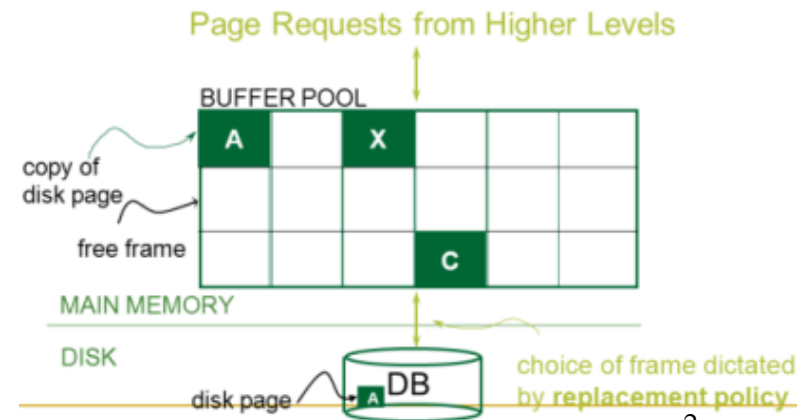


Crash Recovery

SUN YAT-SEN UNIVERSITY

Review

- 事务的4个特性 ACID
- 事务的提交与中止
- 并发控制，严格 2PL
- Buffer Management, 脏页



Review: The ACID properties

- **Atomicity:** All actions in the **Xact (transaction)** happen, or none happen.
- **Consistency:** If each Xact is consistent, and the DB starts consistent, it ends up consistent.
- **Isolation:** Execution of one Xact is isolated from that of other Xacts.
- **Durability:** If an Xact commits, its effects persist.
- The **Recovery Manager** (恢复管理器) guarantees Atomicity & Durability.

Motivation 动机

The **Recovery Manager** guarantees Atomicity & Durability.

■ **Atomicity:** All actions in the Xact (transaction) happen, or none happen.

□ Transactions may abort (“**Rollback**”).

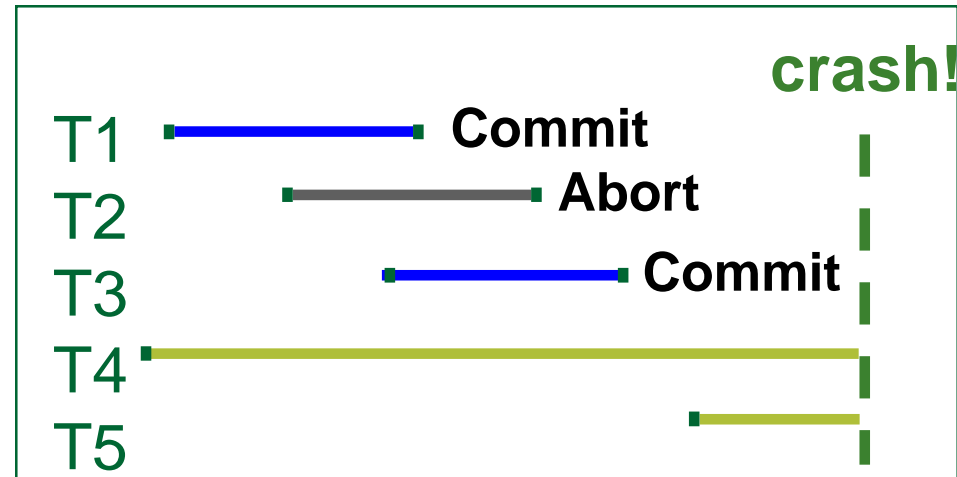
■ **Durability:** If an Xact commits, its effects persist.

□ What if DBMS stops running? (Causes?)

✓ **Desired state** after system restarts:

– **T1** & **T3** should be **durable**.

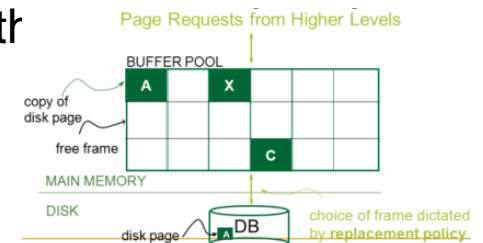
– **T2**, **T4** & **T5** should be **aborted** (effects should not be seen).



Assumptions

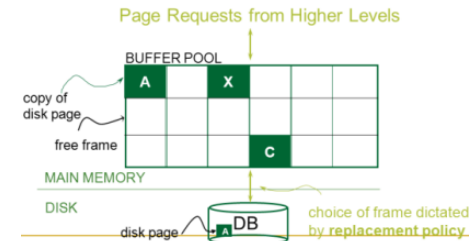
Crash Recovery

- Concurrency control is in effect.
 - Strict 2PL, in particular.
- Updates are happening “in place”.
事务对数据库的更新最终反映在磁盘中
 - i.e., data is overwritten on (deleted from) the disk.
- Atomic Writes: **writing a page to disk is an atomic action.**
 - In practice, additional details are needed to deal with



