Database Recovery Techniques

Truong Tuan Anh

CSE-HCMUT

Outline

- Purpose of Database Recovery
- Recovery Concepts
- Recovery Based on Deferred Update
- Recovery Based on Immediate Update
- Shadow paging
- ARIES Recovery Algorithm
- Recovery in Multidatabase System

Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

- Example:
  - If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value.

  → Thus, the database must be restored to the state before the transaction modified any of the accounts.

Recovery Concepts

Types of Failure

- The database may become unavailable for use due to
  - **Transaction failure**: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
  - **System failure**: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
  - **Media failure**: Disk head crash, power disruption, etc.

Recovery Concepts

Transaction Log

- For recovery from any type of failure, data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required.
- These values and other information is stored in a sequential file called Transaction log. A sample log is given below. Back P and Next P point to the previous and next log records of the same transaction.

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Recovery Concepts

Data Caching

- Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
- The flushing is controlled by **Modified** and **Pin-Unpin** bits.
  - **Pin-Unpin**: Instructs the operating system not to flush the data item.
  - **Modified**: Indicates the AFIM of the data item.

Recovery Concepts

Data Update:

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update**: All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update**: The disk version of the data item is overwritten by the cache version.

Recovery Concepts

Transaction **Roll-back (Undo)** and **Roll-Forward (Redo)**

- To maintain atomicity, a transaction's operations are redone or undone.
  - **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
  - **Redo**: Restore all AFIMs on to disk.

- Database recovery is achieved either by performing only Undo or only Redo or by a combination of the two. These operations are recorded in the log as they happen.

Recovery Concepts

Write-Ahead Logging

- When **in-place** update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by **Write-Ahead Logging (WAL)** protocol. WAL states that:
  - **For Undo**: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
  - **For Redo**: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Recovery Concepts

Steal/No-Steal and Force/No-Force

- Possible ways for flushing database cache to database disk:
  - **Steal/No-Steal:**
  1. Steal: Cache can be flushed before transaction commits.
  2. No-Steal: Cache cannot be flushed before transaction commit.
  - **Force/No-Force:**
  1. Force: Cache is immediately flushed (forced) to disk before the transaction commit.
  2. No-Force: Otherwise.

Recovery Concepts

Steal/No-Steal and Force/No-Force

- These give rise to four different ways for handling recovery:
  - Steal/No-Force (Undo/Redo)
  - Steal/Force (Undo/No-redo)
  - No-Steal/No-Force (Redo/No-undo)
  - No-Steal/Force (No-undo/No-redo)

11

Recovery Concepts

Checkpointing

- Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:
  1. Suspend execution of transactions temporarily.
  2. Force write modified buffer data to disk.
  3. Write a [checkpoint] record to the log, save the log to disk.
  4. Resume normal transaction execution.
- During recovery redo or undo is required to transactions appearing after [checkpoint] record.

Recovery Concepts

Fuzzy Checkpointing

- The time need to force-write all modified memory buffers may delay transaction processing

→ Need fuzzy checkpointing

- System can resume transaction processing after a [begin_checkpoint] record is written to the log without having to wait for step 2 to finish.
- When step 2 is completed → [end_checkpoint] record is written to the log.
- Until step 2 is completed, the previous checkpoint record should maintain valid.

Recovery Based on Deferred Update

- Deferred Update (No Undo/Redo)
- The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme these updates are saved on database disk.
  - After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Recovery Based on Deferred Update

- **Deferred Update in a single-user system**
  There is no concurrent data sharing in a single user system. The data update goes as follows:
  - A set of transactions records their updates in the log.
  - At commit point under WAL scheme these updates are saved on database disk.
- After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

T_1	T_2
read_item (A)	read_item (B)
read_item (D)	write_item (B)
write_item (D)	read_item (D)
	write_item (D)

