Discussion Worksheet: https://tinyurl.com/y6flpzul
Discussion Slides: https://tinyurl.com/186-fa20-disc9

Discussion 9

Recovery

Please turn on your videos!

Announcements

Project 4 Part 1 due Sunday, Nov 8th at 11:59 PM. Part 2 due Thurs. Nov 12th.

Vitamin 8 (Recovery) is due **Monday, Nov 2nd at 11:59 PM**.

Exam Prep 4 is Friday, October 30th from 6 - 8 PM PST

- Will cover Transactions and Recovery

Midterm 2 is Monday, **November 2nd** from **5:30 PM PST - 7:30 PM PST**.

- Midterm 2 Review Session is scheduled for 10/29 from 7-9 pm PST.

Write-Ahead Logging

ACID (recap)

- We want transactions to obey **ACID**
  - **atomicity**: *all* operations in a transaction happen, or *none* of them
  - **consistency**: database consistency (unique constraints, etc) is maintained
  - **isolation**: should look like we only run 1 transaction at a time (even if we run multiple concurrently)
  - **durability**: once a transaction commits, it persists

ACID (recap)

- We want transactions to obey **ACID**
    - **atomicity**: *all* operations in a transaction happen, or *none* of them
    - **consistency**: database consistency (unique constraints, etc) is maintained
    - **isolation**: should look like we only run 1 transaction at a time (even if we run multiple concurrently)
    - **durability**: once a transaction commits, it persists

How do we efficiently maintain **atomicity** and **durability**, if the database can crash at any time?

Steal/No-steal

Whether or not modified pages from an uncommitted xact can be flushed to disk

- **No-Steal** = Don’t Allow
  - Easy to maintain atomicity
  - If we crash mid-transaction, none of the changes were written to disk!
- **Steal** = Allow
  - Faster, avoids some problems
  - Allows buffer manager to use frames optimally
  - Allows uncommitted changes to make it to disk so we may need to undo these

Force/No-force

Whether or not modified pages from a transaction are forced to disk on commit

- **Force** = Enforce
  - Easy to maintain durability
  - If we crash after committing, everything was already written to disk!
- **No-Force** = Don’t Enforce
  - Faster, more difficult to maintain durability
  - Don’t have to do unnecessary writes
  - If DB crashes mid-transaction, we’ll need to redo any changes that didn’t make it to disk

No-Steal, Force Example

T_1	Write(A)	Write(B)			*CRASH*	Commit
T_2			Write(B)	Commit		

- **No-Steal** = Don't Allow modified pages to be flushed to disk before Commit
  Force = Force modified pages to be flushed to disk on Commit
  - T_1: page A will not be modified on disk. T_1 crashed before Commit (*No-steal*)
  - T_2: page B is modified on disk. (*Force*)
- Durability of database maintained. Committed Transactions have their modifications flushed to disk.
- Atomicity is not maintained. T_1 didn't commit, but it's changes to page B still made it to disk!

Steal, No-Force Example

T_1	Write(A)	Write(C)	*Flush to disk*		*CRASH*	Commit
T_2			Write(B)	Commit		

- **Steal** = Allow modified pages to be flushed to disk before Commit,
  No-Force = Do not force modified pages to be flushed to disk on Commit
  - T_1: Pages A, C are modified on disk. (*Steal*)
  - T_2: page B not modified on disk (*No-force*)
- Without additional policies in place, Atomicity and Durability are *not* maintained:
  T_1's changes are flushed to disk despite not committing,
  T_2's changes are not flushed to disk despite committing.

Steal, No-Force

Usually implement Steal, No-Force because they are the most performant policies (faster to execute)

- **Steal**: Allows uncommitted changes to end up on disk
  - To maintain atomicity, we need an UNDO phase to remove the changes on disk from a transaction that aborts before committing
- **No-Force**: Doesn’t guarantee changes are flushed to disk on commit
  - To maintain durability, we need a REDO phase to flush changes to disk from a transaction that aborts after committing

Logging

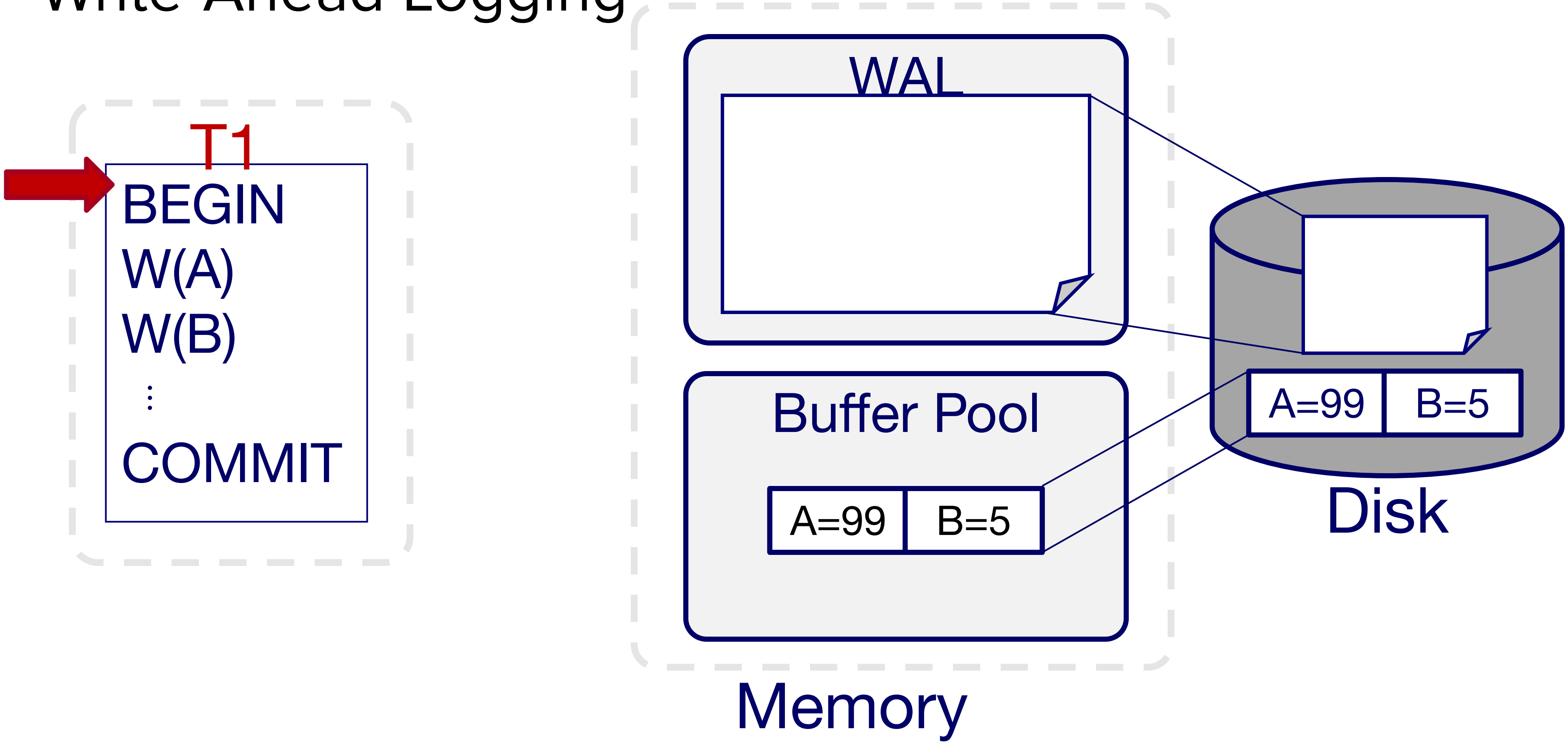
- Log: append-only file containing log records
- Need some way of knowing what changes were made so we can later REDO/UNDO them if the system crashes
- Any kind of DB write operation (insert, update, delete) gets an UPDATE log record.
- Also have COMMIT, ABORT, and END log records

