CompSci 516
Database Systems

Lecture 17
Transactions – Recovery
(ARIES)

Instructor: Sudeepa Roy

Announcements

- Midterm report due on Wednesday, 11/01
- HW3 to be released soon
  – will be due in ~2 weeks after it is released

Reading Material

- Main reading
  "Concurrency Control and Recovery"
  Michael Franklin, 1997
  – Chapters 2.2, 3.2
  – A very good read for other topics on transactions as well
  – pdf on the website
- [RG]
  – Chapter 18.1-18.6

Acknowledgement: Slides by Prof. Magda Balazinska were consulted to prepare some of these slides.

Last Lecture

- UNDO
- REDO
- UNDO/REDO
- Checkpointing
- Recovery
- The main concepts of these logs were discussed

Today

- Last topic in recovery
- Physical/Logical/Physiological Logging
- ARIES protocol by IBM
  – An efficient implementation of UNDO/REDO log
  – Including the data structures that are used for the recovery

Log: old/new concepts

- Each entry in the log is called a log record
  – e.g. recall <T, X, u, v> for undo/redo log
- When a log record is created, it is assigned a unique Log Sequence Number (LSN)
  – typically, monotonically increasing to provide relative position in the log
- Update to a data item in buffer:
  – a log record is created
  – Many systems write the LSN of this log record into the page containing the data item
    - relates the state of a data page to logged updates
- How do these log records look in practice?

WAL for UNDO/REDO log

- Write Ahead Log (WAL)
  - from Lecture 16
- All log records for an update are first written to disk before the update (the modified page) is written to disk
- A transaction is not considered committed, until all its log records + COMMIT record are on disk
- Allows STEAL + NO FORCE (good!)

Physical Logging

- Physical log records indicate location of modified data in the database
  - e.g. position on a particular page
  - if a new tuple is inserted in a relation, log records may contain changes for
    - space allocation
    - index updates
    - reorganization etc.

Logical Logging

- Logical log records indicate high level info about operations performed
  - if a new tuple is inserted in a relation, log records may indicate
    - the insertion has taken place
    - value of the inserted tuple

Physical vs. Logical Logging

Logical logging

- Advantages
  - Minimizes the amount of data that must be written to the log
  - Hides many implementation details and complex operations under UNDO/REDO logic
- Disadvantages
  - Difficult to implement, as logging operations may not be atomic

Physiological Logging

- Log records are constrained to refer to a single page
  - but may reflect logical operations on that page
- e.g. for insert on a page,
  - specify the value of the tuple that is inserted
  - do not specify any free-space manipulation
- Tradeoff between physical and logical
  - atomic like physical and less logging records like logical

ARIES protocol

- A detailed but simple implementation details of logging protocol
  - Developed in IBM, but now used in many DBMS

ARIES : Main Ideas

- Write Ahead Logging (WAL)
  - before an update goes to the disk, the log record for that update must go to the disk
- Physiological Logging
- Page-oriented REDO
  - REDO operations involve pages
  - the affected page is specified in the log record
- Logical UNDO
  - operations performed to undo an update do not need to be the exact inverses of the operations of the original update
  - e.g. if T1 updates an index entry in page P1, later P1 is split (dynamic hashing!) and entry is moved to P2, we do not want to do UNDO operation on P1, but on the new location P2

Next – in detail

ARIES Data Structures

Dirty page table

pageID	recoveryLSN

Log

LSN	prevLSN	transID	pageID	Log entry	Type	undoNextLSN

Buffer Pool

P500 A = ... B = ... PageLSN= -		P600 B = ... PageLSN= -
	P505 C = ... PageLSN= -	

Transaction table

transID	lastLSN	Status

ARIES Data Structures: Page

- Buffer pool contains multiple page
- Each page contains a pageLSN = the LSN of the log record for the latest update to the page
  - used during recovery to determine whether or not an update for a page has to be UNDOne
  - also determines the point in the log from which the REDO pass must commence during recovery