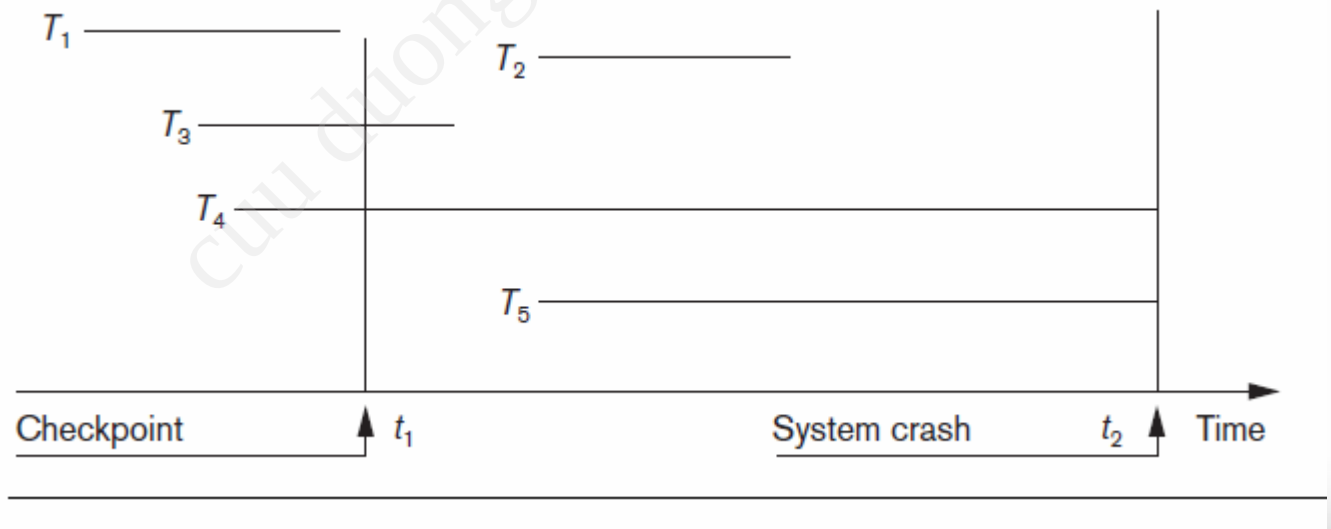
--- log ---

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25] ← system crash

Redo [write_item, T_1, D, 20] of T_1
Ignore T_2

Recovery Based on Deferred Update

- Deferred Update with concurrent users
  - This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the



(a)

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(A)
write_item(A)
read_item(C)
write_item(C)

T_4
read_item(B)
write_item(B)
read_item(A)
write_item(A)

(b)

[start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item, T_2, D, 25] ← System crash

T_2 and T_3 are ignored because they did not reach their commit points.

T_4 is redone because its commit point is after the last system checkpoint.

Recovery Based on Deferred Update

Deferred Update with concurrent users

- Two tables are required for implementing this protocol:
  - **Active table**: All active transactions are entered in this table.
  - **Commit table**: Transactions to be committed are entered in this table.

- During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is "**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

--- log ---
[start_transaction, T_1]
[write_item, T_1, D, 20]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[start_transaction T_2]
[commit, T_1]
[write_item, T_4, A, 20]
[commit, T_4]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item, T_2, D, 25] ← system crash

