

# Database Applications (15-415)

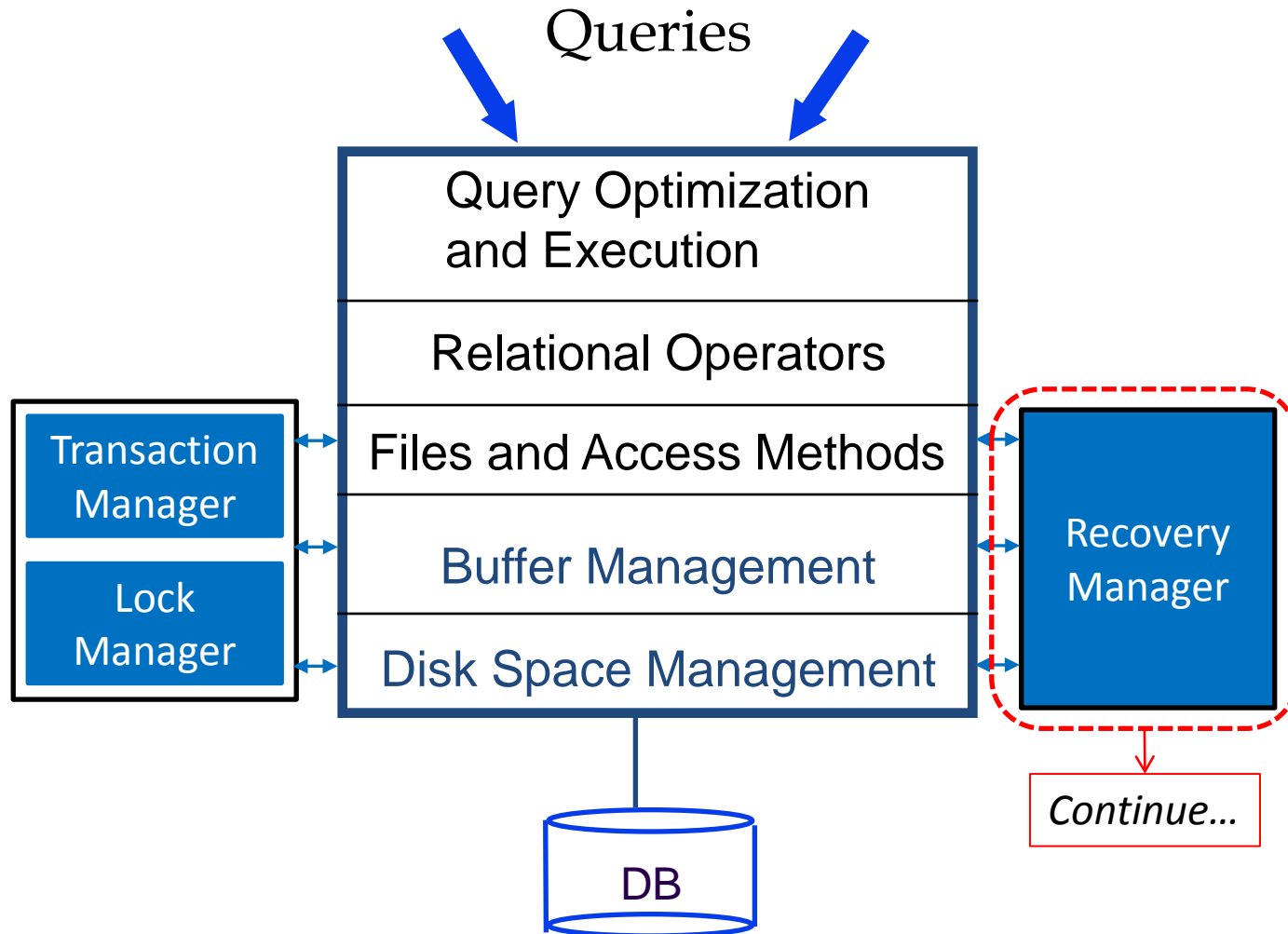
DBMS Internals- Part XIV  
Lecture 22, April 16, 2014

Mohammad Hammoud

# Today...

- Last Session:
  - Transaction Management (*finished*)
    - Non-Lock Based Protocols
  - Recovery Management
- Today's Session:
  - Recovery Management (*Cont'd*)
- Announcements:
  - Final Exam is on Sunday April 27<sup>th</sup>, at 9:00AM in room 2051 (*all materials are included- open book, open notes*)
  - PS5 (the “last” assignment) will be posted by tomorrow

# DBMS Layers



# Outline

The Log ✓

A Simple Transaction Abort

Checkpointing

The ARIES Algorithm

# The Log

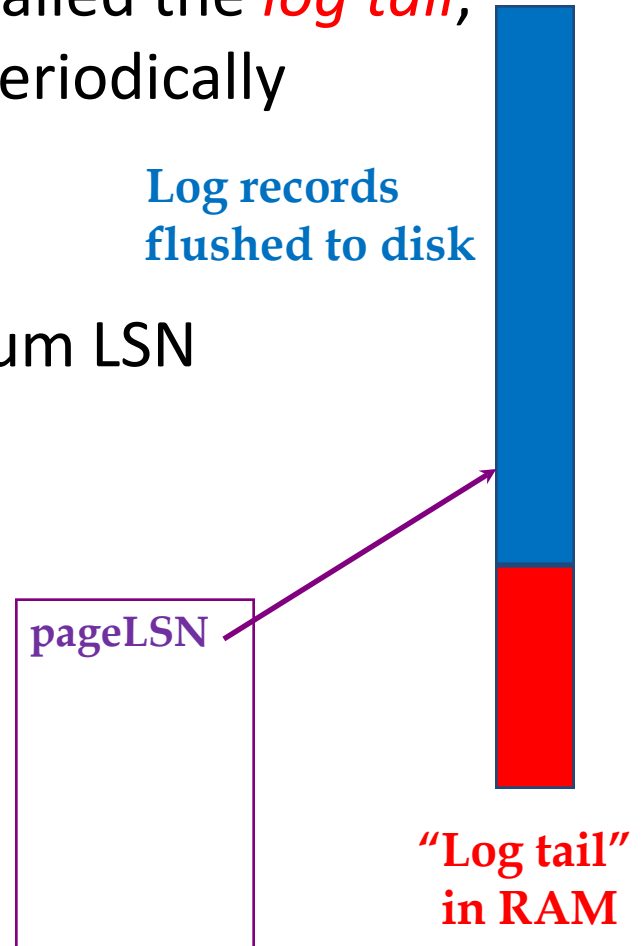
- The **log** is a file of records stored in stable storage
- Every log record is given a unique id called the **Log Sequence Number (LSN)**
  - LSNs are assigned in a monotonically increasing order (this is required by the ARIES recovery algorithm- *later*)
- Every page contains the LSN of the *most recent* log record, which describes a change to this page
  - This is called the **pageLSN**

