WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 764: Topics in Database Management Systems
Lecture 10: Aries Recovery

Xiangyao Yu
10/7/2020

Announcement

Submit a **1-page** course project proposal by **Oct. 21**

- Project name
- Author list
- Abstract (1-2 paragraphs about your idea)
- Introduction (Why is the problem interesting; what's your contribution)
- Methodology (how do you plan to approach the problem)
- Task-list and timeline (List of tasks and when you plan to achieve them)

Submission website: https://wisc-cs764-f20.hotcrp.com

VLDB 2020 format: https://vldb2020.org/formatting-guidelines.html

A list of project ideas are updated to the course website (http://pages.cs.wisc.edu/~yxy/cs764-f20/CS764-Fall2020-project-ideas.pdf)

Post your ideas on Piazza to look for teammates

Today's Paper: Aries Recovery

ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging

C. MOHAN
IBM Almaden Research Center
and
DON HADERLE
IBM Santa Teresa Laboratory
and
BRUCE LINDSAY, HAMID PIRAHESH and PETER SCHWARZ
IBM Almaden Research Center

In this paper we present a simple and efficient method, called ARIES (*Algorithm for Recovery and Isolation Exploiting Semantics*), which supports partial rollbacks of transactions, fine-granularity (e.g., record) locking and recovery using write-ahead logging (WAL). We introduce the paradigm of *repeating history* to redo all missing updates *before* performing the rollbacks of the loser transactions during restart after a system failure. ARIES uses a log sequence number in each page to correlate the state of a page with respect to logged updates of that page. All updates of a transaction are logged, including those performed during rollbacks. By appropriate chaining of the log records written during rollbacks to those written during forward progress, a bounded amount of logging is ensured during rollbacks even in the face of repeated failures during restart or of nested rollbacks We deal with a variety of features that are very important in building and operating an *industrial-strength* transaction processing system ARIES supports fuzzy checkpoints, selective and deferred restart, fuzzy image copies, media recovery, and high concurrency lock modes (e.g., increment/decrement) which exploit the semantics of the operations and require the ability to perform operation logging. ARIES is flexible with respect to the kinds of buffer management policies that can be implemented. It supports objects of varying length efficiently. By enabling parallelism during restart, page-oriented redo, and logical undo, it enhances concurrency and performance. We show why some of the System R paradigms for logging and recovery, which were based on the shadow page technique, need to be changed in the context of WAL. We compare ARIES to the WAL-based recovery methods of

ACM Trans. Database Syst. 1992.

Agenda

Durability

Force vs. No Force and Steal vs. No Steal

ARIES recovery

Durability

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back

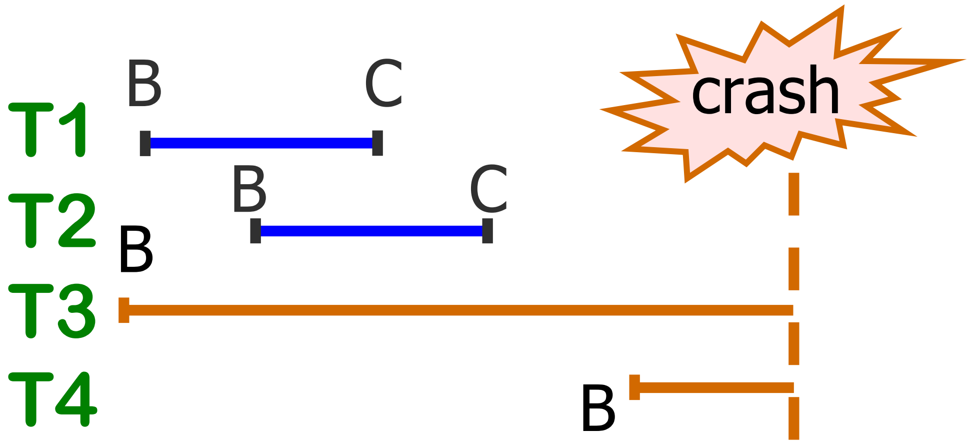
Durability

Durability: The database must recover to a valid state no matter when a crash occurs

- Committed transactions should persist
- Uncommitted transactions should roll back

