Crash Recovery

Yanlei Diao

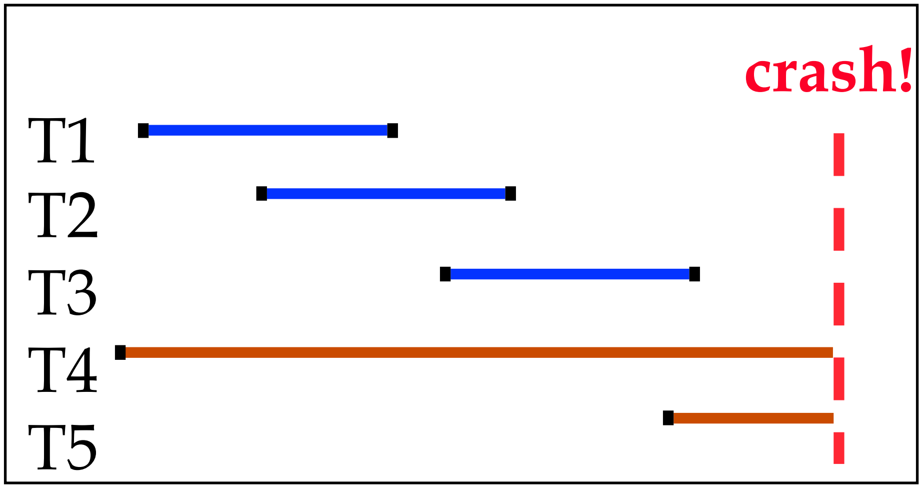
Slides Courtesy of R. Ramakrishnan and J. Gehrke


USER FRIENDLY by J.D. "Illiad" Frazer
YOU DIDN'T MAKE BACKUPS?
THE ANSWER TO THAT WOULD BE "NO."
COPYRIGHT © 2007 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
I JUST WANT TO BE ABSOLUTELY CLEAR: YOU DID NOT BACK UP DATA THAT WE NEED TO GENERATE REVENUE.
CORRECT.
AND WHAT EXACTLY WERE YOU PLANNING ON DOING IF YOU LOST ALL THAT CRUCIAL DATA?
...CALLING YOU.
mighty perky nana

Motivation

- **Atomicity:** *all-or-none*
  - Transactions may abort ("Rollback").
- **Durability:**
  - Effects of committed xacts should survive crashes.

- Desired Behavior after system restarts:
  - T1, T2 & T3 should be durable.
  - T4 & T5 should be aborted (effects not seen).



Assumptions

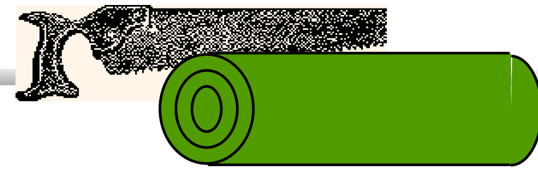
- Concurrency control is in effect.
  - Strict 2PL, in particular.
- Updates are happening "in place".
  - i.e. updates of an object are written from memory to the only copy of it of on disk.
- A simple scheme to guarantee Atomicity & Durability?

Handling the Buffer Pool

- Force every write to disk at commit time?
  - Provides durability.
  - Poor response time. Why?
  - No force, how can we ensure durability?
- Steal buffer-pool frames from uncommitted Xacts?
  - If not, poor throughput. Why?
  - If steal, how can we ensure atomicity?

	No Steal	Steal
Force	Trivial	
No Force		Desired

Basic Idea: Logging



- Log: A history of actions executed by DBMS
  - Records for REDO/UNDO information for every update

    <XID, pageID, offset, length, old data, new data>
  - All commits and aborts
  - And additional control info (which we'll see soon).

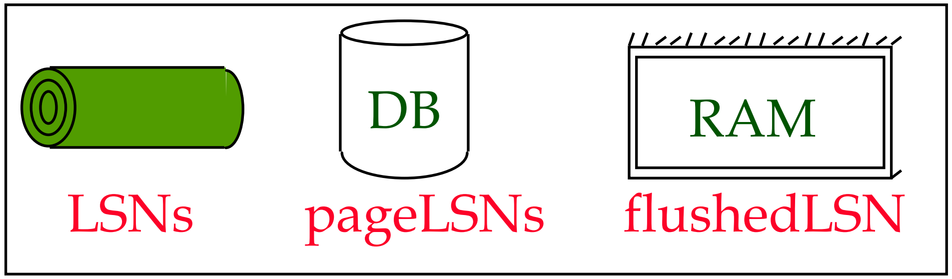
- Writing *log records* to disk is more efficient than *data pages*
  - Sequential writes to log (put it on a separate disk).
  - Minimal info written to log, often smaller than a data record; multiple updates fit in a single log page.