Write-Ahead Logging

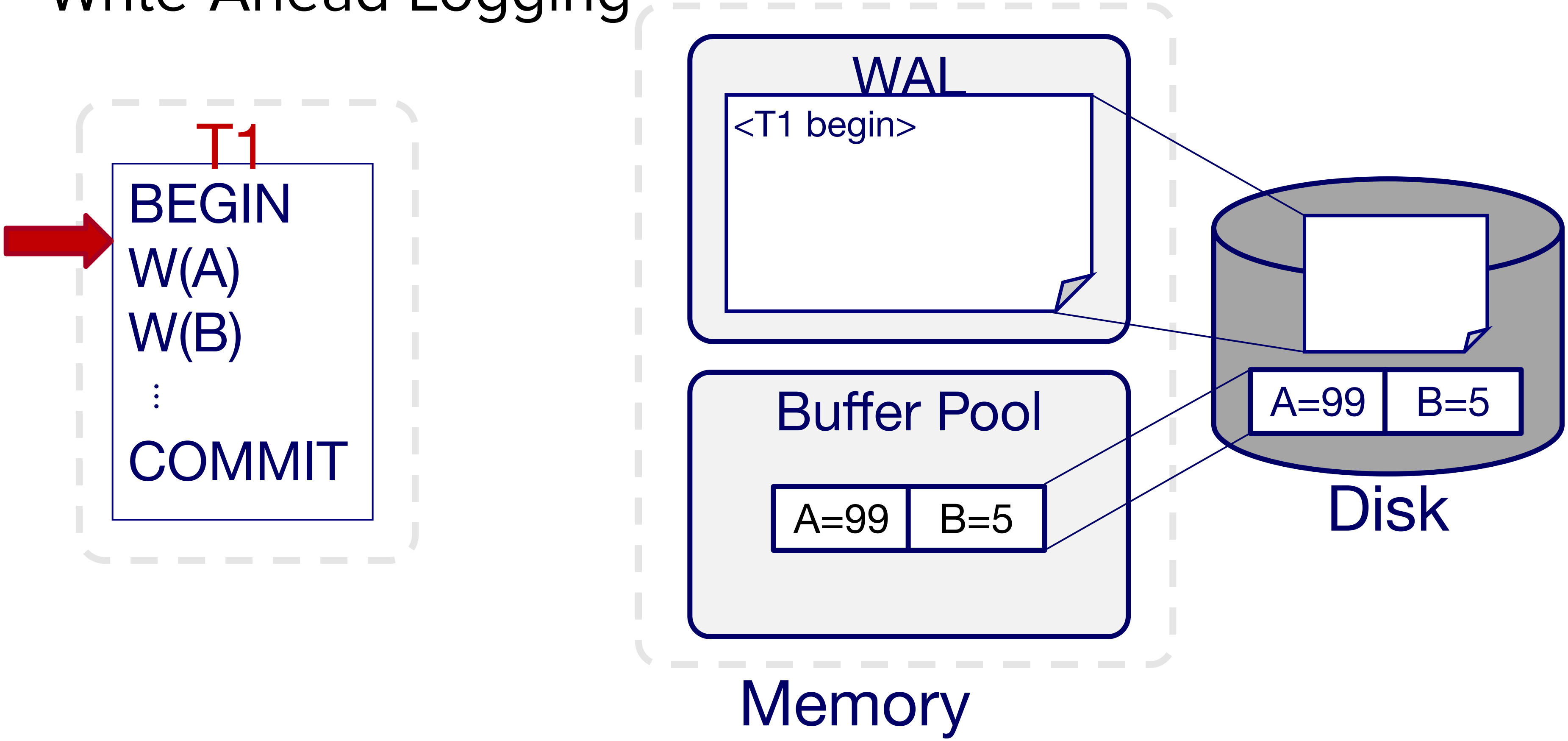
- Two Rules:
  - A transaction is not committed until the log records for all its changes have been written to disk
  - Changes must be logged before the actual data is modified on disk

- With these two rules we can develop a system that recovers properly from failures