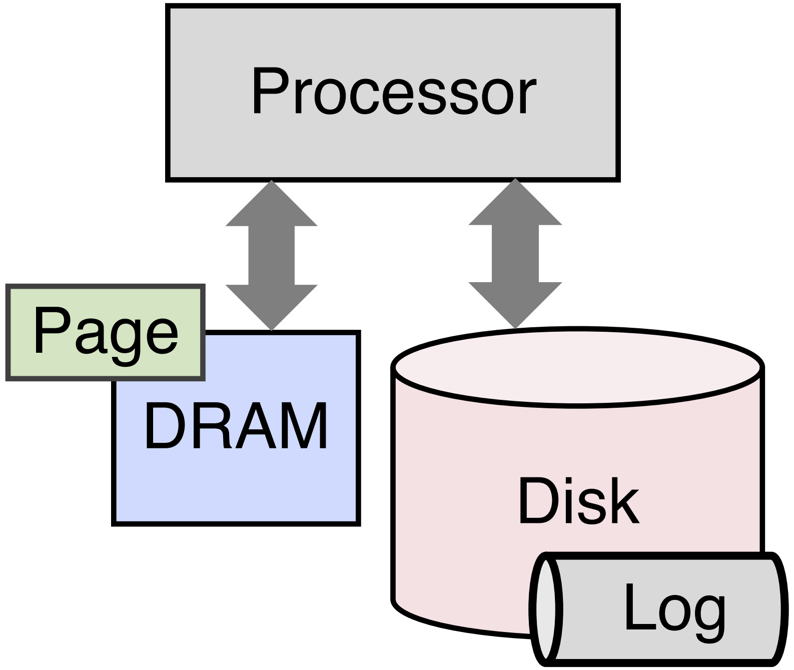
Desired Behavior after system restarts

- T1, T2 should be durable
- T3, T4 should be aborted



Write-Ahead Logging (WAL)

On a crash, data in disk persists, data in memory disappears

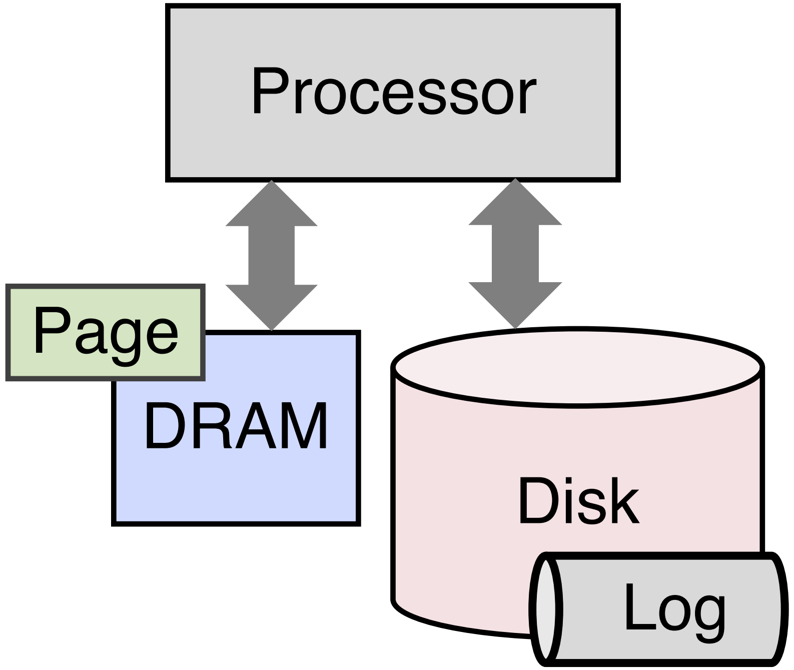


Write-Ahead Logging (WAL)

On a crash, data in disk persists, data in memory disappears

Write-ahead logging

- Flush the log record for an update before the corresponding data page gets to disk
- Write all log records for a transaction before commit

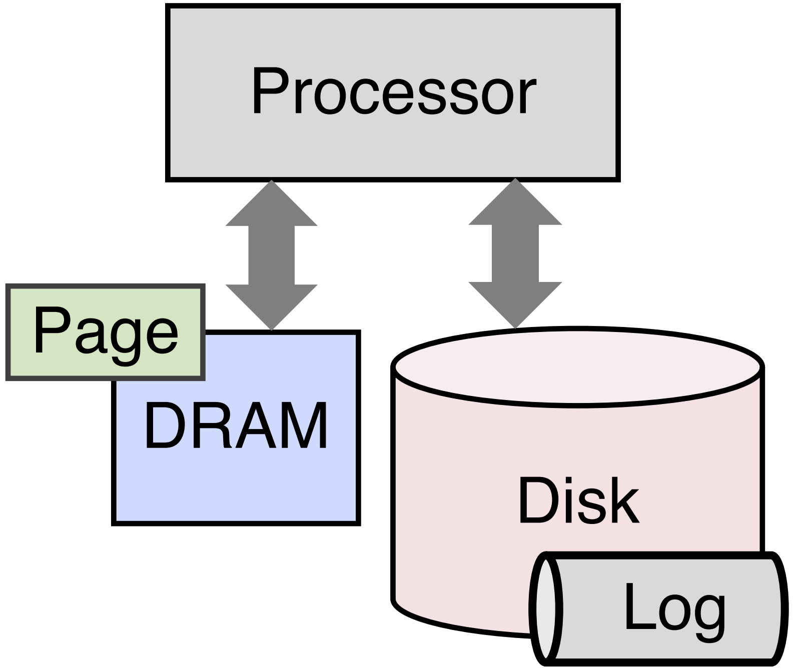


Write-Ahead Logging (WAL)

On a crash, data in disk persists, data in memory disappears

Write-ahead logging

- Flush the log record for an update before the corresponding data page gets to disk
- Write all log records for a transaction before commit



"... a DBMS is really two DBMSs, one managing the database as we know it and a second one managing the log."

