מימוש בסיסי נתונים

Recovery

Robert Moskovitch, PhD
Software and Information Systems Engineering
Ben Gurion University

Recovery

What is it good for?

Example of a transaction

- Bank accounts: A, B
- Transfer 100 from A to B
- The transaction:
  - A := A – 100
  - B := B + 100
- What happens if the server falls after the first operation?
- A's balance was decreased, but we didn't increase B
- The program was ran incorrectly, and our database state is inconsistent

Atomic Execution

- A transaction has to be performed in an atomic fashion
  - It runs completely, or
  - Not at all

- A transaction COMMITs only after it is finished
  - If the transaction COMMITed, then all the operations are stored
  - Otherwise, any operation that was performed is not saved in the database

Recovery Mechanism

- For any database system, there is a recovery mechanism that is responsible for:
  - Canceling all the operations of the program (transaction) if it ABORTed before COMMITing
  - To assure that all the operations of the program are stored in the database if the program performed COMMITed

Which Failures Have to Be Anticipated?

- Transaction failures
- System failures
- Media failures
- Communication failure

ACID

- Atomicity – a transaction is performed as an atomic operation – either it happens completely, or not at all
- Consistency – if each transaction is consistent (correct) and in the beginning the database is consistent, then also at the end it is consistent
- Isolation – a transaction is ran isolated from any influences of other transactions
- Durability – if a transaction COMMITs then its impact is stored

ACID

- Atomicity – a transaction is performed as an atomic operation – either it happens completely, or not at all
- Consistency – if each transaction is consistent (correct) and in the beginning the database is consistent, then also at the end it is consistent
- Isolation – a transaction is ran isolated from any influences of other transactions
- Durability – if a transaction COMMITs then its impact is stored

- Consistency and Isolation are properties that are assured by concurrency control mechanisms (Strict 2PL)
- Atomicity and Durability are properties that are assured by the recovery mechanism

Recoverable Schedules

- An equivalent serial schedule refers to consistency of the database, assuming there are no failures

T_i	T_j
R(A)	
W(A)	
	R(A)
	Commit
R(B)	
Commit	

The schedule is conflict serializable

Recoverable Schedules

- What will happen if there will be a failure after the COMMIT of transaction j but before transaction i COMMITed?
- It is impossible to perform a rollback for transaction j, since it COMMITed already
- Thus, the schedule is not recoverable, since there is no way to finalize Tj and not cancel

T_i	T_j	
R(A)		
W(A)		
	R(A)	
	Commit	
	→	Failure
R(B)		
Commit		

Cascade of ABORTs

- Assuming the transaction Tj didn't yet COMMIT
- But later transaction Ti is canceled
  - Transaction Tj has to be canceled too (since it read a value that is not updated)
  - Now we have a cascade of ABORTs

T_i	T_j	
R(A)		
W(A)		
	R(A)	
R(B)		
Rollback		Failure

Recoverable Schedule

- A schedule is recoverable if a transaction COMMITs only after all the transactions it read their written values COMMITed

- In practice the conditions are more severe
  - A transaction is allowed to READ only values that were written by transactions that COMMITed already

- A schedule that confirms this condition enables recovery and prevents a cascade of ABORTs

Strict 2PL

- Like 2PL (first phase - all the locks are taken, second phase – all are released) with the following additional requirement
  - A transaction releases locks that it holds only after it COMMITed

- Strict 2PL prevents cascade of ABORTs, and enables recovery, when a transaction has to be canceled

Transaction Canceling

- When a transaction is canceled
  - Its changes to the DB have to be canceled (Rollback)

- If Strict 2PL is used
  - There is no need to cancel other transactions, based on an ABORT of a transaction
  - There is no cascade of ABORTs

