

# CSE 444: Database Internals

Lectures 17-19  
Transactions: Recovery

CSE 444 - Winter 2019 1

## The Usual Reminders

- HW3 is due tonight
  - Only a single problem
- Lab3 is due on Monday
- Quiz grades should be returned on Gradescope tomorrow

CSE 444 - Winter 2019 2

## Readings for Lectures 17-19

Main textbook (Garcia-Molina)

- Ch. 17.2-4, 18.1-3, 18.8-9

Second textbook (Ramakrishnan)

- Ch. 16-18

Also: M. J. Franklin. Concurrency Control and Recovery. The Handbook of Computer Science and Engineering, A. Tucker, ed., CRC Press, Boca Raton, 1997.

CSE 444 - Winter 2019 3

## Transaction Management

Two parts:

- Concurrency control: ACID
- Recovery from crashes: ACID

We already discussed concurrency control  
You are implementing locking in lab3

Today, we start recovery

CSE 444 - Winter 2019 4

## System Crash

Client 1:

```

BEGIN TRANSACTION
UPDATE Account1
SET balance= balance - 500

```

Crash!

```

UPDATE Account2
SET balance = balance + 500
COMMIT

```

CSE 444 - Winter 2019 5

## Recovery

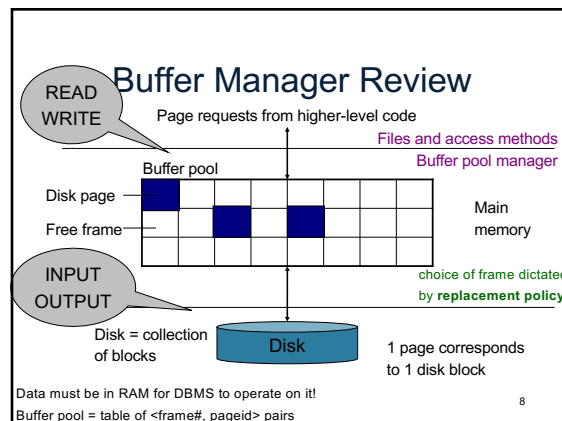
Type of Crash	Prevention
Wrong data entry	Constraints and Data cleaning
Disk crashes	Redundancy: e.g. RAID, archive
Data center failures	Remote backups or replicas
System failures: e.g. power	DATABASE RECOVERY

## System Failures

- Each transaction has *internal state*
- When system crashes, internal state is lost
  - Don't know which parts executed and which didn't
  - Need ability to *undo* and *redo*

CSE 444 - Winter 2019

7



8

## Buffer Manager Review

- Enables higher layers of the DBMS to assume that needed data is in main memory
- Caches data in memory. Problems when crash occurs:
  - If committed data was not yet written to disk
  - If uncommitted data was flushed to disk

CSE 444 - Winter 2019

9

## Transactions

- Assumption: the database is composed of elements.
- 1 element can be either:
  - 1 page = physical logging
  - 1 record = logical logging
- Aries uses physiological logging
  - (will discuss later)

CSE 444 - Winter 2019

10

## Primitive Operations of Transactions

- READ(X,t)
  - copy element X to transaction local variable t
- WRITE(X,t)
  - copy transaction local variable t to element X
- INPUT(X)
  - read element X to memory buffer
- OUTPUT(X)
  - write element X to disk

CSE 444 - Winter 2019

11

## Running Example

```
BEGIN TRANSACTION
```

```
READ(A,t);
```

```
t := t*2;
```

```
WRITE(A,t);
```

```
READ(B,t);
```

```
t := t*2;
```

```
WR
```

```
CO
```

Initially, A=B=8.

**Atomicity** requires that either  
(1) T commits and A=B=16, or  
(2) T does not commit and A=B=8.

Will look at various crash scenarios

What behavior do we want in each case?