Buffer Management Policy Plays a Key Role

Crash Recovery



1、提交前写策略（即偷帧）？

- Shall we allow “dirty pages”（脏页）in the buffer pool caused by an Xact T to be written to disk before T commits?
 - ❑ If so, a second Xact T' can “steal” a frame from T . -偷帧
 - ❑ In contrast, No-Steal. （易实现UNDO）（提高并发度）

2、提交后不写策略（强制写）？

- When an Xact T commits, must we ensure that all the “dirty pages” of T are immediately forced to disk?
 - （易实现Durability）
 - ❑ If so, we say that a “force” approach is used. – 强制写
 - ❑ In contrast, No-Force. 好处：降低IO开销（如：同一“脏页”多次写）

Variants of Buffer Management Policy

	(易实现UNDO)	(高并发度)
	No Steal	Steal
(降低IO开销) No Force		Fastest
Force (易实现Durability)	Slowest	
	Performance Implications	

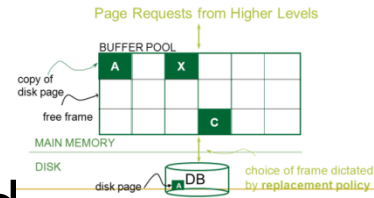
恢复策略？

	No Steal	Steal
No Force	No UNDO REDO	UNDO REDO
Force	No UNDO No REDO	UNDO No REDO
	Logging/Recovery Implications	

Preferred Policy: Steal/No-Force

1、偷帧策略（高并发）

2、非强制写策略（低IO）



- **STEAL** (偷帧, why enforcing Atomicity is hard)
 - *To steal **frame F**: Current page in F (say P) is written to disk; some Xact holds lock on P.*
 - 问题: What if the Xact with the lock on P aborts?
 - 解决: Must remember the **old value** of P at steal time (to support **UNDO**ing the write to page P).
- **NO FORCE** (非强制写, why enforcing Durability is hard)
 - 问题: What if system crashes before an updated page is written to disk?
 - 解决: Write as little as possible, in a convenient place, at commit time, to support **REDO**ing modifications.

Basic Idea: Logging-写日志



- Record REDO and UNDO information, for **every update**, in a *log(日志)*.
 - Sequential writes to log (put it on a separate disk).
 - Minimal information (difference) written to log, so multiple updates fit in a single log page.
- Log: An ordered list of REDO/UNDO actions
 - Log record contains:
 - <XID, pageID, offset, length, old data, new data>
 - and additional control info (which we'll see soon).

The Write-Ahead Logging (WAL) Protocol

先写日志

先写 UNDO action

#1 Must **force** the **log record** for an update to disk before the corresponding **data page** gets to disk.

在将脏页写入磁盘之前先将如何变脏的更新日志记录写入磁盘中

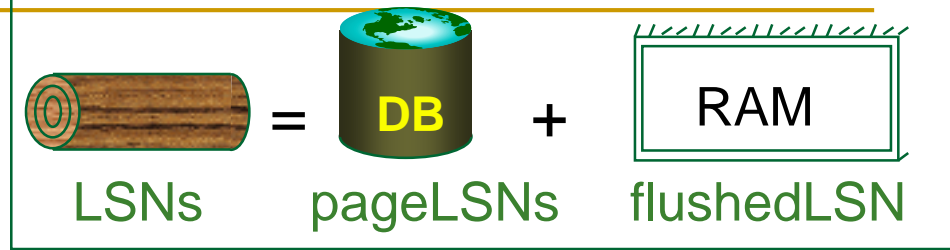
- This rule helps guarantee **Atomicity**.
- This allows us to implement **Steal** policy.

#2 Must **write all log records** for an Xact to disk before commit.

从实现角度定义“事务是否已提交？”

- I.e. transaction is not committed until all of its log records including its “commit” record are written to disk.
- This rule helps guarantee **Durability**.
- This allows us to implement **No-Force** policy.
- Exactly how is logging (and recovery!) done?
 - We'll look at the **ARIES algorithms** from IBM.

WAL & the Log



(使用LOG可以重构DB, 即每个data page)

- Each log record has a unique **Log Sequence Number (LSN)**. —日志序列号
 - LSNs is always increasing.
- System keeps track of **flushedLSN**. -已刷盘LSN
 - The max LSN flushed so far.
- Each **data page** contains a **pageLSN**.
 - The LSN of the **most recent log record** for an **update** to that page.
- WAL**: Before page **i** is written to disk, log must satisfy:
 $\text{pageLSN}_i \leq \text{flushedLSN}$,
i.e., the log record identified by pageLSN_i must have been written to disk for page **i** is written to disk.