Buffer Pool

P500 A = ... B = ... PageLSN= -		P600 B = ... PageLSN= -
	P505 C = ... PageLSN= -	

ARIES Data Structures: Transaction Table

- Contains status information about each transaction that is currently running
- transID (later tID)
  - unique transaction ID
- lastLSN for each transaction
  - LSN of the most recent log record written by the transaction
- Status
  - Running/Committed/Aborted/…
  - Unknown (while recovery)

Transaction table

transID	lastLSN	Status

ARIES Data Structures: Dirty Page Table

Dirty page table

pageID	recoveryLSN

- Contains an entry for each dirty page
- dirty page = contains an update that is not written to disk yet
- recoveryLSN
  - LSN of the earliest log record that might need to be "REDO"ne for the page during restart
- Recall: we care about dirty pages in memory only for REDO, not for UNDO

ARIES Data Structures: Log

- LSN
  - unique id of log in increasing sequence
- transID (later tID)
  - Id of the transaction making the changes
- pageID (later pID)
  - which page is being modified
- Log entry
  - actual changes
  - e.g. WRITE A : "ab" -> "cd"

Log

later

LSN	prevLSN	transID	pageID	Log entry	Type	undoNextLSN

- Note: ARIES is UNDO/REDO
  - maintains both previous and new value
  - i.e. everything in <T, A, u, v> is being maintained

ARIES Data Structures: Log

prevLSN
- Log records belonging to the same transaction are linked backwards in time using a field in each log record
- when a new log record is written
  - the value of the lastLSN field from the Transaction table is written as prevLSN
  - new record's LSN is entered as lastLSN in the Transaction table

Log

LSN	prevLSN	transID	pageID	Log entry	Type	undoNextLSN

Transaction table

transID	lastLSN	Status

ARIES Data Structures: Log

Type
- Update
  - e.g.
  WRITE A : "ab" -> "cd"
- Commit
- Abort
- END
- CLR

- Details later

Log

LSN	prevLSN	transID	pageID	Log entry	Type	undoNextLSN

Checkpointing in ARIES

- Checkpoints are periodically taken
- ARIES uses a form of fuzzy checkpoint that is extremely inexpensive
- When a checkpoint is taken
  - a checkpoint record is constructed
  - includes the contents of the Transaction Table and Dirty Page Table
- Checkpoints are efficient
  - no operation is quiesced (stalled)
  - no database pages are flushed to disk from memory!
- But the log that has to be maintained is not much reduced
  - limited in part by the earliest recoveryLSN of the dirty pages at the checkpointing time
  - writing dirty pages periodically to disk might help

Running Example:
Maintaining Data Structures

Example actions

Example.
1. T_{1000} changes the value of A from "abc" to "def" on page P500
2. T_{2000} changes the value of B from "hij" to "klm" on page P600
3. T_{2000} changes the value of D from "mnp" to "qrs" on page P500
4. T_{1000} changes the value of C from "tuv" to "wxy" on page P505
5. T_{2000} commits and the END log record is written
6. T_{1000} changes the value of E from "pq" to "rs" on page P700
7. P600 is flushed to disk
8. Crash!!

Example is adapted from [RG]

ARIES Data Structures

Dirty page table

pageID	recoveryLSN

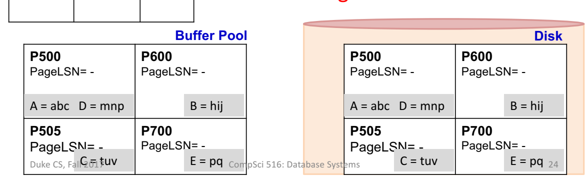
Transaction table

transID	lastLSN	status

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101						

initial configuration



First operation:

1. T_{1000} changes the value of A from "abc" to "def" on page P500?

Dirty page table

pageID	recoveryLSN

Transaction table

transID	lastLSN	status

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101						

Buffer Pool

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Changes

1. T_{1000} changes the value of A from "abc" to "def" on page P500

Dirty page table

pageID	recoveryLSN
P500	101

Transaction table

transID	lastLSN	status
T_{1000}	101	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Next:

2. T_{2000} changes the value of B from "hij" to "klm" on page P600 ?

Dirty page table

pageID	recoveryLSN
P500	101

Transaction table

transID	lastLSN	status
T_{1000}	101	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Changes:

2. T_{2000} changes the value of B from "hij" to "klm" on page P600 ?

Dirty page table

pageID	recoveryLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	102	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	**P600** PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Next:

3. T_{2000} changes the value of D from "mnp" to "qrs" on page P500?

Dirty page table

pageID	recoveryLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	102	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	**P600** PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Changes:

3. T_{2000} changes the value of D from "mnp" to "qrs" on page P500

Dirty page table

Unchanged

pageID	recoveryLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	103	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Next:

4. T_{1000} changes the value of C from "tuv" to "wxy" on page P505?

Dirty page table

pageID	recoveryLSN
P500	101
P600	102

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	101	Running
T_{2000}	103	Running

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Changes:

4. T_{1000} changes the value of C from "tuv" to "wxy" on page P505?

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	103	Running

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Next:

5. T_{2000} commits and the end log record is written

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	103	Running

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Pink = On disk | White = In memory

Changes:

5. T_{2000} commits and the end log record is written --- step 1

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
T_{2000}	105	Committed

Log written to disk
Note: no force = not the dirty pages changed by T_{2000}! – less cost

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Pink = On disk | White = In memory

Changes:

5. T_{2000} commits and the end log record is written --- step 2

Dirty page table

pageID	recoveryLSN
P500	101
P600	
P505	104

T2000 removed from transaction table

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102			P600	Write B "hij" -> "klm"	Update	-
		T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Transaction table

transID	lastLSN	status
T_{1000}	104	Running
~~T_{2000}~~	~~105~~	~~Committed~~

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

assume an extra flush log

Pink = On disk | White = In memory

Changes:

5. T_{2000} commits and the end log record is written --- step 3

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Transaction table

transID	lastLSN	status
T_{1000}	104	Running

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	P600 PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	P700 PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= - B = hij
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

- Whenever a transaction commits, log is flushed to the disk: i.e the "log-tail" (whatever is not on disk) is written to disk
  Assume a "force-write" of log after "commit" is written

  The dirty pages are not needed to be flushed to disk (NO-FORCE)

NOTE:
1. The "Commit" record is required to be flushed (i.e. all logs up to and including that commit record)
2. The "End" record is not required to be flushed, in this case we are only assuming that it has been flushed as well
 (so that we have a good example while doing recovery)

Log Record "Types"

- **Update**: standard
- **Commit**: log-tail forced-written to disk, up to & including commit
  (note that still no-force, the actual modified pages may not be written, and much smaller cost)
- **Abort**: abort type log record is written + undo is initiated for this transaction
- **End**: when a transaction is aborted or committed, some additional actions are performed, after that an end record is written
- **CLR: (later)**
  Undoing updates (during abort or recovery from crash),
  for every update record undone, write a CLR (Compensation Log Record)

Next:

6. T_{1000} changes the value of E from "pq" to "rs" on page P700

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T_{1000}	104	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	**P700** PageLSN= - E = pq

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Changes:

6. T_{1000} changes the value of E from "pq" to "rs" on page P700

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104
P700	107

Transaction table

transID	lastLSN	status
T_{1000}	107	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107	104	T_{1000}	P700	Write E "pq" -> "rs"	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	**P700** PageLSN= 107 E = rs

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Next:

7. Page P600 is flushed to disk

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104
P700	107

Transaction table

transID	lastLSN	status
T_{1000}	107	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107	104	T_{1000}	P700	Write E "pq" -> "rs"	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	**P700** PageLSN= 107 E = rs

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= - B = hij
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Next:

7. Page P600 is flushed to disk

Dirty page table

pageID	recoveryLSN
P500	101
~~P600~~	~~102~~
P505	104
P700	107

Transaction table

transID	lastLSN	status
T_{1000}	107	Running

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107	104	T_{1000}	P700	Write E "pq" -> "rs"	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	**P700** PageLSN= 107 E = rs

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Next:

7. Page P600 is flushed to disk (after)

Dirty page table

pageID	recoveryLSN
P500	101
P505	104
P700	107

Transaction table

transID	lastLSN	status
T_{1000}	107	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107	104	T_{1000}	P700	Write E "pq" -> "rs"	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN= 102 B = klm
P505 PageLSN= 104 C = tuv	**P700** PageLSN= 107 E = rs

Disk

P500 PageLSN= - A = abc D = mnp	**P600** **PageLSN= 102** **B = klm**
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Flush page P600 to disk

1. Write current content of P600 to disk (along with pageLSN)
2. Remove from Dirty Page table

NOTE: Write Ahead Log

1. All LSNs changing that page must be written to disk
2. In this case it is okay, since the last (not flushed) log record involves P700 while P600 is being flushed (LSN 102 is already on disk)
3. When a page is written, we need to ensure that all log records up to the pageLSN are on disk

-- still no force (only logs – that are much smaller than set of pages changed by the transaction are written to disk)
-- further, log is maintained as asequential file – much cheaper write

Note: Log is always written to disk in order, i.e. we can never skip some log entries in between

Checkpointing at ARIES

- Like before -- <START CKPT> and <END CKPT>
- Writes
  - Transaction table
  - Dirty page table
  - the state as of the time of <START CKPT>
- Called "fuzzy checkpointing"
  - Non-quiescent : new transactions can start
  - Does not require pages in buffer pool to be written
  - But effectiveness limited to earliest possible "recoveryLSN" in the dirty page table – has to start REDO from there
- Periodically writing dirty pages to disk helps
- After checkpointing, both transaction table and dirty page tables are empty

8. Crash!! ---- These are gone from memory

Dirty pa ble

P505	
P700	106

Transaction table

transID	lastLSN	status
		g

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107						

Buffer Pool

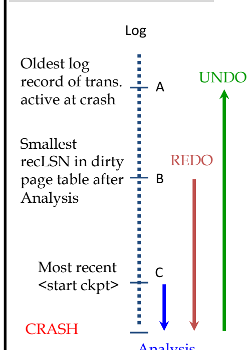
P500 PageLSN= 103 A = de	
P505 PageLSN C = tuv	N= 107 E = rs

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

Crash Recovery: Three Phases - Big Picture

Note: the order of A, B, C may vary



1. **Analysis**
   - Start from last checkpoint (from C)
   - Go forward until the last log record
   - Figure out which trans. committed since checkpoint, which failed
   - Reconstructs (not exact) dirty page table and transaction table
2. **REDO**
   - repeat history in forward direction
   - start with the earliest recoveryLSN returned by Analysis phase (from B)
   - redo all changes to all pages that "might have been dirty"
3. **UNDO**
   - undo effects of active transactions at crash returned by Analysis phase
   - go in backward direction (until A)

1. Analysis Pass

Analysis Pass

- It has a threefold job:

1. Determines the point in the log at which to start the REDO pass
2. Determines which pages "could have been" dirty at the time of the crash to avoid unnecessary I/O in the REDO pass
   – a conservative superset of actual dirty pages
3. Determines which transactions had not committed at the time of the crash and will therefore need to be UNDOne.

Analysis Pass: Details

- Begin at the most recent checkpoint
- Reconstruct Dirty Page Table and Transaction Table
  – to determine the state of the system at the time of crash
- Scan forward to the end of the log
  – Contents of these two tables are modified according to the log records encountered in the forward scan

Read yourself later after seeing examples

Analysis Pass: More Details

- When a log record for a transaction that does not appear in the Transaction Table is encountered
  – that transaction is added to the transaction table
- When an END record is encountered
  – that transaction is removed from the transaction table
- When an UPDATE log record for a page not in the Dirty Page Table is encountered
  – that page is added to the dirty page table
  – LSN of the record is recorded as recoveryLSN for that page
  – LastLSN is modified
- All like before!