Write-Ahead Logging (WAL)

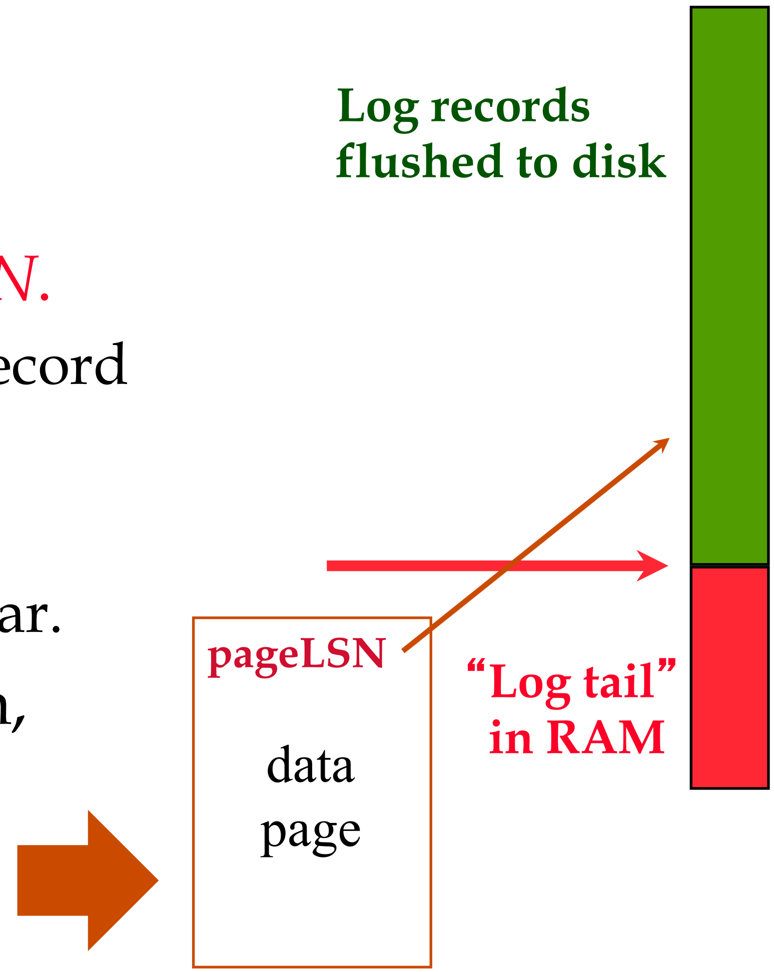
- Write-Ahead Logging (WAL) Protocol:
  1. Must force the **log record** for an update *before* the corresponding **data page** gets to disk (when steal).
     - Guarantees Atomicity
  2. Must write **all log records** for a Xact *before commit*.
     - Guarantees Durability

- Exactly how is logging (and recovery) done?
  - We'll study the **ARIES** algorithm.

ARIES



- Each log record has a unique *Log Sequence Number* (LSN).
  - LSNs always increasing.
- Each data page contains a *pageLSN.*
  - The LSN of the **most recent** log record for an update to that page.
- System keeps track of *flushedLSN.*
  - The **max LSN** written to disk so far.
- WAL (1): Before a page is written, flush its log record:
  - pageLSN ≤ flushedLSN