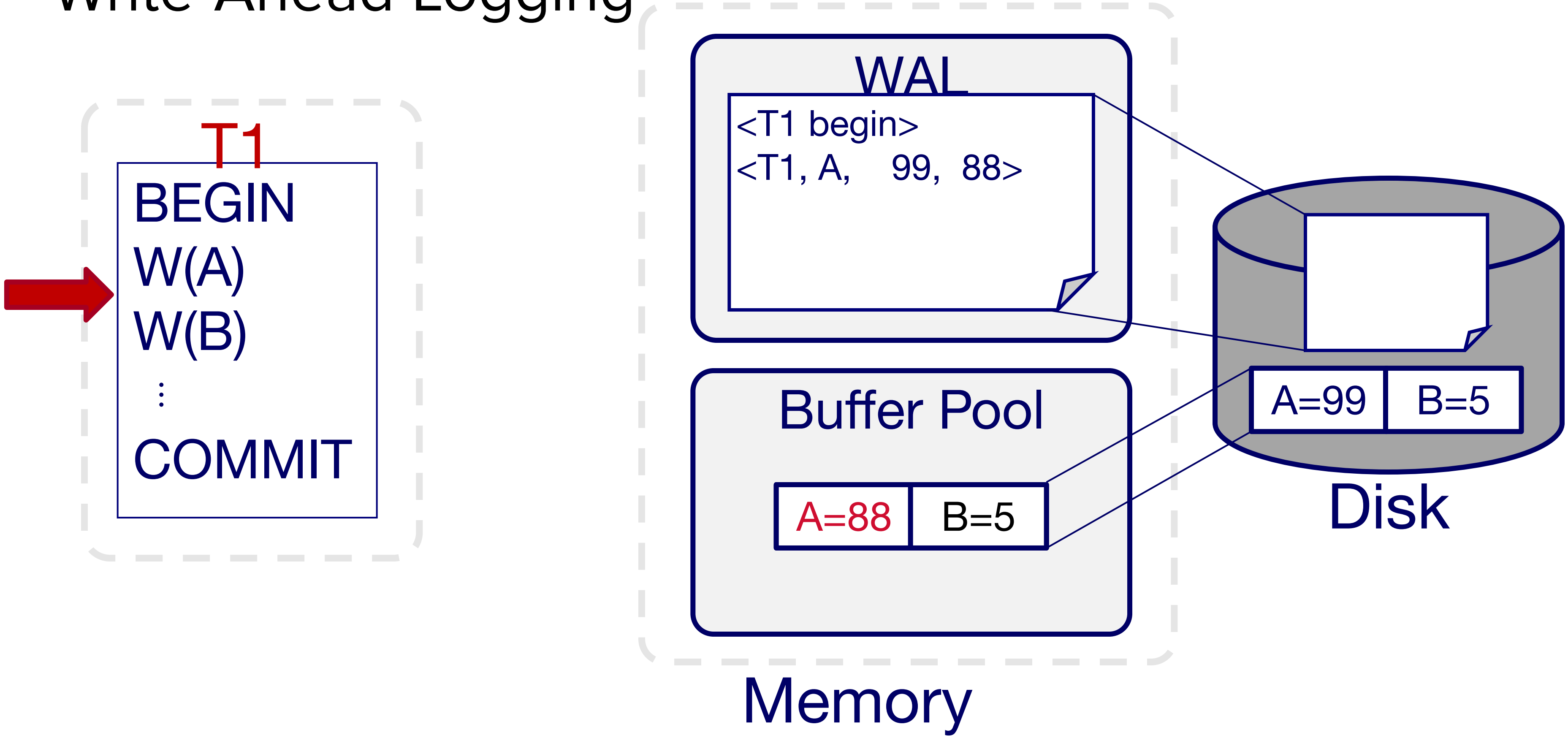
Write-Ahead Logging



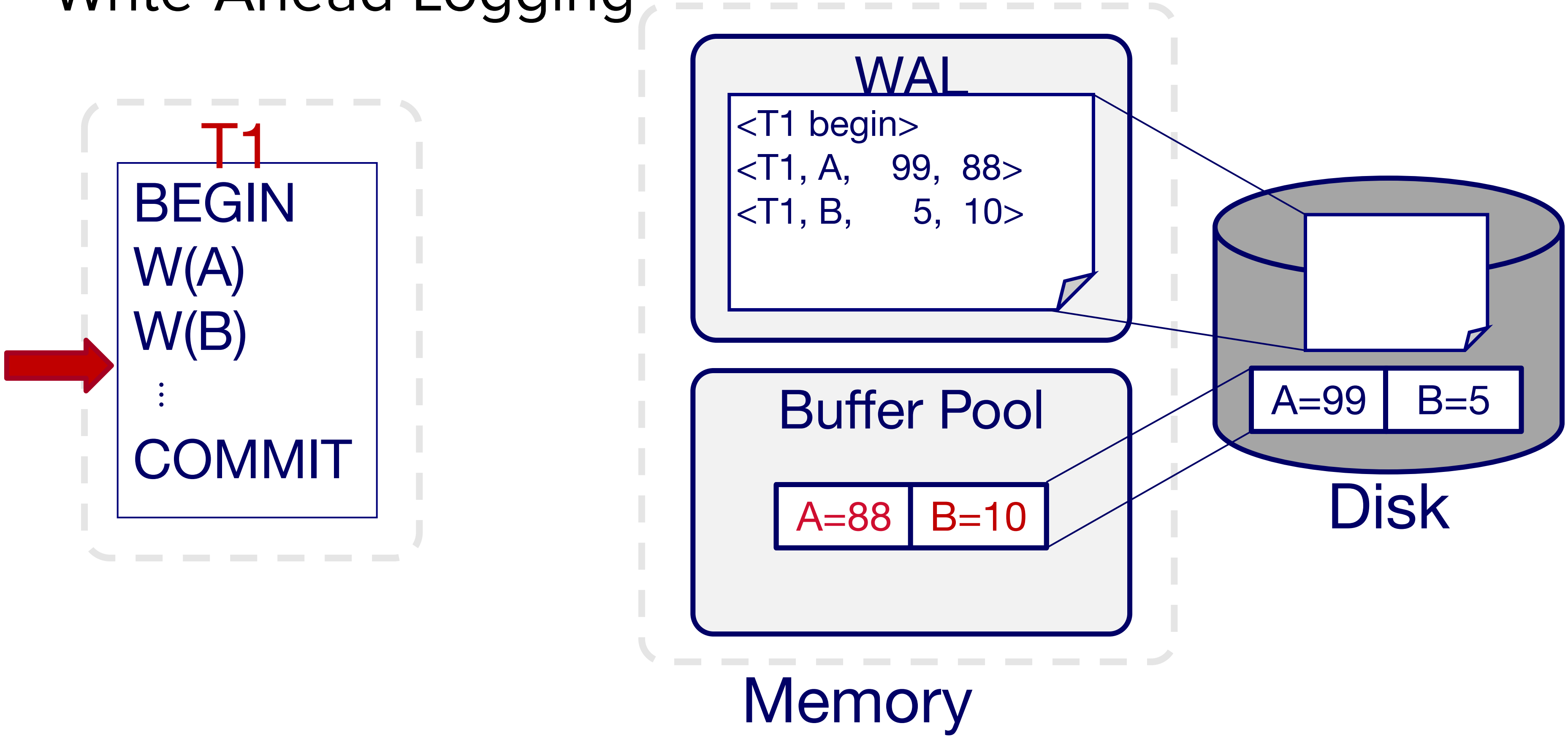
Write-Ahead Logging



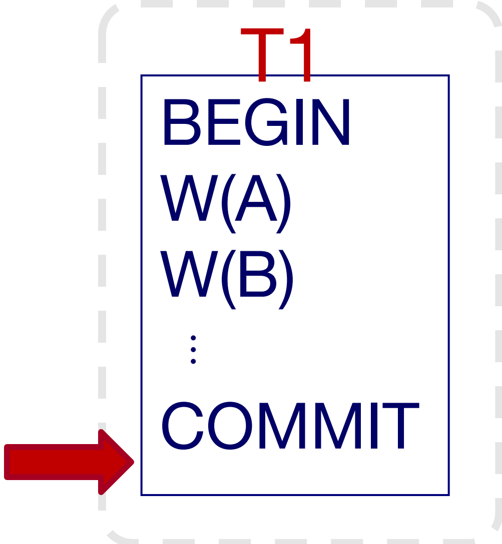
Write-Ahead Logging

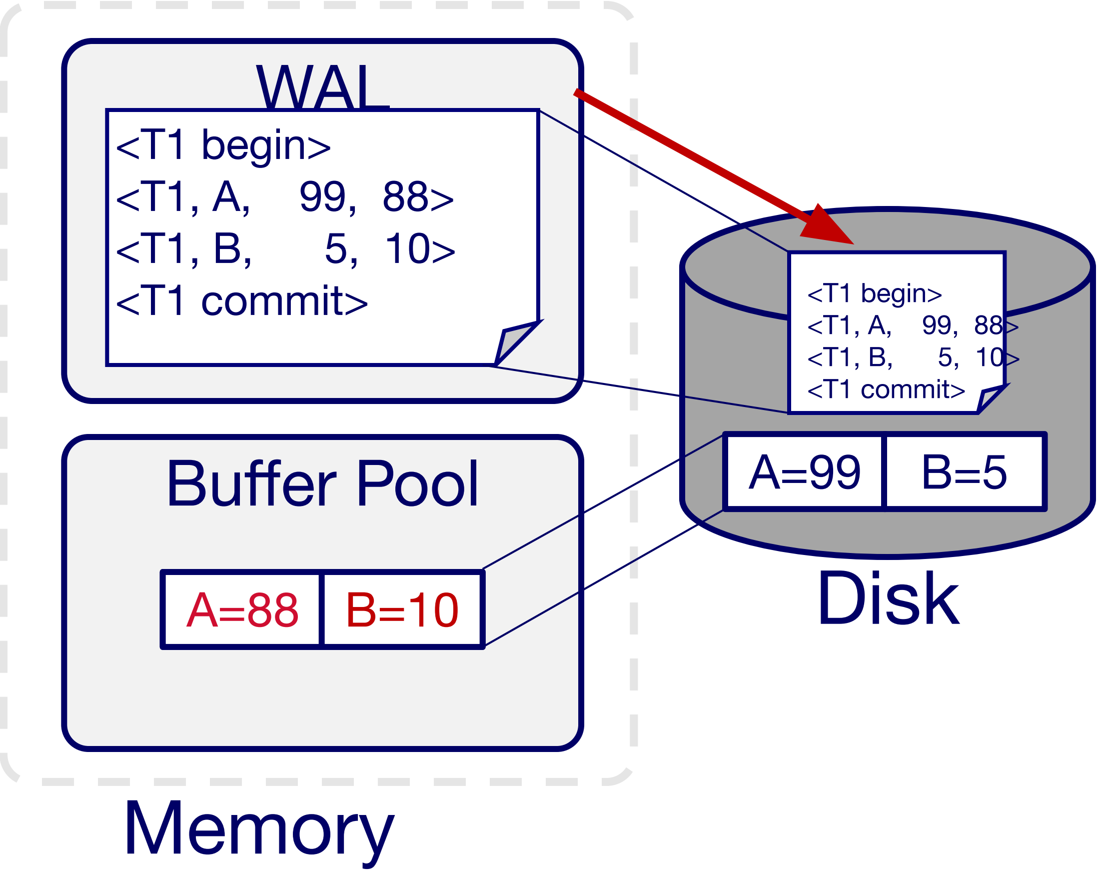


Write-Ahead Logging