# The Log (*Cont'd*)

- The most recent portion of the log, called the *log tail*, is kept in main memory and *forced* periodically to disk

- The DBMS keeps track of the maximum LSN flushed to disk so far
  - This is called the **flushedLSN**

- As per the WAL protocol, before a page is written to disk,  
 $\text{pageLSN} \leq \text{flushedLSN}$



# When to Write Log Records?

- A log record is written after:
  - **Updating a Page**
    - An *update log record* is appended to the log tail
    - The pageLSN of the page is set to the LSN of the update log record
  - **Committing a Transaction**
    - A *commit log record* is appended to the log tail
    - The log tail is written to stable storage, up to and including the commit log record
  - **Aborting a Transaction**
    - An *abort log record* is appended to the log tail
    - An undo is initiated for this transaction

# When to Write Log Records?

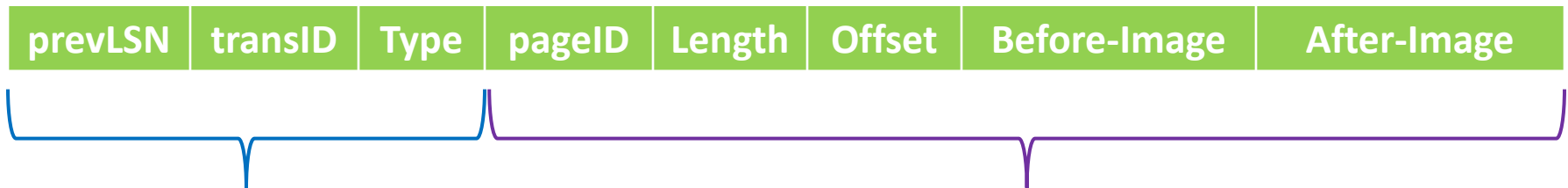
- A log record is written after:
  - Ending (After Aborting or Committing) a Transaction:
    - Additional steps are completed (*later*)
    - An *end log record* is appended to the log tail
  - Undoing an Update
    - When the action (described by an update log record) is undone, a *compensation log record* (CLR) is appended to the log tail
    - CLR describes the action taken to undo the action recorded in the corresponding update log record



# Log Records

- The fields of a log record are usually as follows:

Can be used to *redo* and *undo* the changes!



- Fields common to *all* log records:

- Update Log Records
- Commit Log Records
- Abort Log Records
- End Log Records
- Compensation Log Records

Additional Fields for only the Update Log Records

# Other Recovery-Related Structures

- In addition to the log, the following two tables are maintained:
  - **The Transaction Table**
    - One entry  $E$  for each active transaction
    - $E$  fields are:
      - *Transaction ID*
      - *Status*, which can be “*Progress*”, “*Committed*” or “*Aborted*”
      - *lastLSN*, which is the most recent log record for this transaction
  - **The Dirty Page Table**
    - One entry  $E'$  for each dirty page in the buffer pool
    - $E'$  fields are:
      - *Page ID*
      - *recLSN*, which is the LSN of the first log record that caused the page to become dirty

# An Example

PageID	recLSN
P500	
P600	

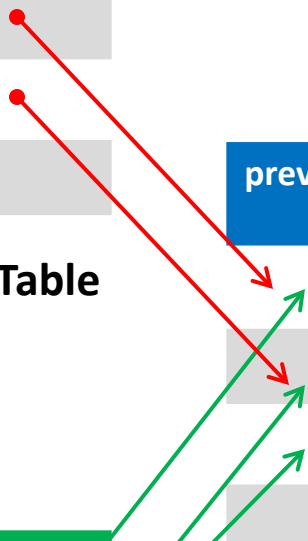
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS

LOG



# An Example

PageID	recLSN
P500	
P600	

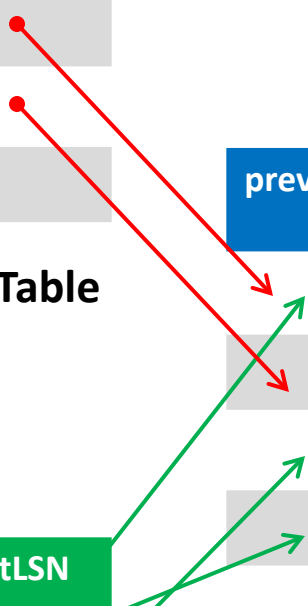
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
	T1000	Update	P505	3	21	TUV	WXY

