RECOVERY MANAGER

E0 261

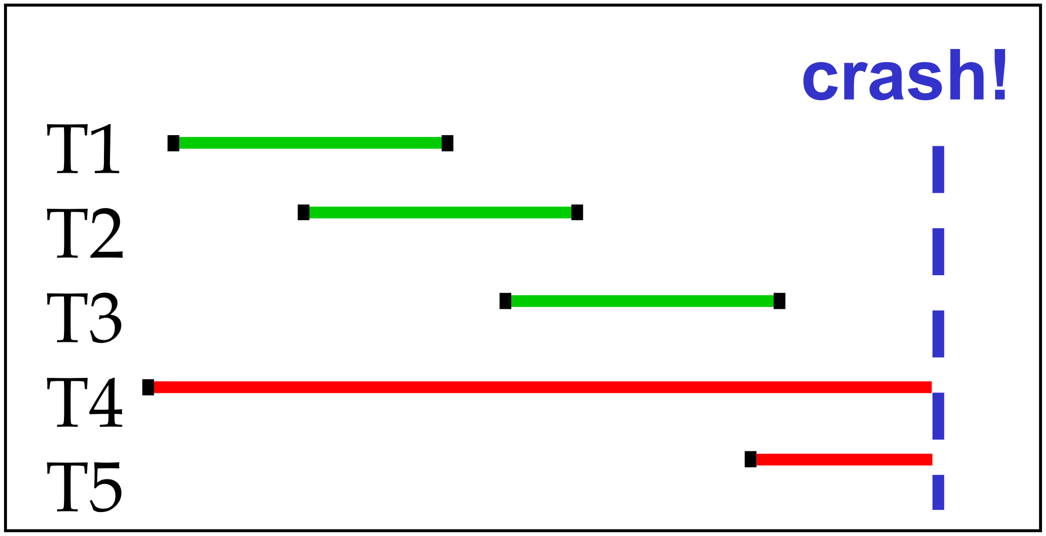
Jayant Haritsa
Computer Science and Automation
Indian Institute of Science

RECOVERY MANAGER

- Atomicity: All actions in the Xact happen, or none happen.
- Durability: If a Xact commits, its effects persist.

- The **Recovery Manager** guarantees these properties.

Example Scenario



- Desired Behavior after system restarts:
  - T1, T2 & T3 should be durable.
  - T4 & T5 should be erased.

Design Issues

- Database
  - Updates: In-place or to "shadow" copy ?
- Buffer Pool
  - Commit-Force: Force every memory data write to disk at commit time, or No-Force ?
  - Frame-Steal: Allow buffer frames of uncommitted Xsactions to be taken by others, or No-Steal?

Simple Solution

- Shadow updates (instant recovery)
- Force (provides durability),
  No-steal (provides atomicity)
- But,
  - Shadow results in fragmentation
  - Force results in poor response time
  - No-steal results in poor throughput

High-performance Solution

- In-place, No-force, Steal
- Mechanisms:
  - In-place : By using logging
  - No-force : By recording new value of P at commit time to support REDO of write to P
  - Steal : By recording old value of P at steal time to support UNDO of write to P

