

Database System Internals

Transactions: Recovery (part 3)

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

ARIES

Undo/Redo protocol

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

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.

ARIES Recovery Manager

Rule:

- If T modifies X, then $\langle T, X, u, v \rangle$ must be written to disk before **OUTPUT(X)**

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

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

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

$W_{T100}(P7)$

$W_{T200}(P5)$

$W_{T200}(P6)$

$W_{T100}(P5)$

Data Structures

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Log (WAL) - tail of the log may be in memory

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 ?

Log (WAL) - tail of the log may be in memory

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

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 ?

- Write $\langle T, P, u, v \rangle$ in the **Log** – no need to flush to disk yet
- **pageLSN=LSN**
- **prevLSN=lastLSN**
- **lastLSN=LSN**
- **recLSN**=if isNull then **LSN**

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Log (WAL) - tail of the log may be in memory

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

Buffer manager wants to OUTPUT(P)

- What do we do ?

Buffer manager wants INPUT(P)

- What do we do ?

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Active transactions

transID	lastLSN
T100	104
T200	103

Log (WAL) - tail of the log may be in memory

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

Buffer Pool

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

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 ?

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Log (WAL) - tail of the log may be in memory

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

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

Log (WAL) - tail of the log may be in memory

Dirty pages		LSN	prevLSN	transID	pageID	Log entry
pageID	recLSN	101	-	T100	P7	
P5	102	102	-	T200	P5	
P6	103	103	102	T200	P6	
P7	101	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

Transaction T starts

- What do we do ?

Transaction T commits/aborts

- What do we do ?

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Active transactions

transID	lastLSN
T100	104
T200	103

Log (WAL) - tail of the log may be in memory

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

Buffer Pool

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

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 ?

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Active transactions

transID	lastLSN
T100	104
T200	103

Log (WAL) - tail of the log may be in memory

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

Buffer Pool

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

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>** and remove T from the Active Transactions

Dirty pages

pageID	recLSN
P5	102
P6	103
P7	101

Active transactions

transID	lastLSN
T100	104
T200	103

Log (WAL) - tail of the log may be in memory

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

Buffer Pool

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

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

ARIES Recovery

1. Analysis pass

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

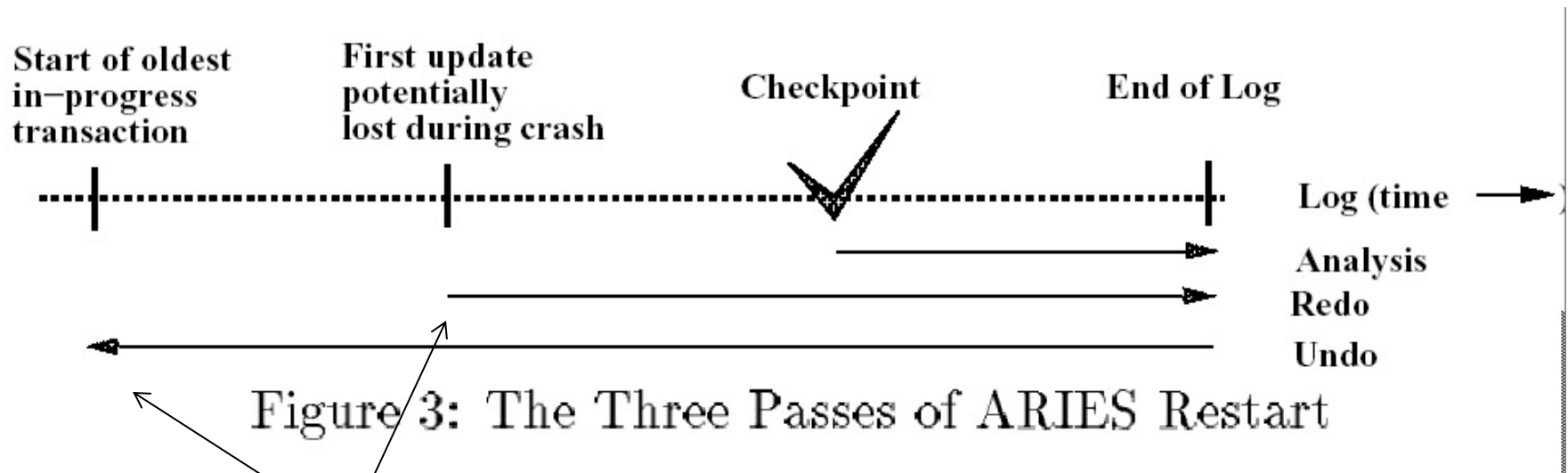
2. Redo pass (repeating history principle)