Transaction Atomicity

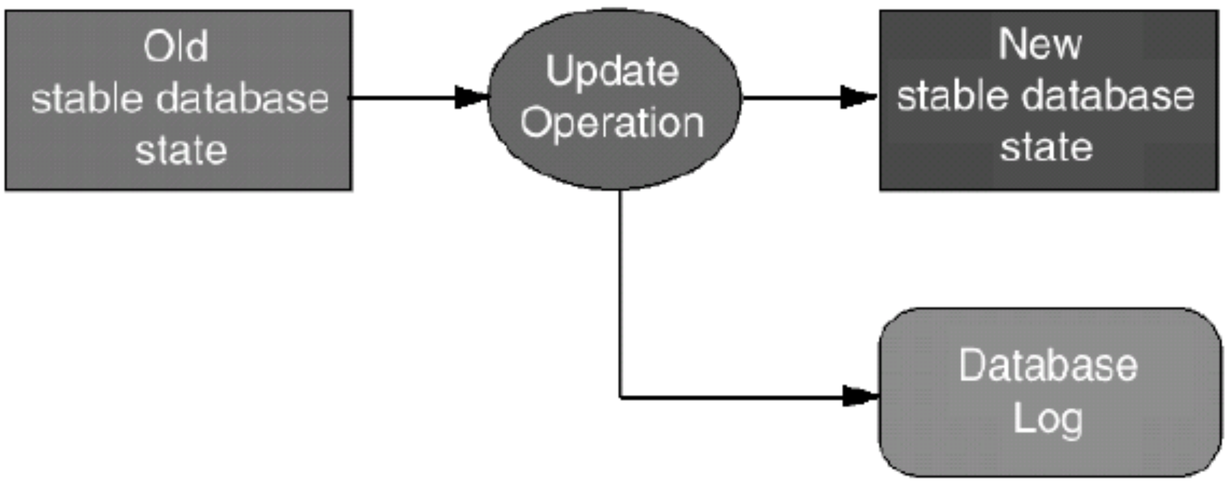
- A transaction writes to the disk the changes it made
- How can we know if the transaction COMMITed?
  - Will write to the disk (in the proper area) that the transaction COMMITed
  - The transaction has to write to the disk all the changes it performed before it writes to the disk that it COMMITs
- If the machine falls before the transaction wrote to the disk COMMIT, how can we cancel the changes it made?
  - It is required to write to the disk the changes twice (once while performing, and second after COMMIT)

Write-Ahead Log (AWL)

- The log (LOG) is a dedicated area in the disk in which the transaction writes:
  - All the changes it makes to the database
  - Additional crucial information, such as that the transaction COMMITed

- Changes are written to the database only after they were written to the LOG on the disk

LOG


Old
stable database
state
Update
Operation
New
stable database
state
Database
Log

Execute Transaction as Atomic

- When it ends, the transaction writes COMMIT to the LOG
  - This write is ATOMIC

- If in the LOG it is written - COMMIT
  - It is possible to recover the changes the transaction made

- If in the LOG there is no COMMIT
  - It is possible to cancel the changes that the transaction made

Multiple Writing – to the LOG and then to the DB – Brings Efficiency

- The LOG is located on a specific disk
- Writing to the LOG is sequential
- The amount of information in the LOG is small, such that:
  - The LOG requires, for each transaction, small number of blocks
  - The LOG enables No-Force and Steal, and for that it is efficient

Buffer Pool

- Blocks (pages) are being read from the disks to the buffers in the internal memory
- Transactions read from the buffers and write to them
- In principle, at the end, a transaction has to write to the disks the blocks it changed in the buffers
- But, in practice, it is better to keep these blocks in the memory, since maybe another transaction will need them
- **No Force** – a transaction that COMMITed is not forced to write to the DB (disk), since it is possible to restore from the LOG

What to Do When the Buffers are Full?

- In the buffers it is possible to store simultanuously a limited number of blocks

- If a transaction has to read from the disk an additional block of the data and there is no room in the memory, then:
  - If there is in the memory a buffer (blocks) that is **not locked** by any transaction – it is possible to free this block, but first it has to be written to the disk

Steal

- If all the buffers in the internal memory are occupied, then:
  - It is possible to "steal" a buffer, but first it has to be written to the disk
    - Part of the changes are made by transactions that COMMITed and ended
    - Some changes are made by transactions that currently lock records that are in the block and were not ended yet
  - Using the LOG, it is always possible to CANCEL the changes that were made in the buffer by transactions that didn't end yet

UNDO,REDO – Steal and No Force

- Steal – cache can be flushed before the transaction Commits (UNDO)
- No-Force – cache is deferred until some transactions commit (REDO)

	Steal	**No-steal**
Force	Undo, No-Redo	No-Undo, No-Redo
No-force	Undo, Redo	No-Undo, Redo

What the LOG Enables – a Summary

- The LOG enables to restore in the database changes that were performed by transactions that COMMITed

- The LOG enables to CANCEL in the database changes that were made by transactions that didn’t COMMIT

- The LOG enables No Force and Steal and that way shortens the response time of the system

Simple LOG Model

- For each record in the LOG there is a field with an identifier that is called – Log Sequence Number – LSN
  - The records are written to the LOG incrementally, according to the LSN
- Additionally, there are fields for:
  - Transaction id – related to the record
  - Type of record (Update/Commit/Abort/End)
  - The previous LSN of the same transaction id (the LOG records for a specific transaction are linked backwards)
- There are additional fields according to the type of transaction

A LOG record for Update

- For each item A that is updated, there is in the LOG a record with:
  - LSN – an identifier
  - The LSN of a previous record for the same transaction
  - The identifier of the transaction
  - The type of record (here only update)
  - The value of A before the update (Before Image)
  - The value of A after the update (After Image)