Michael Stonebraker [1]

[1] M. Stonebraker. The land sharks are on the squawk box. Commun. ACM, 2016

Buffer Management Policy

No Steal: Dirty pages stay in DRAM until the transaction commits

Steal: Dirty pages can be flushed to disk before the transaction commits

- Advantage: other transactions can use the buffer slot in DRAM
- Challenge: system crashes after flushing dirty pages but before the transaction commits
  => Dirty data on disk
- Solution: **UNDO logging** before each update

Buffer Management Policy

Force: All dirty pages must be flushed when the transaction commits

No Force: Dirty pages may stay in memory after the transaction commits

- Advantage: reduce # random IO
- Challenge: system crashes after the transaction commits but before the dirty pages are flushed
  => missing updates from committed transactions
- Solution: **REDO logging** before each update

Flushing logs can be much cheaper than flushing data pages:

- Log record can be much smaller than a data page
- Logging incurs sequential IO; data page updates incur random IO

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Disk-based DB

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only
	Disk-based DB	**Main memory DB**

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Non-volatile memory DB

Disk-based DB

Main memory DB

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Non-volatile memory DB

Disk-based DB

Main memory DB

[1] Philip Bernstein, Vassos Hadzilacos, Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, 1987

Buffer Management Policy

	Steal	No Steal
Force	**UNDO** only	No REDO nor UNDO
No Force	**REDO and UNDO** logging (ARIES)	**REDO** only

Focus of this lecture

ARIES Logging

Data Structures – Log Records

Update log record: contains REDO and UNDO information

Compensate log record (CLR): contains REDO information that rolls back a previous update log record

Data Structures – Log Records

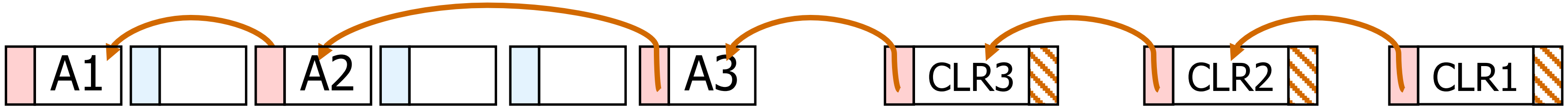
Log record fields

- **LSN**: address of the first byte of the log record (not actually stored)
- **Type**: ‘update’, ‘compensate log record (CLR)’, etc.
- **TransID**: transaction ID
- **PageID**: identifier of the page to which the updates of this record were applied
- **Data**: the actual redo and undo record

Data Structures – Log Records

Log record fields

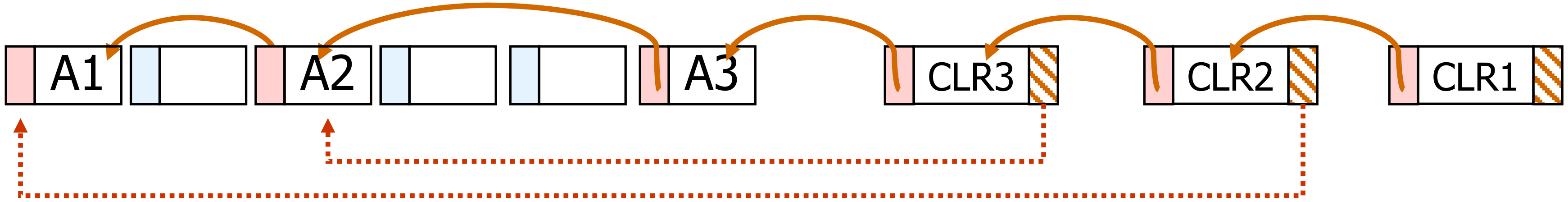
- **LSN**: address of the first byte of the log record (not actually stored)
- **Type**: ‘update’, ‘compensate log record (CLR)’, etc.
- **TransID**: transaction ID
- **PageID**: identifier of the page to which the updates of this record were applied
- **Data**: the actual redo and undo record
- **prevLSN**: preceding log record written by the same transaction



Data Structures – Log Records

Log record fields

- **LSN**: address of the first byte of the log record (not actually stored)
- **Type**: ‘update’, ‘compensate log record (CLR)’, etc.
- **TransID**: transaction ID
- **PageID**: identifier of the page to which the updates of this record were applied
- **Data**: the actual redo and undo record
- **prevLSN**: preceding log record written by the same transaction
- **UndoNxtLSN**: (For CLR) LSN of the next log record of this transaction that is to be processed during rollback



Data Structures – Data Page

Page_LSN: LSN of the log record that describes the latest update to the page.

