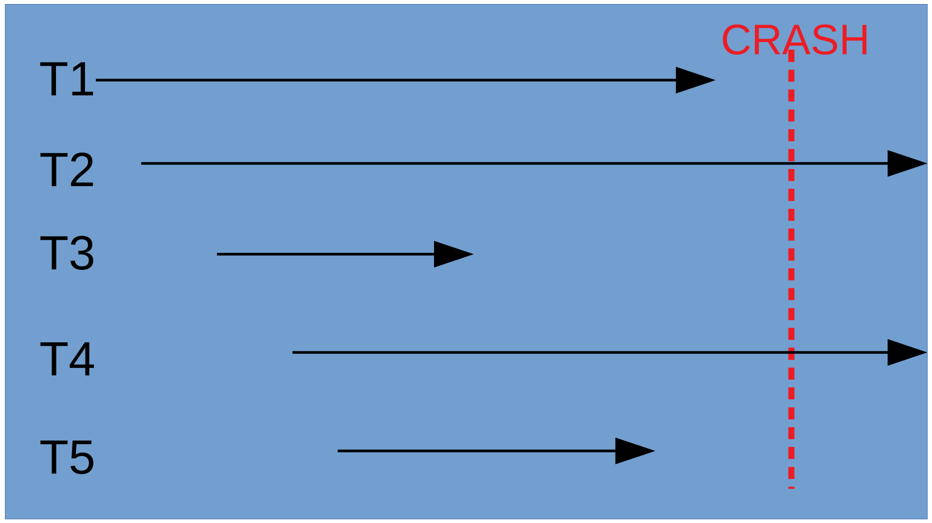
DBMS UNIT V

Introduction to Crash Recovery,
Write Ahead Log,
ARIES Protocol

Database Crash

- If the system or database crashes while processing transactions, how can the database be recovered to a consistent state?
- Database may crash due to
  - Power failure
  - Hardware error
  - Operating system errors
  - Database software
  - Any other error



- Which transactions must be available when the system restarts after the CRASH?

Database Startup

- Transactions T1, T3 and T5 must be available and parts of transactions T2 and T4 if stored in the disk must be rolled back.
- Every time the DBMS starts up
  - ***Recovery manager*** reads a log file
  - Applies all the transactions to the database.
    - The committed transactions which are not saved in database files are redone (repeated).
    - The partial updates of uncommitted, aborted and incomplete transactions are rolled-back(undone).
- Recovery Manager: is the module that brings the database to a consistent state on database startup.

ACID Properties

- The database recovery manager is responsible for ensuring the following two properties
  - Atomicity
    - The effect of incomplete transactions must be rolled back if they were written to disk.
  - Durability
    - All the completed transactions must be available (even though they were in memory buffers and not written to the disk at the time of the CRASH).

Why incomplete txns in Disk?

- Recall from database architecture-
  - Database buffers in memory are updated (and not files on disk).
  - Database buffers are written to the disk to make space for new database blocks which have to be read from the disk (as system memory is limited).
  - As a result
    - Some uncommitted (in progress) transaction data may be written to disk because the buffer space was required for another transaction.
  - Therefore all transaction data (complete and incomplete) is written to the ***log file*** even though it may not have been saved in ***database file***.
      - Q: If we can write to log file after each transaction, why can't we write to database file?
      - A: The log file records contain bare minimum amount of data where as database files have to be written along with data of neighbouring rows, related data structures like indexes etc. Also, the log file is written in append mode where as database file has to be written in such a manner that the buffer in memory must be placed at the correct position on the disk.

Why committed txns not in Disk?

- Any changes made to the database is made to the copy of the database in memory (called database buffer). Database buffers in memory are updated (<u>and not files on disk</u>).
- When the transaction is committed, the data is first written to the log file on disk. If the log file writing succeeded, then the database buffer data is changed.
- Database buffers are not written to the disk immediately but only when the memory (buffer space) is required for other transactions.
- Some committed transaction data may still be in buffer and not in disk.
- If there is a CRASH, this data in memory is lost (but is available in log file).
  - Q: Does the log file contain only record of changes made to the database by committed transactions?
  - A: The log file contains a record for every database update/change made. Whether the update/change is for an ongoing transaction or a committed transaction. There is a log buffer which records every change to the database. On commit, the contents of the log buffer is written to the log file on disk.