LOG

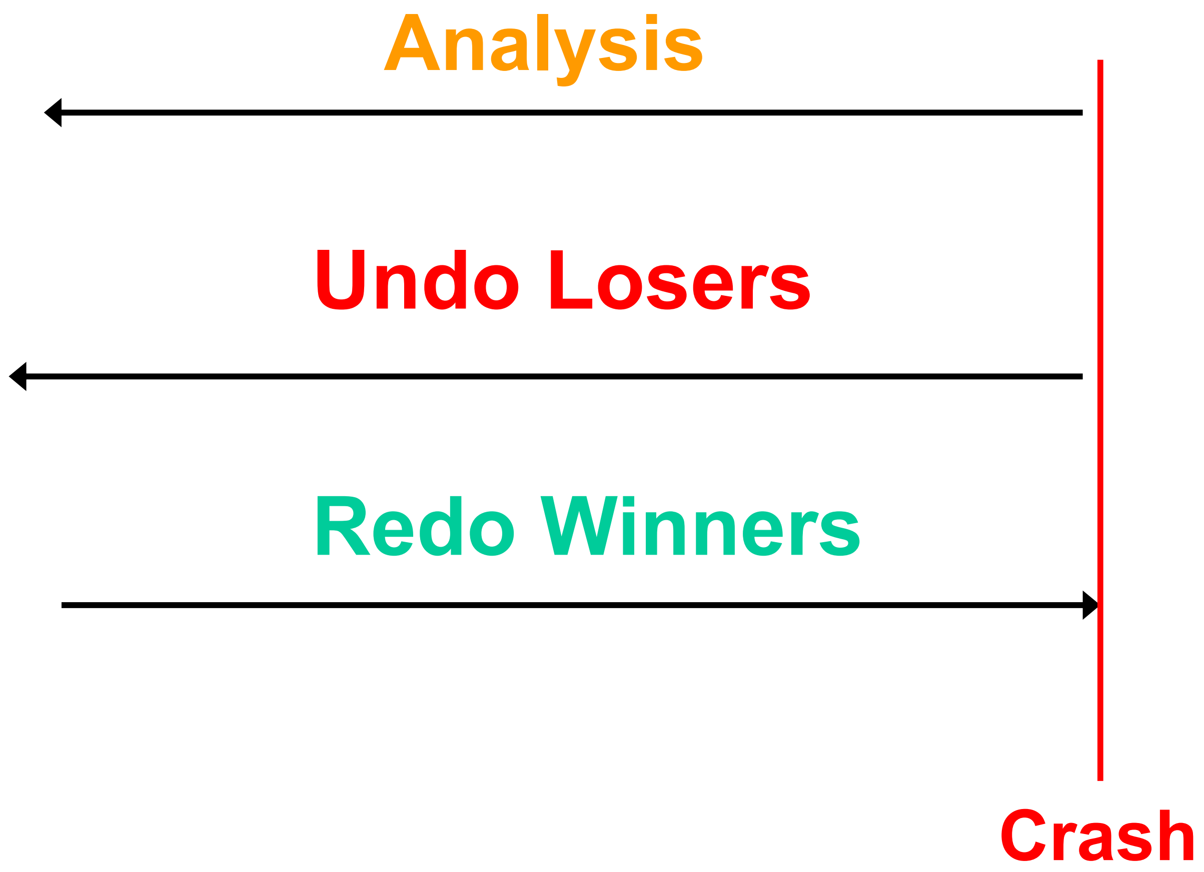
- A temporally-ordered list of REDO/UNDO actions
- One record for each update, containing <XID, pageID, offset, length, old data, new data>
  - <T420, 3873, 200, 1, B, A>
- Sequentially written to separate disk
- Several updates captured in single log page

Write-Ahead Logging (WAL)

- Must force the log record for an update before the corresponding data page gets to disk: guarantees Atomicity

- Must force all log records for a Xaction at commit: guarantees Durability.

Recovery Protocol



Assumptions

- Page-level locking
- Simple lock types (S, X)
- Physical logging (before-image, after-image)
  - Trivially guaranteed idempotency of undo and redo operations (ensuring no impact of crashes during the recovery process)

To Increase Concurrency

- Support operation logging
  - describe operation, not effects of operation
- Support fancy lock modes
  - e.g. increment/decrement
- Support fine-grained locking
  - record-level

Implications

- Because of operation logging, no longer trivially idempotent ! (e.g. repeated undo of an increment operation is not equal to that of single undo)
- Because of fancy lock types, the value of a data item may reflect the effect of multiple uncommitted updates from different transactions.
- Because of record-level locking, a page may simultaneously contain updates of (eventual) losers and winners.

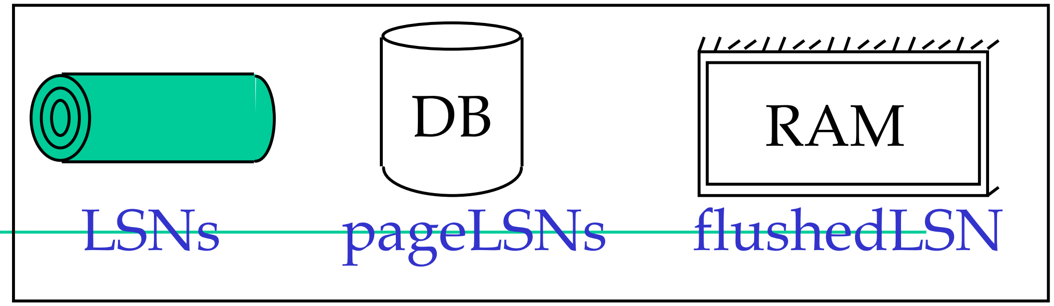
$\Rightarrow$ (Very) Careful design of recovery protocol

Solution: (SC) ARIES !