Big Picture: What's Stored Where

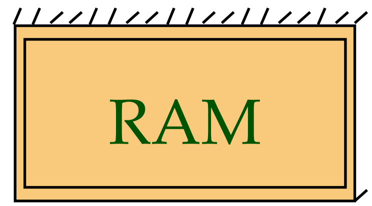
Data pages

each w. a pageLSN

master record

LogRecords

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



Xact Table

lastLSN
status

Dirty Page Table

recLSN

flushedLSN

1. Transaction Commit

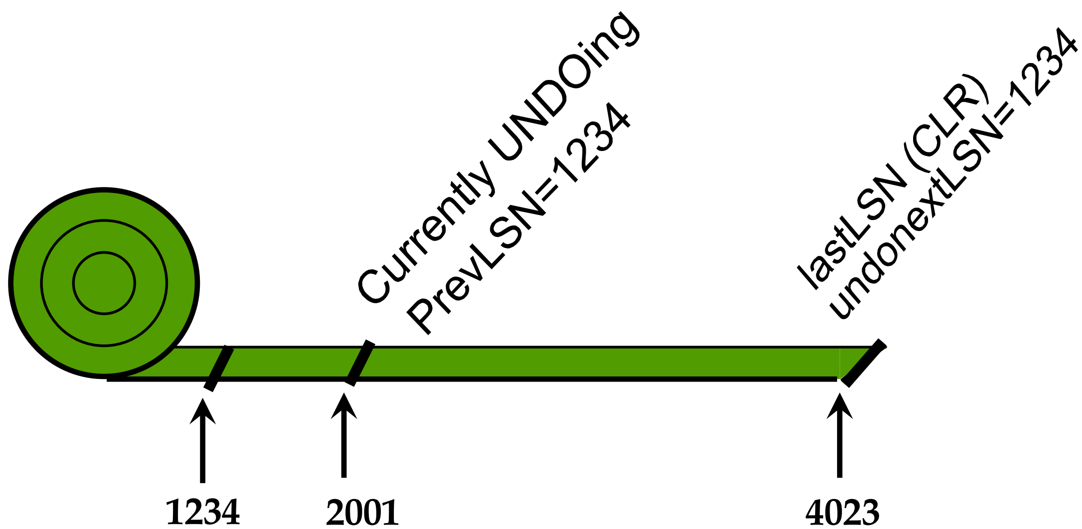
- When an Xact is running,
  - Generate a log record with LSN for each update operation.
  - Update Xact table and Dirty Page table in memory.
- Upon commit,
  - Write *commit* record to log.
  - WAL (2): All log records up to Xact's lastLSN are flushed.
    - Guarantees that flushedLSN ≥ lastLSN.
    - Log writes are sequential; many log records per log page.
  - When commit returns, write end record to log.

2. *Simple Transaction Abort*

- For now, consider an explicit abort of a Xact.
  - No crash involved.
- "Play back" the log in reverse order, UNDOing updates.
  - Get lastLSN of Xact from Xact table.
  - Follow chain of log records backward via prevLSN field.
    - To perform UNDO, also need to have a lock on data!
- Logging continues in UNDOs!

1. Before starting UNDO, write an *Abort* log record.
   - For recovering from crash during UNDO!

Abort (Contd.)



2. Before restoring old value of a page, write a *Compensation Log Record* (CLR):
   - CLR has one extra field, undonextLSN, pointing to the next LSN to undo (i.e. the prevLSN of the record we're undoing now).
3. At end of UNDO, write an *end* log record.

Example of Rollback

RAM

Xact Table (TT)
XID
lastLSN
status

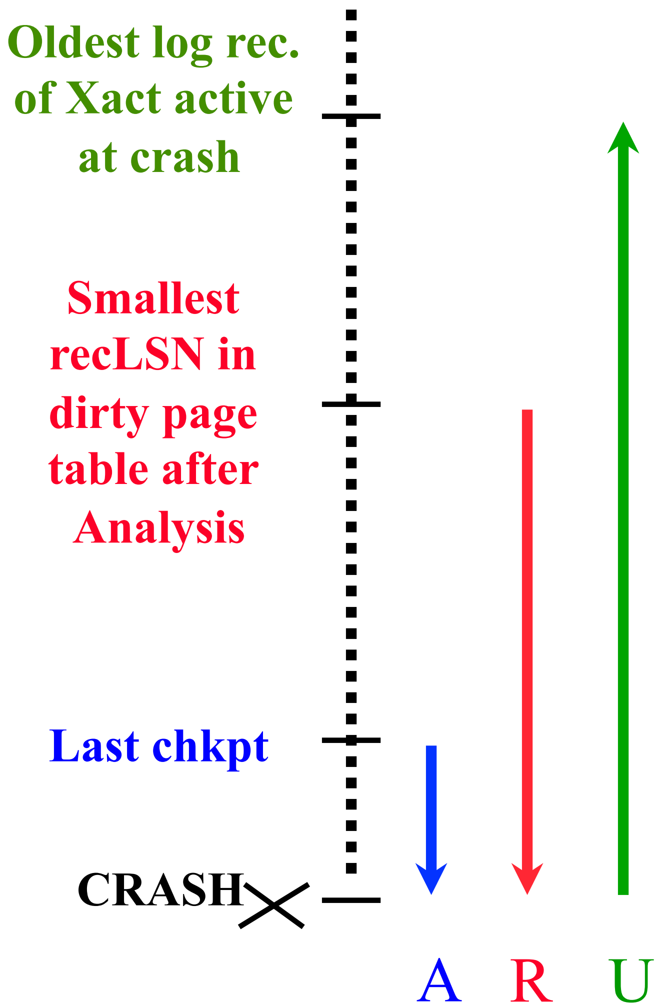
Dirty Page Table (DPT)
recLSN (earliest)