- 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

ARIES Method Illustration



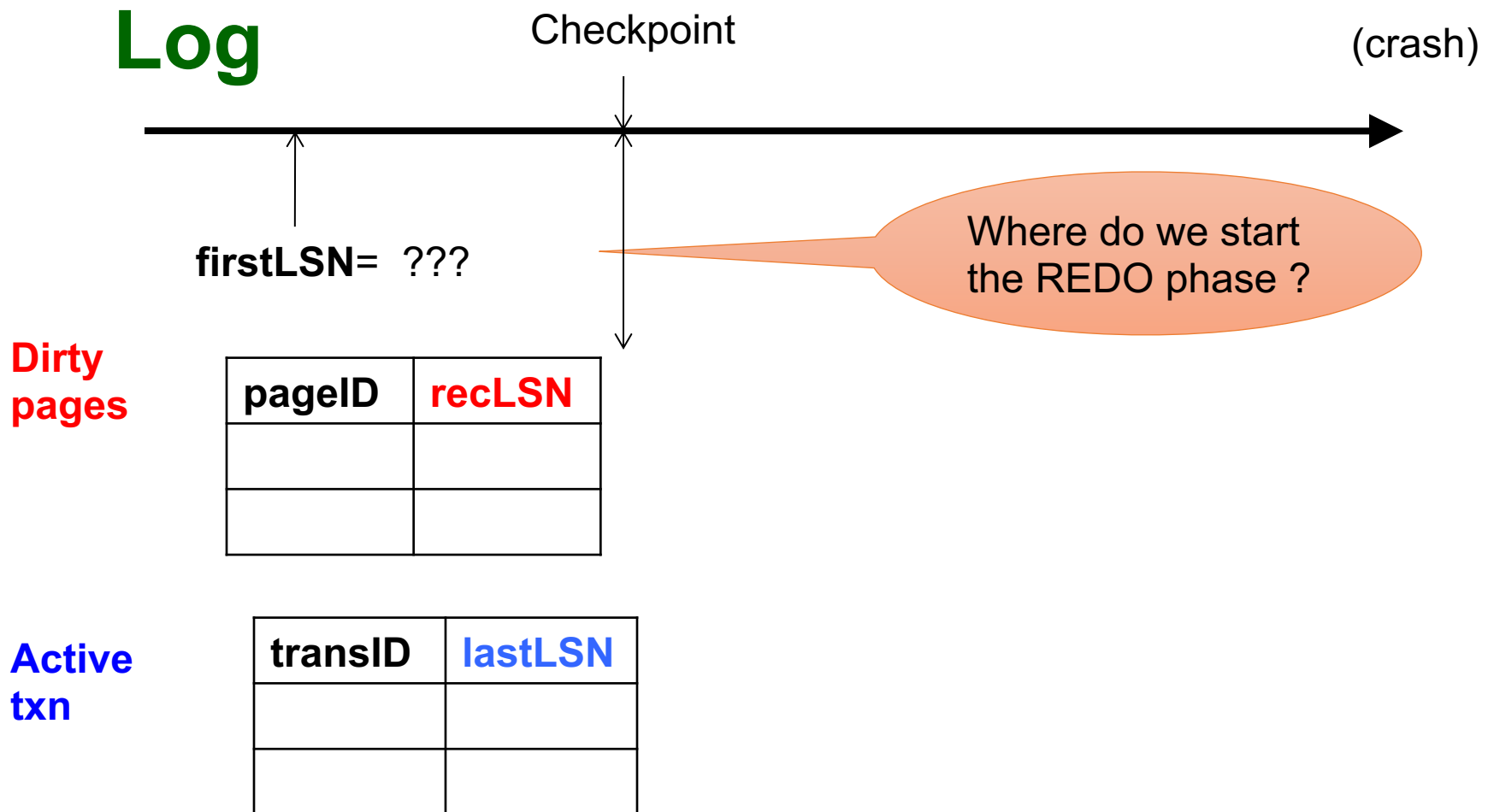
First undo and first redo log entry might be in reverse order

[Figure 3 from Franklin97]