- Ignore: T_2 & T_3
- Redo: T_1 & T_4

D ← 20
B ← 15
A ← 20

Recovery Based on Immediate Update

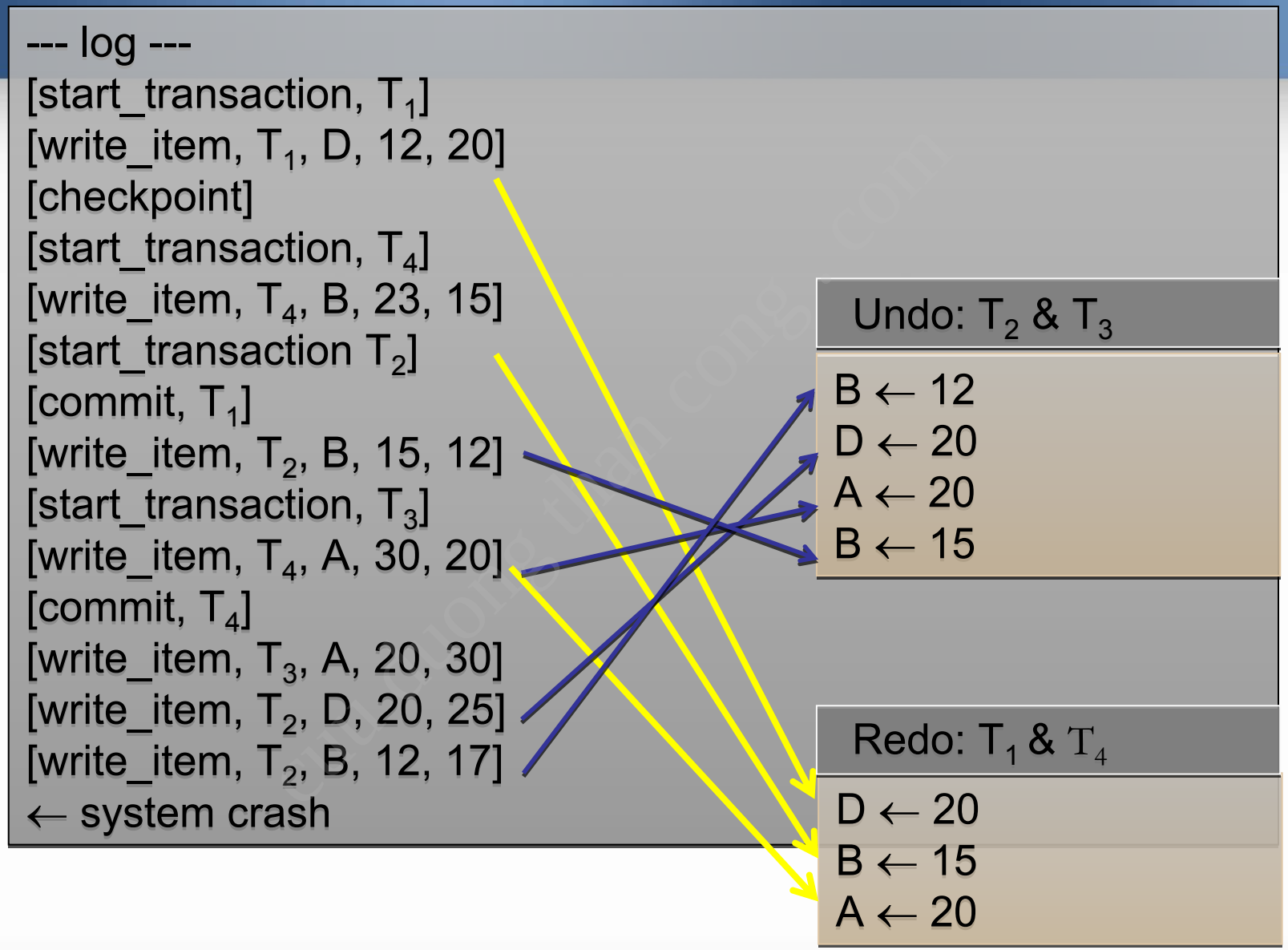
- **Undo/No-redo Algorithm**
  - In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits.
  - For this reason the recovery manager **undoes** all transactions during recovery.
  - No transaction is **redone**.
  - It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Recovery Based on Immediate Update

- **Undo/Redo Algorithm** (**Single-user environment**)
  - Recovery schemes of this category apply **undo** and also **redo** for recovery.
  - In a single-user environment no concurrency control is required but a log is maintained under WAL.
  - Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
  - The recovery manager performs:
    - **Undo** of a transaction if it is in the **active** table.
    - **Redo** of a transaction if it is in the **commit** table.

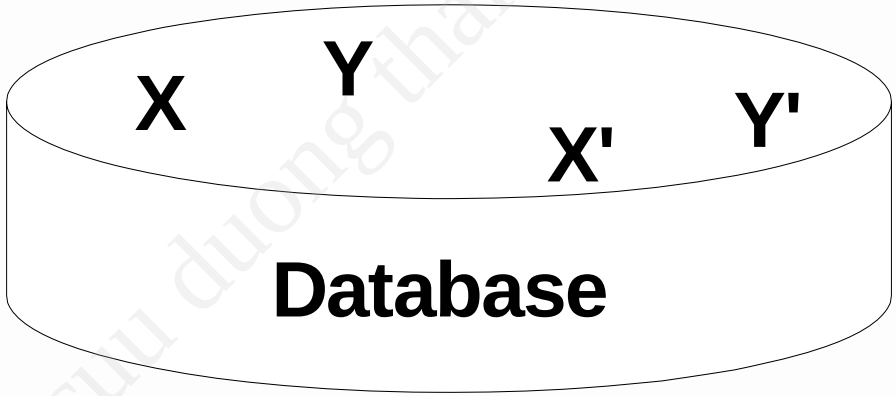
Recovery Based on Immediate Update

- **Undo/Redo Algorithm** (**Concurrent execution**)
- Recovery schemes of this category applies **undo** and also **redo** to recover the database from failure.
- In concurrent execution environment a concurrency control is required and log is maintained under WAL.
- Commit table records transactions to be committed and active table records active transactions. To minimize the work of the recovery manager checkpointing is used.
- The recovery performs:
  - **Undo** of a transaction if it is in the **active** table.
  - **Redo** of a transaction if it is in the **commit** table.