flushedLSN

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	update: T1 writes P2
40	T1 aborts
45	CLR: Undo T1 LSN 30
50	CLR: Undo T1 LSN 10
60	T1 ends

prevLSNs

3. Crash Recovery: Big Picture



- Analysis figures which Xacts had committed or failed at crash time.
  - *Checkpoint*: a snapshot of xact table and dirty page table.
  - Start from most recent checkpoint (via master record).
  - Construct Xact and Dirty Page tables
- REDO <u>all</u> actions; repeat history
  - Start from smallest recLSN in Dirty Page Table.
- UNDO effects of failed Xacts.
  - Back to oldest LSN of a running xact at crash.

(1) *Recovery: The Analysis Phase*

- First, reconstruct state at checkpoint.
  - Get *begin_checkpoint* record via the *master* recod.
  - Find its *end_checkpoint* record, read the xact table and dirty page table (DPT).
- Reconstruct Xact Table and Dirty Page Table at crash; scan log forward (from last checkpoint).
  - Xact Table (TT):
    - *End* record: Remove xact from Xact Table.
    - *Other records*: Add xact to Xact Table (if not there), set its lastLSN to this LSN, change xact status upon commit/abort.
  - Dirty Page Table (DPT):
    - *Update* record: If page P not in Dirty Page Table, add P to D.P.T., set its recLSN (earliest update) to this LSN.

(2) *Recovery: The REDO Phase*

- Repeat history to reconstruct *DB state at crash*:
  - Reapply all updates, even those of aborted Xacts and redo CLRs.

a. Scan forward from the log record containing smallest *recLSN* in D.P.T.

b. For each update or CLR log record, REDO the action.
   - Optimizations are in textbook, but not required in this class.
   - No additional logging!

(3) *Recovery: The UNDO Phase*

- Take a set of loser xacts, undo all in reverse order of LSN!

ToUndo = { *l* | *l*: lastLSN of a "loser" xact}

Repeat:

- Choose 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.

Until ToUndo is empty.

Example of Recovery

Atomicity & *Durability*:
none of T1, T2, T3 should have any effect on DB!

RAM

Xact Table (TT)
XID,
lastLSN,
status

Dirty Page Table (DPT)
recLSN (earliest)

flushedLSN

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 aborts
40	CLR: Undo T1 LSN 10
45	T1 ends
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

prevLSNs

Example of Recovery

Recovery algorithm:
ANALYSIS: TT & DPT?
REDO: T1, T2, T3
UNDO: T2, T3

RAM

Xact Table (TT)
lastLSN, status

T2: 60, running
T3: 50, running

Dirty Page Table (DPT)
recLSN (earliest)

P5: 10
P3: 20
P1: 50

flushedLSN

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update: T2 writes P3
30	T1 aborts
40	CLR: Undo T1 LSN 10
45	T1 ends
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

prevLSNs

Crash During Restart!

Atomicity & *Durability*:
none of T1, T2, T3 should have any effect on DB!