LOG



# An Example

PageID	recLSN
P500	
P600	
P505	

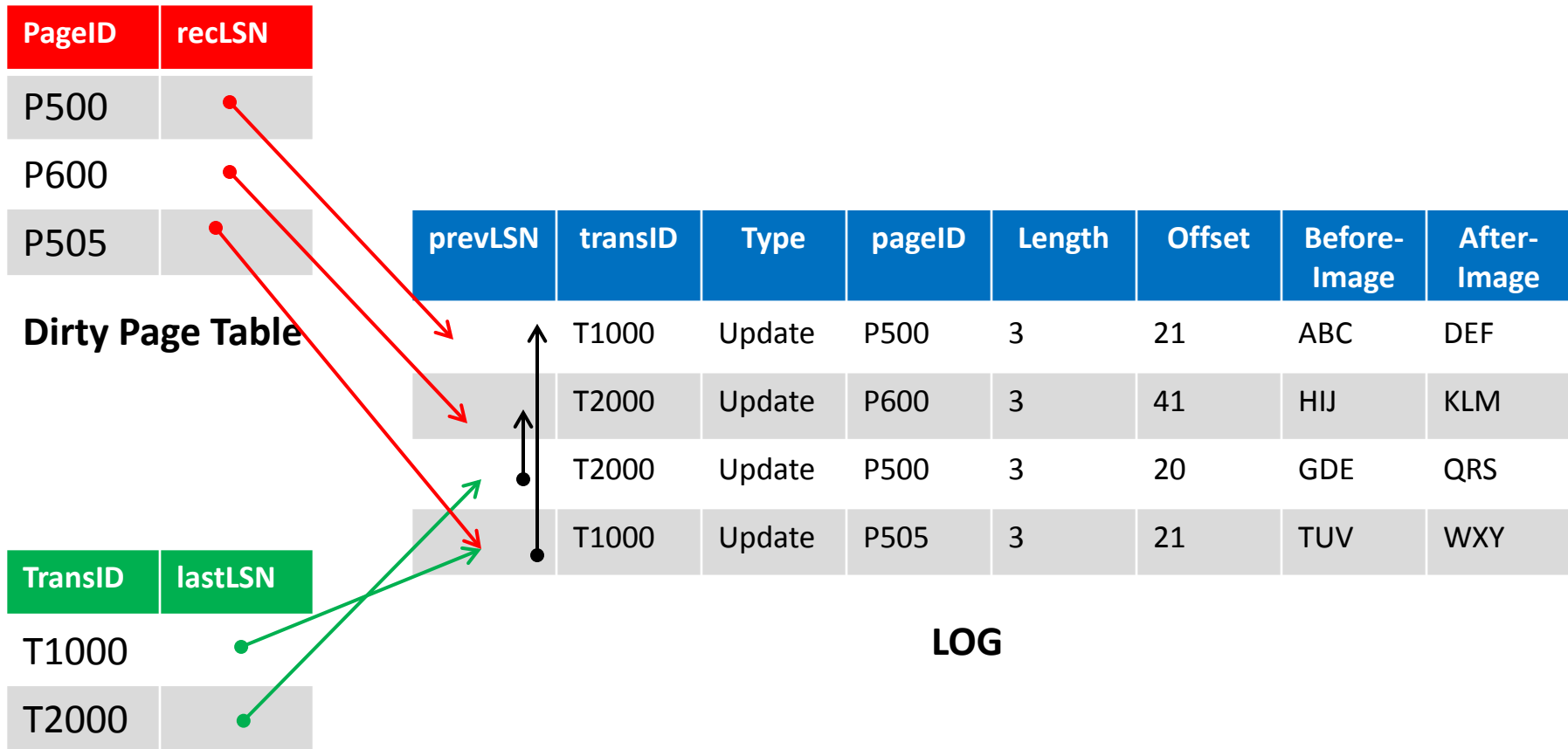
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
	T1000	Update	P505	3	21	TUV	WXY

LOG



# Outline

The Log

A Simple Transaction Abort ✓

Checkpointing

The ARIES Algorithm

# A Simple Transaction Abort

- For now, let us consider an “explicit” abort of a transaction  $T$ 
  - That is, no system crash is involved
- We want to “play back” the log in reverse order, *undoing*  $T$ 's updates
  - **Step 1:** We get the *lastLSN* of  $T$  from the Transaction table
  - **Step 2:** We lock the corresponding data to be undone (we can use strict 2PL)

# A Simple Transaction Abort (*Cont'd*)

- **Step 3:** before restoring an old value of a page, we write a respective *Compensation Log Record* (CLR)
  - CLR has one extra field, that is, *undoNextLSN*, which points to the next LSN to undo
    - That is, the *prevLSN* of the record we are currently undoing
  - CLR's are never undone (but they might be Redone)
- **Step 4:** repeat **steps 2** and **3** by following a *chain* of log records backward via the *prevLSN* field
- **Last Step:** at the end of UNDO, write an *end log record*



# An Example

Let us assume T1000 is aborted!

PageID	recLSN
P500	
P600	
P505	

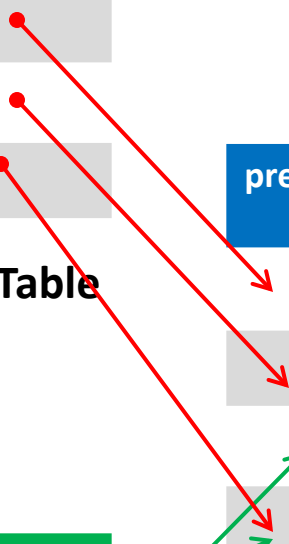
Dirty Page Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY

LOG

TransID	lastLSN
T1000	
T2000	

Transaction Table



# An Example

Step 1: Get the lastLSN of T1000 from the Transaction table

PageID	recLSN
P500	
P600	
P505	

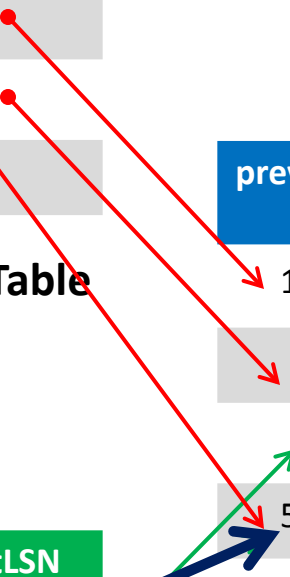
Dirty Page Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY

LOG

TransID	lastLSN
T1000	
T2000	

Transaction Table



# An Example

Step 2: Lock P505

PageID	recLSN
P500	
P600	
P505	

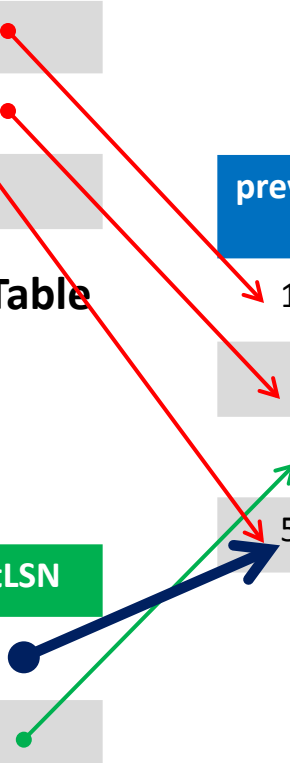
Dirty Page Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY

LOG

TransID	lastLSN
T1000	
T2000	

Transaction Table



# An Example

Step 3: Write CLR

PageID	recLSN
P500	
P600	
P505	

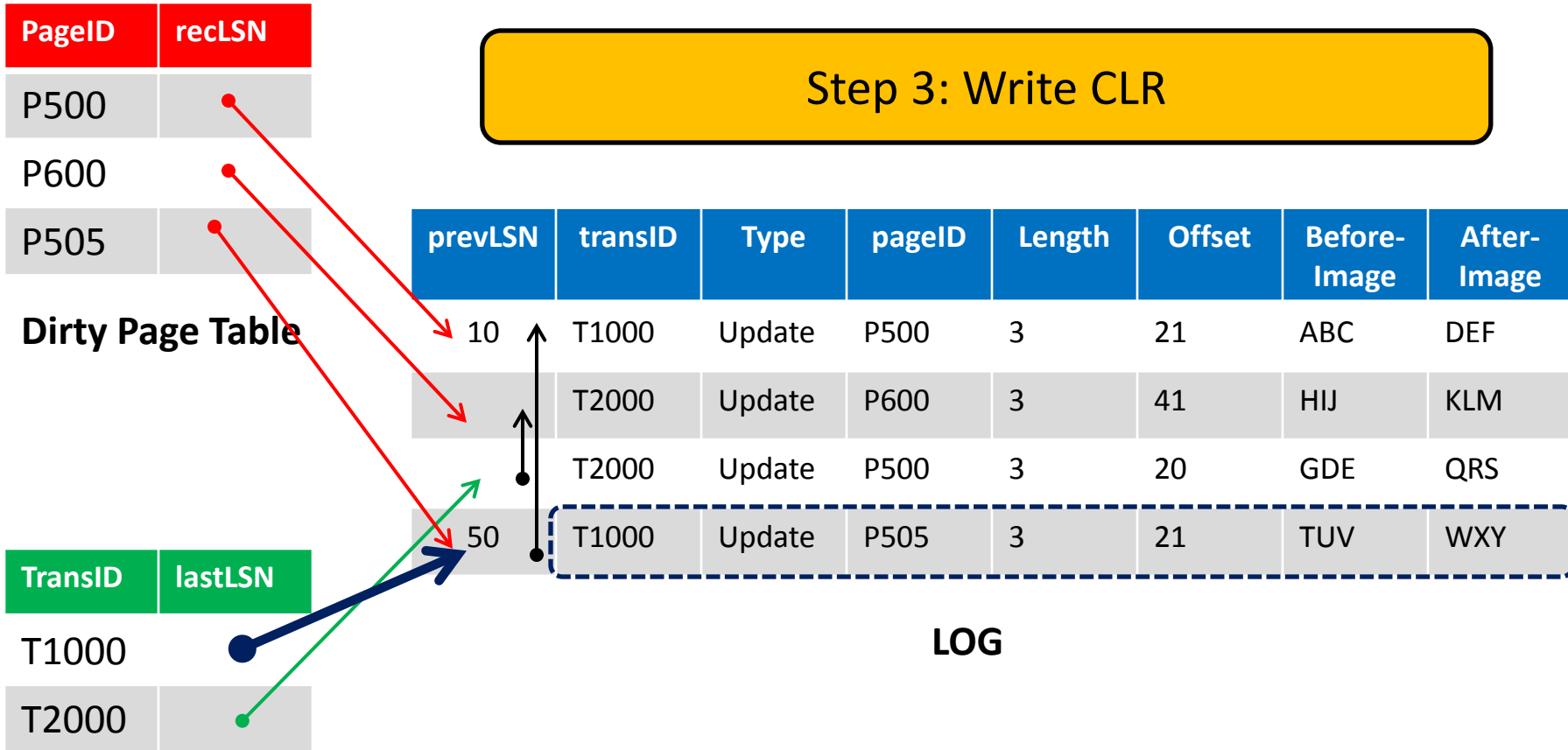
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY

LOG



# An Example

Step 3: Write CLR

PageID	recLSN
P500	
P600	
P505	

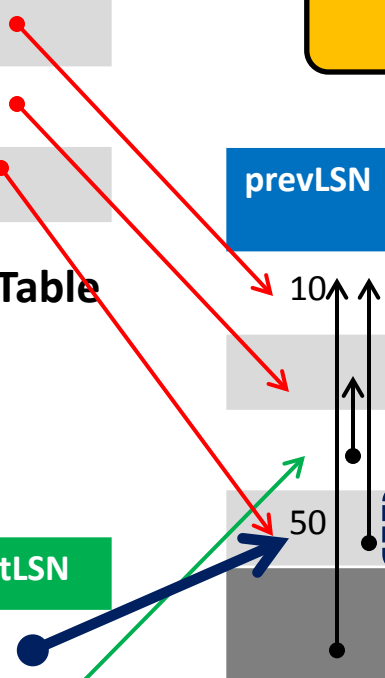
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000- LSN 50	CLR					