Steal and No Steal

- When a new transaction needs to get disk blocks into the database buffer, the blocks in the database buffer have to be written to the disk (or discarded if no changes were made)
- What if a transaction is still using the block in memory?
- Should the new transaction ***<u>“steal”</u>*** the memory frame?
- Or should it wait for the currently running transaction to complete? ***<u>“No-Steal”</u>***
- ***<u>Clearly: No-Steal makes transactions wait and is undesirable.</u>***

Force and No-Force

- When a transaction is committed,
  - Should we write the database block in the buffer to the disk after every transaction? If yes then it is called-***”Force”*** write.
  - Or should we wait till the buffer space is required by another transaction? ***“No-Force”*** write.
- ***Clearly: A No-Force write avoids excessive disk writing and reduces IO operations.***
- ***So most databases choose a Steal-No-Force approach.***
- ***This is the reason why we have***
  - ***Some committed transactions not in Disk***
  - ***Some incomplete or aborted transactions in Disk***

WAL-Write Ahead Logging

- Logging
  - WAL - write ahead log, redo log, or simply the log file.
    - records every transaction on the database.
- A background process called the log-writer (wal process) is constantly writing data from log-buffer in memory to the log-file.
- A transaction commit forces a write to the log file in the disk.
- All the log records of the transaction as well as changes made by other (running/uncommitted transactions) are also written.
- The recovery manager reads the log file and performs the actions to restore the database to a consistent state after a system failure or crash

Analogy for logging

- A Cash Register has
  - A roll of paper which prints the receipts for customers
  - A roll of paper which records all the transactions for the business (which acts like a backup if cash register memory fails).
  - This roll of paper can be considered as a transaction log. Except that in case of failure of memory, this log file is read manually whereas database log file is read by the recovery manager to restore the database to a consistent state after a system failure or crash.

Checkpoint

- How far back in the log should we go to perform recover the database after a crash?
  - Answer: To the last “Checkpoint”
- **Checkpoint**: A Checkpoint is a time/instant when the database is in a consistent state.
- By taking checkpoints frequently, the amount of work done during restart is reduced.
- Fuzzy Checkpoint: (database continues to operate during checkpoint)
  - A begin checkpoint log record is written.
  - The contents of transaction table and dirty page table are written to the log
  - An end checkpoint log record is written.
  - Between the begin checkpoint and the end checkpoint records there may be transaction log records.
  - Transactions are allowed during checkpoint process- This is called Fuzzy checkpoint

ARIES Algorithm

- ARIES: Algorithms for Recovery and Isolation Exploiting Semantics
- Principles behind ARIES
  - Write Ahead Logging
    - All changes to database objects (transactions) are recorded here. Log is written to disk. Database buffers may be written to disk later.
  - Repeating History during Redo
    - On restart, ARIES retraces all the actions in the log file that were not written to disk. The Uncommitted transactions are rolled back.
  - Logging Changes during Undo
    - To ensure that the same actions are not repeated in later restarts, the rollback of uncommitted transactions is recorded in the log file.

ARIES – Three Phases

- **Analysis Phase:** Identify (dirty pages) buffers not written to disk at the time of crash.
- **Redo Phase:** Repeats all the actions from the point in the log file to the time of crash.
- **Undo Phase:** Undo all the actions (of the last transactions) that were not committed at the time of crash.

Analysis Phase

- The analysis phase involves three tasks-
  - It determines the point in the log at which to start the Redo Phase. (from the most recent checkpoint).
  - It determines the pages in the database buffer that were dirty at the point of the CRASH.
    - **Dirty page**: blocks in database buffer that has some updates but was not saved to the disk.
  - It identifies the transactions that were active at the time of the CRASH that will have to be undone.
    - T2 and T4 for the example in the beginning.

Summary of Analysis Phase

- In the analysis phase, we have
  - Created a Transaction Table.
  - What does the Transaction Table contain?
    - All incomplete transactions.
  - Created a Dirty Page Table.
  - What does the Dirty Page Table Contain?
    - All pages that have updates (possibly) not written to the disk.
- If the database would not have crashed, this would be the status of the transaction table and the dirty page table.
- The Analysis phase creates the transaction table and dirty page table to the state they were in at the time of the CRASH.
- At a checkpoint the transaction table and dirty page table is written to the log.