CSE 444 - Winter 2019

12

Transaction		Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)					
t:=t*2					
WRITE(A,t)					
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

Transaction		Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2					
WRITE(A,t)					
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

Transaction		Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)					
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

Transaction		Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)					
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

Transaction		Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)					
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

Transaction		Buffer pool		Disk	
Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2					
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

Transaction Buffer pool Disk

READ(A,t); t := t\*2; WRITE(A,t);  
READ(B,t); t := t\*2; WRITE(B,t)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	8
READ(B,t)	8	16	8	8	8
t:=t*2	16	16	8	8	8
WRITE(B,t)					
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

9

Transaction Buffer pool Disk

READ(A,t); t := t\*2; WRITE(A,t);  
READ(B,t); t := t\*2; WRITE(B,t)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)					
OUTPUT(B)					
COMMIT					

0

Transaction Buffer pool Disk

READ(A,t); t := t\*2; WRITE(A,t);  
READ(B,t); t := t\*2; WRITE(B,t)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)					
COMMIT					

1

Transaction Buffer pool Disk

READ(A,t); t := t\*2; WRITE(A,t);  
READ(B,t); t := t\*2; WRITE(B,t)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

2

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash!

3

Is this bad ?

Yes it's bad: A=16, B=8....

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash!

4

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !  
5

Is this bad ? Yes it's bad: A=B=16, but not committed

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !  
6

Is this bad ?

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !  
7

Is this bad ? No: that's OK

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16
COMMIT					

Crash !  
8

OUTPUT can also happen **after** COMMIT (details coming)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

9

OUTPUT can also happen **after** COMMIT (details coming)

Action	t	Mem A	Mem B	Disk A	Disk B
INPUT(A)		8		8	8
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	8
INPUT(B)	16	16	8	8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Crash !  
10

## Atomic Transactions

- **FORCE or NO-FORCE**
  - Should all updates of a transaction be forced to disk before the transaction commits?
- **STEAL or NO-STEAL**
  - Can an update made by an uncommitted transaction overwrite the most recent committed value of a data item on disk?

CSE 444 - Winter 2019 31

## Force/No-steal

- **FORCE**: Pages of committed transactions must be forced to disk before commit
- **NO-STEAL**: Pages of uncommitted transactions cannot be written to disk

Easy to implement (how?) and ensures atomicity

CSE 444 - Winter 2019 32

## No-Force/Steal

- **NO-FORCE**: Pages of committed transactions need not be written to disk
- **STEAL**: Pages of uncommitted transactions may be written to disk

In either case, need a Write Ahead Log (WAL) to provide atomicity in face of failures

CSE 444 - Winter 2019 33

## Write-Ahead Log (WAL)

**The Log**: append-only file containing log records

- Records every single action of every TXN
- Forces log entries to disk as needed
- After a system crash, use log to recover

Three types: UNDO, REDO, UNDO-REDO  
Aries: is an UNDO-REDO log

CSE 444 - Winter 2019 34

## Policies and Logs

	NO-STEAL	STEAL
FORCE	Lab 3	Undo Log
NO-FORCE	Redo Log	Undo-Redo Log

CSE 444 - Winter 2019 35

## UNDO Log

FORCE and STEAL

CSE 444 - Winter 2019 36

## Undo Logging

Log records

- <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 old value was v
  - *Idempotent, physical log records*

CSE 444 - Winter 2019

37

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

CSE 444 - Winter 2019

38

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

CSE 444 - Winter 2019

39

WHAT DO WE DO ?

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

WHAT DO WE DO ?

We UNDO by setting B=8 and A=8

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

CSE 444 - Winter 2019

40

What do we do now ?

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

CSE 444

What do we do now ?

Nothing: log contains COMMIT

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

CSE 444 - Winter 2019 43

### After Crash

- This is all we see (for example):

Disk A	Disk B	<START T>
8	16	<T,A,8>
		<T,B,8>

CSE 444 - Winter 2019 44

### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T>
8	16	<T,A,8>
		<T,B,8>

CSE 444 - Winter 2019 45

### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T>
8	16	<T,A,8>
		<T,B,8>

- What direction?