Running Example: Analysis Pass

Checkpointing in the example

- This example has no checkpointing == Checkpointing at the beginning
- Analysis phase in the recovery starts with empty Dirty Page table and empty Transaction Table
  - If checkpoint was available, the latest copies of these tables have to be read from disk from the last checkpoint

Analysis Pass

Dirty page table

pageID	recoveryLSN

Transaction table

transID	lastLSN	status

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Buffer Pool

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Will not show the buffer pool from the next slide

Analysis Pass

Dirty page table

pageID	recoveryLSN
P500	101

Transaction table

transID	lastLSN	status
T1000	101	U= Unknown

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Analysis Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T1000	101	U
T2000	102	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Analysis Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102

Transaction table

transID	lastLSN	status
T1000	101	U
T2000	103	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Analysis Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U
T2000	103	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Analysis Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U
T2000	105	C

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Write A or Abort if you see an Abort log instead

Disk

P500 PageLSN= - A = abc D = mnp	P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv	P700 PageLSN= - E = pq

Analysis Pass

Log

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U
~~T2000~~	~~105~~	~~C~~

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T_{1000}	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp
P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv
P700 PageLSN= - E = pq

Remove entry from Transaction Table if you see an End record (both for Aborted and Committed transactions)

Analysis Pass

Log

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T_{1000}	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Already written to disk, but reappears (conservative construction of Dirty Page Table)

Disk

P500 PageLSN= - A = abc D = mnp
P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv
P700 PageLSN= - E = pq

Compare previous slide with Dirty Table and Transaction Table “right before Crash”

Lost update during crash, but write ahead log, so safe!

Dirty page table

pageID	recoveryLSN
P500	101
P505	104
P700	107

Transaction table

transID	lastLSN	status
T_{1000}	107	Running

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A “abc” -> “def”	Update	-
102	-	T_{2000}	P600	Write B “hij” -> “klm”	Update	-
103	102	T_{2000}	P500	Write D “mnp” -> “qrs”	Update	-
104	101	T_{1000}	P505	Write C “tuv” -> “wxy”	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107	104	T_{1000}	P700	Write E “pq” -> “rs”	Update	-

Buffer Pool

P500 PageLSN= 103 A = def D = qrs
P505 PageLSN= 104 C = tuv
P700 PageLSN= 107 E = rs

Disk

P500 PageLSN= - A = abc D = mnp
P600 PageLSN= 102 B = klm
P505 PageLSN= - C = tuv
P700 PageLSN= - E = pq

Analysis Pass: NOTE

- At the end of this pass,
  - the Dirty Page Table is conservative
  - Some pages may already have been flushed to disk
  - It lists all pages that “could have been” dirty at the time of crash
  - Transaction Table contains entries for transactions that would “actually” require UNDO

2. REDO Pass

ARIES: REDO Pass

- REDO in ARIES = Repeating History
- REDO updates for all transactions
  - committed as well as for transactions to be aborted in UNDO
  - both “UPDATE” and “CLR” records (later)
- at the end of REDO, the database would be in the same state w.r.t. logged updates at the time of crash

When REDO is NOT needed

1. If the affected page is not in the Dirty Page Table
   – all changes to this page was written to disk
2. Otherwise, the page's recoveryLSN > LSN of the record being checked
   – RecoverLSN is the first update to a page that may not have been written to disk
   – current update has gone to disk
3. Otherwise, the pageLSN >= LSN of the record being checked
   – may need to load the page from disk (stored on the page)
   – checked last because it needs a page I/O
   – either this update or a later update was already written to disk

