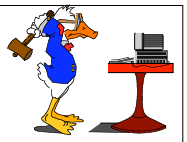
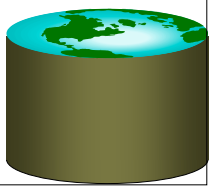
Crash Recovery, Part 1



R&G - Chapter 18

If you are going to be in the logging business, one of the things that you have to do is to learn about heavy equipment.

Robert VanNatta,
Logging History of Columbia County



Review: The ACID properties

- Atomicity: All actions in the Xact 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 a Xact commits, its effects persist.

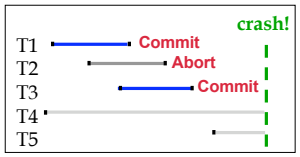
- Question: which ones does the Recovery Manager help with?

Atomicity & Durability (and also used for Consistency-related rollbacks)

Motivation

- **Atomicity:**
  - Transactions may abort ("Rollback").
- **Durability:**
  - What if DBMS stops running? (Causes?)

❖ Desired state after system restarts:
- T1 & T3 should be durable.
- T2, 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. data is overwritten on (deleted from) the actual page copies (not private copies).

- **Can you think of a simple scheme (requiring no logging) to guarantee Atomicity & Durability?**
  - What happens during normal execution (what is the minimum lock granularity)?
  - What happens when a transaction commits?
  - What happens when a transaction aborts?

Buffer Management plays a key role

	No Steal	Steal
No Force		Fastest
Force	Slowest	

Performance Implications

	No Steal	Steal
No Force	No UNDO REDO	UNDO REDO
Force	No UNDO No REDO	UNDO No REDO

Logging/Recovery Implications

Basic Idea: Logging

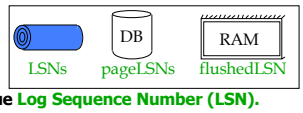
- **Desired properties:**
  - Sequential writes to log (put it on a separate disk).
  - Minimal info (diff) written to log, so multiple updates fit in a single log page.

- **Log: An ordered list of REDO/UNDO info**
- **Log record contains:**
  <XID, pageID, offset, length, old data, new data>
  - and additional control info (which we'll see soon).

Write-Ahead Logging (WAL)

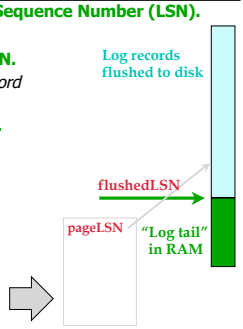
- **The Write-Ahead Logging Protocol:**
  1. Must force the log record for an update *before* the corresponding data page gets to disk.
  2. Must force all log records for a Xact *before commit*.
- **#1 (with UNDO info) helps guarantee Atomicity.**
- **#2 (with REDO info) helps guarantee Durability.**
- **This allows us to implement Steal/No-Force**
- **Exactly how is logging (and recovery!) done?**
  - We'll look at the ARIES algorithm from IBM.

WAL & the Log



- **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 flushed so far.
- **WAL: For a page i to be written must flush log at least to the point where:**

  $pageLSN_i \leq flushedLSN$



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)**
  - for UNDO actions
- End **(end of commit or abort)**

Other Log-Related State

- **In-memory table:**
- **Transaction Table**
  - One entry per currently active Xact.
    - entry removed when Xact commits or aborts
  - Contains XID, status (running/committing/aborting), and lastLSN (most recent LSN written by Xact).
- **Also: Dirty Page Table (will cover later …)**

Normal Execution of an Xact

- **Series of reads & writes, followed by commit or abort.**
  - We will assume that disk write is atomic.
    - In practice, additional details to deal with non-atomic writes.
- **Strict 2PL.**
- **STEAL, NO-FORCE buffer management, with Write-Ahead Logging.**

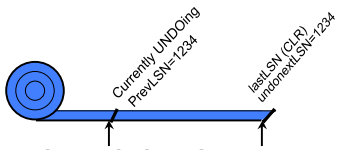
Transaction Commit

- **Write commit record to log.**
- **All log records up to Xact's commit record are flushed to disk.**
  - Guarantees that flushedLSN ≥ lastLSN.
  - Note that log flushes are sequential, synchronous writes to disk.
  - Many log records per log page.
- **Commit() returns.**
- **Write end record to log.**

Simple Transaction Abort

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

Abort, cont.



- **To perform UNDO, must have a lock on data!**
  - No problem (we're doing 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).
  - CLRs *never* Undone (but they might be Redone when repeating history: guarantees Atomicity!)
- **At end of UNDO, write an "end" log record.**

To be continued . . .