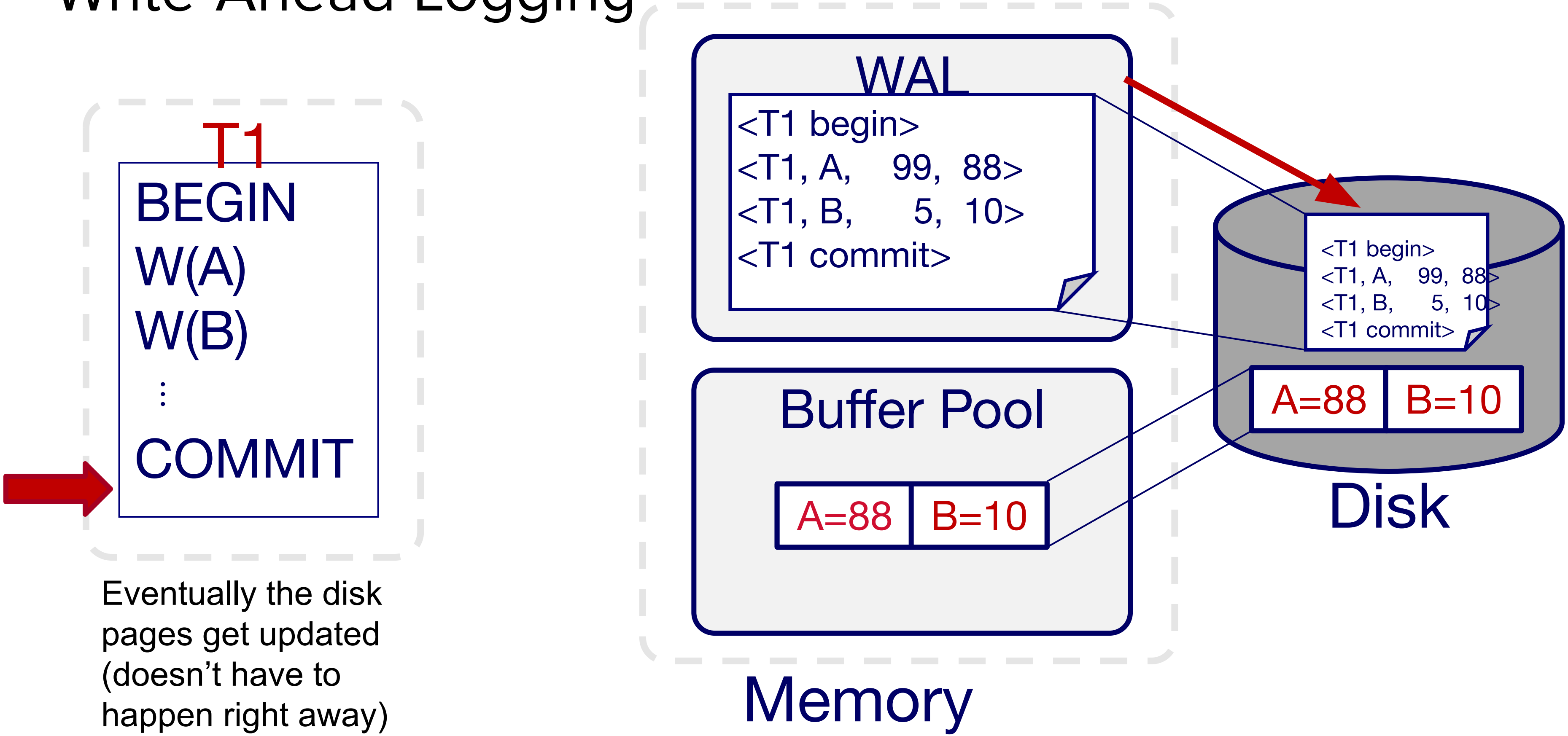
Write-Ahead Logging





Because log records are on disk, it is safe to update disk values now

Write-Ahead Logging



Eventually the disk pages get updated (doesn't have to happen right away)