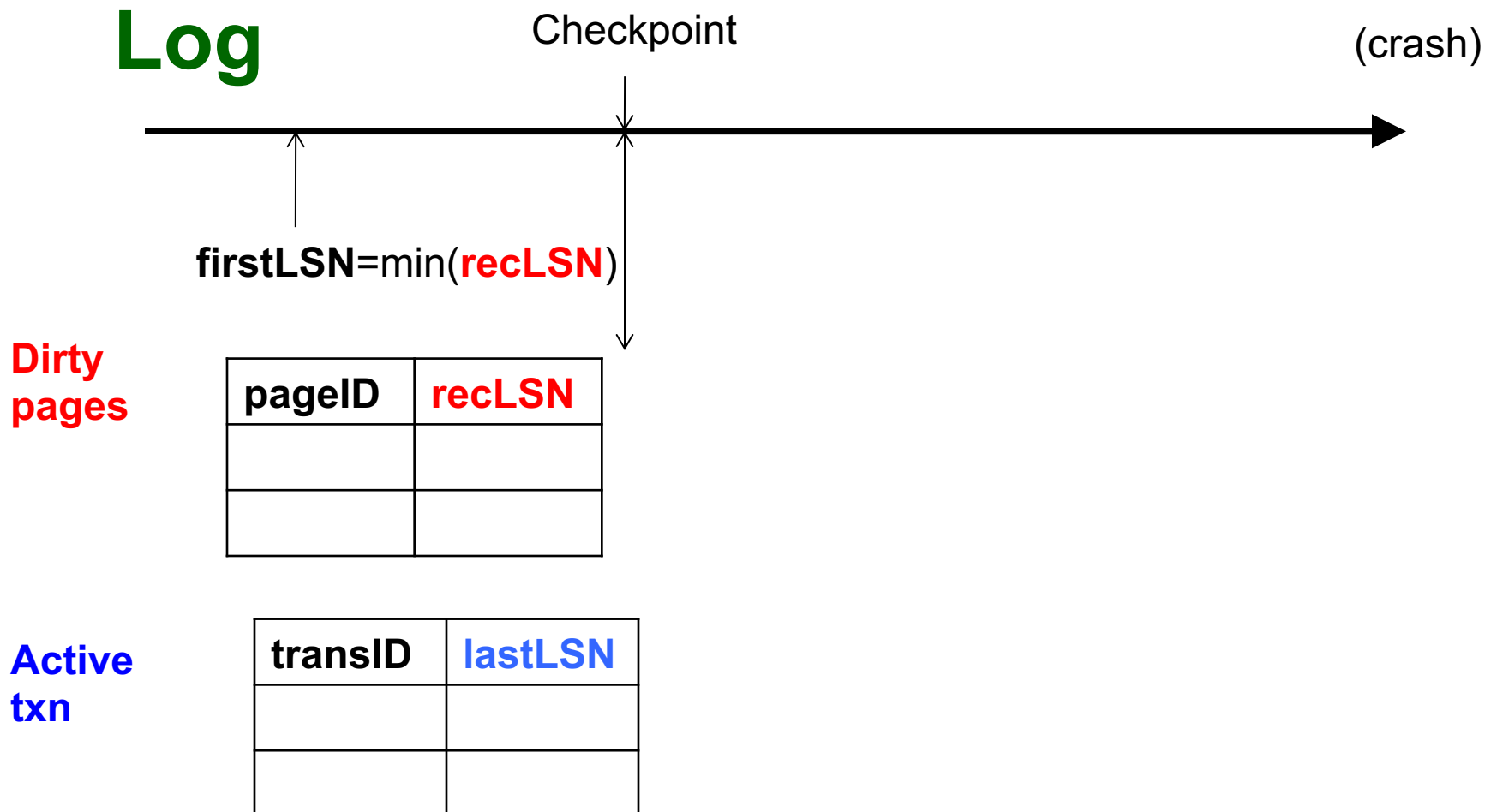
1. Analysis Phase

- 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**

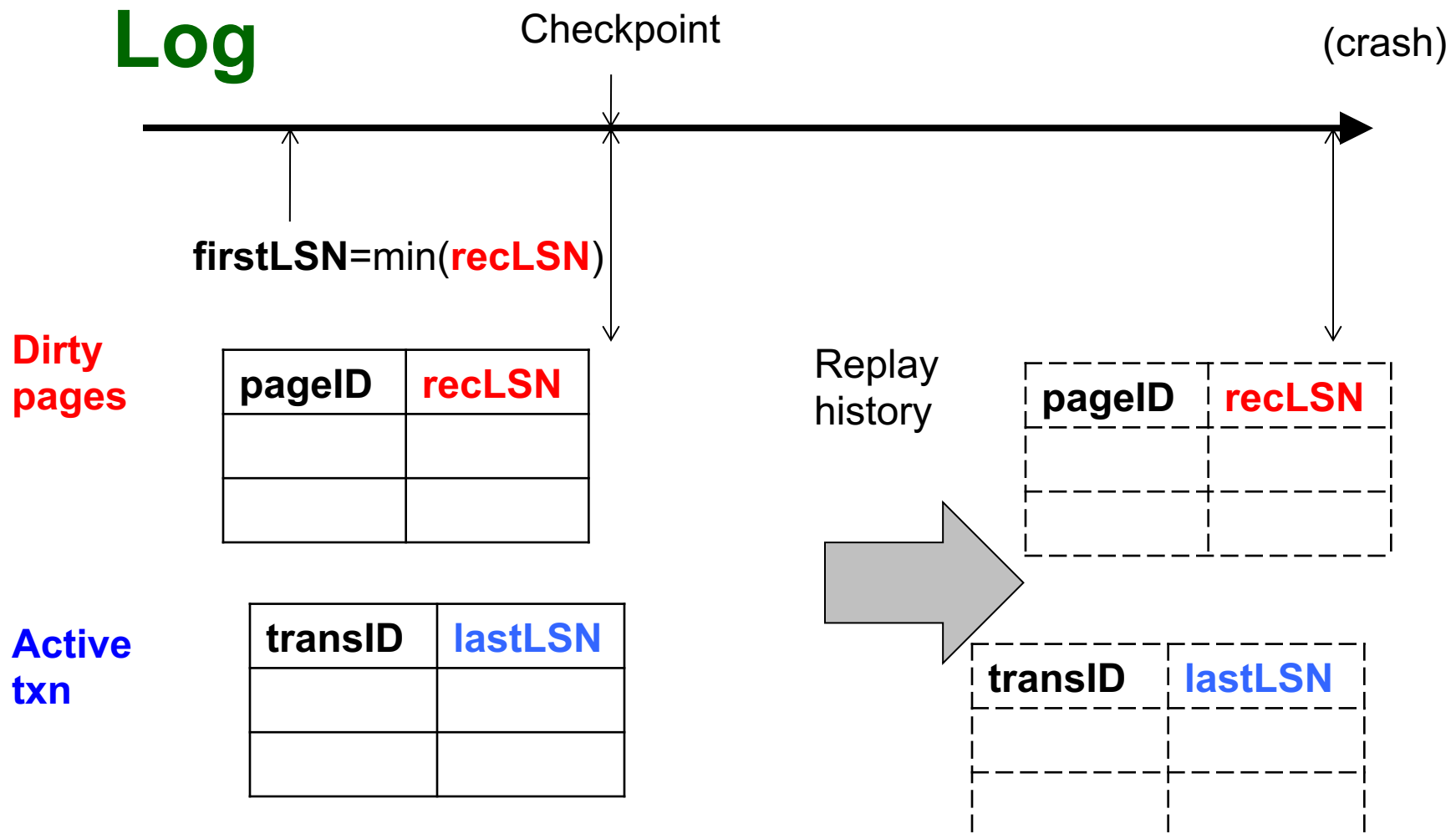
1. Analysis Phase



1. Analysis Phase



1. Analysis Phase



2. Redo Phase

Main principle: replay history

- Process Log forward, starting from **firstLSN**
- Read every log record, sequentially
- Redo actions are not recorded in the log
- Needs the **Dirty Page Table**

2. Redo Phase: Details

For each **Log** entry record **LSN: $\langle T, P, u, v \rangle$**

- Redo the action $P=v$ and $WRITE(P)$
- Only redo actions that need to be redone

2. Redo Phase: Details

For each **Log** entry record **LSN**: $\langle T, P, u, v \rangle$

- If **P** is not in **Dirty Page** then **no update**
- If **recLSN** $>$ **LSN**, then **no update**
- Read page from disk:
If **pageLSN** \geq **LSN**, then **no update**
- Otherwise perform update

2. Redo Phase: Details

What happens if system crashes during REDO ?

2. Redo Phase: Details

What happens if system crashes during REDO ?

We REDO again ! REDO is idempotent.