CSE 444 - Winter 2019 46

### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T>
8	16	<T,A,8>
		<T,B,8>

↑

- What direction?
- In UNDO log, we start at the most recent and go backwards in time

CSE 444 - Winter 2019 47

### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T>
8	16	<T,A,8>
		<T,B,8>

↑

- What direction?
- In UNDO log, we start at the most recent and go backwards in time

CSE 444 - Winter 2019 48



### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T> <T,A,8> <T,B,8>	↑
8	16		

- What direction?
- In UNDO log, we start at the most recent and go backwards in time

49

### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T> <T,A,8> <T,B,8>	↑
8	8		

- What direction?
- In UNDO log, we start at the most recent and go backwards in time

50

### After Crash

- This is all we see (for example):
- Need to step through the log

Disk A	Disk B	<START T> <T,A,8> <T,B,8>	↑
8	8		

- What direction?
- In UNDO log, we start at the most recent and go backwards in time

51

### After Crash

- If we see NO Commit statement:
  - We UNDO both changes: A=8, B=8
  - The transaction is atomic, since none of its actions have been executed
- In we see that T has a Commit statement
  - We don't undo anything
  - The transaction is atomic, since both it's actions have been executed

CSE 444 - Winter 2019 52

### Recovery with Undo Log

After system's crash, run recovery manager

- Decide for each transaction T whether it is completed or not
  - <START T>.....<COMMIT T>..... = yes
  - <START T>.....<ABORT T>..... = yes
  - <START T>..... = no
- Undo all modifications by **incomplete** transactions

CSE 444 - Winter 2019 53

### Recovery with Undo Log

Recovery manager:

- Read log from the end; cases:
  - <COMMIT T>: mark T as completed
  - <ABORT T>: mark T as completed
  - <T,X,v>: if T is not completed
    - then write X=v to disk
    - else ignore
  - <START T>: ignore

CSE 444 - Winter 2019 54

### Recovery with Undo Log

...

...

<T6,X6,v6>

...

...

<START T5>

<START T4>

<T1,X1,v1>

<T5,X5,v5>

<T4,X4,v4>

<COMMIT T5>

<T3,X3,v3>

<T2,X2,v2>

**Question 1:** Which updates are undone ?

**Question 2:** How far back do we need to read in the log ?

**Question 3:** What happens if second crash during recovery?

55

### Recovery with Undo Log

...

...

<T6,X6,v6>

...

...

<START T5>

<START T4>

<T1,X1,v1>

<T5,X5,v5>

<T4,X4,v4>

<COMMIT T5>

<T3,X3,v3>

<T2,X2,v2>

**Question 1:** Which updates are undone ?

**Question 2:** How far back do we need to read in the log ?  
**To the beginning.**

**Question 3:** What happens if second crash during recovery?

56

### Recovery with Undo Log

...

...

<T6,X6,v6>

...

...

<START T5>

<START T4>

<T1,X1,v1>

<T5,X5,v5>

<T4,X4,v4>

<COMMIT T5>

<T3,X3,v3>

<T2,X2,v2>

**Question 1:** Which updates are undone ?

**Question 2:** How far back do we need to read in the log ?  
**To the beginning.**

**Question 3:** What happens if second crash during recovery?  
**No problem! Log records are idempotent. Can reapply.**

58

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

CSE 444 - Winter 2019

Action	t	Mem A	Mem B	Disk A	Disk B	UNDO Log
INPUT(A)						<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>
INPUT(B)	16	16	8	8	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>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	
COMMIT						<COMMIT T>

FORCE

RULES: log entry *before* OUTPUT *before* COMMIT

### Undo-Logging Rules

U1: If T modifies X, then <T,X,v> must be written to disk before OUTPUT(X)

U2: If T commits, then OUTPUT(X) must be written to disk before <COMMIT T>

FORCE

- Hence: OUTPUTs are done early, before the transaction commits

CSE 444 - Winter 2019

## Checkpointing

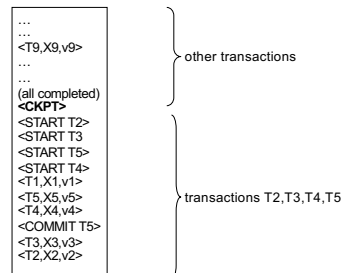
- Checkpoint the database periodically
- Stop accepting new transactions
  - Wait until all current transactions complete
  - Flush log to disk
  - Write a <CKPT> log record, flush
  - Resume transactions

CSE 444 - Winter 2019

61

## Undo Recovery with Checkpointing

During recovery,  
Can stop at first  
<CKPT>



62

## Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

Quiescent = being quiet, still, or at rest; inactive  
Non-quiescent = allowing transactions to be active