Undo Logging

- Write log records to ensure **Atomicity** after a system crash:
  - **`<START T>`**: transaction T has begun
  - **`<COMMIT T>`**: T has committed
  - **`<ABORT T>`**: T has aborted
  - **`<T,X,v>`**: T has updated element X, and its <u>old</u> value was v
- If T commits, then **FLUSH(X)** must be written to disk before **`<COMMIT T>`**
  - **Force** – we can UNDO any modifications if a Xact crashes before COMMIT
- If T modifies X, then **`<T,X,v>`** log entry must be written to disk before **FLUSH(X)**
  - **Steal** – we can UNDO any modifications if a Xact crashes before FLUSH

Undo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	<T,A,8>
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	<T,B,8>
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

Undo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						`<START T>`
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	`<T,A,8>`
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	`<T,B,8>`
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	
COMMIT						`<COMMIT T>`

CRASH!

If the system crashes *after* `<COMMIT T>`, under the **Force** paradigm, we know the changes have been flushed to disk.

We don't have to undo anything!

Undo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						`<START T>`
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	`<T,A,8>`
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	`<T,B,8>`
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	
COMMIT						`<COMMIT T>`

CRASH!

If the system crashes *before* `<COMMIT T>`, under the **Steal** paradigm, we don't know whether changes have been pushed to disk.

We need to undo any changes because T never committed.

Undo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
						<START T>
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	<T,A,8>
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	<T,B,8>
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

CRASH!

We can UNDO by writing the old values stored in the UNDO log records.

➔ Set A=8, B=8.

Undo Log Q1a

Operation	Mem A	Mem B	Disk A	Disk B	UNDO Log
READ(A)					
READ(B)					
WRITE(A, A+B)					
WRITE(B, A-B)					
FLUSH(A)					
FLUSH(B)					
COMMIT					

Given the column of Operations and that Disk A=7, Disk B=3 at the start, fill in columns Mem A, Mem B, Disk A, Disk B.

Undo Log Q1a

Operation	Mem A	Mem B	Disk A	Disk B	UNDO Log
READ(A)	7		7	3	
READ(B)	7	3	7	3	
WRITE(A, A+B)	10	3	7	3	
WRITE(B, A-B)	10	7	7	3	
FLUSH(A)	10	7	10	3	
FLUSH(B)	10	7	10	7	
COMMIT	10	7	10	7	

Given the column of Operations and that Disk A=7, Disk B=3 at the start, fill in columns Mem A, Mem B, Disk A, Disk B.

Undo Log Q1b

Operation	Mem A	Mem B	Disk A	Disk B	UNDO Log
READ(A)	7		7	3	
READ(B)	7	3	7	3	
WRITE(A, A+B)	10	3	7	3	
WRITE(B, A-B)	10	7	7	3	
FLUSH(A)	10	7	10	3	
FLUSH(B)	10	7	10	7	
COMMIT	10	7	10	7	

CRASH!

If the system crashes right before COMMIT, how do we recover? Fill in the UNDO Log column.

Undo Log Q1b

Operation	Mem A	Mem B	Disk A	Disk B	UNDO Log
					<START T>
READ(A)	7		7	3	
READ(B)	7	3	7	3	
WRITE(A, A+B)	10	3	7	3	<T, A, 7>
WRITE(B, A-B)	10	7	7	3	<T, B, 3>
FLUSH(A)	10	7	10	3	
FLUSH(B)	10	7	10	7	
COMMIT	10	7	10	7	<COMMIT T>

CRASH!

If the system crashes right before COMMIT, how do we recover? Fill in the UNDO Log column.

Undo by setting A = 7, B = 3.

Undo Log Q1c

Operation	Mem A	Mem B	Disk A	Disk B	UNDO Log
					<START T>
READ(A)	7		7	3	
READ(B)	7	3	7	3	
WRITE(A, A+B)	10	3	7	3	<T, A, 7>
WRITE(B, A-B)	10	7	7	3	<T, B, 3>
FLUSH(A)	10	7	10	3	
FLUSH(B)	10	7	10	7	
COMMIT	10	7	10	7	<COMMIT T>

CRASH!

What happens if the system crashes again while we're undoing? How do we recover?

Undo Log Q1c

Operation	Mem A	Mem B	Disk A	Disk B	UNDO Log
					<START T>
READ(A)	7		7	3	
READ(B)	7	3	7	3	
WRITE(A, A+B)	10	3	7	3	<T, A, 7>
WRITE(B, A-B)	10	7	7	3	<T, B, 3>
FLUSH(A)	10	7	10	3	
FLUSH(B)	10	7	10	7	
COMMIT	10	7	10	7	<COMMIT T>

CRASH!

What happens if the system crashes again while we're undoing? How do we recover?

If the system crashes, then we've done some of the undos but maybe not all of them. We need to do all the undos again.

Redo Logging

- Write log records to ensure **Durability** after a system crash:
  - `<START T>`: transaction T has begun
  - `<COMMIT T>`: T has committed
  - `<ABORT T>`: T has aborted
  - *`<T,X,v>`: T has updated element X, and its <u>new</u> value was v*

  Same as in Undo Logging

- If T modifies X, then *both* `<T,X,v>` and `<COMMIT T>` must be written to disk before **FLUSH(X)**
  - **No-Steal, No-Force** – we can REDO any modifications if a Xact crashes before FLUSH

Redo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	REDO Log
						`<START T>`
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	`<T,A,16>`
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	`<T,B,16>`
COMMIT						`<COMMIT T>`
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	

Redo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	REDO Log
						<START T>
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	<T,A,16>
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	<T,B,16>
COMMIT						<COMMIT T>
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	

CRASH!

If the system crashes *after* `<COMMIT T>`, under the **No-Force** paradigm, we aren't sure if changes have been flushed to disk.

We can recover by REDOing the REDO log records.

➜ Set A=16, B=16.