Log records
flushed to disk

flushedLSN

pageLSN

“Log tail”
in RAM

Log Records

LogRecord fields:

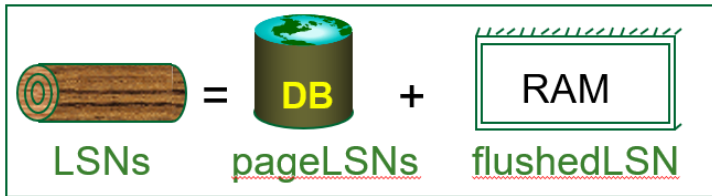
LSN
prevLSN
XID
type
update records only { pageID
length
offset
before-image
after-image

prevLSN is the LSN of the previous log record written by *this* Xact (so records of an Xact form a linked list backwards in time)

Possible log record types:

- Update, Commit, Abort
- Checkpoint (for log maintenance)
- Compensation Log Records (CLRs)-补偿日志记录
 - for UNDO actions
- End (end of commit or abort)

Other Log-Related Data Structures



■ Two in-memory tables:

■ Transaction Table-事务表

- ❑ One entry per currently active Xact.
 - entry removed when Xact commits or aborts
- ❑ Contains **XID**, **status** (running/committing/aborting), **lastLSN** (most recent LSN written by Xact).

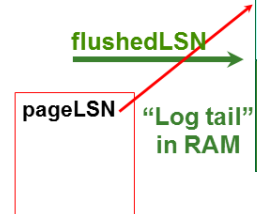
transID	lastLSN
T1000	
T2000	

TRANSACTION TABLE

LogRecord fields:

- LSN
 - prevLSN
 - XID
 - type
 - pageID
 - length
 - offset
 - before-image
 - after-image
- update records only

Log records
flushed to disk



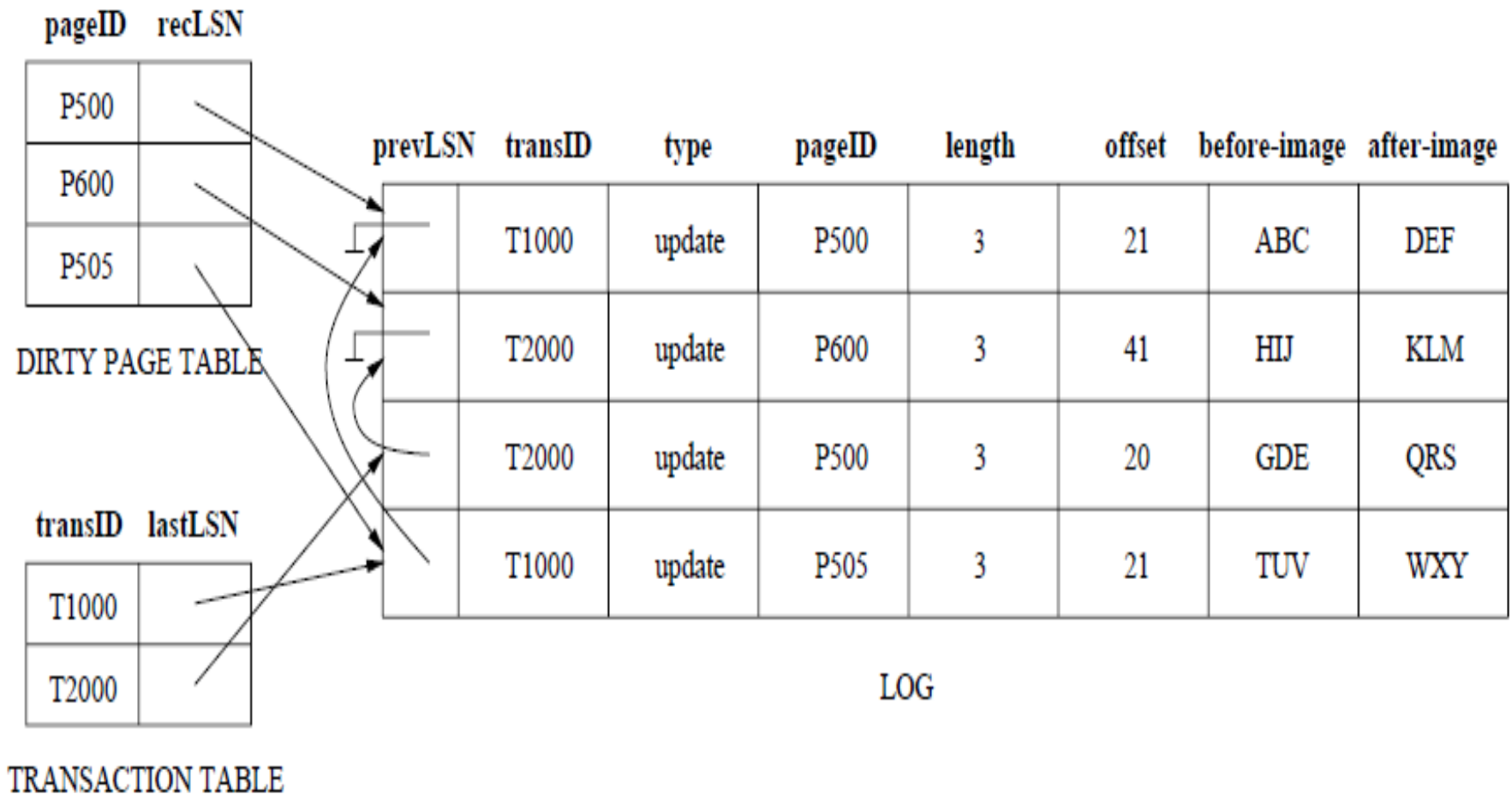
■ Dirty Page Table-脏页表:

- ❑ One entry per dirty page currently in buffer pool.
 - ❑ Contains **recLSN** -- the LSN of the log record which **first** caused the page to be dirty. (顺序扫描, 支持 REDO)
- 最早使该页变脏的更新操作的日志记录的LSN

pageID	recLSN
P500	
P600	
P505	

DIRTY PAGE TABLE

An Example

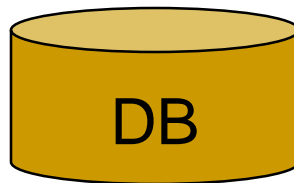


The Big Picture: What's Stored Where

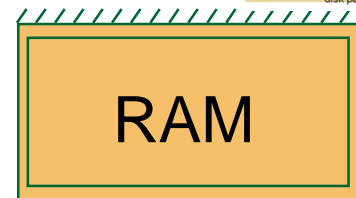
(使用LOG可以重构DB, 即每个data page)



=



+



LogRecords

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

Log records
flushed to disk

flushedLSN

pageLSN

"Log tail"
in RAM

Data pages

each
with a
pageLSN

Master record

LSN
of most recent
checkpoint record

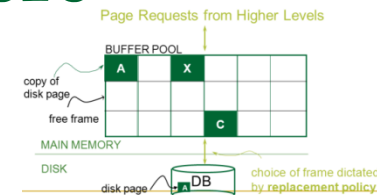
Xact Table

lastLSN
status

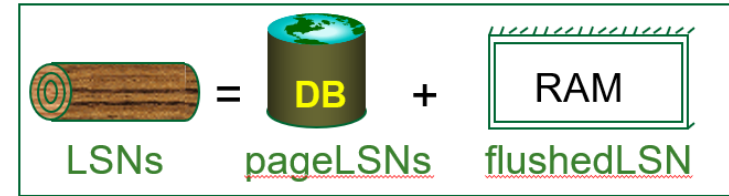
Dirty Page Table

recLSN

flushedLSN



Normal Execution of an Xact



通过Log 实现崩溃恢复，**要求**事务的执行满足以下条件：

- Series of **reads** & **writes**, followed by **commit** or **abort**.
- **Strict 2PL**.
- **STEAL, NO-FORCE** buffer management, with **Write-Ahead Logging**.

Transaction Commit

Log records
flushed to disk

flushedLSN

pageLSN
“Log tail”
in RAM

- Write *commit* record to log.
- All log records up to Xact's *commit record* are flushed to disk.

从实现角度定义“事务是否已提交？”

□ Guarantees that $\text{flushedLSN} \geq \text{lastLSN}$.

□ Note that

- log flushes are sequential, synchronous writes to disk.
- Many log records per log page.

transID	lastLSN
T1000	
T2000	

TRANSACTION TABLE

强制写日志的情况：

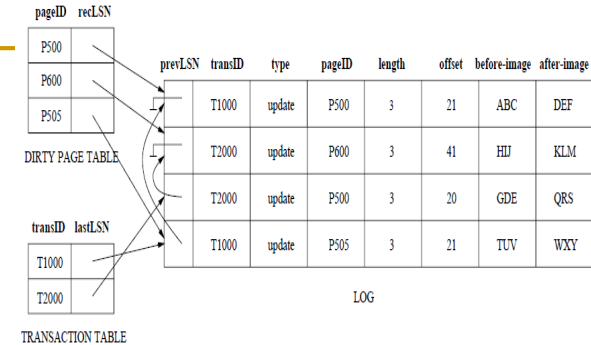
1. 事务提交
2. 偷帧
3. 页满？

- Commit() returns.
- Write *end* record to log.

Possible log record types:

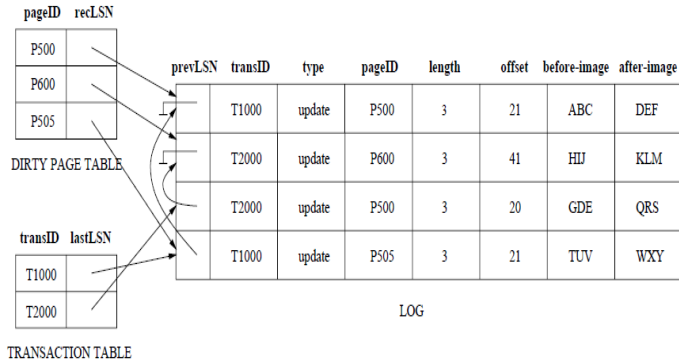
- Update, Commit, Abort
- Checkpoint (for log maintenance)
- Compensation Log Records (CLRs)
 - for UNDO actions
- End (end of commit or abort)

Simple Transaction Abort



- For now, consider an explicit abort of an Xact.
 - No crash involved.
- We want to “play back” the log in reverse order, UNDOing updates.
 - Get **lastLSN** of Xact from Xact table.
 - Write an **Abort log record** before starting to rollback operations
 - Can follow chain of log records backward via the **prevLSN** field. (反向链表, 支持 UNDO)
 - Write a “**CLR**” (compensation log record-补偿日志记录) for each undone operation.

Abort, cont.