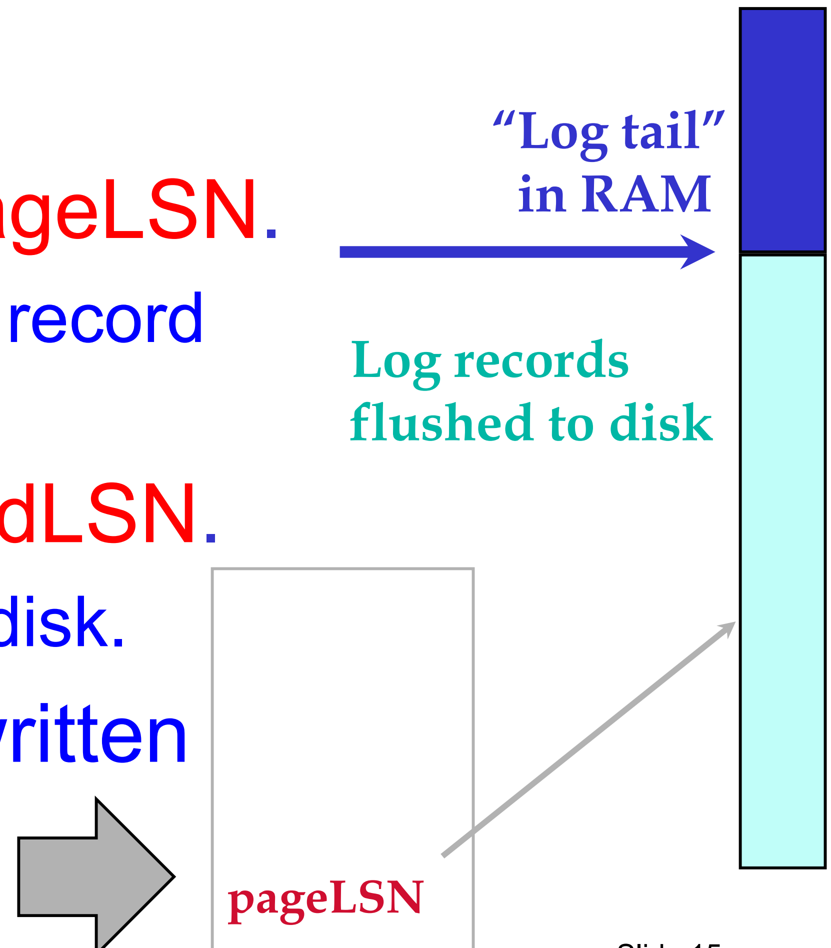
- Algorithm for Recovery and Isolation Exploiting Semantics
- C. Mohan (IBM Fellow)
- Industrial Strength Algorithm
- Implemented in several commercial products (e.g. IBM DB2) and research prototypes (e.g. Shore)
- Integrates well with other components of the system (data CC, index CC, ...)
- Main features: CLRs and REDO-ALL

The ARIES Method

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 to disk.
- WAL: Before a data page is written to disk, pageLSN ≤ flushedLSN



Log Records

LogRecord fields:

	type
	XID
	prevLSN
update & CLR records only	pageID length offset before-image after-image
CLRs only	UndoNxtLSN

Possible log record types:

- **Update**
- **Prepared** **(for distributed)**
- **Commit**
- **Abort**
- **End**
- Compensation Log Records
- Checkpoint Records

(Instead of before/after image, logical logging is also permitted)

Compensation Log Records

- CLRs are redo-only records of the undos of the updates of aborted transactions
- Explicitly provide idempotency by keeping track of the rollback status of a transaction
- Permits operation logging, page-oriented redo (for efficient recovery), and logical undo (high concurrency during normal processing)
- A NO-OP Xaction is equivalent to a committed "aborted + CLR" transaction, hence ensures uniform treatment of losers and winners