--- log ---
[start_transaction, T1]
[write_item, T1, D, 12, 20]
[checkpoint]
[start_transaction, T4]
[write_item, T4, B, 23, 15]
[start_transaction T2]
[commit, T1]
[write_item, T2, B, 15, 12]
[start_transaction, T3]
[write_item, T4, A, 30, 20]
[commit, T4]
[write_item, T3, A, 20, 30]
[write_item, T2, D, 20, 25]
[write_item, T2, B, 12, 17]
← system crash
Undo: T2 & T3
B ← 12
D ← 20
A ← 20
B ← 15
Redo: T1 & T4
D ← 20
B ← 15
A ← 20

Shadow Paging

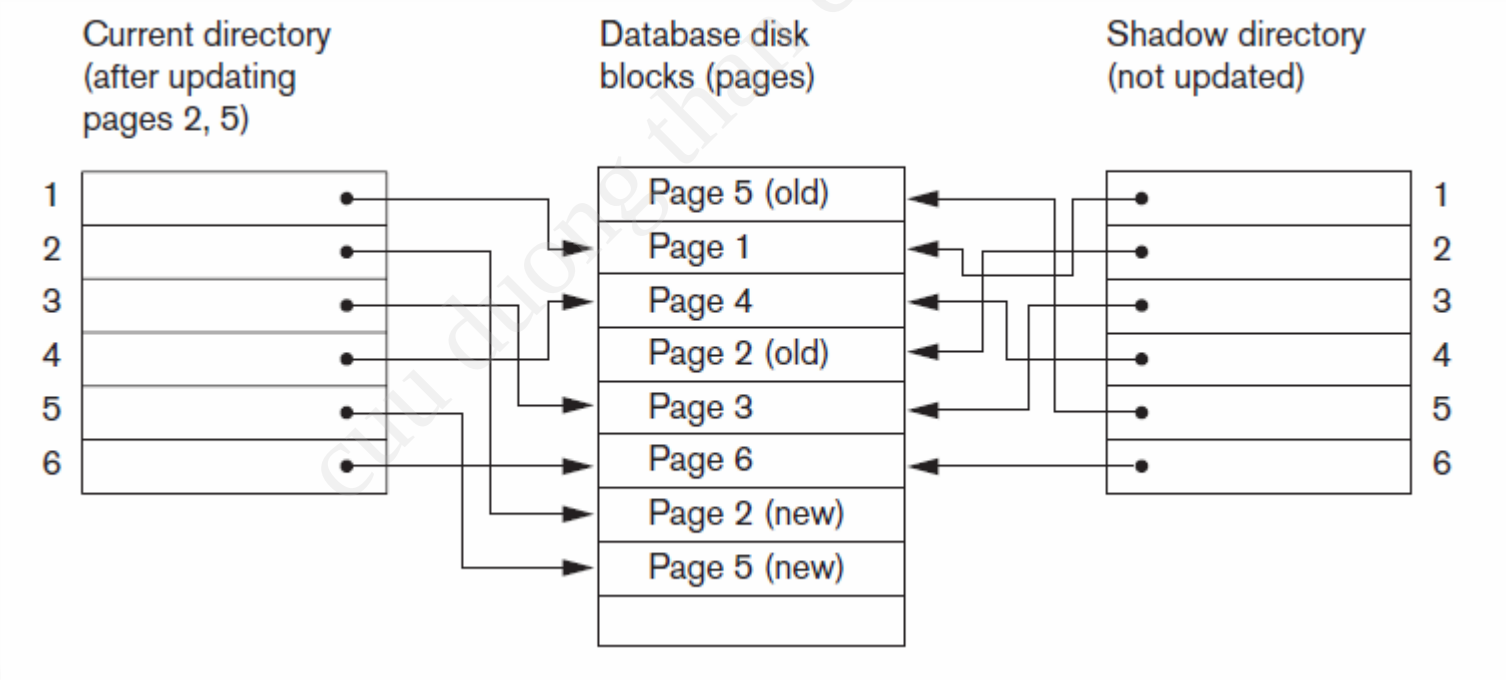
- The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items
X' and Y': Current copies of data items

Shadow Paging

- To manage access of data items by concurrent transactions two directories (current and shadow) are used.
  - The directory arrangement is illustrated below. Here a



Shadow Paging

- Recovery:
  - Free the modified database pages and to discard the current directory (reinstating the shadow directory)
- Committing a transaction corresponding to discarding the previous shadow directory.
- NO-UNDO/ NO-REDO
- In a multiuser environment→ use logs and checkpoints

27

ARIES Recovery Algorithm

- Steal/no-force (UNDO/REDO)
- The ARIES Recovery Algorithm is based on:
  - **WAL** (Write Ahead Logging)
  - **Repeating history during redo**:
    - ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred.
  - **Logging changes during undo**:
    - It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

ARIES Recovery Algorithm

- The ARIES recovery algorithm consists of three steps:
  1. **Analysis**: step identifies the **dirty (updated) pages** in the buffer and the **set of transactions active** at the time of crash. The appropriate point in the log where redo is to start is also determined.
  2. **Redo**: necessary redo operations are applied.
  3. **Undo**: log is scanned backwards and the operations of transactions active at the time of crash are undone in reverse order.