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			

ABORT of a Specific Transaction

- If we want to CANCEL a transaction that didn't COMMIT yet
  - The transaction still holds locks on all its items that were updated
- Write an ABORT record to the LOG for the transaction
- Perform UNDO to the transaction operations

Perform UNDO for a Specific Operation

- Go backwards through the LOG records from the end to the beginning
  - Foreach UPDATE record of the transaction, write to the Database the value of the item before the UPDATE

- Eventually, after going backwards through all the updates, all the items that were updated by the transaction will return to their previous values

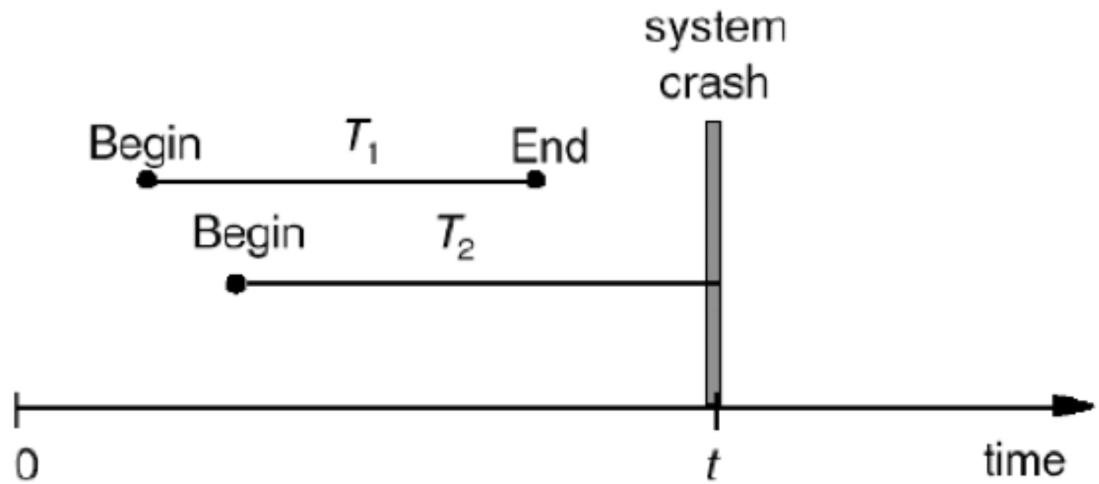
- Release all the locks the transaction held

Recovery from a Failure

- When the system falls, we have to go over the LOG (from the beginning) and do three things:
  - Analysis Phase
  - Redo Phase
  - Undo Phase

Recovery from a Failure

- Upon recovery:
  - All T_1's effects should be reflected in the database (REDO if necessary)
  - All of T_2's effects should be reflected in the database (UNDO if necessary)



Analysis Phase

- Going over the LOG (from the beginning) forward and check which transactions in the log COMMITed and which didn't COMMIT
- Changes that were made by transactions that COMMITed – REDO
- Changes that were made by transactions that didn't COMMIT - UNDO

REDO Phase

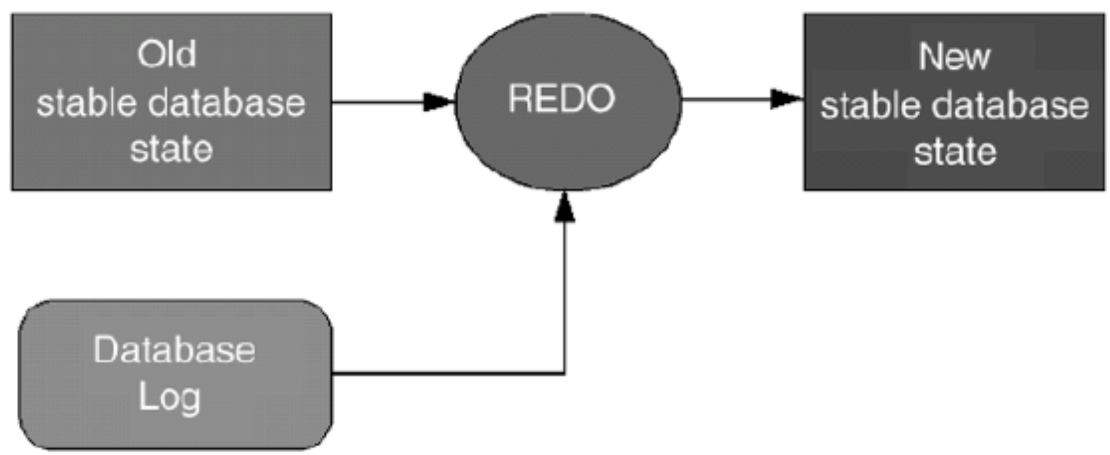
- In this phase the operations that were made by transactions that COMMITed already are redone