Transaction Table (TT)

- One entry per active Xaction
  - XID (transaction identifier)
  - status (running/prepared/committed/aborted)
  - lastLSN (latest log record written by Xaction)
  - UndoNxtLSN (LSN of next log record to be undone)

    Use: ensures no repetition of previously done work
- Meant for UNDO pass

Dirty Page Table (DPT)

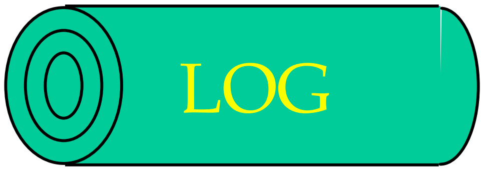
- One entry per dirty page in buffer pool
  - PageID (page identifier)
  - recLSN (LSN of first log record that caused the page to be dirty).

    Use: indicates earliest log record which might have to be redone
- Meant for REDO pass

Checkpointing

- Periodically, the DBMS creates a checkpoint, in order to minimize the time taken to recover in the event of a system crash. Write to log:
  - begin_checkpoint record: Indicates when checkpoint began.
  - end_checkpoint record: Contains current *transaction table* and *dirty page table*.
    - Other Xacts continue to run; so these tables are guaranteed to be uptodate only as of the time of the begin_checkpoint record.
    - No attempt to force dirty pages to disk; effectiveness of checkpoint limited by oldest un-forced change to a dirty page. (So it's a good idea to periodically flush dirty pages to disk!)
  - Store LSN of begin_checkpoint record in a safe place (*master* record).

The Big Picture: What's Stored Where



LogRecords

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

Data pages

- pageLSN

master record

Xact Table

- Xid
- lastLSN
- status
- undoNxtLSN

Dirty Page Table

- PageID
- recLSN

flushedLSN

Normal Execution of a Transaction

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

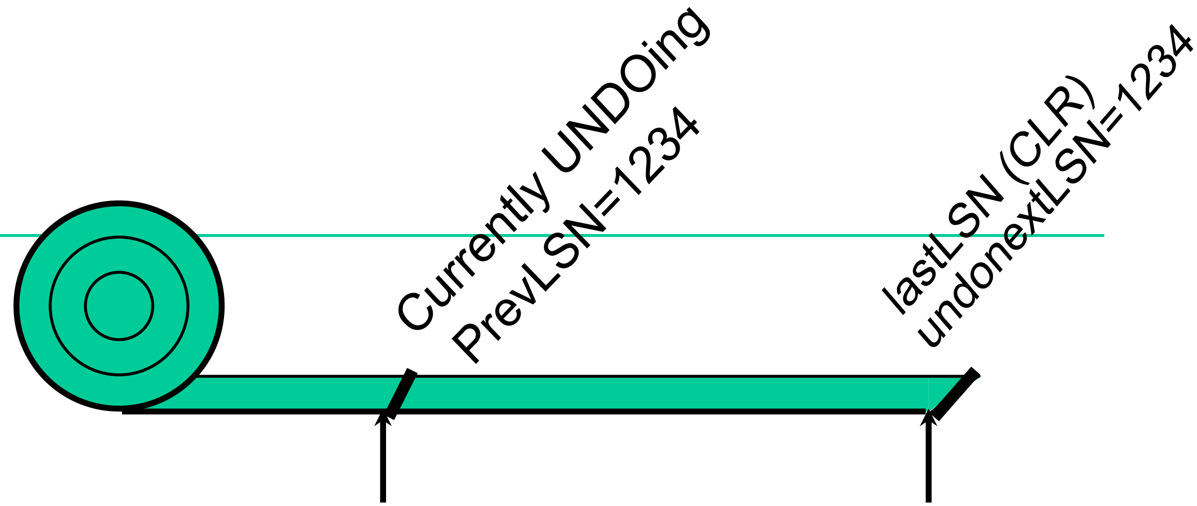
Transaction Commit

- Write commit record to log.
- All log records up to Xact's lastLSN are flushed.
  - Guarantees that flushedLSN $\geq$ lastLSN.
- Commit() returns.
- Write end record to log.