CSE 444 - Winter 2019

63

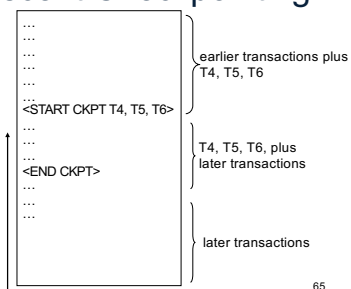
## Nonquiescent Checkpointing

- Write a <START CKPT(T1,...,Tk)> where T1,...,Tk are all active transactions. Flush log to disk
- Continue normal operation
- When all of T1,...,Tk have completed, write <END CKPT>, flush log to disk

64

## Undo Recovery with Nonquiescent Checkpointing

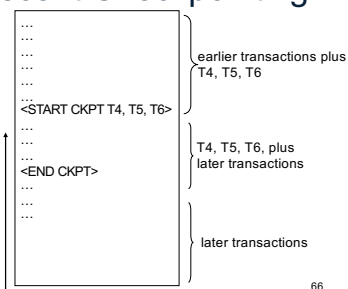
Need to read  
Back to start of  
T4, T5, T6



65

## Undo Recovery with Nonquiescent Checkpointing

Need to read  
Back to start of  
T4, T5, T6



66

Q: do we need  
<END CKPT> ?

### Undo Recovery with Nonquiescent Checkpointing

Need to read Back to start of T4, T5, T6

```

...
...
...
...
...
<START CKPT T4, T5, T6>
...
...
...
<END CKPT>
...
...
...

```

earlier transactions plus T4, T5, T6

T4, T5, T6, plus later transactions

later transactions

**Q: do we need <END CKPT> Not really, it's implicit in seeing T4,T5,T6 commits**

### Implementing ROLLBACK

- Recall: a transaction can end in COMMIT or ROLLBACK
- Idea: use the undo-log to implement ROLLBACK
- How ?
  - LSN = Log Sequence Number
  - Log entries for the same transaction are linked, using the LSN's
  - Read log in reverse, using LSN pointers

CSE 444 - Winter 2019 68

...  
 ...  
 <T9,X9,v9>  
 ...  
 (all completed)  
 <CKPT>  
 <START T2>  
 <START T3>  
 <START T5>  
 <START T4>  
 <T1,X1,v1>  
 <T5,X5,v5>  
 <T2,X1,v2>  
 <T4,X4,v4>  
 <COMMIT T5>  
 <T3,X3,v3>  
 <T2,X2,v2>

• Recall  
 or  
 • Idea  
 ROLLBACK  
 • How ?

69

### REDO Log

NO-FORCE and NO-STEAL

CSE 444 - Winter 2019 70

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

CSE 444 - Winter 2019 71

Is this bad ?


Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16

Crash!

CSE 444 - Winter 2019 72

Is this bad ? Yes, it's bad: A=16, B=8


Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



CSE 444 - Winter 2019 73

Is this bad ?


Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



CSE 444 - Winter 2019 74

Is this bad ? Yes, it's bad: lost update


Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



CSE 444 - Winter 2019 75

Is this bad ?


Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



CSE 444 - Winter 2019 76

Is this bad ? No: that's OK.

Action	t	Mem A	Mem B	Disk A	Disk B
READ(A,t)	8	8		8	8
t:=t*2	16	8		8	8
WRITE(A,t)	16	16		8	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
COMMIT					
OUTPUT(A)	16	16	16	16	8
OUTPUT(B)	16	16	16	16	16



CSE 444 - Winter 2019 77

## Redo Logging


One minor change to the undo log:

- $\langle T, X, v \rangle = T$  has updated element X, and its new value is v


CSE 444 - Winter 2019 78

Action	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>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

CSE 444 - Winter 2019 79

Action	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>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

How do we recover ? CSE 444 - Winter 2019 80

Action	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>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

How do we recover ? We REDO by setting A=16 and B=16

## Recovery with Redo Log

After system's crash, run recovery manager

- Step 1. Decide for each transaction T whether it is committed or not
  - <START T>....<COMMIT T>..... = yes
  - <START T>....<ABORT T>..... = no
  - <START T>..... = no
- Step 2. Read log from the beginning, redo all updates of committed transactions