Currently UNDOing
PrevLSN=1234

lastLSN (CLR)
undonextLSN=1234

- To perform UNDO, must have a lock on data!
 - No problem! (原因: Strict 2PL.)
- Before restoring old value of a page, write a CLR:
 - You continue logging while you UNDO!!
 - CLR has one extra field: undonextLSN
 - Points to the next LSN to undo (i.e. the prevLSN of the record we're currently undoing).
 - CLR contains REDO info – 也称 “redo only” 日志记录
 - CLR's never Undone
 - Undo needn't be idempotent (幂等的, >1 UNDO won't happen)
 - But they might be Redone when repeating history (=1 UNDO guaranteed)
- At end of all UNDOs, write an “end” log record.

..., before-image, after-image

(使用LOG可以重构DB, 即每个data page)

Checkpointing



- Usually, we **can not keep** log around for all time.
- Periodically, the DBMS creates a checkpoint
 - ❑ Minimizes recovery time after crash.
 - ❑ Write to log:
 - **begin_checkpoint** record: Indicates when the **checkpoint** began.
 - **end_checkpoint** record: Contains **current Xact table** and **dirty page table**. A 'fuzzy checkpoint'-模糊检查点:
 - ❑ Other Xacts continue to run; so these tables accurate only as of the time of the **begin_checkpoint** record.
 - ❑ **No attempt to force dirty pages** to disk; effectiveness of checkpoint limited by oldest unwritten change to a dirty page.

pageID	recLSN
P500	
P600	
P505	

DIRTY PAGE TABLE

transID	lastLSN
T1000	
T2000	

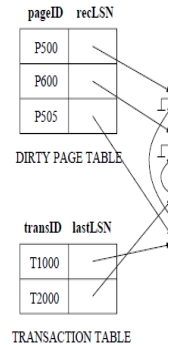
TRANSACTION TABLE

强制写日志的情况：

1. 事务提交
2. 偷帧
3. 页满？
4. 写 master record

Store LSN of most recent checkpoint record in a safe place (**master** record).

Crash Recovery: Big Picture

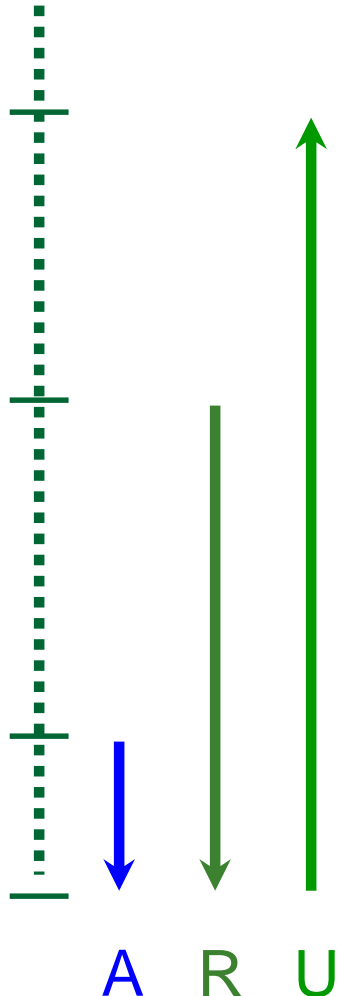


Oldest log
rec. of Xact
active at
crash

Smallest
recLSN in
dirty page
table after
Analysis

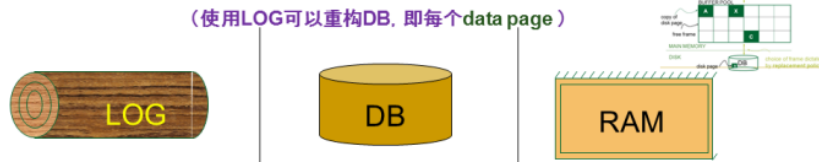
Last chkpt

CRASH

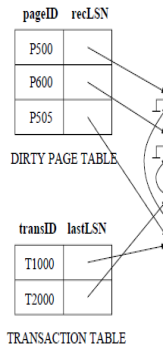


- ✓ Start from a **checkpoint** (found via **master** record).
- ✓ Three phases. Need to do:
 - ① **Analysis** - Figure out which **Xacts committed** since checkpoint, **which failed**.
 - ② **REDO** *all* actions.
 - (repeat history)
 - ③ **UNDO** effects of failed Xacts.

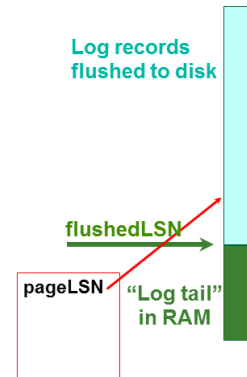
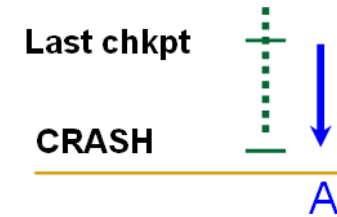
Recovery: The Analysis Phase



恢复事务表与脏页表



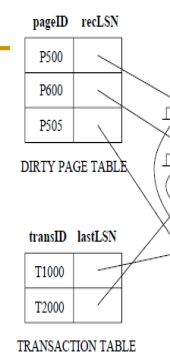
1. Re-establish knowledge of state at checkpoint.
 - ❑ via transaction table and dirty page table stored in the checkpoint
 2. Scan log forward from the most recent checkpoint.
 - ❑ End record: Remove Xact from Xact table.
 - ❑ All Other records: Add Xact to Xact table, set lastLSN=LSN,
 - change Xact status on commit record.
 - also, for Update records: If page P not in Dirty Page Table (DPT), Add P to DPT, set its recLSN=LSN.
- At the end of Analysis...
- ❑ Xact table says which xacts were active at time of crash.
 - ❑ DPT says which dirty pages might not have been written to disk.