ARIES: REDO Pass - details

- Work with the Dirty Page Table
- Find the smallest recoveryLSN in the dirty page table = FirstLSN
  - from the analysis phase
- Redo the "Update" (and "CLR" - later) actions, unless (in this order)
  - Affected page is not in the dirty page table
  - Or, recoveryLSN > LSN being checked
  - Or, pageLSN >= LSN being checked
- End/Commit/Abort LSNs are "skipped"

Running Example: REDO Pass

State After Analysis Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500	P600
PageLSN= -	PageLSN=102
A = abc D = mnp	B = klm
P505	**P700**
PageLSN= -	PageLSN= -
C = tuv	E = pq

REDO Pass: find firstLSN

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500	P600
PageLSN= -	PageLSN=102
A = abc D = mnp	B = klm
P505	**P700**
PageLSN= -	PageLSN= -
C = tuv	E = pq

firstLSN:
- The earliest recoveryLSN of all the entries in the Dirty Page Table spot in the log from which to begin the REDO pass

REDO: Step 1

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Buffer Pool

P500	
PageLSN= "-" to 101	
A = def D = mnp	

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500	P600
PageLSN= -	PageLSN=102
A = abc D = mnp	B = klm
P505	**P700**
PageLSN= -	PageLSN= -
C = tuv	E = pq

- Affected page is not in the dirty page table: **N**
- Else, recoveryLSN > LSN being checked: **N**
- Else, pageLSN >= LSN being checked: **N**
- **REDO**

REDO: Step 2

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Buffer Pool

P500 PageLSN= 101 A = def D = mnp	**P600** PageLSN=102 B = klm

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN=102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

- Affected page is not in the dirty page table: **N**
- Else, recoveryLSN > LSN being checked: **N**
- Else, pageLSN >= LSN being checked: **Y**
- **NO REDO = SKIPPED**

REDO: Step 3

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Buffer Pool

P500 PageLSN= 101 to 103 A = def D = qrs	**P600** PageLSN=102 B = klm

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN=102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

- Affected page is not in the dirty page table: **N**
- Else, recoveryLSN > LSN being checked: **N**
- Else, pageLSN >= LSN being checked: **N**
- **REDO**

REDO: Step 4

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Buffer Pool

P500 PageLSN= 103 A = def D = qrs	**P600** PageLSN=102 B = klm
P505 PageLSN="" to 104 C =wxy	

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	

Disk

P500 PageLSN= - A = abc D = mnp	**P600** PageLSN=102 B = klm
P505 PageLSN= - C = tuv	**P700** PageLSN= - E = pq

- Affected page is not in the dirty page table: **N**
- Else, recoveryLSN > LSN being checked: **N**
- Else, pageLSN >= LSN being checked: **N**
- **REDO**

3. UNDO Pass

ARIES : UNDO Pass

- Scan backward from the end of the log
- All transactions that have not committed at the time of the crash, should be undone
- UNDO is an unconditional operation on ARIES
  - – i.e. the pageLSN s not checked because always the UNDO has to be done
  - – Can do this because of the prior REDO phase – applied all logged updates to the page

Compensation Log Record (CLR) and UndoNxtLSN

- CLR is added after an update is undone
  - so that no "Undo" action is undone
  - e.g. as the result of a system crash during an abort
- UndoNxtLSN
  - additional field for CLR
  - LSN of the next log record that must be undone for the transaction
  - set to the value of the prevLSN of the log record being undone

Recall: ARIES Method Illustration

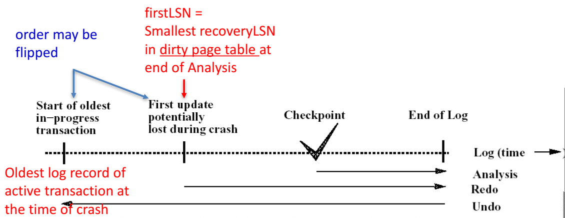


Figure 3: The Three Passes of ARIES Restart

i.e. start with lastLSN of "U" transactions in the Transaction Table and then trace back using the log