CSE 444 - Winter 2019 82

## Recovery with Redo Log

↓

<START T1>  
 <T1,X1,v1>  
 <START T2>  
 <T2, X2, v2>  
 <START T3>  
 <T1,X3,v3>  
 <COMMIT T2>  
 <T3,X4,v4>  
 <T1,X5,v5>

Show actions during recovery



CSE 444 - Winter 2019 83

## Nonquiescent Checkpointing

- Write a <START CKPT(T1,...,Tk)> where T1,...,Tk are all active txn's
- Flush to disk all **blocks of committed transactions** (*dirty blocks*)
- Meantime, continue normal operation
- When **all blocks have been written**, write <END CKPT>

END CKPT has different meaning here than in Undo log 84

### Nonquiescent Checkpointing

Step 1: look for the last <END CKPT>

All OUTPUTs of T1 are known to be on disk

Cannot use

Step 2: redo from the earliest start of T4, T5, T6 ignoring transactions committed earlier

```

<START T1>
<COMMIT T1>
...
<START T4>
<START CKPT T4, T5, T6>
...
<END CKPT>
...
<START CKPT T9, T10>
    
```

CSE 444 - Winter 2019 85

Action	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>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

CSE 444 - Winter 2019 86

Action	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		NO-STEAL				<COMMIT T>
OUTPUT(A)	16	16	16	16	8	
OUTPUT(B)	16	16	16	16	16	

RULE: OUTPUT after COMMIT 87

### Redo-Logging Rules

R1: If T modifies X, then both <T,X,v> and <COMMIT T> must be written to disk before OUTPUT(X)

NO-STEAL

- Hence: OUTPUTs are done late

CSE 444 - Winter 2019 88

### Comparison Undo/Redo

- Undo logging:** Steal/Force
  - OUTPUT must be done early
  - If <COMMIT T> is seen, T definitely has written all its data to disk (hence, don't need to redo) – inefficient
- Redo logging:** No-Steal/No-Force
  - OUTPUT must be done late
  - If <COMMIT T> is not seen, T definitely has not written any of its data to disk (hence there is not dirty data on disk, no need to undo) – inflexible
- Would like more flexibility on when to OUTPUT: Steal/No-Force undo/redo logging (next)

CSE 444 - Winter 2019 90

### Undo/Redo Logging

Log records, only one change

- <T,X,u,v>= T has updated element X, its old value was u, and its new value is v

CSE 444 - Winter 2019 91

## Undo/Redo-Logging Rule

UR1: If T modifies X, then  $\langle T, X, u, v \rangle$  must be written to disk before  $\text{OUTPUT}(X)$

Note: we are free to  $\text{OUTPUT}$  early or late relative to  $\langle \text{COMMIT } T \rangle$

CSE 444 - Winter 2019

92

Action	T	Mem A	Mem B	Disk A	Disk B	Log
						$\langle \text{START } T \rangle$
READ(A,t)	8	8		8	8	
t:=t'2	16	8		8	8	
WRITE(A,t)	16	16		8	8	$\langle T, A, 8, 16 \rangle$
READ(B,t)	8	16	8	8	8	
t:=t'2	16	16	8	8	8	
WRITE(B,t)	16	16	16	8	8	$\langle T, B, 8, 16 \rangle$
OUTPUT(A)	16	16	16	16	8	
						$\langle \text{COMMIT } T \rangle$
OUTPUT(B)	16	16	16	16	16	

Can  $\text{OUTPUT}$  whenever we want: before/after  $\text{COMMIT}$  <sup>93</sup>

## Recovery with Undo/Redo Log

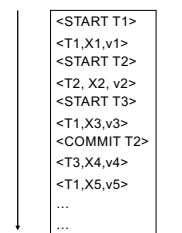
After system's crash, run recovery manager

- Redo all committed transaction, top-down
- Undo all uncommitted transactions, bottom-up

CSE 444 - Winter 2019

94

## Recovery with Undo/Redo Log



CSE 444 - Winter 2019

95

## ARIES

CSE 444 - Winter 2019

96

## Aries

- ARIES pieces together several techniques into a comprehensive algorithm
- Developed at IBM Almaden, by Mohan
- IBM botched the patent, so everyone uses it now
- Several variations, e.g. for distributed transactions