Data Structures – Transaction Table

Transaction table: One entry per **active** transaction

Each entry contains

- **TransID**: Transaction ID
- **Status**: prepared (P) or **unprepared** (U)
- **LastLSN**: LSN of the last log record written by the transaction
- **UndoNxtLSN**: LSN of the next record to be processed during rollback

Data Structures – Dirty Page Table (DPT)

Dirty page table: One entry per dirty page in buffer pool

Each entry contains

- **PageID**: ID of the page
- **RecLSN**: LSN of the first log record since when the page is dirty
  (the page on disk is up to date before RecLSN)

Data Structures – Checkpoint

A **checkpoint** is a snapshot of the database

- Reduces recovery time

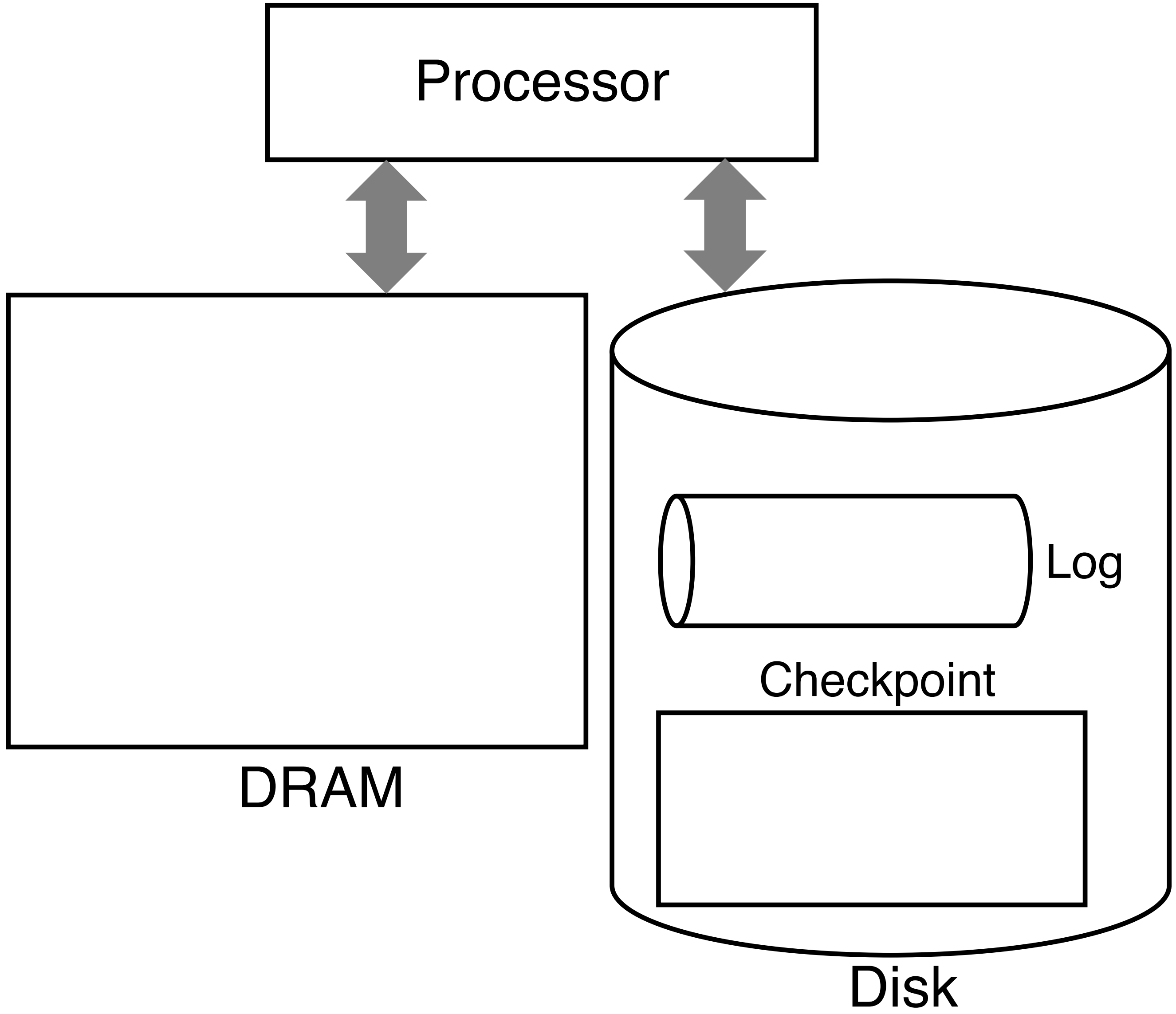
In ARIES, A checkpoint contains

- Transaction Table
- Dirty page table