Simple Transaction Abort

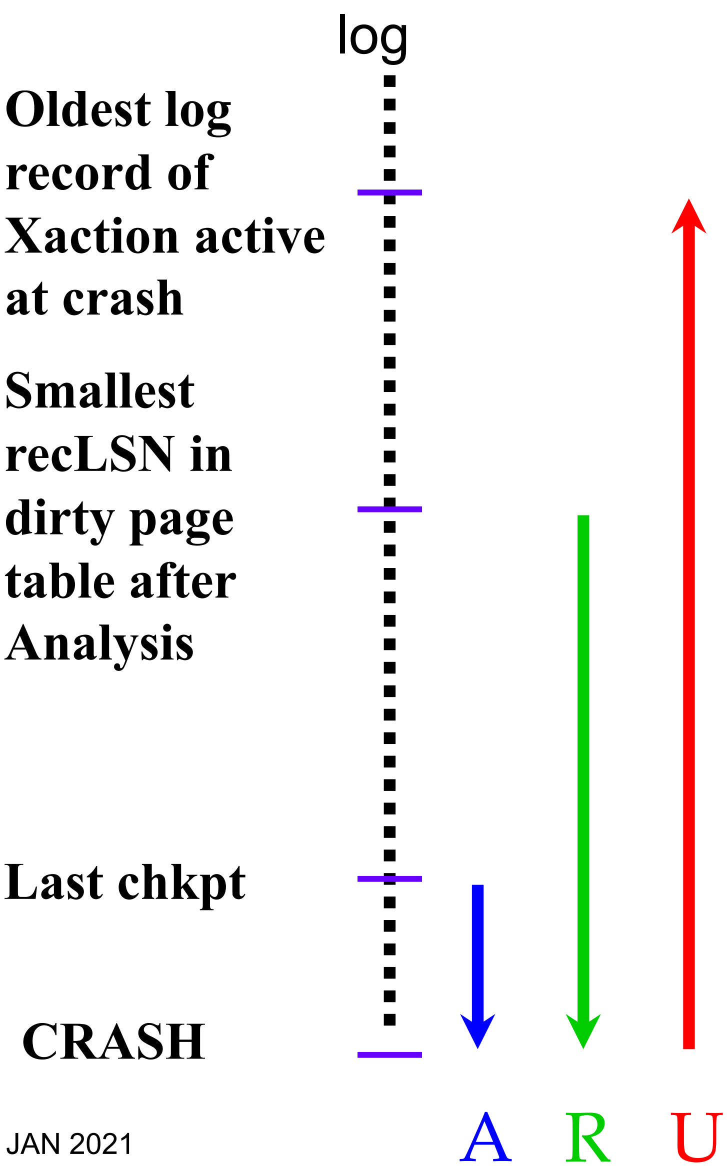
- Explicit abort of a Xaction, no crash involved
- Write an Abort log record.
- “Play back” the log in reverse order, UNDOing updates.
  - Get lastLSN of Xaction from TT .
  - Follow chain of log records backward via the prevLSN field.

Abort (contd)



- To perform UNDO, must have a lock on data!
  - No problem (because of 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.

Crash Recovery: Big Picture



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

Recovery: The Analysis Phase

- Reconstruct state at checkpoint.
  - via end_checkpoint record.
- Scan log forward from begin_checkpoint.
  - End record: Remove Xaction from TT.
  - Other records: Add Xaction to TT if not already there, set lastLSN=LSN, change Xaction status for control records.
    - Update record: If page P not in DPT, add P to DPT, set its recLSN=LSN.

Output of Analysis Phase

- TT is accurate as of crash, and gives the list of all transactions that were active at the time of the crash.
- DPT is also accurate as of crash, but may be "conservative" – may include some pages that may have been written to disk.
  - Could be eliminated by writing an end_write log record at the end of each data page write, but again CISC versus RISC argument holds.

Recovery: The REDO Phase

- We repeat History to reconstruct state at crash:
  - Reapply all updates including CLRs.
- Scan forward from log record containing smallest recLSN in DPT. For each CLR or update log record, REDO the action unless:
  - Affected page is not in DPT, or
  - Affected page is in DPT, but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN.
- To REDO an action:
  - Reapply logged action.
  - Set pageLSN = LSN . No additional logging!