LOG



# An Example

Step 4: Restore old value "TUV"

PageID	recLSN
P500	
P600	
P505	

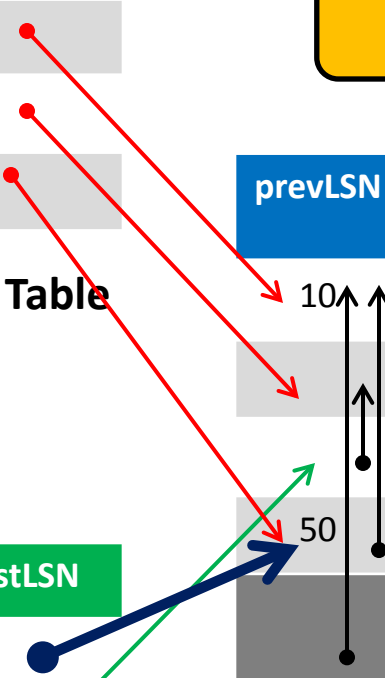
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					

LOG



# An Example

Step 4: Restore old value "TUV"

PageID	recLSN
P500	
P600	
P505	

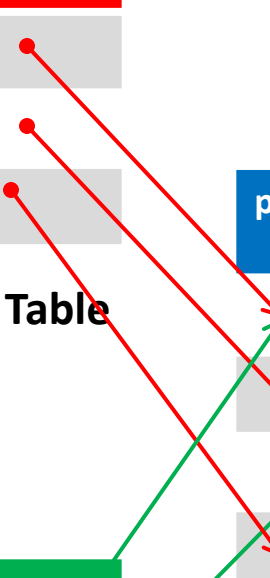
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					

LOG



# An Example

Step 4: Restore old value "TUV"

PageID	recLSN
P500	
P600	

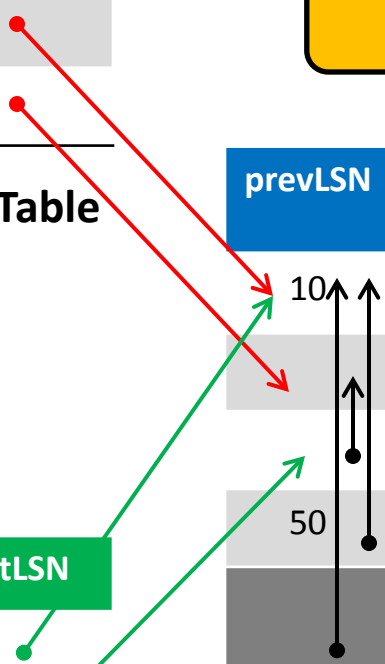
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					

LOG





# An Example

Step 5: Lock P500

PageID	recLSN
P500	
P600	

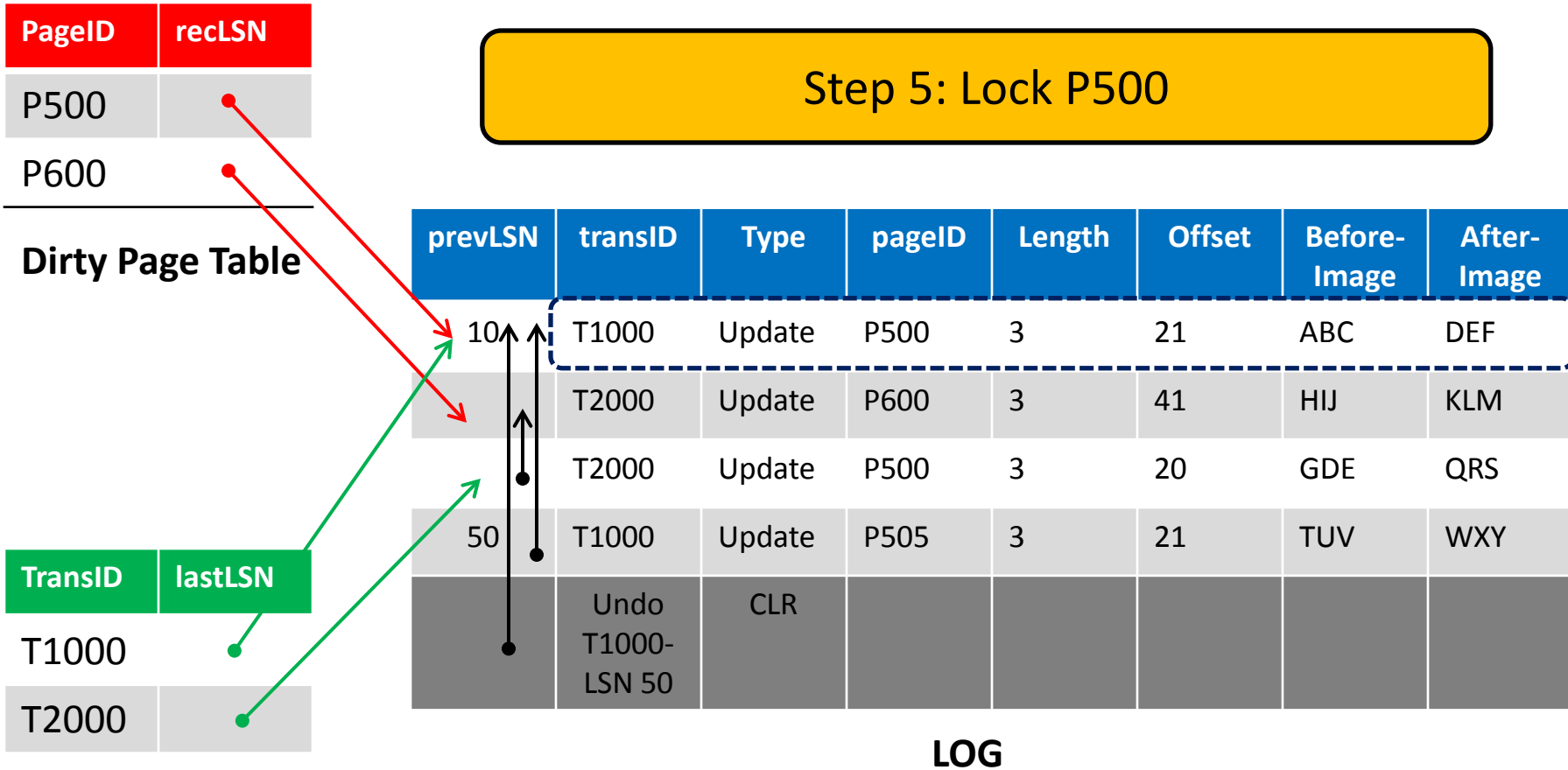
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					