Phase 2: The REDO Phase

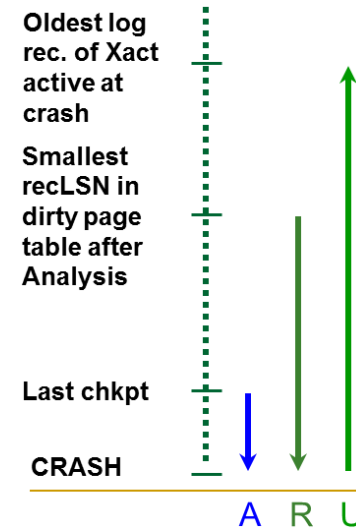
恢复脏页

(使用LOG可以重构DB, 即每个data page)



- We *Repeat History* to reconstruct state at crash:
 - Reapply *all* updates (even of aborted Xacts!), redo CLR.s.
- Scan forward from log record containing **smallest recLSN** in **DPT**.
- For each update log record or CLR with a given **LSN**, REDO the action unless:
 - Affected page is not in the **DPT**, or
 - Affected page is in **DPT**, but has
 - $recLSN > LSN$, or
 - $pageLSN$ (in DB) $\geq LSN$. (this last case requires I/O)
- To **REDO** an action:

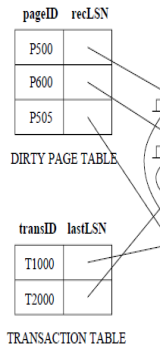
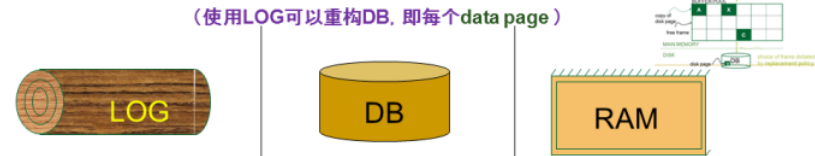
..., before-image, after-image



- Reapply logged action. (注: 在脏页上修改)
- Set **pageLSN** to **LSN**. No additional logging, **no forcing**!

Phase 3: The UNDO Phase

UNDO未完成事务

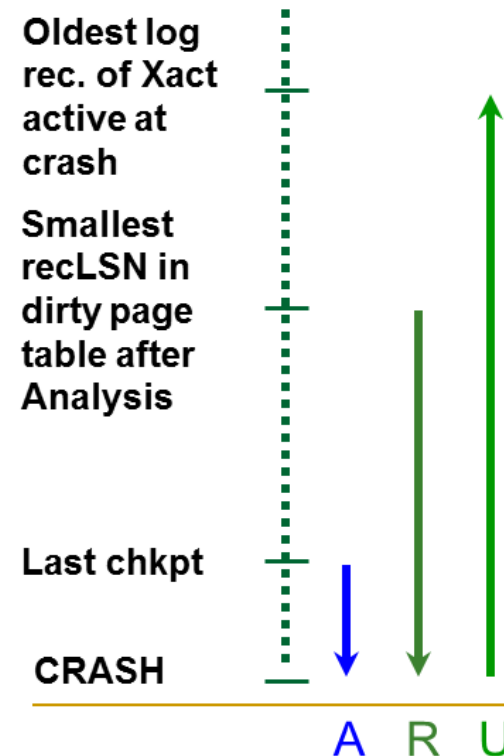


■ A Naïve solution:

- ❑ The xacts in the Xact Table are losers(失败事务).
- ❑ For each loser, perform simple transaction abort.

❑ Problems?

- 多次反向扫描LOG



Phase 3: The UNDO Phase

transID	lastLSN
T1000	
T2000	

TRANSACTION TABLE

Oldest log
rec. of Xact
active at
crash

Smallest
recLSN in
dirty page
table after
Analysis

Last chkpt

CRASH

A R U

ToUndo={lastLSNs of all Xacts in the Xact Table}
a.k.a. “losers”

Repeat:

- ❑ Choose (**and remove**) largest LSN among ToUndo.
- ❑ If this LSN is a CLR and `undonextLSN==NULL`
 - Write an **End** record for this Xact.
- ❑ If this LSN is a CLR, and `undonextLSN != NULL`
 - Add `undonextLSN` to ToUndo
- ❑ Else this LSN is an **update**. Undo the update, write a CLR, add `prevLSN` to ToUndo.