CSE 444 - Winter 2019

97



### Log Granularity

Two basic types of log records for update operations

- **Physical log records**
  - Position on a particular page where update occurred
  - Both before and after image for undo/redo logs
  - Benefits: Idempotent & updates are fast to redo/undo
- **Logical log records**
  - Record only high-level information about the operation
  - Benefit: Smaller log
  - BUT difficult to implement because crashes can occur in the middle of an operation

CSE 444 - Winter 2019 98

### ARIES Recovery Manager

Log entries:

- <START T> -- when T begins
- Update: <T,X,u,v>
  - T updates X, old value=u, new value=v
  - Logical description of the change
- <COMMIT T> or <ABORT T> then <END>
- <CLR> – we'll talk about them later.

CSE 444 - Winter 2019 100

### ARIES Recovery Manager

Rule:

- If T modifies X, then <T,X,u,v> must be written to disk before OUTPUT(X)

We are free to OUTPUT early or late w.r.t commits

CSE 444 - Winter 2019 101

### LSN = Log Sequence Number

- **LSN** = identifier of a log entry
  - Log entries belonging to the same TXN are linked with extra entry for previous LSN
- Each page contains a **pageLSN**:
  - LSN of log record for latest update to that page

CSE 444 - Winter 2019 102

### ARIES Data Structures

- **Active Transactions Table**
  - Lists all active TXN's
  - For each TXN: **lastLSN** = its most recent update LSN
- **Dirty Page Table**
  - Lists all dirty pages
  - For each dirty page: **recoveryLSN (recLSN)**= first LSN that caused page to become dirty
- **Write Ahead Log**
  - LSN, **prevLSN** = previous LSN for same txn

CSE 444 - Winter 2019 103

### ARIES Data Structures

W<sub>T100</sub>(P7)  
 W<sub>T200</sub>(P5)  
 W<sub>T200</sub>(P6)  
 W<sub>T100</sub>(P5)

#### Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

#### Log (WAL)

LSN	prevLSN	transID	pageID	Log entry
101	-	T100	P7	
102	-	T200	P5	
103	102	T200	P6	
104	101	T100	P5	

#### Active transactions

transID	lastLSN
T100	104
T200	103

#### Buffer Pool

P8	P2	...
P5 PageLSN=104	P6 PageLSN=103	P7 PageLSN=101

## ARIES Normal Operation

T writes page P

- What do we do ?

CSE 444 - Winter 2019

105

## ARIES Normal Operation

T writes page P

- What do we do ?
- Write  $\langle T, P, u, v \rangle$  in the **Log**
- **pageLSN**=LSN
- **prevLSN**=lastLSN
- **lastLSN**=LSN
- **recLSN**=if isNull then **LSN**

CSE 444 - Winter 2019

106

## ARIES Normal Operation

Buffer manager wants to OUTPUT(P)

- What do we do ?

Buffer manager wants INPUT(P)

- What do we do ?

CSE 444 - Winter 2019

107

## ARIES Normal Operation

Buffer manager wants to OUTPUT(P)

- Flush log up to **pageLSN**
  - Remove P from **Dirty Pages** table
- Buffer manager wants INPUT(P)
- What do we do ?

CSE 444 - Winter 2019

108

## ARIES Normal Operation

Buffer manager wants to OUTPUT(P)

- Flush log up to **pageLSN**
  - Remove P from **Dirty Pages** table
- Buffer manager wants INPUT(P)
- Create entry in **Dirty Pages** table  
**recLSN** = NULL

CSE 444 - Winter 2019

109

## ARIES Normal Operation

Transaction T starts

- What do we do ?

Transaction T commits/aborts

- What do we do ?

CSE 444 - Winter 2019

110

## ARIES Normal Operation

Transaction T starts

- Write **<START T>** in the log
- New entry T in **Active TXN**;  
lastLSN = null

Transaction T commits

- What do we do ?