LOG



# An Example

Step 6: Write CLR

PageID	recLSN
P500	
P600	

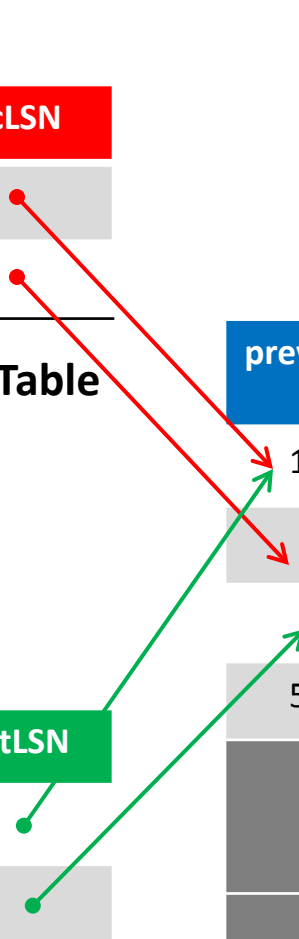
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					
	Undo T1000-LSN 10	CLR					

LOG



# An Example

Step 7: Restore old value "ABC"

PageID	recLSN
P500	
P600	

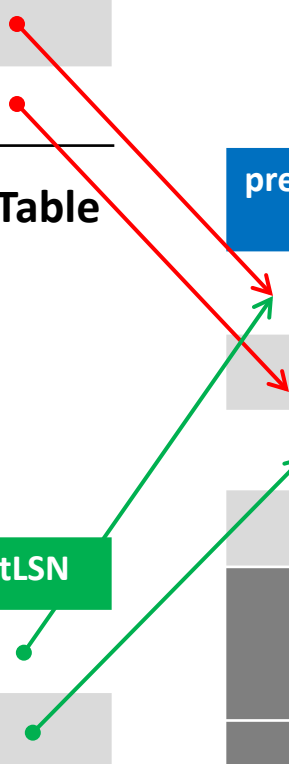
Dirty Page Table

TransID	lastLSN
T1000	
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					
	Undo T1000-LSN 10	CLR					

LOG



# An Example

Step 7: Restore old value "ABC"