- Going through the LOG forward from the beginning to the end

  - Foreach update record, the value of the item after the update is written to the database

REDO

- REDOing an action means performing it again
- The REDO operation uses the log information and performs the action that might have been done before, or not done due to failures
- The REDO operation generates the new image

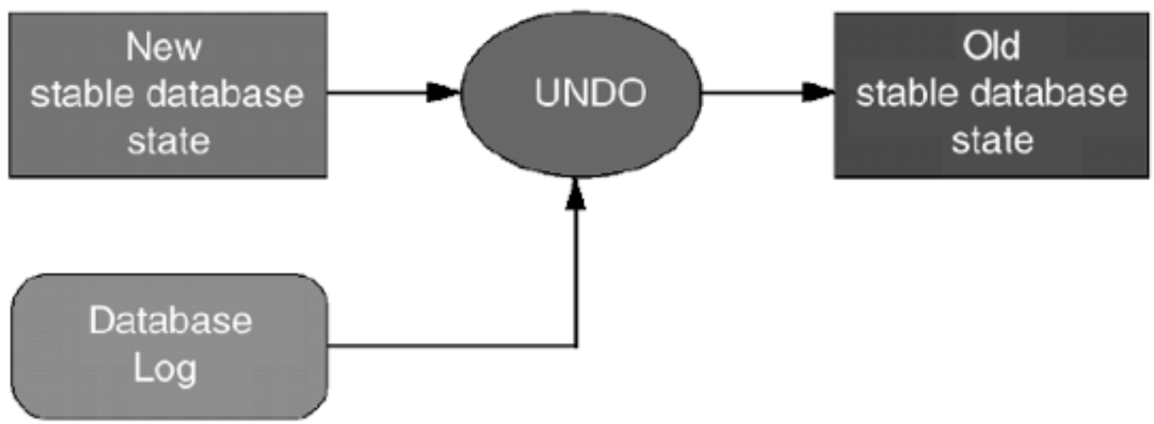


UNDO Phase

- In this phase all the updates that were written by transactions that didn’t COMMIT are canceled
- Going over the LOG backwards from the end to the beginning
  - Foreach update record of a transaction that didn’t COMMIT, the value of the item before the update is written to the database
- The result is – all the changes to the database reflect exactly the writes that were performed by transactions that COMMITed

UNDO

- UNDOing an action means to restore the object to its before image
- The UNDO operation uses the log information and restores the old value of the object



Failure While Performing Recovery

- If there is a failure while performing recovery, then we restart the recovery process
- The result of the recovery process is dependent only on:
  - What is written in the LOG
  - And not in what is written in the database

Checkpoint

- Performing recovery since the very beginning of the LOG is inefficient

- For that reason, from time to time, a checkpoint is performed
  - New transactions are not accepted and executed, and finishing all the transactions that already run
  - Write all the changes to the database
  - Write a checkpoint record to the LOG
  - Continuing to accept and execute transactions

- Recovery will be performed from the last Checkpoint

Backup

- The database disks have to be backed-up, to enable to restore the database in case of a disk destruction

- Write to the LOG that a backup was performed

- If the disks were destructed, it is required to copy to the DB the backup copy and perform recovery (using the LOG) from the last backup record (the latest) that is in the LOG

The Aries Recovery Algorithm

- The Aries algorithm enables
  - Fuzzy checkpoint and backup
  - To decrease the number of writing operations to the DB (disk) by storing simple information
  - Easy (relatively) to understand and implement

The Aries Recovery Algorithm

The ARIES recovery algorithm consists of three steps:

- **Analysis**: step identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of crash
- **Redo**: necessary redo operations are applied
- **Undo**: the operations of transactions are undone in reverse order

The Aries Recovery Algorithm

- During checkpointing the following tables are stored:
  - Transaction Table: transaction ID, status and the LSN of the most recent log record for the transaction
  - Dirty Page Table: entry for each dirty page in the buffer: page ID, LSN corresponding to the earliest update to the that page

The 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
    - Writes the LSN of the begin_checkpoint record to a special file. Thus special file is accessed during recovery to locate the last checkpoint information.
  - ARIES uses “fuzzy checkpointing”

ARIES: Analysis Phase

- Aim: rebuild the dirty pages and transaction table
  - Start at the begin_checkpoint record and proceed to the end of the LOG
  - Transactions table and dirty page tables are updated
  - Create the undo_set consists of uncommitted transactions
  - NOTE: during this phase, some other LOG records may be written to the LOG and transaction table may be modified