Recovery: The UNDO Phase

ToUndo={ l | l is a lastLSN of a loser Xaction}

Repeat:

- Choose largest LSN among ToUndo.
  - If this LSN is a CLR and undonextLSN==NULL
    - Write an End record for this Xaction
  - If this LSN is a CLR, and undonextLSN != NULL
    - Add undonextLSN to ToUndo
  - If this LSN is an update. Undo the update, write a CLR, add prevLSN to ToUndo.
  - If this LSN is “abort” or “prepare”, add prevLSN to ToUndo.
- Remove LSN from ToUndo

Until ToUndo is empty.

Example of Recovery

Xact Table
- lastLSN
- status
- undoNxtLSN

Dirty Page Table
- recLSN

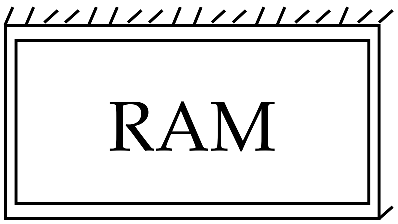
flushedLSN

ToUndo

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

prevLSNs

Example: Crash During Restart!



Xact Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

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?
  - Restart the Analysis phase again

  During REDO?
  - Some redos will not be redone since pageLSN will now be equal to update record's LSN.

- How to limit the amount of work in REDO?
  - Flush asynchronously in the background.

- How to limit the amount of work in UNDO?
  - Avoid long-running Xactions.