PageID	recLSN
P500	
P600	

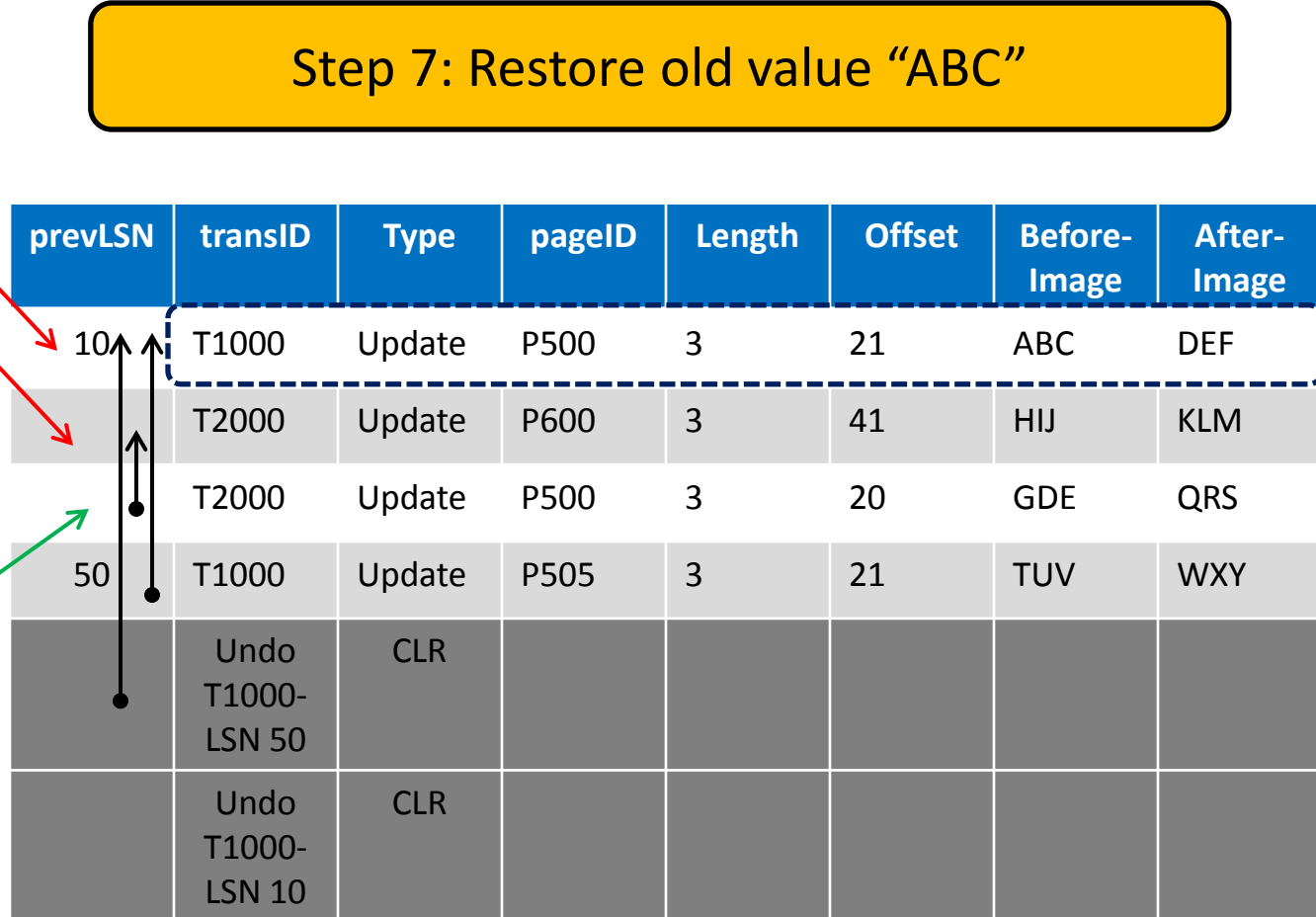
Dirty Page Table

TransID	lastLSN
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					
	Undo T1000-LSN 10	CLR					

LOG



# An Example

Step 7: Restore old value "ABC"

PageID	recLSN
P500	
P600	

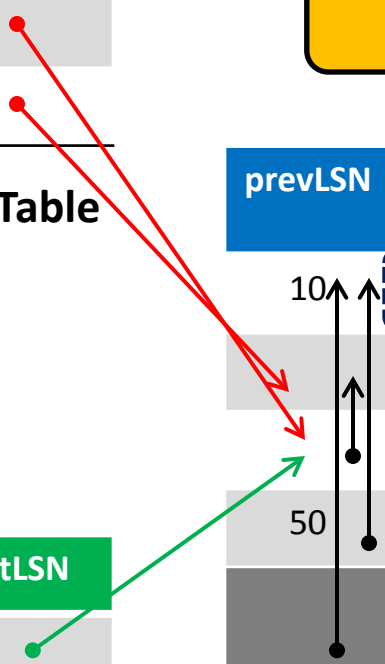
Dirty Page Table

TransID	lastLSN
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					
	Undo T1000-LSN 10	CLR					

LOG



# An Example

Step 8: Write an end log record

PageID	recLSN
P500	
P600	

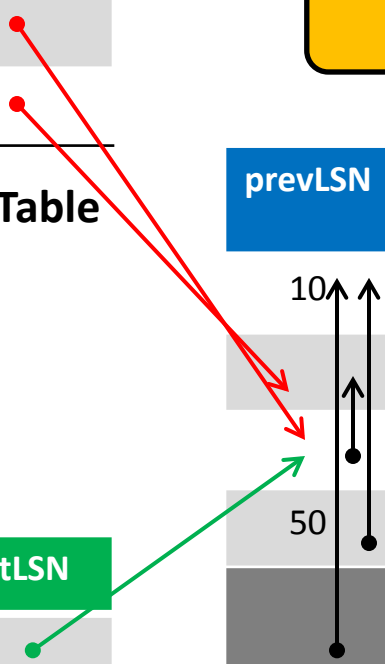
Dirty Page Table

TransID	lastLSN
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					
	Undo T1000-LSN 10	CLR					

LOG



# An Example

Step 8: Write an end log record

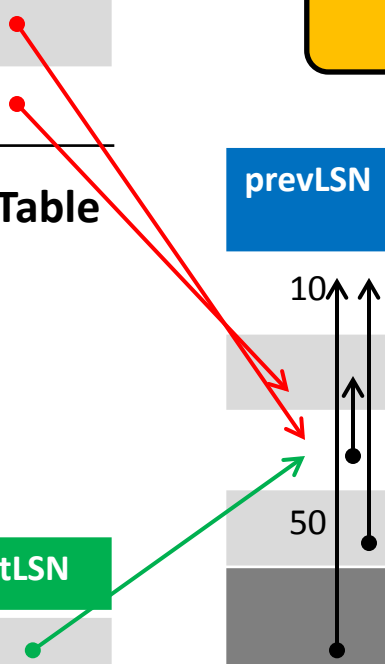
PageID	recLSN
P500	
P600	

Dirty Page Table

TransID	lastLSN
T2000	

Transaction Table

prevLSN	transID	Type	pageID	Length	Offset	Before-Image	After-Image
10	T1000	Update	P500	3	21	ABC	DEF
	T2000	Update	P600	3	41	HIJ	KLM
	T2000	Update	P500	3	20	GDE	QRS
50	T1000	Update	P505	3	21	TUV	WXY
	Undo T1000-LSN 50	CLR					
	Undo T1000-LSN 10	CLR					
	T1000 End-LSN 10	END					



# Outline

The Log

A Simple Transaction Abort

Checkpointing ✓

The ARIES Algorithm



# Checkpointing

- To reduce the amount of work to do during recovery, DBMSs typically take *checkpoints*
- A checkpoint is like a *snapshot of a DBMS state*
- A checkpoint can be taken by writing to the log:
  - **A begin\_checkpoint record**
    - This indicates the start of the checkpoint
  - **An end\_checkpoint record**
    - This indicates the end of the checkpoint
    - It includes the contents of the Transaction and the Dirty Page tables
  - **A master record**
    - This contains the LSN of the begin\_checkpoint record