Redo Phase

- In the Redo phase, all the changes made to the database (that were lost due to CRASH) is repeated (re-applied).
- Every page on the disk has a field to record the last log serial number (called pageLSN).
  - If the pageLSN is greater than the current log record serial number being redone, than the page was written to disk by a more recent transaction.
  - Else, the changes recorded in log record is applied.

Undo Phase

- In the Undo phase, the changes made to the database by the (incomplete) running transactions at the time of the CRASH are undone.
- That is, the changes made to the database by incomplete transactions must be “cleared” to ensure atomicity and database consistency.
- The undo actions must be done in the reverse order of the actions of the transactions. The most recent change must be undone first and the

The Three Phases of ARIES

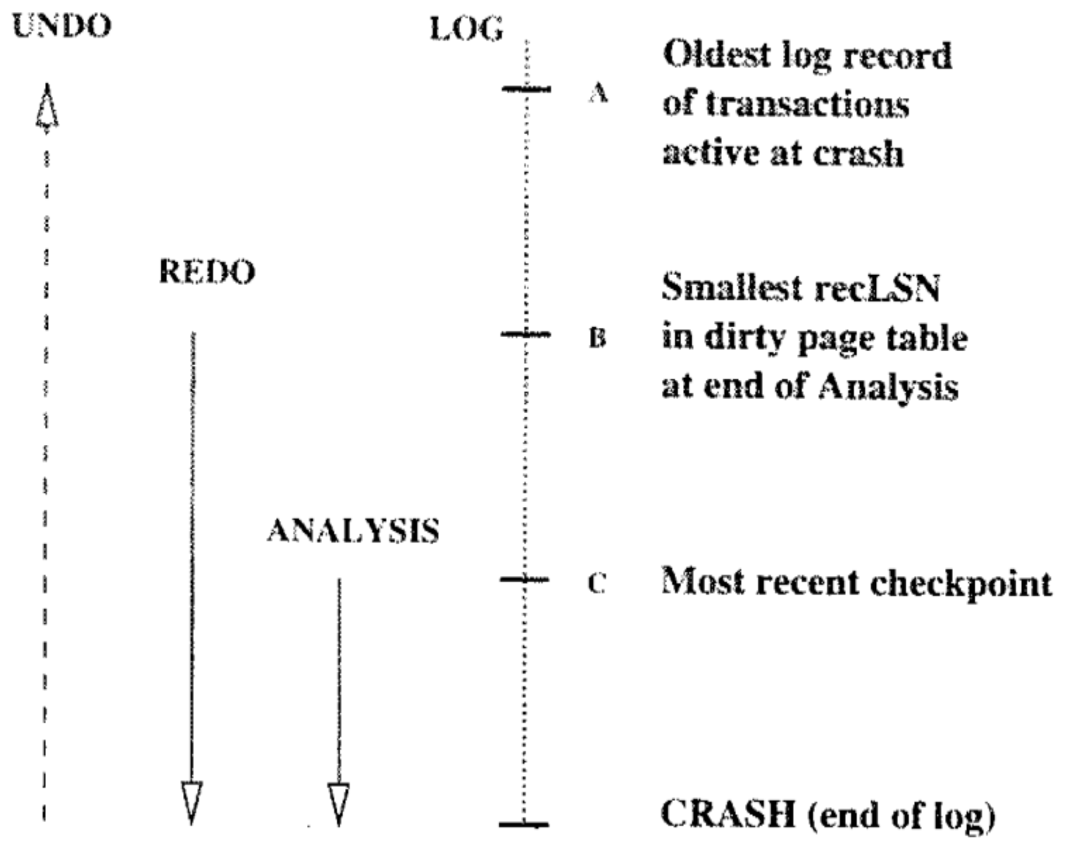

UNDO
LOG
Oldest log record of transactions active at crash
A
REDO
Smallest recLSN in dirty page table at end of Analysis
B
ANALYSIS
C
Most recent checkpoint
CRASH (end of log)

Figure 18.4 Three Phases of Restart in ARIES

Data Structures Used

- Redo Log
- Transaction Table
- Dirty Page Table

Log Records

- Every log record has:
  - Log Sequence Number or LSN
  - The prevLSN
    - This is the LSN of the previous operation for a transaction. So, a linked list of all operations can be created out of the log records.
  - Transaction id – an ID for every transaction.
  - Type – the type of the log record.
- Log record types
  -
  - Checkpoint-begin and end checkpoint, The current state of the transaction table and the dirty page table is stored in the log.
  - CLR-Compensation log record is a record of undoing the transaction during the recovery phase.