3. Undo Phase

- We could “unplay” history the same way as we “replay” history
- However, we cannot do this selectively, for one transactions that wants to ROLLBACK
- Reason 1:
 - TXN T1 updates one record on page X
 - TXN T2 updates another record on same page X
 - TXN T2 commits
 - TXN T1 wants to ROLLBACK: cannot restore X
- Reason 2: indexes; similar issue
- Explanation: limits of our abstract model...

3. Undo Phase

Main principle: “logical” undo

- Start from end of **Log**, move backwards
- Read only affected log entries
- UNDO is not idempotent!
- Solution: log the UNDO's as special log entries:
CLR (Compensating Log Records)
- **CLR**s are redone, but never undone

3. Undo Phase: Details

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

3. Undo Phase: Details

While **ToUndo** not empty:

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

3. Undo Phase: Details

What happens if system crashes during UNDO ?

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

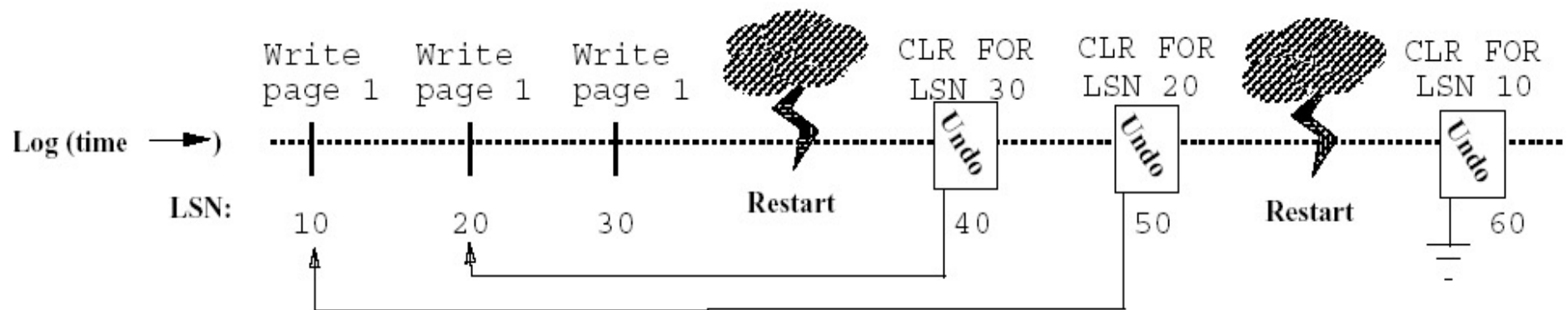


Figure 4: The Use of CLR for UNDO

[Figure 4 from Franklin97]

Example

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		

Example

ToUndo = {30} from the Active TXN table

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		

Example

ToUndo = {30,20}

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		
40	CLR(T,X,c)		20

Example

ToUndo = {30,20,10}

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		
40	CLR(T,X,c)		20
50	CLR(T,X,b)		10

Example

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		
40	CLR(T,X,c)		20
50	CLR(T,X,b)		10
	CRASH/RESTART		

Example

Redo phase: update $X=b$, $X=c$, $X=d$, $X=c$, $X=b$

LSN	Log Entry	prevLSN	undoNextLSN
10	$\langle T, X, a, b \rangle$	0	
20	$\langle T, X, b, c \rangle$	10	
30	$\langle T, X, c, d \rangle$	20	
	CRASH/RESTART		
40	CLR(T, X, c)		20
50	CLR(T, X, b)		10
	CRASH/RESTART		

Note: during REDO the CLR's are like regular update entries

Example

ToUndo = {50} this is the new **LastLSN** of T

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		
40	CLR(T,X,c)		20
50	CLR(T,X,b)		10
	CRASH/RESTART		

Example

ToUndo = {50}

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		
40	CLR(T,X,c)		20
50	CLR(T,X,b)		10
	CRASH/RESTART		

LSN 50 is a CLR; we do nothing, instead follow **undoNextLSN**

Example

ToUndo = {50,0} DONE!

LSN	Log Entry	prevLSN	undoNextLSN
10	<T,X,a,b>	0	
20	<T,X,b,c>	10	
30	<T,X,c,d>	20	
	CRASH/RESTART		
40	CLR(T,X,c)		20
50	CLR(T,X,b)		10
	CRASH/RESTART		
60	CLR(T,X,a)		0