ARIES: REDO Phase

- Starts from the smallest LSN of all the dirty pages in the Dirty Page Table
- Scans forward to the end of the LOG
- For each change recorded in the LOG:
  - Check if the update has to be reapplied
  - If a change recorded in the LOG refers to page P that is not in the Dirty Page Table, then this change is already on disk and does not need to be reapplied
  - If a change recorded in the LOG (with LSN = N, say) refer to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then the change is already present
  - If neither of these two conditions hold, page P is read from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), rewritten to disk
  - If non of the above apply then write the page to the disk

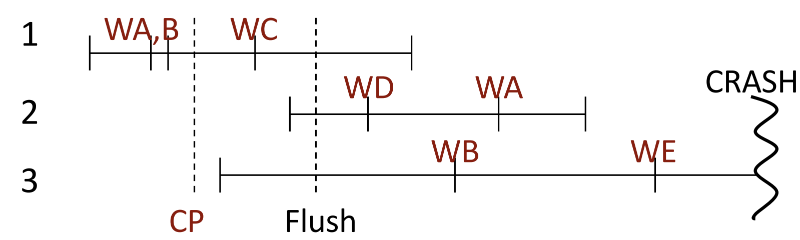
ARIES: UNDO Phase

- Starts from the end backwards
  - Undo all actions of transactions in the undo_set
  - For each undo writes the “before: in the LOG

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM



Main Memory

Transaction Table

Tid	Last LSN	Status

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0

CP

CP

Flush
(reset Dirty Pages)

Disk

Transaction Table

Tid	Last LSN	Status

Dirty Page

Page	LSN

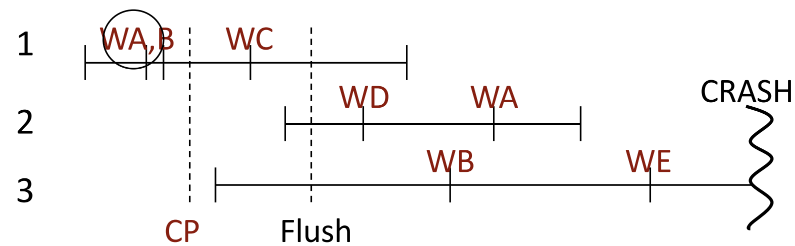
Data

Page	Img	LSN
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1



Transaction Table

Tid	Last LSN	Status
1	1	Active

Dirty Page

Page	LSN
A	1

Data

Page	Img	LSN
A	1	1
B	0	0
C	0	0
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status

Dirty Page

Page	LSN

Data

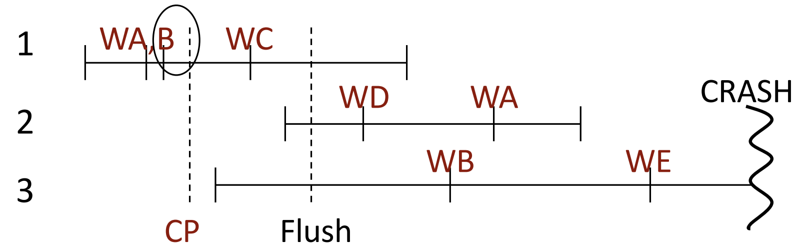
Page	Img	LSN
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2



Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	0	0
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status

Dirty Page

Page	LSN

Data

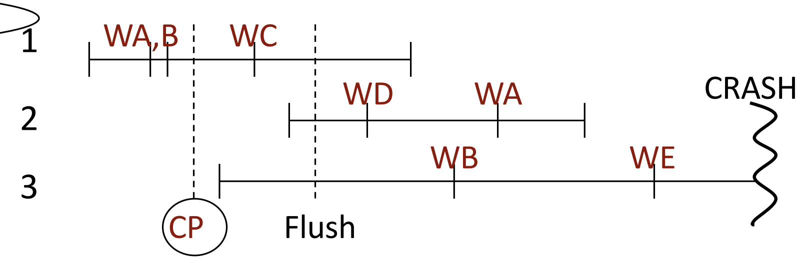
Page	Img	LSN
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					



Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

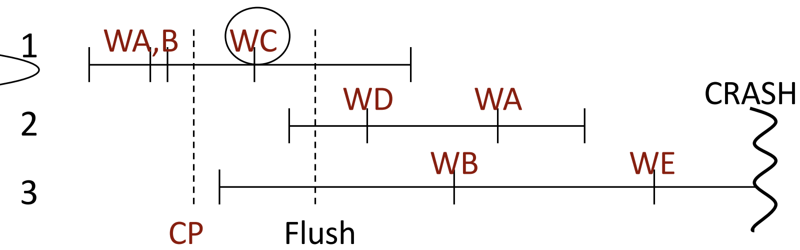
Data

Page	Img	LSN
A	1	1
B	2	2
C	0	0
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N