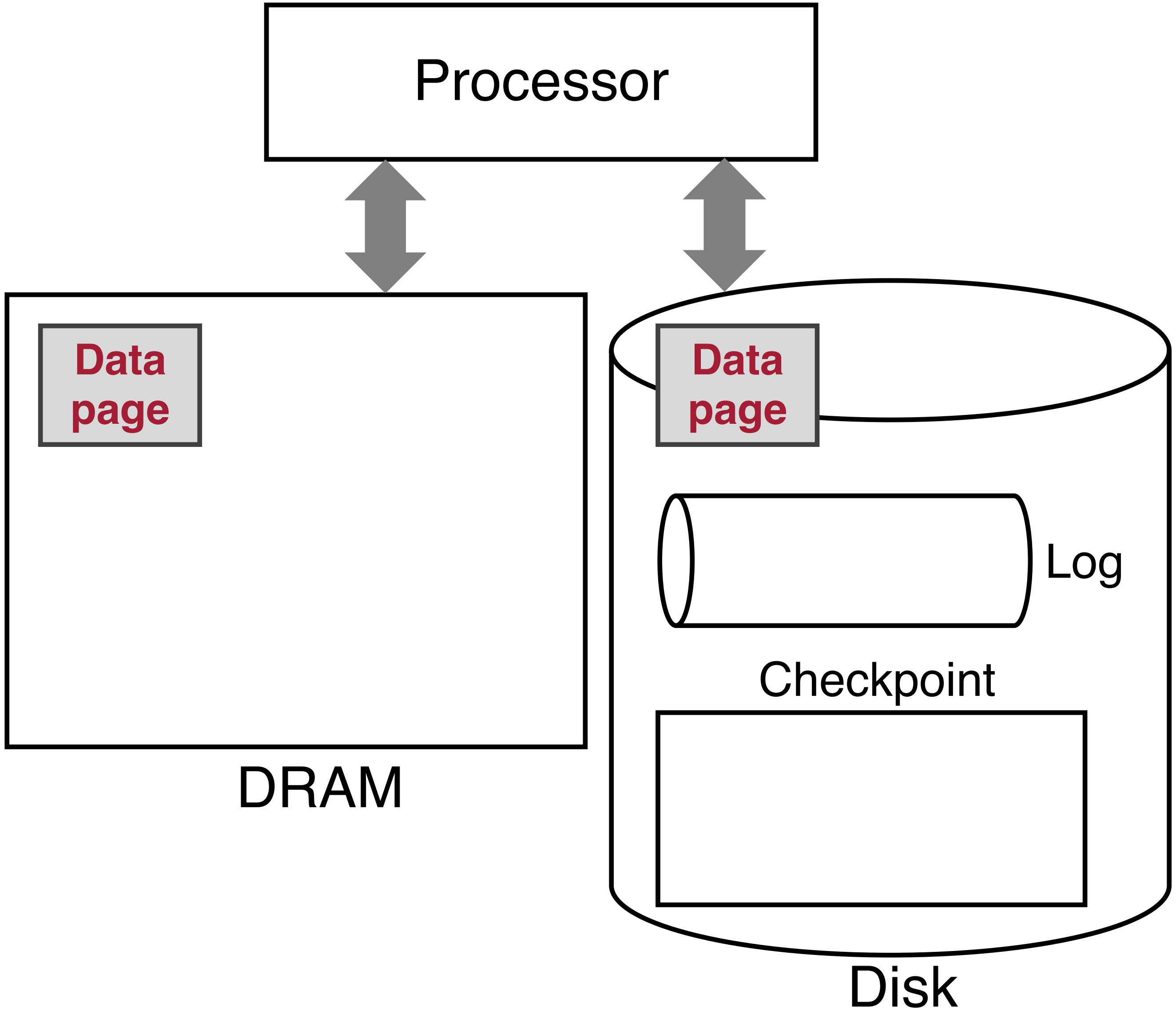
Fuzzy checkpoint

- Checkpoints can be taken asynchronously

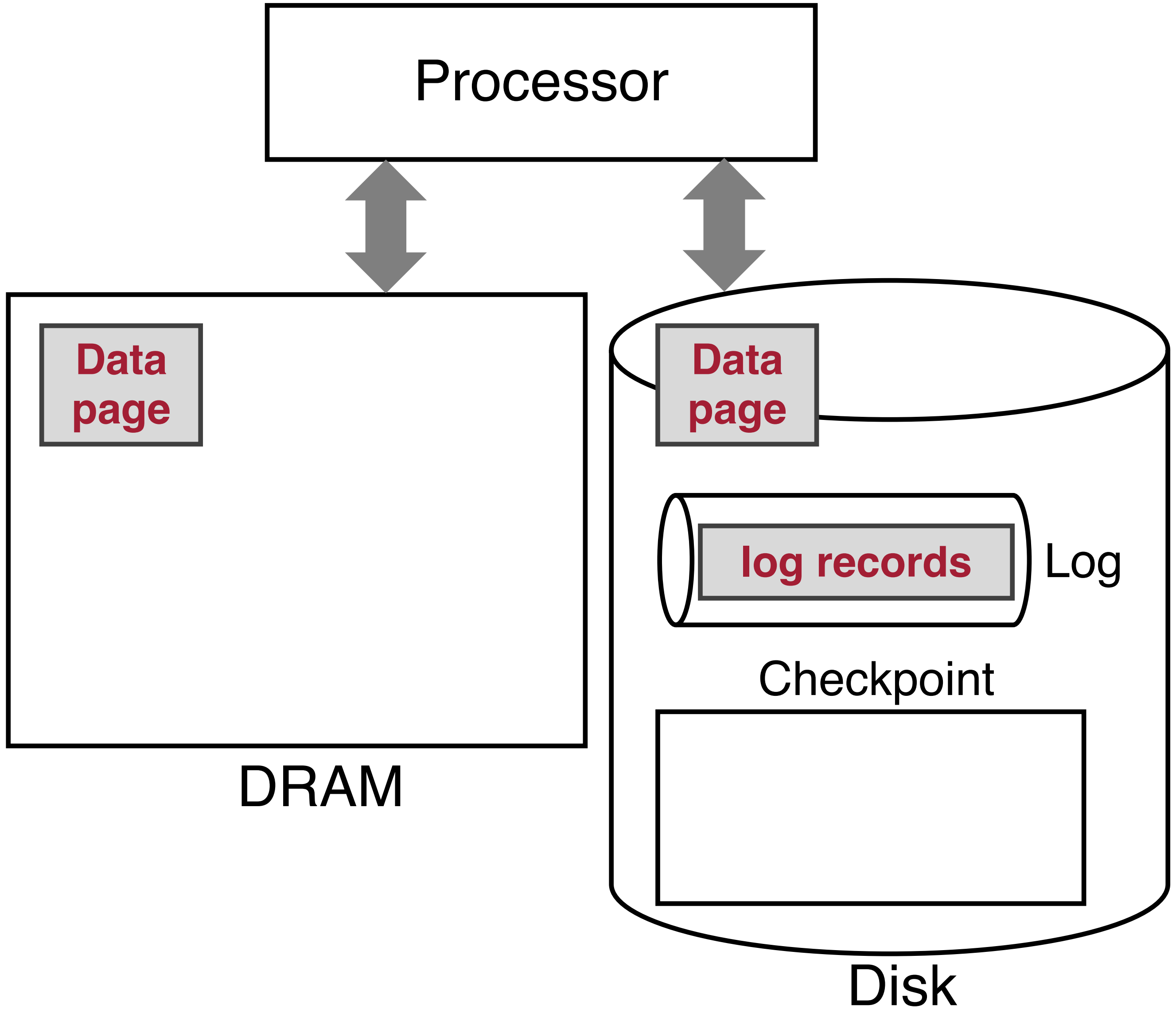
Data Structures – Big Picture



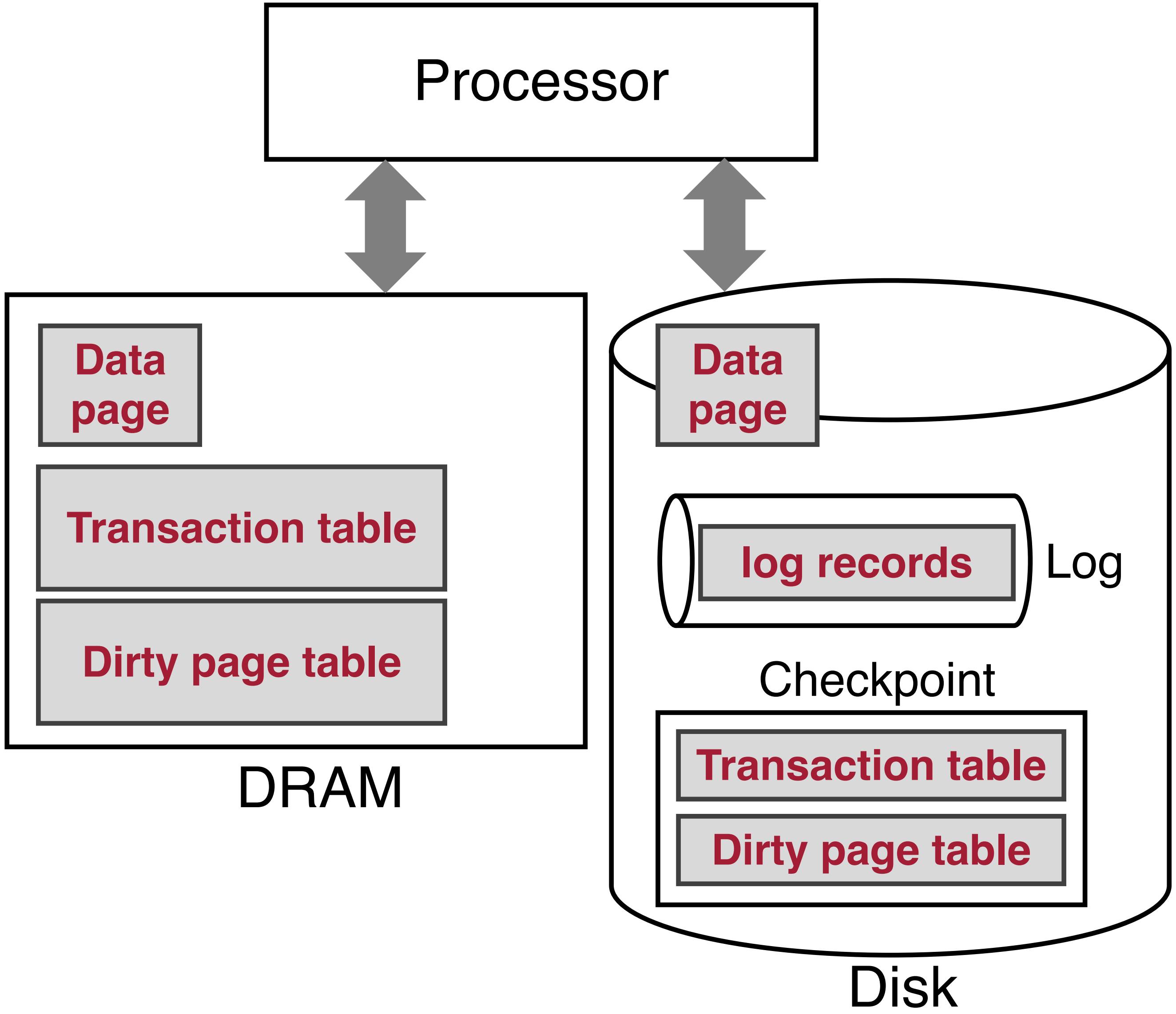
Data Structures – Big Picture



Data Structures – Big Picture



Data Structures – Big Picture



Normal Processing

Write-ahead logging

- Flush log before flushing the corresponding data page
- Flush all logs before committing the transaction

Maintain the transaction table and dirty page table

Rollback

- Must UNDO previous update
- Write compensate log record (CLR) for the UNDO operation

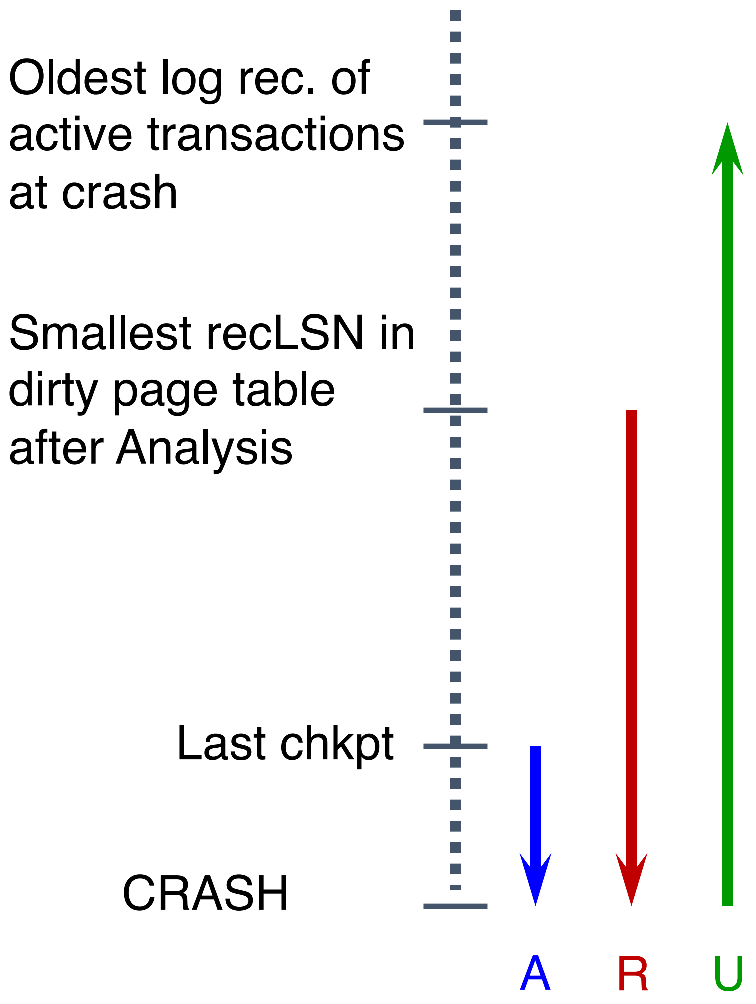
Checkpoint

- Periodically flush transaction table and dirty page table to disk

Crash Recovery – Big Picture

Goal: Bring the database to the state before the crash (REDO phase) and rollback uncommitted transactions (UNDO phase)