Log Records

- Update Log Record
  - PageID is the page id of the modified page.
  - Length is the number of bytes being modified.
  - Offset is the number of bytes from beginning of the page where the modification is being done.
  - Before-image is the value of the bytes before update

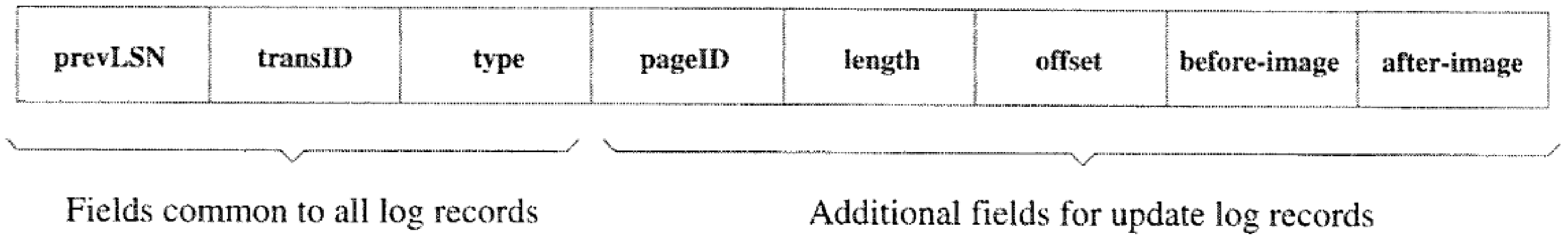
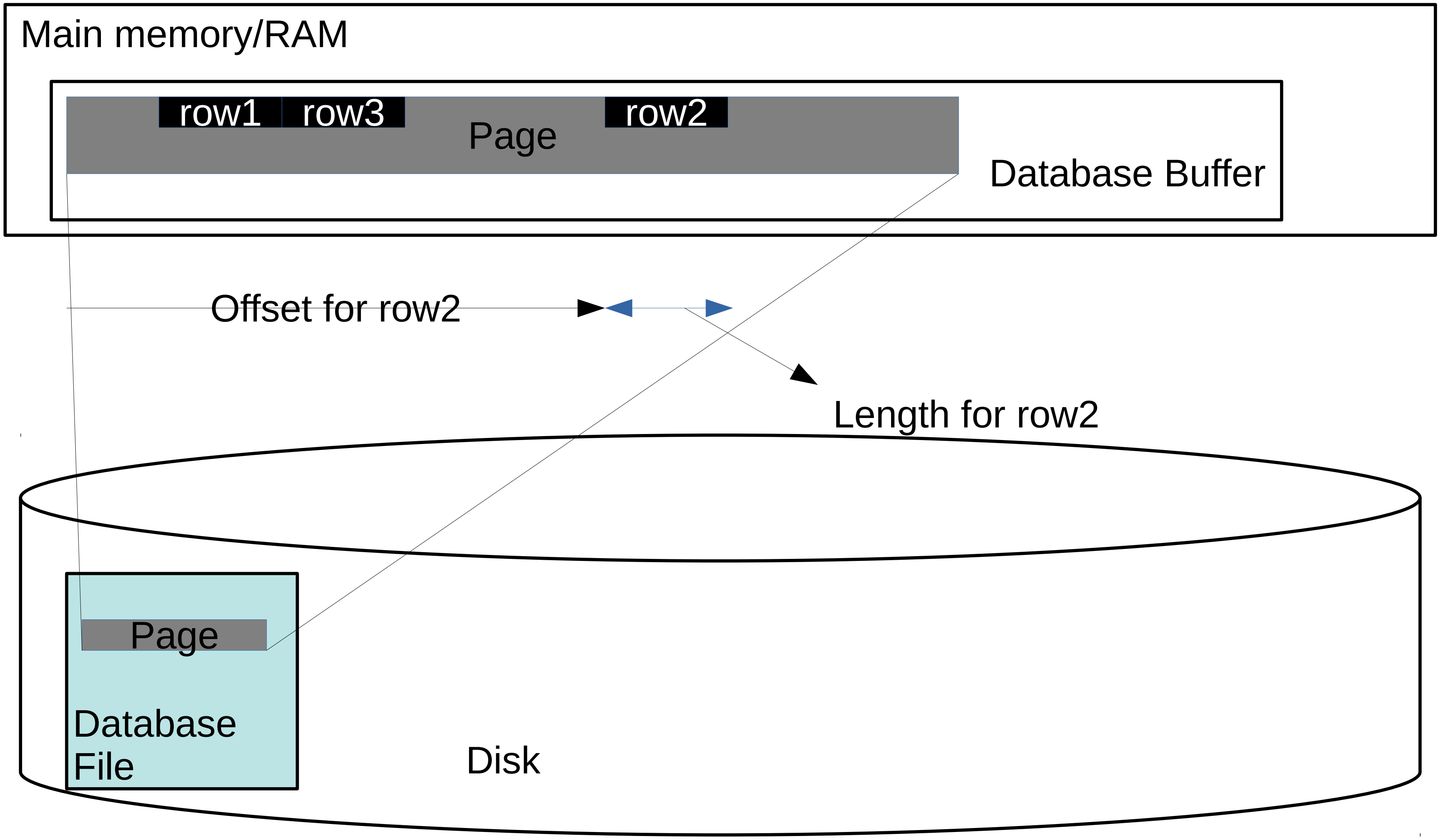