See the details in Franklin97-Transaction.pdf

Running Example: UNDO Pass

UNDO Pass

- Efficient implementation:
- Maintain a set ToUndo
  - Initialize to lastLSNs of all "U" (unknown) transactions at Transaction Table
  - undo the "largest LSN" in ToUndo at each step (the latest one in bottom-up order)

UNDO Pass

Dirty page table

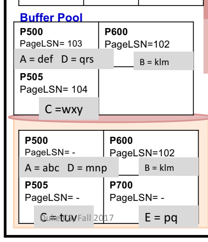
pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	



ToUNDO = {104}

UNDO Pass

Dirty page table

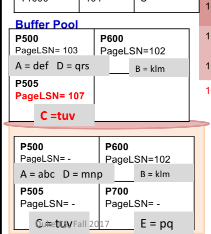
pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107		T_{1000}		UndoT_{1000} LSN104	CLR	101



- A CLR is written
- PageLSN = LSN (CLR)
- Value of C is undone

ToUNDO = {101}

UNDO Pass

Dirty page table

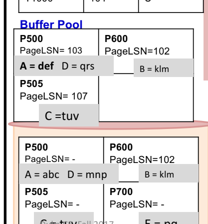
pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107		T_{1000}		UndoT_{1000} LSN104	CLR	101



ToUNDO = {101}

UNDO Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Buffer Pool

P500 PageLSN= 108 A = abc D = qrs
P600 PageLSN=102 B = klm
P505 PageLSN= 107 C =tuv

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107		T_{1000}		UndoT_{1000} LSN104	CLR	101
108		T_{1000}		UndoT_{1000} LSN101	CLR	-

Disk

P500 PageLSN= - A = abc D = mnp
P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv
P700 PageLSN= - E = pq

ToUNDO = {}

UNDO Pass

Dirty page table

pageID	recoveryLSN
P500	101
P600	102
P505	104

Transaction table

transID	lastLSN	status
T1000	104	U

Buffer Pool

P500 PageLSN= 108 A = abc D = qrs
P600 PageLSN=102 B = klm
P505 PageLSN= 107 C =tuv

Log

LSN	prevLSN	tID	pID	Log entry	Type	undoNextLSN
101	-	T1000	P500	Write A "abc" -> "def"	Update	-
102	-	T_{2000}	P600	Write B "hij" -> "klm"	Update	-
103	102	T_{2000}	P500	Write D "mnp" -> "qrs"	Update	-
104	101	T_{1000}	P505	Write C "tuv" -> "wxy"	Update	-
105	103	T_{2000}			Commit	
106	105	T_{2000}			End	
107		T_{1000}		UndoT_{1000} LSN104	CLR	101
108		T_{1000}		UndoT_{1000} LSN101	CLR	-
109		T_{1000}			End	-

Disk

P500 PageLSN= - A = abc D = mnp
P600 PageLSN=102 B = klm
P505 PageLSN= - C = tuv
P700 PageLSN= - E = pq

Write an END record

When a CLR is encountered during backward scan…

- No operation is performed on the page
  - backward scan continues to the log record pointed to by UndoNxtLSN
  - "jump over" undone update and all other updates for the transactions already undone
  - does not undo an "UNDO"

Use of CLR in UNDO

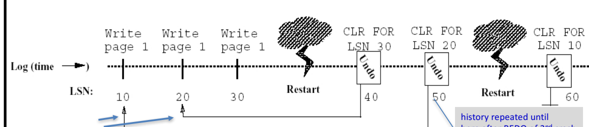


Figure 4: The Use of CLRs for UNDO

starts to examine from here but skips until LSN 10

See the details in Franklin97-Transaction.pdf (3.2.4)

If some CLR records are written to disk during an UNDO phase, then a crash happens (e.g. here LSN 40, 50 are written to disk before the second crash), then the next UNDO phase will skip undoing those CLRs.

Note: REDO re-does CLRs!