Redo Log Example

Operation	t	Mem A	Mem B	Disk A	Disk B	REDO Log
						`<START T>`
READ(A, t)	8	8		8	8	
t := t*2	16	8		8	8	
WRITE(A, t)	16	16		8	8	`<T,A,16>`
READ(B, t)	8	16	8	8	8	
t := t*2	16	16	8	8	8	
WRITE(B, t)	16	16	16	8	8	`<T,B,16>`
COMMIT						`<COMMIT T>`
FLUSH(A)	16	16	16	16	8	
FLUSH(B)	16	16	16	16	16	

CRASH!

If the system crashes *before* `<COMMIT T>`, under the **No-Steal** paradigm, we know that changes haven't been pushed to disk.

We don't need to do anything to recover.

Redo Log Q1a

Operation	Mem A	Mem B	Disk A	Disk B	REDO Log
READ(A)					
READ(B)					
WRITE(A, A+B)					
WRITE(B, A-B)					
COMMIT					
FLUSH(A)					
FLUSH(B)					

Given the column of Operations and that Disk A=5, Disk B=4 at the start, fill in columns Mem A, Mem B, Disk A, Disk B.

Redo Log Q1a

Operation	Mem A	Mem B	Disk A	Disk B	REDO Log
READ(A)	5		5	4	
READ(B)	5	4	5	4	
WRITE(A, A+B)	9	4	5	4	
WRITE(B, A-B)	9	5	5	4	
COMMIT					
FLUSH(A)	9	5	9	4	
FLUSH(B)	9	5	9	5	

Given the column of Operations and that Disk A=5, Disk B=4 at the start, fill in columns Mem A, Mem B, Disk A, Disk B.

Redo Log Q1b

Operation	Mem A	Mem B	Disk A	Disk B	REDO Log
READ(A)	5		5	4	
READ(B)	5	4	5	4	
WRITE(A, A+B)	9	4	5	4	
WRITE(B, A-B)	9	5	5	4	
COMMIT					
FLUSH(A)	9	5	9	4	
FLUSH(B)	9	5	9	5	

CRASH!

If the system crashes right after COMMIT, how do we recover? Fill in the REDO Log column.

Redo Log Q1b

Operation	Mem A	Mem B	Disk A	Disk B	REDO Log
					<START T>
READ(A)	5		5	4	
READ(B)	5	4	5	4	
WRITE(A, A+B)	9	4	5	4	<T, A, 9>
WRITE(B, A-B)	9	5	5	4	<T, B, 5>
COMMIT					<COMMIT>
FLUSH(A)	9	5	9	4	
FLUSH(B)	9	5	9	5	

CRASH!

How do we recover if the system crashes right after COMMIT? Fill in the REDO Log column.

Redo by setting A = 9, B = 5.

Undo/Redo Logging

- UNDO logging provides **atomicity**
  - Undoes all updates for ***running*** transactions
- REDO logging provides **durability**
  - Redoes all updates for ***committed*** transactions
- Requires us to recover from the beginning of the log!
  - Computationally expensive
- What if we could start mid-way through the log?
- What if we could get the best of both? **Steal / No-Force**

ARIES

ARIES Recovery System

- Combines **UNDO** logging and **REDO** logging
- Provides a Steal / No-Force policy
  - Unlike REDO, **uncommitted data** can be flushed to disk due to Steal
    - *Benefit:* Increases the amount of memory available!
  - Unlike UNDO, **data pages** do not need to be flushed before a transaction can commit due to No-Force
    - *Benefit:* Reduces the latency of transactions!
- Recovery from crash can be sped up with **checkpointing**

ARIES: The Log (WAL)

- Each log record is assigned a **Log Sequence Number (LSN)**, an increasing timestamp

- Each log record has a **prevLSN**, which is the LSN of the last log record for the same transaction

LSN	prevLSN	XID	Log Payload
10	-	1	START
20	10	1	UPDATE
30	-	2	START
40	20	1	COMMIT

Traverse Linked list of prevLSNs to get all log records for a transaction T

ARIES: The Log (WAL)

- Log records are assigned a **Log Sequence Number (LSN)** (increasing "timestamp")
  - **prevLSN**: LSN of previous log record for transaction

- **Compensation Log Record (CLR)**: log entry undoing another (non-CLR) log entry - can never be undone
  - **undoNextLSN**: next-to-be-undone LSN

ARIES: Keeping Track of Page Updates

- **pageLSN**: LSN of last log record that modified the page
  - Stored on the page itself
  - Updated (in memory) when page modified
  - Flushed to disk with page - indicates how up to date the page on disk actually is
  - In-memory pageLSN and on-disk pageLSN may be different!

ARIES: In-memory Data Structure - Transactions Table

- Tracks active transactions
- **lastLSN**: LSN of latest log record for transaction
- Status of transaction (running, aborting, committing)

Transaction Table

XID	Status	lastLSN
1	R	33
2	C	42

ARIES: In-memory Data Structure - Dirty Page Table

- Tracks dirty pages in buffer
- **recLSN**: recovery LSN indicating ***first*** log record that caused page to be dirty (modified in memory but changes not on disk yet)

Dirty Page Table

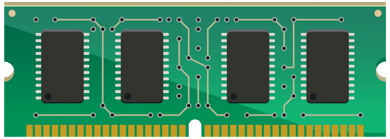
PageID	recLSN
46	11
63	24

ARIES: In-memory Data Structure - Flushed LSN

- One final state ARIES tracks in memory
  - **flushedLSN**: max LSN that was successfully flushed to disk
- $pageLSN_A <= flushedLSN$
  - Before page **A** can be flushed to disk, we need to ensure that the corresponding log records are on disk
  - Necessary for **atomicity**

ARIES Overview

MEMORY



Transaction Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

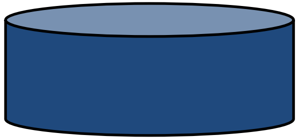
Buffer pool
- Each page has an in-memory pageLSN

Log tail

LOG

LogRecords
- LSN
- prevLSN
- XID
- type
- pageID
- Length (*bytes*)
- Offset (*bytes*)
- Before-image (*old-value*)
- After-image (*new-value*)

DISK



Data pages
- Each page has a disk pageLSN that may differ from the in-memory pageLSN