prevLSN
transID
type
pageID
length
offset
before-image
after-image
Fields common to all log records
Additional fields for update log records

Figure 18.2 Contents of an Update Log Record

Database File, Disk, Buffer & Page



Transaction Table

- Transaction table has one entry for each active transaction at the time of CRASH.
- The status of the transaction can be “in-progress”, committed or aborted.
- The “in-progress” transactions have to be undone.

Dirty Page Table

- The Dirty Page Table contains an entry for each dirty page in the buffer.
  - That is: every page that has changes that were not reflected on the disk at the time of the CRASH.

End of Portions for Unit-5

The Example will help you understand the concepts explained upto now.

Analysis Phase Example-1

- In this example, we have the “tail” of the log file.
- We start with an empty Transaction Table and empty Dirty Page Table.
- Reading the first entry of the log file.
- Transaction T1000 has updated Page with PageID P500

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN

Transaction Table

pageID	recLSN

Dirty Page Table

Analysis Phase Example-1...

- Transaction T1000 has updated Page with PageID P500.
- We enter the transaction T1000 and the Log Sequence Number (LSN) 00 in the transaction table.
- We enter the updated PageID P500 and the LSN 00 in the Dirty Page Table.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	00

Transaction Table

pageID	recLSN
P500	00

Dirty Page Table

Analysis Phase Example-1...

- Enter T2000 and LSN 10 in the transaction table.
- Enter the page id P600 and LSN 10 in Dirty Page Table.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	00
T2000	10

Transaction Table

pageID	recLSN
P500	00
P600	10

Dirty Page Table

Analysis Phase Example-1...

- T2000 is already in transaction table. But a more recent update has occurred. Update lastLSN to 20 (the LSN of the log record being read) in the transaction table.
- The page id P500 is already in the Dirty Page Table.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	00
T2000	20

Transaction Table

pageID	recLSN
P500	00
P600	10

Dirty Page Table

Analysis Phase Example-1...

- T1000 is already in transaction table. Update lastLSN to 30 (the LSN of the log record being read) in the transaction table.
- Enter the page id P505 and LSN 30 in Dirty Page Table.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
T2000	20

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Analysis Phase Example-1...

- This is a commit record for the T2000 transaction. So, we have to remove the transaction ID from the transaction table.
- There are no more log records. (perhaps there was a CRASH).

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Summary of Analysis Phase

- In the analysis phase, we have
  - Created a Transaction Table.
  - What does the Transaction Table contain?
    - All incomplete transactions.
  - Created a Dirty Page Table.
  - What does the Dirty Page Table Contain?
    - All pages that have updates (possibly) not written to the disk.
- If the database would not have crashed, this would be the status of the transaction table and the dirty page table.
- The Analysis phase creates the transaction table and dirty page table to the state they were in at the time of the CRASH.
- At a checkpoint the transaction table and dirty page table is written to the log.