ARIES Recovery Algorithm

- **The Log and Log Sequence Number (LSN)**
  - A log record is written for:
    - (a) data update
    - (b) transaction commit
    - (c) transaction abort
    - (d) undo
      - In the case of undo a compensating log record is written.
    - (e) transaction end

ARIES Recovery Algorithm

- **The Log and Log Sequence Number (LSN) (cont.)**
  - A unique LSN is associated with every log record.
    - LSN increases monotonically and indicates the disk address of the log record it is associated with.
    - In addition, each data page stores the LSN of the latest log record corresponding to a change for that page.
  - A log record stores
    - (a) the previous LSN of that transaction. It links the log record of each transaction. It is like a back pointer points to the previous record of the same transaction
    - (b) the transaction ID
    - (c) the type of log record.

ARIES Recovery Algorithm

- **The Log and Log Sequence Number (LSN) (cont.)**
  - For a write operation the following additional information is logged:

1. Page ID for the page that includes the item
2. Length of the updated item
3. Its offset from the beginning of the page
4. BFIM of the item
5. AFIM of the item

ARIES Recovery Algorithm

- The **Transaction table** and the **Dirty Page table**
  - For efficient recovery following tables are also stored in the log during checkpointing:
    - **Transaction table**: Contains an entry for each active transaction, with information such as transaction ID, transaction status and the LSN of the most recent log record for the transaction.
    - **Dirty Page table**: Contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the **earliest update** to that page.

ARIES Recovery Algorithm

- Checkpointing
  - A checkpointing does the following:
    - Writes a begin_checkpoint record in the log
    - Writes an end_checkpoint record in the log. With this record the contents of transaction table and dirty page table are appended to the end of the log.
    - Writes the LSN of the begin_checkpoint record to a special file. This special file is accessed during recovery to locate the last checkpoint information.
  - To reduce the cost of checkpointing and allow the system to continue to execute transactions, ARIES uses “fuzzy checkpointing”.

ARIES Recovery Algorithm

- The following steps are performed for recovery:
  - **Analysis phase**:
    - Start at the begin_checkpoint record and proceed to the end_checkpoint record.
    - Access transaction table and dirty page table are appended to the end of the log.
    - Modify transaction table and dirty page table:
      - An end log record was encountered for T → delete entry T from transaction table
      - Some other type of log record is encountered for T' → insert an entry T' into transaction table if not already present, or the last LSN is modified.
      - The log record corresponds to a change for page P → insert an entry P (if not present) with the associated LSN in dirty page table
    - The analysis phase compiles the set of redo and undo to be performed and ends.

ARIES Recovery Algorithm

- The following steps are performed for recovery:
  - **Redo phase**: Starts redoing at a point in the log where it knows that previous changes to dirty pages *have already been applied to disk.*
  - Where?
    - Finding the smallest LSN, M of all the dirty pages in the Dirty Page Table.
  - Redo starts at the log record with LSN = M and scan forward to the end of the log.
  - Verify whether or not the change has to be reapplied.
    - A change recorded in the log pertains to the page P that is not in the Dirty Page Table → no redo
    - A change recorded in the log (LSN = N) pertain to Page P and the Dirty Page Table contains an entry for P with LSN > N → no redo.
    - Page P is read from disk and the LSN stored on that page > N → no redo.

ARIES Recovery Algorithm

- The following steps are performed for recovery:
  - **Undo phase**: Starts from the end of the log and proceeds backward while performing appropriate undo. For each undo it writes a compensating record in the log.

(a)

Lsn	Last_lsn	Tran_id	Type	Page_id	Other_information
1	0	T_1	update	C	...
2	0	T_2	update	B	...
3	1	T_1	commit		...
4	begin checkpoint				
5	end checkpoint				
6	0	T_3	update	A	...
7	2	T_2	update	C	...
8	7	T_2	commit		...

(b)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	2	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2

(c)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	8	commit
T_3	6	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2
A	6

Recovery in Multi-database System

- A multi-database system is a special distributed database system where one node may be running relational database system under UNIX, another may be running object-oriented system under Windows and so on.
- A transaction may run in a distributed fashion at multiple nodes.
- In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing.
- This commit scheme is referred to as "**two-phase commit**" (**2PC**).
  - If any one of these nodes fails or cannot commit the part of the transaction, then the transaction is aborted.
- Each node recovers the transaction under its own recovery protocol.