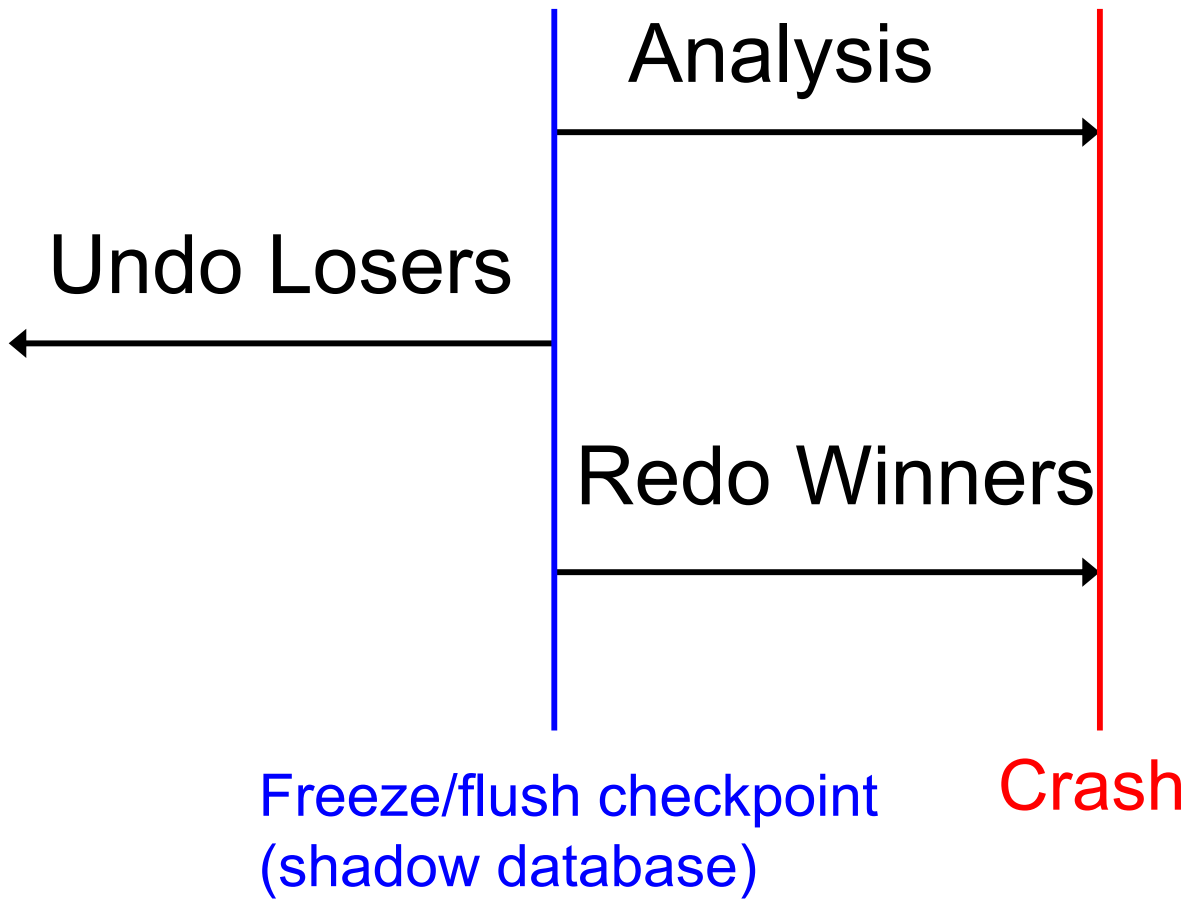
SUMMARY of ARIES PRINCIPLES

- WAL
- Repeating history during REDO
  - Make db accurate as of CRASH
- Logging changes (with CLRs) during UNDO
  - Bounded recovery effort

REDO-WINNERS PARADIGM

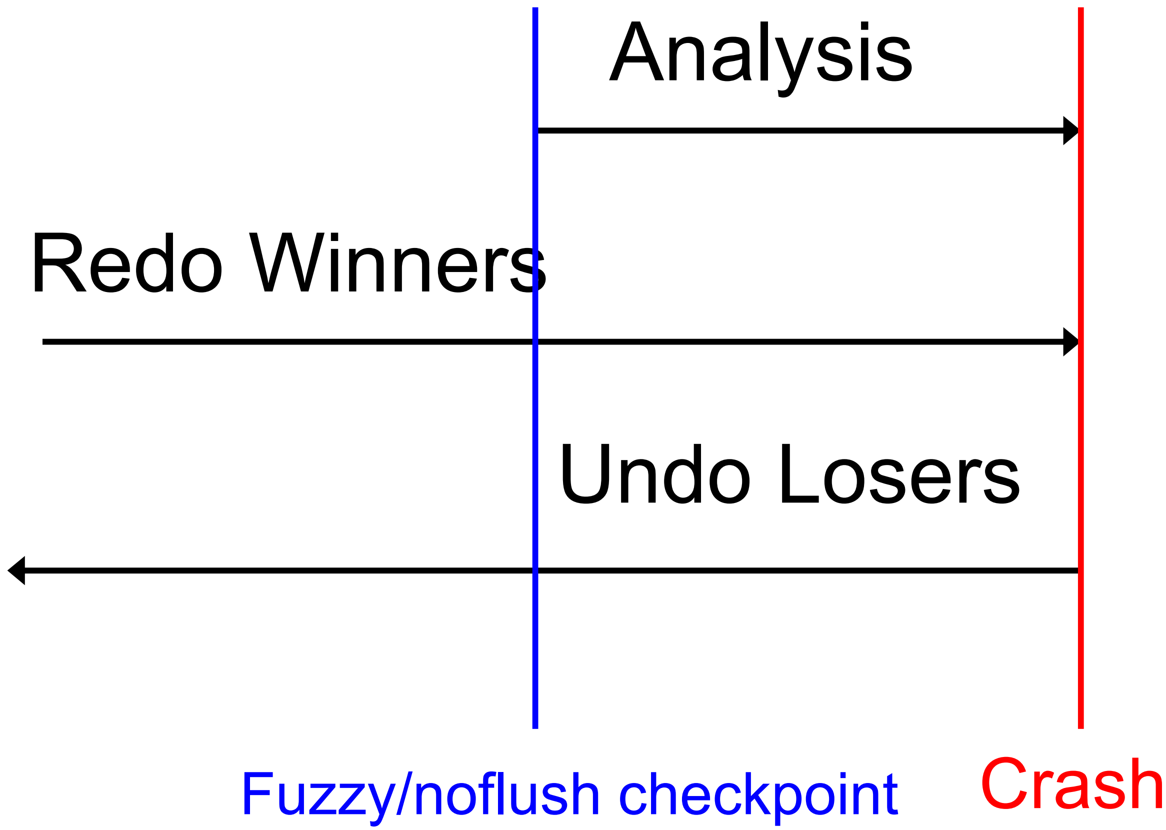
- Protocol
  - Analysis Phase
  - Undo Losers
    - correct database state of the past, not present !
  - Redo Winners
- Used in System R
- Variation used in DB2