Crash Recovery – Big Picture



Goal: Bring the database to the state before the crash (REDO phase) and rollback uncommitted transactions (UNDO phase)

Start from the last complete checkpoint

- **Analysis phase**: rebuild transaction table (for undo phase) and dirty page table (for redo phase)
- **REDO phase**: redo transactions whose effects may not be persistent before the crash
- **UNDO phase**: undo transactions that did not commit before the crash

Crash Recovery – Analysis Phase

Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint

Crash Recovery – Analysis Phase

Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint

(update transaction table) For each log record:

- If ‘update’ or ‘CLR’: insert to transaction table if not exists
- If ‘end’: delete from transaction table

Crash Recovery – Analysis Phase

Goal: Rebuild transaction table (for undo phase) and dirty page table (for redo phase) based on the ones in the last checkpoint

(update transaction table) For each log record:

- If 'update' or 'CLR': insert to transaction table if not exists
- If 'end': delete from transaction table

(update dirty page table) For each log record:

- If 'update' or 'CLR': insert to dirty page table if not exists (PageID, RecLSN)

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN

Dirty page table

PageID	RecLSN

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	**update: T1 writes P5**
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T1	**10**

Dirty page table

PageID	RecLSN
P5	**10**

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	**update T2 writes P3**
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T1	10
T2	**20**