Flushed Log Records

ARIES: Commit Protocol

1. Write commit record to log

2. Flush log up to the commit record

3. Write end record

ARIES: Commit Protocol

MEMORY

LSN	prevLSN	XID	Log Payload
30	-	T2	START
40	20	T1	UPDATE
50	30	T2	UPDATE

flushedLSN: **20**

DISK

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	10	T1	UPDATE

ARIES: Commit Protocol

1. **Write Commit log record**

MEMORY

LSN	prevLSN	XID	Log Payload
30	-	T2	START
40	20	T1	UPDATE
50	30	T2	UPDATE
60	40	T1	COMMIT

flushedLSN: **20**

DISK

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	10	T1	UPDATE

ARIES: Commit Protocol

2. **Flush log up to Commit log record**

MEMORY

LSN	prevLSN	XID	Log Payload
30	-	T2	START
40	20	T1	UPDATE
50	30	T2	UPDATE
60	40	T1	COMMIT

flushedLSN: **60**

DISK

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	10	T1	UPDATE
30	-	T2	START
40	20	T1	UPDATE
50	30	T2	UPDATE
60	40	T1	COMMIT

ARIES: Commit Protocol

2. **Flush log up to Commit log record**

MEMORY

LSN	prevLSN	XID	Log Payload

flushedLSN: **60**

DISK

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	10	T1	UPDATE
30	-	T2	START
40	20	T1	UPDATE
50	30	T2	UPDATE
60	40	T1	COMMIT

ARIES: Commit Protocol

3. Write End log record

MEMORY

LSN	prevLSN	XID	Log Payload
70	60	T1	END

flushedLSN: **60**

DISK

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	10	T1	UPDATE
30	-	T2	START
40	20	T1	UPDATE
50	30	T2	UPDATE
60	40	T1	COMMIT

ARIES: CLR (Compensation Log Record)

- Perform UNDO operations to abort transactions
- Before overwriting actual data, write a **CLR**
- Almost the same as an Update log record
  - Contains **undoNextLSN** - next LSN to undo
    - Can be determined from prevLSN of current log record being undone

ARIES: Abort Protocol

1. Write abort record to log

2. Undo all changes made by the transaction. Start from lastLSN, and go backwards via prevLSN
   a. Undo all log records that can be undone by adding CLRs to log

3. Write end record

ARIES: Abort Protocol

Due to some deadlock, we decide to abort T2.

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	-	T2	START
30	20	T2	UPDATE
40	10	T1	UPDATE
50	30	T2	UPDATE

ARIES: Abort Protocol

2. **Undo all changes made by the transaction. Start from lastLSN, and go backwards via prevLSN**
   a. **Undo all log records that can be undone by adding CLRs to log**

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	-	T2	START
30	20	T2	UPDATE
40	10	T1	UPDATE
50	30	T2	UPDATE
60	50	T2	ABORT
70	60	T2	CLR

ARIES: Abort Protocol

2. **Undo all changes made by the transaction. Start from lastLSN, and go backwards via prevLSN**
   a. **Undo all log records that can be undone by adding CLRs to log**

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	-	T2	START
30	20	T2	UPDATE
40	10	T1	UPDATE
50	30	T2	UPDATE
60	50	T2	ABORT
70	60	T2	CLR
80	70	T2	CLR

ARIES: Abort Protocol

3. Write End log record.

LSN	prevLSN	XID	Log Payload
10	-	T1	START
20	-	T2	START
30	20	T2	UPDATE
40	10	T1	UPDATE
50	30	T2	UPDATE
60	50	T2	ABORT
70	60	T2	CLR
80	70	T2	CLR
90	80	T2	END

ARIES: Abort Protocol

1. **Write Abort log record**

LSN	**prevLSN**	**XID**	**Log Payload**
10	-	T1	START
20	-	T2	START
30	20	T2	UPDATE
40	10	T1	UPDATE
50	30	T2	UPDATE
60	50	T2	ABORT

ARIES: Recovery

- Three phases of recovering (from a crash)
  1. **Analysis**: read through log to recreate Xact table/DPT at time of crash
  2. **Redo**: repeat *all* actions that were not written to disk
  3. **Undo**: abort and undo transactions from before crash that were not successfully committed

ARIES: Checkpoints

- Don’t want to start from beginning of log each time we crash!
- Write a **begin checkpoint** record
  - This is the latest record we can start analysis from - later records’ changes may or may not be reflected in checkpoint
- Write an **end checkpoint** record (sometime later)
  - This record contains DPT/xact table after begin checkpoint record (but we don’t know exactly when)

ARIES: Analysis Phase

- Goal: Rebuild the Xact Table and DPT at time of crash
- Start from checkpoint (LSN after BEGIN_CHECKPOINT)
- If record != END
  - Add xact to **Xact Table** if not there
  - Set **lastLSN** of xact to record.LSN
- If record = COMMIT or record = ABORT
  - Change status in **Xact Table** accordingly
- If record = UPDATE and record.page NOT IN **DPT**
  - Add to **DPT** & Set **recLSN** to record.LSN in **DPT**
- If record = END
  - Remove xact from Xact Table

ARIES: Analysis Phase

- At the end of the Analysis Phase
  - If xact is Committing
    - Write END record and remove from Xact Table
  - If xact is Running
    - Write ABORT record and change status to Aborting

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

We read all log records after the last begin checkpoint record, but in this case we don’t have one, so we read all log records
(updating the transaction table and dirty page table as we go)

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T1	10	Running

PageID	recLSN
P1	10

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T1	10	Running
T2	20	Running

PageID	recLSN
P1	10
P3	20

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T1	30	Committting
T2	20	Running

PageID	recLSN
P1	10
P3	20

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T1	30	Committing
T2	20	Running
T3	40	Running

PageID	recLSN
P1	10
P3	20
P4	40

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T1	30	Committing
T2	50	Running
T3	40	Running

PageID	recLSN
P1	10
P3	20
P4	40

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
~~T1~~	~~30~~	~~Committing~~
T2	50	Running
T3	40	Running

PageID	recLSN
P1	10
P3	20
P4	40

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T2	50	Running
T3	70	Running

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	70	Running

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

Worksheet #1

(a) During analysis, what log records are read? What are the contents of the transaction table and the dirty page table at the end of the analysis stage?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

At the end of analysis, all Xacts still "Running" should be moved to "Aborting". ➜ write an Abort record

ARIES: Redo

- Redo *everything* from the earliest recLSN in the DPT to get back unflushed changes from before crash, unless:
  - Page not in DPT
    - Page on disk must be up to date, since we have no changes!
  - recLSN of page > LSN
    - No need to undo here: recLSN of page is *first* record that dirtied page, so this change must have been flushed
  - pageLSN >= LSN
    - Page LSN is the authoritative source for determining which changes have been applied already

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

Start at the smallest recLSN from the DPT, 10.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

P1 in DPT, (recLSN = 10) <= (LSN = 10). Redo this record.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

P3 in DPT, (recLSN = 20) <= (LSN = 20). Redo this record.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

No updates to redo here.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

P4 in DPT, (recLSN = 40) <= (LSN = 40). Redo this record.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

P1 in DPT, (recLSN = 10) <= (LSN = 50). Redo this record.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

No updates to redo here.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

P2 in DPT, (recLSN = 70) <= (LSN = 70). Redo this record.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

No update to redo here.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

No update to redo here.