..., before-image, after-image

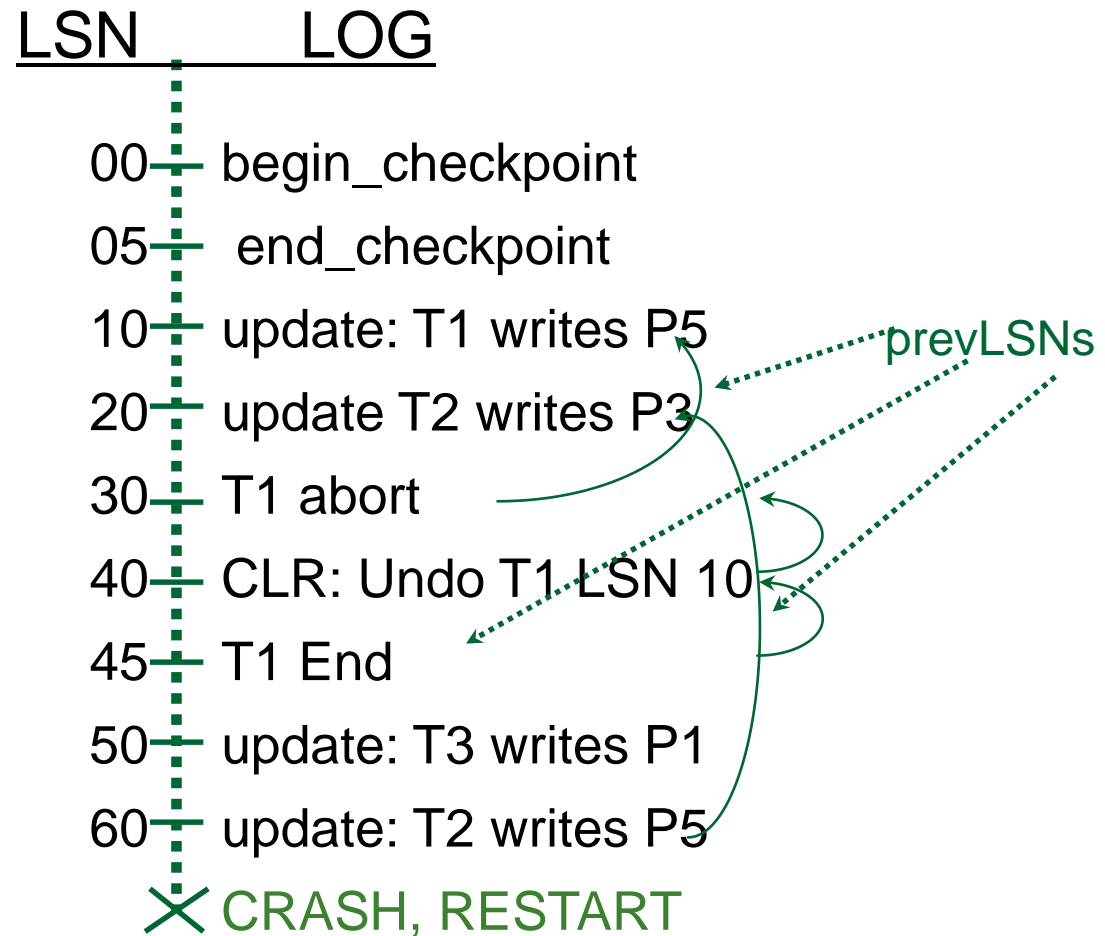
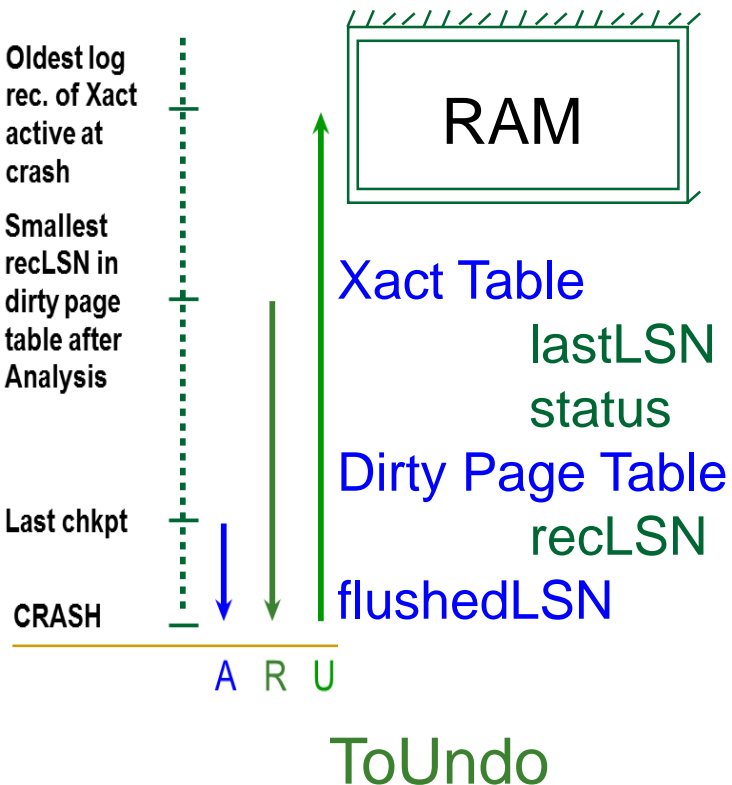
Until ToUndo is empty.