CSE 444 - Winter 2019

111

## ARIES Normal Operation

Transaction T starts

- Write **<START T>** in the log
- New entry T in **Active TXN**;  
lastLSN = null

Transaction T commits

- Write **<COMMIT T>** in the log
- Flush log up to this entry
- Write **<END>**

CSE 444 - Winter 2019

112

## Checkpoints

Write into the log

- Entire **active transactions table**
- Entire **dirty pages table**

Recovery always starts by analyzing latest checkpoint

Background process periodically flushes dirty pages to disk

CSE 444 - Winter 2019

113

## ARIES Recovery

### 1. Analysis pass

- Figure out what was going on at time of crash
- List of dirty pages and active transactions

### 2. Redo pass (repeating history principle)

- Redo all operations, even for transactions that will not commit
- Get back to state at the moment of the crash

### 3. Undo pass

- Remove effects of all uncommitted transactions
- Log changes during undo in case of another crash during undo

CSE 444 - Winter 2019

114

## Announcements

- Lab 4 out tomorrow
- Lab 5 due dates extended
  - No late days allowed (will take that into consideration when setting deadline)
- HW 6 released tomorrow
  - On parallel database concepts

CSE 444 - Winter 2019

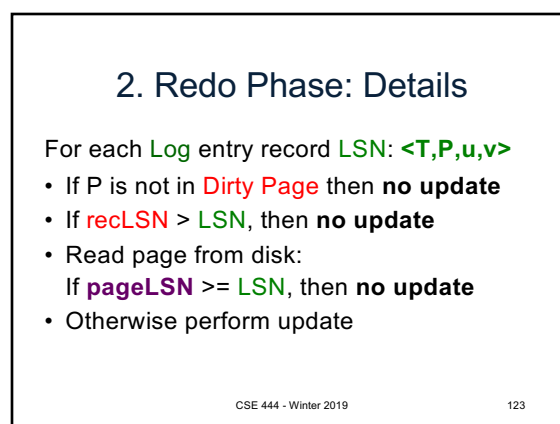
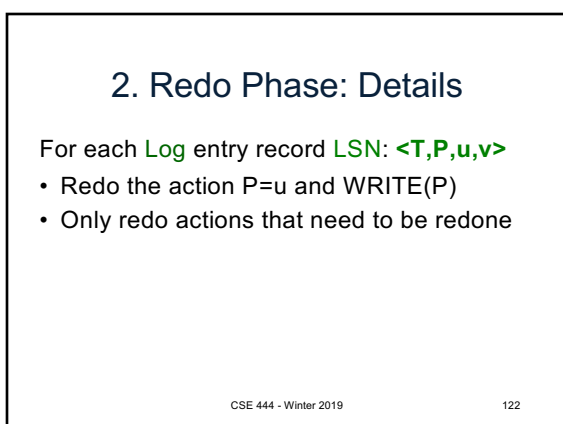
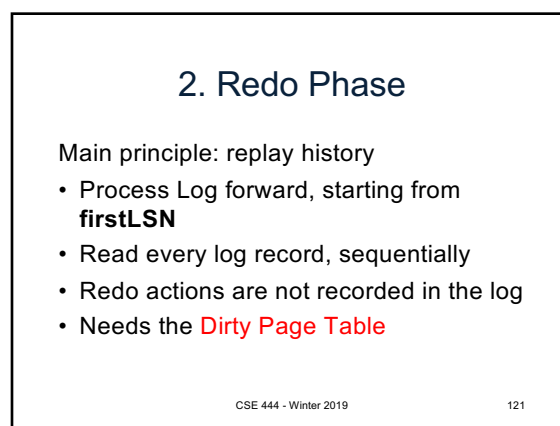
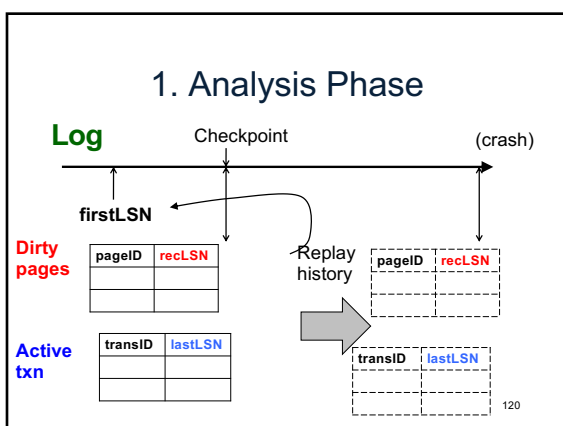
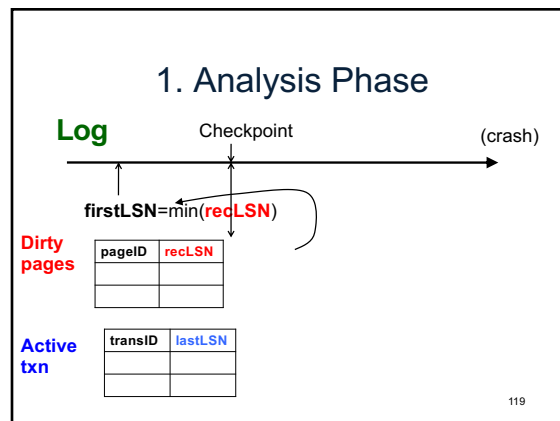
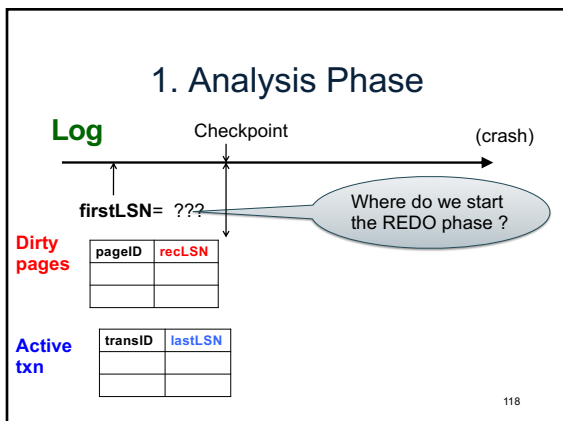
116