Dirty page table

PageID	RecLSN
P5	10
P3	**20**

Analysis Phase – Example

LSN LOG

00 begin_checkpoint
05 end_checkpoint
10 update: T1 writes P5
20 update T2 writes P3
30 **T1 abort**
40 **CLR: Undo T1 LSN 10**
45 **T1 End**
50 update: T3 writes P1
60 update: T2 writes P5
CRASH, RESTART

Transaction Table

TransID	LastLSN
~~T1~~	~~10~~
T2	20

Dirty page table

PageID	RecLSN
P5	10
P3	20

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	**update: T3 writes P1**
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	**50**
T2	20

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	**50**

Analysis Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	**update: T2 writes P5**
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	**60**

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Crash Recovery – REDO Phase

Repeat history to reconstruct state at crash

- Reapply all updates (even of aborted transactions), redo CLRs

Crash Recovery – REDO Phase

Repeat history to reconstruct state at crash

- Reapply all updates (even of aborted transactions), redo CLRs

Where to start?

- From log record containing **smallest RecLSN** in the dirty page table
- Before this LSN, all redo records have been reflected in data pages on disk

Crash Recovery – REDO Phase

Repeat history to reconstruct state at crash

- Reapply all updates (even of aborted transactions), redo CLRs

Where to start?

- From log record containing **smallest RecLSN** in the dirty page table
- Before this LSN, all redo records have been reflected in data pages on disk

Observation: can **skip a redo record** for the following cases where the corresponding page has already been flushed before the crash

- The page is not in dirty page table (DPT)
- The page is in DPT but redo_record.LSN < DPT[page].recLSN
- After fetching the data page, redo_record.LSN ≤ page.page_LSN

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	**update: T1 writes P5**
20	**update T2 writes P3**
30	T1 abort
40	**CLR: Undo T1 LSN 10**
45	T1 End
50	**update: T3 writes P1**
60	**update: T2 writes P5**
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Data pages

PageID	Page_LSN
P5	40
P3	0
P1	0

REDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	**update: T1 writes P5**
20	update T2 writes P3
30	T1 abort
40	**CLR: Undo T1 LSN 10**
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN
T3	50
T2	60

Dirty page table

PageID	RecLSN
P5	10
P3	20
P1	50

Data pages

PageID	Page_LSN
P5	**40**
P3	0
P1	0

Crash Recovery – UNDO Phase

Rollback uncommitted transactions

- Transactions in transaction table did not commit before the crash
- Undo each update log record of these transactions

Crash Recovery – UNDO Phase

Rollback uncommitted transactions

- Transactions in transaction table did not commit before the crash
- Undo each update log record of these transactions

Repeat until transaction table is empty:

- Choose largest LastLSN among transactions in the transaction table
- If the log record is an ‘update’: Undo the update, write a CLR, add record.prevLSN to transaction table
- If the log record is an ‘CLR’: add CLR.UndoNxtLSN to transaction table
- If prevLSN and UpdoNxtLSN are NULL, remove the transaction from transaction table

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T3	50	50
T2	60	60

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	**update: T2 writes P5**
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T3	50	50
T2	~~60~~ 70	~~60~~ 20

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	**update: T3 writes P1**
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T3	~~50~~ 80	~~50~~ null
T2	70	20

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
~~T3~~	~~80~~	~~null~~
T2	70	20

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	**update T2 writes P3**
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
T2	~~70~~ 90	~~20~~ null

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	
90	CLR: Undo T2, LSN 20,	(null)

UNDO Phase – Example

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
X	CRASH, RESTART

Transaction Table

TransID	LastLSN	UndoNxtLSN
~~T2~~	~~90~~	~~null~~

LSN	LOG	(undoNextLSN)
70	CLR: Undo T2, LSN 60,	(20)
80	CLR: Undo T3, LSN 50,	(null)
85	T3 End	
90	CLR: Undo T2, LSN 20,	(null)
95	T2 End	

Crash During Restart – Example

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40,45	CLR: Undo T1 LSN 10, T1 End
50	update: T3 writes P1
60	update: T2 writes P5
X	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end
X	CRASH, RESTART
90	CLR: Undo T2 LSN 20, T2 end

No need to undo LSN 60 and LSN 50 again due to the CLRs created in the previous restart

Can created a checkpoint to reduce the cost of future restart

Summary

ARIES: WAL supporting STEAL/NO-FORCE

- Checkpointing: A quick way to limit the amount of log to scan on recovery.

Recovery works in 3 phases:

- Analysis: Forward from last checkpoint
- Redo: Forward from oldest RecLSN.
- Undo: Backward from end to first LSN of oldest transaction alive at crash

Upon UNDO, write CLRs

Q/A – Aries Recovery

Too long, too many variables…

What's the main contribution?

Why REDO aborted transactions in the REDO phase?

Why don't we worry about deadlocks when using latches?

Why REDO when log_record.LSN > page.LSN?

Physical and logical consistency?

Latches vs. Locks?

Logs consume large disk space

Log becomes a bottleneck in modern systems?

Technique still valid for modern systems?

Before Next Lecture

Look for teammates for the course project ☺

Submit review before next lecture

- C. Mohan, et al., Transaction Management in the R* Distributed Database Management System. ACM Trans. Database Syst. 1986.