Recovery algorithm:
ANALYSIS: TT & DPT?
REDO: T1, T2, T3
UNDO: T2

RAM

Xact Table (TT)
XID,
lastLSN,
status

Dirty Page Table (DPT)
recLSN (earliest)

flushedLSN

LSN	LOG
00,05	begin_check
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
65,70	T2 abort, CLR: Undo T2 LSN 60
75,80,85	T3 abort, CLR: Undo T3 LSN 50, T3 end
	CRASH, RESTART

undonextLSN

Crash During Restart!

RAM

Xact Table (TT)
XID,
lastLSN,
status

Dirty Page Table (DPT)
recLSN (earliest)

flushedLSN

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
65,70	T2 abort, CLR: Undo T2 LSN 60
75,80,85	T3 abort, CLR: Undo T3 LSN 50, T3 end
	CRASH, RESTART
90,95	CLR: Undo T2 LSN 20, T2 end

undonextLSN

Main Principles behind ARIES

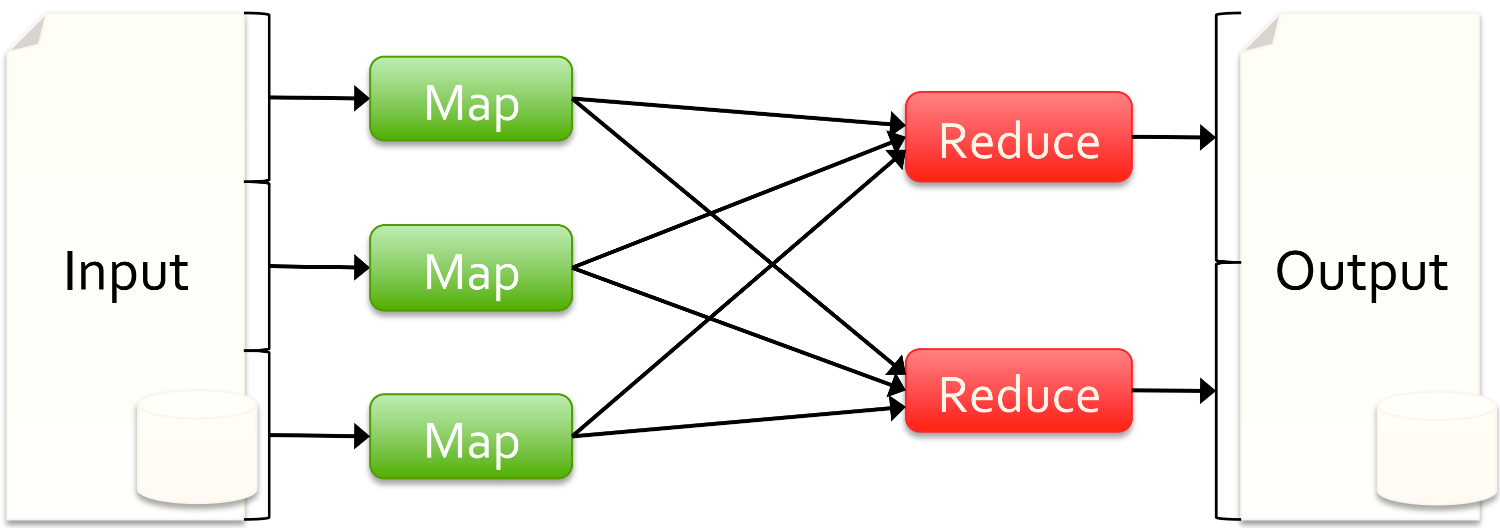
- Write-ahead logging (WAL)
  - Any change to an object is recorded in the log. The log is written to disk *before* the change to the object is written, or upon *commit*.
- Repeating history during REDO
  - On restart after a crash, repeat all actions before the crash, brings the system back to the exact state that it was in before the crash.
  - Then undo the xacts still active at crash time.
- Logging changes during UNDO
  - Changes made to DB while undoing are logged to ensure such an action isn't repeated in the event of repeated restarts.
- Fuzzy checkpointing to expedite recovery
  - An efficient way to create a snapshot of Xact Table & D.P.T.

NO! Logging + Replication + ARIES recovery (w. checkpoints)!

IBM DB2 Microsoft SQL Server ORACLE

Case Study: Fault Tolerance in Spark

- Current popular programming models for clusters transform data flowing from stable storage to stable storage
- E.g., MapReduce:



Questions