## 1. Analysis Phase

- Goal
  - Determine point in log where to start REDO
  - Determine set of dirty pages when crashed
    - Conservative estimate of dirty pages
  - Identify active transactions when crashed
- Approach
  - Rebuild **active transactions table** and **dirty pages table**
  - Reprocess the log from the checkpoint
    - Only update the two data structures
  - Compute: **firstLSN** = smallest of all **recoveryLSN**

CSE 444 - Winter 2019

117



## 2. Redo Phase: Details

What happens if system crashes during REDO ?

CSE 444 - Winter 2019

124

## 2. Redo Phase: Details

What happens if system crashes during REDO ?

We REDO again ! The pageLSN will ensure that we do not reapply a change twice

CSE 444 - Winter 2019

125

## 3. Undo Phase

- Cannot “unplay” history, in the same way as we “replay” history
- WHY NOT ?

CSE 444 - Winter 2019

126

## 3. Undo Phase

- Cannot “unplay” history, in the same way as we “replay” history
- WHY NOT ?
  - Undo only the loser transactions
  - Need to support ROLLBACK: selective undo, for one transaction
- Hence, *logical* undo v.s. *physical* redo

CSE 444 - Winter 2019

127

## 3. Undo Phase

Main principle: “logical” undo

- Start from end of **Log**, move backwards
- Read only affected log entries
- Undo actions *are* written in the Log as special entries: **CLR** (Compensating Log Records)
- **CLRs** are redone, but never undone

CSE 444 - Winter 2019

128

## 3. Undo Phase: Details

- “Loser transactions” = uncommitted transactions in **Active Transactions Table**
- **ToUndo** = set of **lastLSN** of loser transactions

CSE 444 - Winter 2019

129

### 3. Undo Phase: Details

While **ToUndo** not empty:

- Choose most recent (largest) **LSN** in **ToUndo**
- If **LSN** = regular record **<T,P,u,v>**:
  - Undo v
  - Write a **CLR** where **CLR.undoNextLSN** = **LSN.prevLSN**
- If **LSN** = **CLR** record:
  - Don't undo !
- if **CLR.undoNextLSN** not null, insert in **ToUndo** otherwise, write **<END>** in log

CSE 444 - Winter 2019

130

### 3. Undo Phase: Details

What happens if system crashes during UNDO ?

CSE 444 - Winter 2019

132

### 3. Undo Phase: Details

What happens if system crashes during UNDO ?

We do not UNDO again ! Instead, each CLR is a REDO record: we simply redo the undo

CSE 444 - Winter 2019

133