Transaction Table

Tid	Last LSN	Status
1	4	Active

Dirty Page

Page	LSN
A	1
B	2
C	4

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

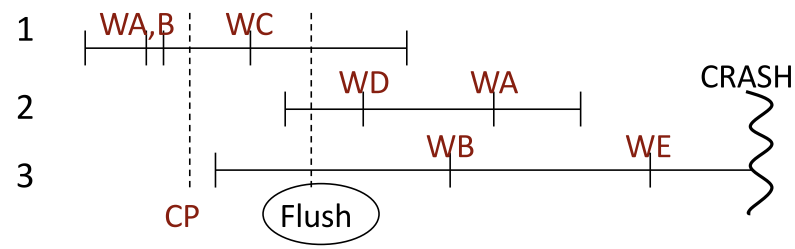
Page	Img	LSN
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N



Transaction Table

Tid	Last LSN	Status
1	4	Active

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

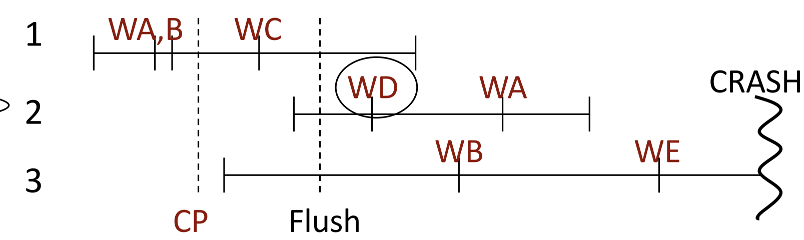
Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q



Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	4	Active
2	5	Active

Dirty Page

Page	LSN
D	5

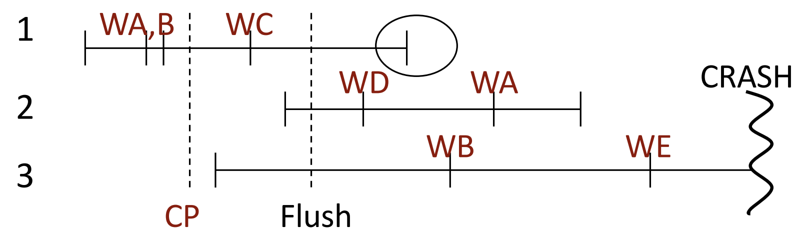
Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	Q	5
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			



Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	5	Active

Dirty Page

Page	LSN
D	5

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	Q	5
E	0	0

Main Memory

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

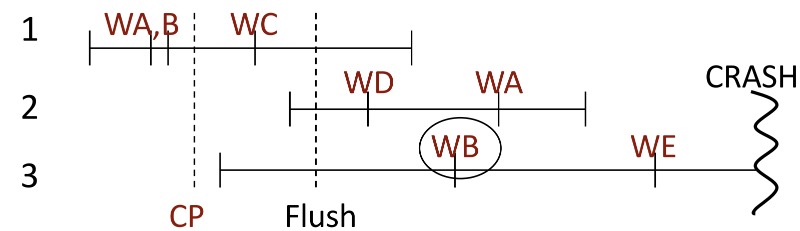
Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S



Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	5	Active
3	7	Active

Dirty Page

Page	LSN
D	5
B	7

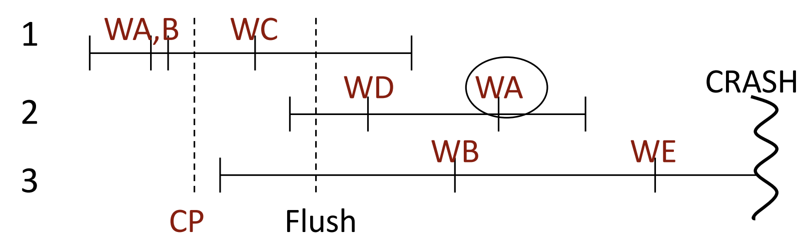
Data

Page	Img	LSN
A	1	1
B	S	7
C	N	4
D	Q	5
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T



Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	8	Active
3	7	Active

Dirty Page

Page	LSN
D	5
B	7
A	8

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	0	0

Main Memory

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

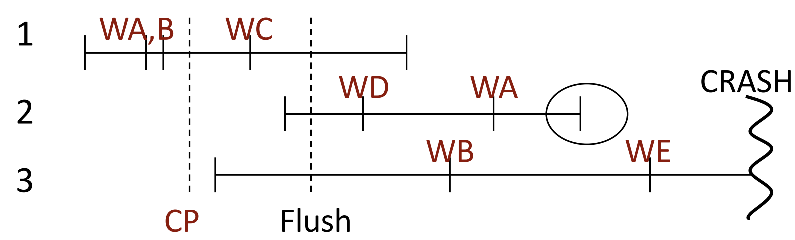
Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			



Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	7	Active

Dirty Page

Page	LSN
D	5
B	7
A	8

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

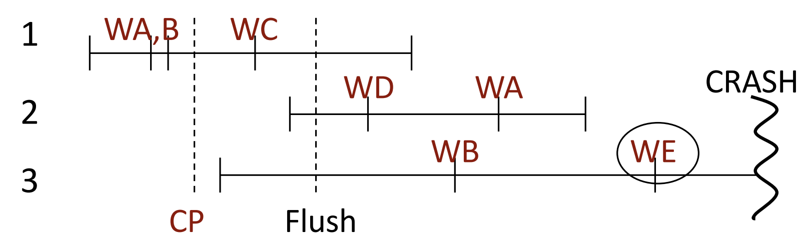
Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2



Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
D	5
B	7
A	8
E	10

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

Main Memory

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

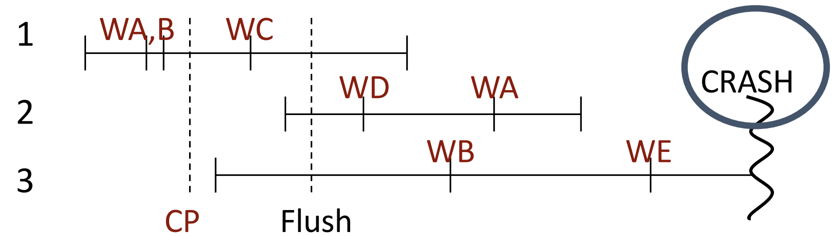
Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0


A problem has been detected and windows has been shut down to prevent damage
to your computer.
DRIVER_IRQL_NOT_LESS_OR_EQUAL
If this is the first time you've seen this Stop error screen,
restart your computer, If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)
ress F86B5A89 base at F86B5000, DateStamp 3dd991eb
O hai
inistrator or technical support group for further

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2



Transaction Table

Tid	Last LSN	Status

Dirty Page

Page	LSN

Data

Page	Img	LSN

Main Memory

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Recovery begins – Analysis Phase

- Aim: rebuild the dirty page and transactions table
  - Start at the begin_checkpoint record and proceed to the end of the LOG
  - Transaction and dirty pages tables are appended according to the LOG
  - Create the undo_set that consists on uncommitted transactions

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Analysis Begins in last CP (LSN=3) and recover the tables

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Transaction Table

Tid	Last LSN	Status
1	4	Active

Dirty Page

Page	LSN
A	1
B	2
C	4

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	4	Active
2	5	Active

Dirty Page

Page	LSN
A	1
B	2
C	4
D	5

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	5	Active

Dirty Page

Page	LSN
A	1
B	2
C	4
D	5

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	5	Active
3	7	Active

Dirty Page

Page	LSN
A	1
B	7
C	4
D	5

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	8	Active
3	7	Active

Dirty Page

Page	LSN
A	8
B	7
C	4
D	5

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	7	Active

Dirty Page

Page	LSN
A	8
B	7
C	4
D	5

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
A	8
B	7
C	4
D	5
E	10

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Analysis Phase is Completed

- We correctly recovered the Transaction Table

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

=

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Recovered by Analysis Phase

Last Table in Memory Before Crash

- The Undo_Set = {3}

Analysis Phase is Completed

- Note that Dirty Page table is not Identical – Because the Flush Operations are not in Log

Dirty Page

Page	LSN
A	8
B	7
C	4
D	5
E	10

Recovered by Analysis Phase

$\neq$

Dirty Page

Page	LSN
D	5
B	7
A	8
E	10

Last Table in Memory Before Crash

- But who cares ? We have the Redo Phase

2. Redo Phase …

ARIES Example

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

Redo Begins from the smallest LSN of the Dirty Page Table (LSN=4)

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
A	8
B	7
C	4
D	5
E	10

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

1. if a change recorded in the log refers to page P that is not in the Dirty Page Table, then this change is already on disk and do nothing.
2. if a change recorded in the log (with LSN = N, say) refers to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then do nothing
3. Read page P from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), then the change has been applied and <u>do nothing</u>.
4. If none of the above apply then write the page to the disk

✗ $P \notin DPT^{M}$ → Nothing

✗ $LSN(P)^{DPM} > N^{log}$ → Nothing

$LSN(P)^{Disk} >= N^{log}$ √ → Nothing

If none → write page to data

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
A	8
B	7
D	5
E	10

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	0	0
E	0	0

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

1. if a change recorded in the log pertains to page P that is not in the Dirty Page Table, then this change is already on disk and do nothing. ✗
2. if a change recorded in the log (with LSN = N, say) pertains to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then do nothing ✗
3. Read page P from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), then the change has been applied and do nothing. ✗
4. If none of the above apply then write the page to the disk √