Worksheet #1

(b) During Redo, what log records are read? What data pages are read? What operations are redone (assuming no updates made it out to disk before the crash)?

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

All pages in DPT are read in from disk. Operations with LSNs 10, 20, 40, 50, 70 are redone.

ARIES: Undo

- Abort all transactions in xact table that aren't already committing (don't actually write the abort record)
  - And write the relevant CLRs

- Performance optimization:
  - Make a set containing the last undoNextLSN of each transaction (or lastLSN if no CLRs written for transaction yet)
  - Undo the largest LSN, repeat

Worksheet #1

(c) During Undo, what log records are read? What operations are undone? Show any new log records that are written for CLR's. Start at LSN 100. Be sure to show the undoNextLSN.

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

Start from largest lastLSN in Xact Table. LSN=90

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

No updates made from aborting a Transaction.

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	90	Aborting

No updates made from aborting a Transaction.

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	100	Aborting

T3's lastLSN is updated

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	80	Aborting
T3	100	Aborting

T1 is not in our Transaction Table.

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	110	Aborting
T3	100	Aborting

T2's lastLSN is updated

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40
110	CLR: undo T2 LSN 50	80	20

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	110	Aborting

T3 is removed from the Xact Table

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40
110	CLR: undo T2 LSN 50	80	20
120	CLR: undo T3 LSN 40	100	null
130	T3 end	120	--

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status
T2	110	Aborting

T1 is not in our Transaction Table.

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40
110	CLR: undo T2 LSN 50	80	20
120	CLR: undo T3 LSN 40	100	null
130	T3 end	120	--

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status

No more updates involving T2, end., remove T2

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40
110	CLR: undo T2 LSN 50	80	20
120	CLR: undo T3 LSN 40	100	null
130	T3 end	120	--
140	CLR: undo T2 LSN 20	110	null
150	T2 end	140	--

LSN	Log Record
10	Update: T1 writes P1
20	Update: T2 writes P3
30	T1 commit
40	Update: T3 writes P4
50	Update: T2 writes P1
60	T1 end
70	Update: T3 writes P2
80	T2 abort
90	T3 abort

Transaction	lastLSN	Status

T1 is not in our Transaction Table.

PageID	recLSN
P1	10
P3	20
P4	40
P2	70

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	90	40
110	CLR: undo T2 LSN 50	80	20
120	CLR: undo T3 LSN 40	100	null
130	T3 end	120	--
140	CLR: undo T2 LSN 20	110	null
150	T2 end	140	--

Worksheet #1

(c) During Undo, what log records are read? What operations are undone? Show any new log records that are written for CLR's. Start at LSN 100. Be sure to show the undoNextLSN.

LSN	Record	prevLSN	undoNextLSN
100	CLR: undo T3 LSN 70	70	40
110	CLR: undo T2 LSN 50	80	20
120	CLR: undo T3 LSN 40	100	null
130	T3 end	120	--
140	CLR: undo T2 LSN 20	110	null
150	T2 end	140	--

Worksheet #2

(a) The log record at LSN 60 says that transaction 2 updated page 5. Was this update to page 5 successfully written to disk?

The log record at LSN 70 says that transaction 1 updated page 2. Was this update to page 2 successfully written to disk?

Worksheet #2

(a) The log record at LSN 60 says that transaction 2 updated page 5. Was this update to page 5 successfully written to disk?

Maybe. It wasn't flushed at the time of the checkpoint, but could have been flushed later on.

The log record at LSN 70 says that transaction 1 updated page 2. Was this update to page 2 successfully written to disk?

Yes - it's not in the DPT.

Worksheet #2

(b) At the end of the analysis phase, what transactions will be in the transaction table, and what pages will be in the dirty page table?

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	70	Running
T2	60	Running
T3	30	Running
T4	50	Running

PageID	recLSN
P5	50
P1	40

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T2	60	Running
T3	30	Running
T4	50	Running

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T2	60	Running
T3	30	Running
T4	50	Running

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T2	110	Running
T3	30	Running
T4	50	Running

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T2	110	Committting
T3	30	Running
T4	50	Running

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T2	110	Committing
T3	30	Running
T4	130	Running

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
~~T2~~	~~110~~	~~Committing~~
T3	30	Running
T4	130	Running

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T3	30	Running
T4	150	Aborting

PageID	recLSN
P5	50
P1	40
P3	90

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T3	30	Running
T4	150	Aborting
T5	160	Running

PageID	recLSN
P5	50
P1	40
P3	90
P2	160

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150

Transaction	lastLSN	Status
T1	90	Running
T3	30	Running
T4	180	Aborting
T5	160	Running

PageID	recLSN
P5	50
P1	40
P3	90
P2	160

LSN	Log Record	prevLSN
30	Update: T3 writes P5	null
40	Update: T4 writes P1	null
50	Update: T4 writes p5	40
60	Update: T2 writes P5	null
70	Update: T1 writes P2	null
80	begin_checkpoint	-
90	Update: T1 writes P3	70
100	end_checkpoint	-
110	Update: T2 writes P3	60
120	T2 commit	110
130	Update: T4 writes P1	50
140	T2 end	120
150	T4 abort	130
160	Update: T5 writes P2	null
180	CLR: undo T4 LSN 130	150
190	T1 abort	90

LSN	Log Record	prevLSN
200	T3 abort	30
210	T5 abort	160

All running xacts need to be changed to aborting and emit an Abort record and update lastLSN!

Transaction	lastLSN	Status
T1	190	Aborting
T3	200	Aborting
T4	180	Aborting
T5	210	Aborting

PageID	recLSN
P5	50
P1	40
P3	90
P2	160

Worksheet #2

(b) At the end of the analysis phase, what transactions will be in the transaction table, and what pages will be in the dirty page table?

Transaction Table			**Dirty Page Table**	
Transaction	**lastLSN**	**Status**	**PageID**	**recLSN**
T1	90	Running	P1	40
T3	30	Running	P2	160
T4	180	Aborting	P3	90
T5	160	Running	P5	50

Worksheet #2

(c) At which LSN in the log should redo begin? Which log records will be redone (list their LSNs)? All other log records will be skipped.

Worksheet #2

(c) At which LSN in the log should redo begin? Which log records will be redone (list their LSNs)? All other log records will be skipped.

Starts at LSN 40 (earliest recLSN in table).

Records redone: 40, 50, 60, 90, 110, 130, 160, 180

70 skipped because P2 recLSN = 160 > 70
Non-update records skipped (no after-image/redo state)

Note: CLR *not* skipped (nor is the record it undoes skipped)