Redo Phase Example - 1

- What should we redo?
  - All the updates made to the pages but not saved to the disk.
- Where is this information?
  - The smallest LSN in the Dirty Page Table. All the log records starting from this point have to be redone.
- How do we apply the changes (Redo)?
  - Find the lowest LSN in the Dirty Page Table
  - Start from the lowest LSN and apply all the changes in the log file starting from this LSN. (In example, LSN 00)
  - Apply the changes using the following rules
    - If the page identified by the log record is not in the Dirty Page Table, ignore this page (It was written to the disk).
    - If the Disk Page has a recLSN ***<u>lesser than the LSN of the log file</u>***-the update has not been applied therefore apply the update0.
    - If the Disk Page has a recLSN ***<u>greater than the LSN of the log file-the update was applied (possibly by dbwriter because some other transaction required the buffer space.</u>***) so, ignore this log record..

Redo Phase Example-1...

- The smallest LSN in Dirty Page Table is 00.
- Start with the LSN 00 in the log.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Redo Phase Example-1...

- PageID P500 is in Dirty Page Table.
- Check the LSN number in the buffer.
  - If higher than 00, then this was updated. Hence ignore.
  - **If less than 00, apply the change.**
- Read the next log entry.

LSN	TransID	Type	PageID
→ 00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
→ P500	00
P600	10
P505	30

Dirty Page Table

Redo Phase Example-1...

- PageID P600 is in Dirty Page Table.
- Check the LSN number in the buffer.
  - **Assume higher than 10, then this was updated. Hence ignore this update.**
  - However, If less than 10, apply the change.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
→ 10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
→ P600	10
P505	30

Dirty Page Table

Redo Phase Example-1...

- PageID P500 is in Dirty Page Table.
- Check the LSN number in the buffer.
  - If higher than 20, then this was updated. Hence ignore this update.
  - However, If less than 20, apply the change.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Redo Phase Example-1...

- PageID P505 is in Dirty Page Table.
- Check the LSN number in the buffer.
  - If higher than 30, then this was updated. Hence ignore this update.
  - However, If less than 30, apply the change.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Redo Phase Example-1...

- This is a commit record.
- There is nothing to Redo.
- At this stage, the database buffers are in the same state as they were before the CRASH.
- The Redo phase brings the database buffers to the state they were in before the CRASH.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
→ 40	T2000	commit	
System Crash			

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Undo Phase

Undo Phase Example-1...

- The Transaction table contains the transactions that were still incomplete at the time of the CRASH and hence have be undone.
- Here the effects of the transaction T1000 has to be undone.
- The last LSN for the transaction T1000 is 30.
- The undo phase starts from this log record.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	

transID	lastLSN
$\rightarrow$ T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Undo Phase Example-1...

- The effects of this update is undone.
- How do we know how to undo this update operation.
- Recall from the type of Log Records that there is a before image and after image stored in the update log record.
- The before image is placed in page ID P505.
- A CLR Compensation Log Entry is added to the Log file to reflect this Undo operation.
- Read the next log entry.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
→ 30	T1000	update	P505
40	T2000	commit	
50	T1000	CLR: 30	P505

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Undo Phase Example-1...

- Which is the next log entry?
  - There is a prevLSN record which gives the log record to be processed next. (Not shown in the table.)
- The prevLSN is 00.
- The before image is placed in page ID P500.
- A CLR Compensation Log Entry is added to the Log file to reflect this Undo operation.
- The Undo operations are performed in the reverse order.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
50	T1000	CLR: 30	P505
60	T1000	CLR: 00	P500

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Undo Phase Example-1...

- The log record for LSN 00 is undone.
- The before image is placed in page ID P500.
- There are no more log records for Transaction T1000 and no more transaction in the Transaction table.
- The Undo Operation is complete.

LSN	TransID	Type	PageID
00	T1000	update	P500
10	T2000	update	P600
20	T2000	update	P500
30	T1000	update	P505
40	T2000	commit	
50	T1000	CLR: 30	P505
60	T1000	CLR: 00	P500

transID	lastLSN
T1000	30
~~T2000~~	~~20~~

Transaction Table

pageID	recLSN
P500	00
P600	10
P505	30

Dirty Page Table

Summary

- Analysis starts from the last checkpoint and recreates the transaction table and the dirty page table at the time of the CRASH.
- Redo starts from the earliest Log Sequence Number found in the Dirty Page Table and Applies all the changes made to the database in the increasing order of LSN number.
- Undo has to undo all the transactions in the transaction table. For each transaction in the transaction table, it reverses the operations made by that transaction by reading the highest LSN entry first and then going backward using the prevLSN (a linked list of undo log records).

Questions