System R



- Recreates state as of checkpoint

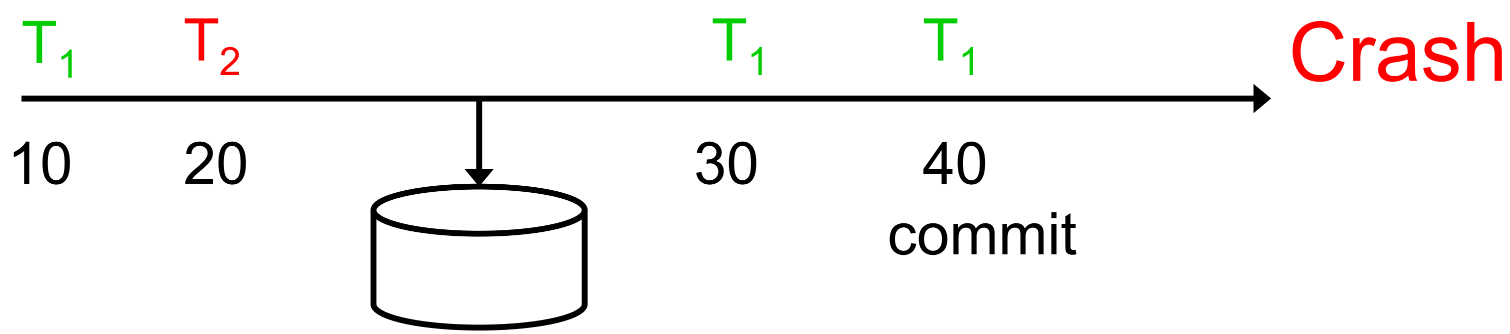
DB2 (Old scheme)



Problems with Redo-Winners instead of Redo-ALL

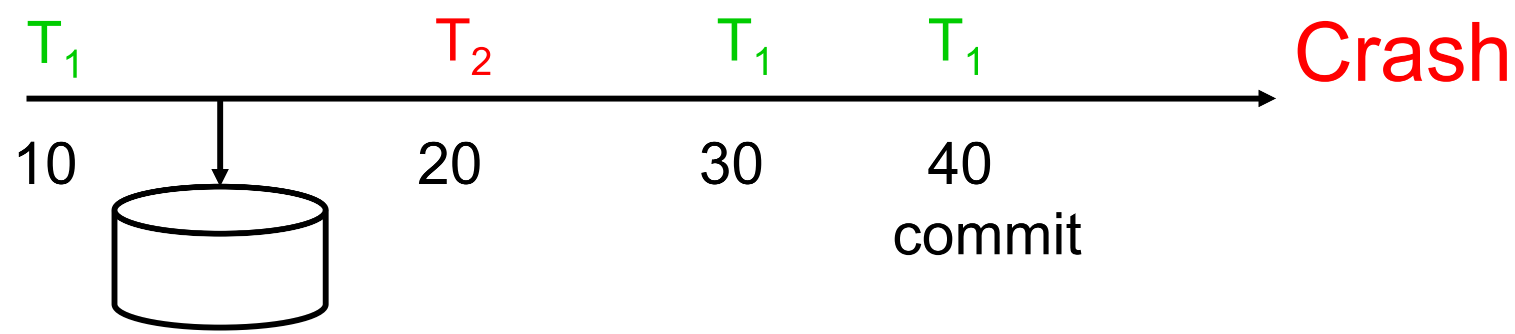
- Does not work with fine-granularity locking and operation logging and fuzzy checkpointing.
- Example scenario: Multiple updates to a page, some by a winner, some by a loser.

Selective Redo



Undo first, redo next: will fail to redo LSN 30 since LSN of page = 50 = LSN of CLR (20), although LSN 30 should be on page

Selective Redo (contd)



Redo first, undo next: will perform undo of LSN 20 since LSN of page = 30, although LSN 20 is not on page

Main Issue

- Basically, the page_LSN is no longer a true indicator of the current state of the page.

END TRANSACTION MANAGEMENT

E0 261

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 CLRs.
- Redo “repeats history”: Simplifies the logic!