$P \notin DPT^{M}$
→ Nothing

$LSN(P)^{DPM} > N^{log}$
→ Nothing

$LSN(P)^{Disk} >= N^{log}$
→ Nothing

If none → write page to data

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	Q	5
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
A	8
B	7
E	10

Data

Page	Img	LSN
A	1	1
B	2	2
C	N	4
D	Q	5
E	0	0

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

1. if a change recorded in the log pertains to page P that is not in the Dirty Page Table, then this change is already on disk and do nothing. ✗
2. if a change recorded in the log (with LSN = N, say) pertains to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then do nothing ✗
3. Read page P from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), then the change has been applied and do nothing. ✗
4. If none of the above apply then write the page to the disk √

$P \notin DPT^M$
→ Nothing

$LSN(P)^{DPM} > N^{log}$
→ Nothing

$LSN(P)^{Disk} >= N^{log}$
→ Nothing

If none → write page to data

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	1	1
B	S	7
C	N	4
D	Q	5
E	0	0

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
A	8
E	10

Data

Page	Img	LSN
A	1	1
B	S	7
C	N	4
D	Q	5
E	0	0

Main Memory

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

1. if a change recorded in the log pertains to page P that is not in the Dirty Page Table, then this change is already on disk and do nothing. ✗
2. if a change recorded in the log (with LSN = N, say) pertains to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then do nothing ✗
3. Read page P from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), then the change has been applied and do nothing. ✗
4. If none of the above apply then write the page to the disk √

$P \notin DPT^{M}$
→ Nothing

$LSN(P)^{DPM} > N^{log}$
→ Nothing

$LSN(P)^{Disk} >= N^{log}$
→ Nothing

If none → write page to data

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	0	0

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN
E	10

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	0	0

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

1. if a change recorded in the log pertains to page P that is not in the Dirty Page Table, then this change is already on disk and do nothing. ✗
2. if a change recorded in the log (with LSN = N, say) pertains to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then do nothing ✗
3. Read page P from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), then the change has been applied and do nothing. ✗
4. If none of the above apply then write the page to the disk √

$P \notin DPT^{M}$
→ Nothing

$LSN(P)^{DPM} > N^{log}$
→ Nothing

$LSN(P)^{Disk} >= N^{log}$
→ Nothing

If none → write page to disk

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

Redo Phase is Completed

- We correctly recovered the data

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

After Redo Phase

=

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

Before Crash

3. Undo Phase…

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2

1. if a change recorded in the log pertains to page P that is not in the Dirty Page Table, then this change is already on disk and do nothing. ✗
2. if a change recorded in the log (with LSN = N, say) pertains to page P and the Dirty Page Table contains an entry for P with LSN greater than N, then do nothing ✗
3. Read page P from disk and the LSN stored on that page, LSN(P), is compared with N. If N <= LSN(P), then the change has been applied and <u>do nothing</u>. ✗
4. If none of the above apply then write the page to the disk √

$P \notin DPT^M$
→ Nothing

$LSN(P)^{DPM} > N^{log}$
→ Nothing

$LSN(P)^{Disk} >= N^{log}$
→ Nothing

If none → write page to disk

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

Main Memory

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	2	10

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2
11	UND	3	10	E	2	0

The Undo_Set = {3}

* Begins with the last operation of T3

* For each update perform undo if the page LSN is equal to the current LSN √

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	0	11

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	T	8
B	S	7
C	N	4
D	Q	5
E	0	11

Main Memory

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2
11	UND	3	10	E	2	0
12	UND	3	11	B	S	2

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	T	8
B	2	12
C	N	4
D	Q	5
E	0	11

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit
3	10	Active

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	T	8
B	2	12
C	N	4
D	Q	5
E	0	11

Main Memory

Log

LSN	Type	Tid	PrevLSN	Page	BFIM	AFIM
1	UP	1	0	A	0	1
2	UP	1	1	B	0	2
3	CP					
4	UP	1	2	C	0	N
5	UP	2	0	D	0	Q
6	CM	1	4			
7	UP	3	0	B	2	S
8	UP	2	5	A	1	T
9	CM	2	8			
10	UP	3	7	E	0	2
11	UND	3	10	E	2	0
12	UND	3	11	B	S	2

* Completed with undo of Transaction 3 – Because last previous LSN is equal to 0
* Update the transaction Table by deleting it from the transaction Table

Disk

Transaction Table

Tid	Last LSN	Status
1	2	Active

Dirty Page

Page	LSN
A	1
B	2

Data

Page	Img	LSN
A	T	8
B	2	12
C	N	4
D	Q	5
E	0	11

Transaction Table

Tid	Last LSN	Status
1	6	Commit
2	9	Commit

Dirty Page

Page	LSN

Data

Page	Img	LSN
A	T	8
B	2	12
C	N	4
D	Q	5
E	0	11

Main Memory

THE END