NOTE: This is simply a **performance optimization** on the naïve solution to do it in one backwards pass of the log!

Example of Recovery

脏页表	
pageID	recLSN
P5	10
P3	20
P1	50

事务表	
transID	lastLSN
T1	40
T2	60
T3	50

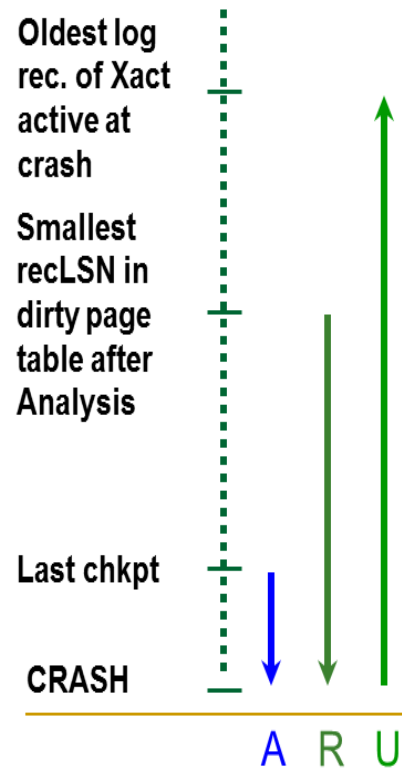


Example:

Crash During Restart!

脏页表	
pageID	recLSN
P5	10
P3	20
P1	50

事务表	
transID	lastLSN
T1	40
T2	70
T3	80



ToUndo

LSN LOG

00,05 — begin_checkpoint, end_checkpoint

10 — update: T1 writes P5

20 — update T2 writes P3

30 — T1 abort

40,45 — CLR: Undo T1 LSN 10, T1 End

50 — update: T3 writes P1

60 — update: T2 writes P5

✗ CRASH, RESTART

70 — CLR: Undo T2 LSN 60

80,85 — CLR: Undo T3 LSN 50, T3 end

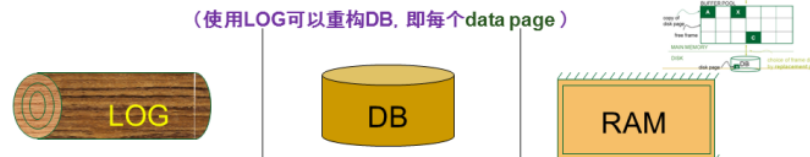
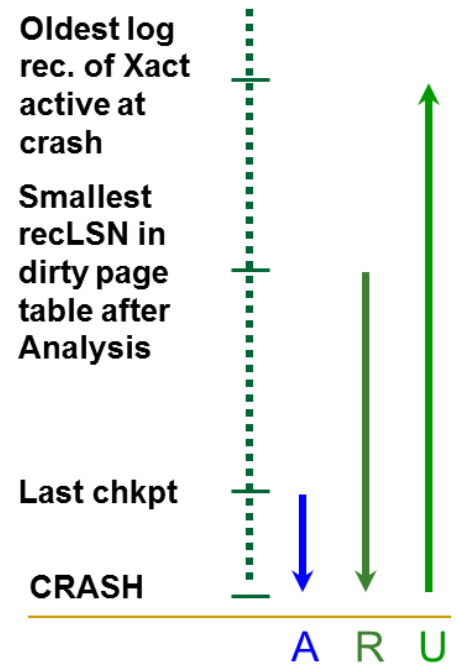
✗ CRASH, RESTART

90 — CLR: Undo T2 LSN 20, T2 end

undonextLSN

Additional Crash Issues

- What happens if system crashes during Analysis? During REDO?
- How do you limit the amount of work in REDO?
 - Flush asynchronously in the background.
- How do you limit the amount of work in UNDO?
 - Avoid long-running Xacts.



- 1、恢复事务表与脏页表
- 2、恢复脏页
- 3、UNDO未完成事务

Summary of Logging/Recovery

- **Recovery Manager** guarantees Atomicity & Durability.
- Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness.
- LSNs identify log records; linked into backwards chains per transaction (via prevLSN).
- pageLSN allows comparison of data page and log records.

Summary (Contd.)

- **Checkpointing:** A quick way to limit the amount of log to scan on recovery.
- Recovery works in 3 phases:
 - **Analysis:** Forward from checkpoint.
 - **Redo:** Forward from oldest recLSN.
 - **Undo:** Backward from end to first LSN of oldest Xact alive at crash.
- Upon Undo, write CLR's.
- Redo “repeats history”: Simplifies the logic!

Summary

- 要求:
 - 理解Crash Recovery的3个阶段算法，相应的例子
 - 能够根据给定日志进行崩溃恢复，即
 - 恢复事务表、脏页表
 - 为日志添加因UNDO而生成的CLR和END日志记录