# Outline

The Log

A Simple Transaction Abort

Checkpointing

The ARIES Algorithm

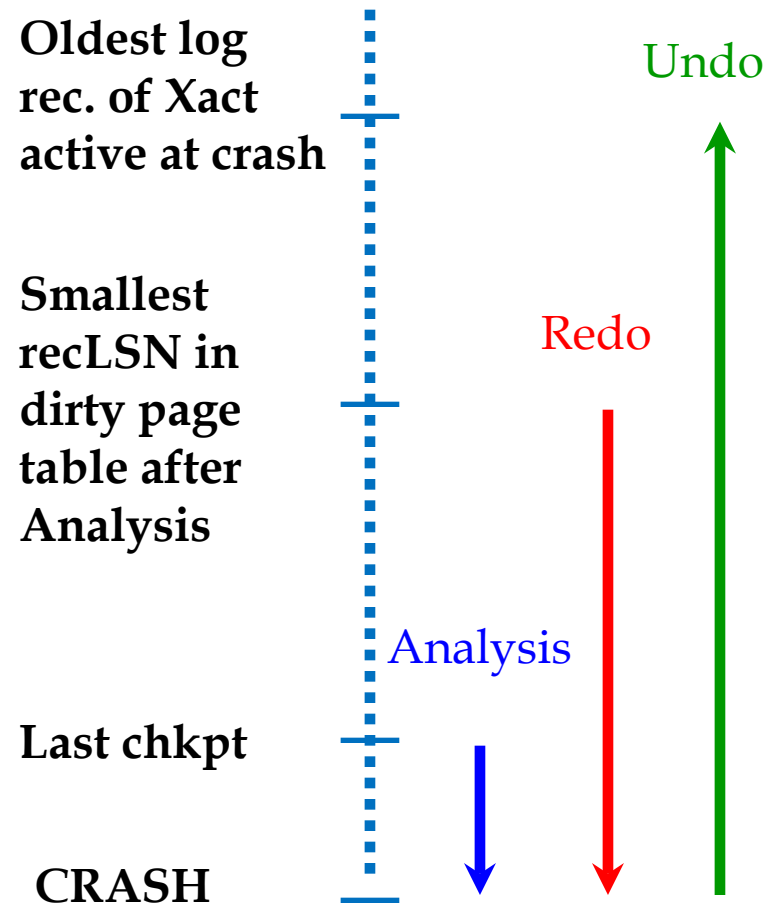


# Recovering From a System Crash: ARIES

- We will study the ARIES algorithm for recovering from system crashes
- ARIES is designed to work with a steal, no-force approach
- When the recovery manager is invoked after a crash, restart proceeds in three phases:
  - Analysis
  - Redo
  - Undo

# Recovering From a System Crash: ARIES

- **The Analysis Phase:**
  - Identifies dirty pages in the buffer pool and active transactions at the time of the crash
- **The Redo Phase:**
  - Redoes *all* actions
- **The Undo Phase:**
  - Undoes the actions of transactions that were active and did not commit



# ARIES: The Analysis Phase

- The Analysis phase encompasses two main steps:
  - **Step 1:** Reconstruct state (i.e., Dirty Page and Transaction tables) via the *end\_checkpoint* record, after the most recent *begin\_checkpoint* record
  - **Step 2:** Scan the log in the forward direction, starting after the checkpoint
    - If an *end log record* is encountered
      - Remove the corresponding transaction from the Transaction table

# ARIES: The Analysis Phase

- The Analysis phase encompasses two main steps:
  - **Step 2 (Cont'd):**
    - If any other record is encountered
      - Add the corresponding transaction to the Transaction table (*if it is not already there*)
      - Set lastLSN to the LSN of the record
      - Set status to **C** for committed transactions, or to **U** (i.e., Undo), otherwise
    - When an *update log record* is encountered
      - If the recorded page, **P**, is not in the Dirty Page table
        - Add **P** to the Dirty Page table and set its recLSN to the LSN of the log record

# ARIES: The Analysis Phase

- At the end of the Analysis phase:
  - The Transaction table contains an “accurate” list of all transactions that were active at the time of the crash
  - The Dirty Page table contains all pages that were dirty at the time of the crash
    - These pages may contain some pages that were written to disk (why?)– Not a Problem!

# ARIES: The Redo Phase

- During the Redo phase, ARIES reapplies the updates of “all” transactions (i.e., *committed* and *aborted*)
  - This paradigm is referred to as *Repeating History*
- The Redo phase scans forward until the end of the log, and redoes every action unless:
  - The affected page is not in the Dirty Page table
  - The affected page is in the Dirty Page table, but its recLSN > the current record’s LSN
  - The pageLSN of the affected page  $\geq$  the current record’s LSN

*Wouldn't checking this be enough?*



# ARIES: The Redo Phase

- During the Redo phase, ARIES reapplies the updates of “all” transactions (i.e., *committed* and *aborted*)
  - This paradigm is referred to as *Repeating History*
- The Redo phase scans forward until the end of the log, and redoes every action unless:
  - The affected page is not in the Dirty Page table
  - The affected page is in the Dirty Page table, but its recLSN > the current record’s LSN
  - The pageLSN of the affected page  $\geq$  the current record’s LSN

*YES, but it requires retrieving the page from the disk, thus made last!*

# ARIES: The Redo Phase

- If the logged action must be redone:
  - The logged action is reapplied
- The pageLSN on the page is set to the LSN of the redone log record
  - No additional record is written at this time!

# ARIES: The Undo Phase

- This phase will undo the actions of all transactions that were active before the crash
  - These transactions are referred to as *loser transactions* and were identified by the Analysis phase
- The Undo phase:
  - Considers the set of lastLSN values for all loser transactions
    - This is denoted as the **ToUndo** set
  - Repeatedly chooses the *largest* (i.e., the most recent) LSN value in ToUndo and processes it, until ToUndo is empty

# ARIES: The Undo Phase

- In particular, the Undo phase proceeds as follows:

## Repeat:

Choose largest LSN among ToUndo

If this LSN is a CLR and undoNextLSN==NULL

Write an End record for this Xact

If this LSN is a CLR, and undoNextLSN != NULL

Add undonextLSN to ToUndo

Else this LSN is an update

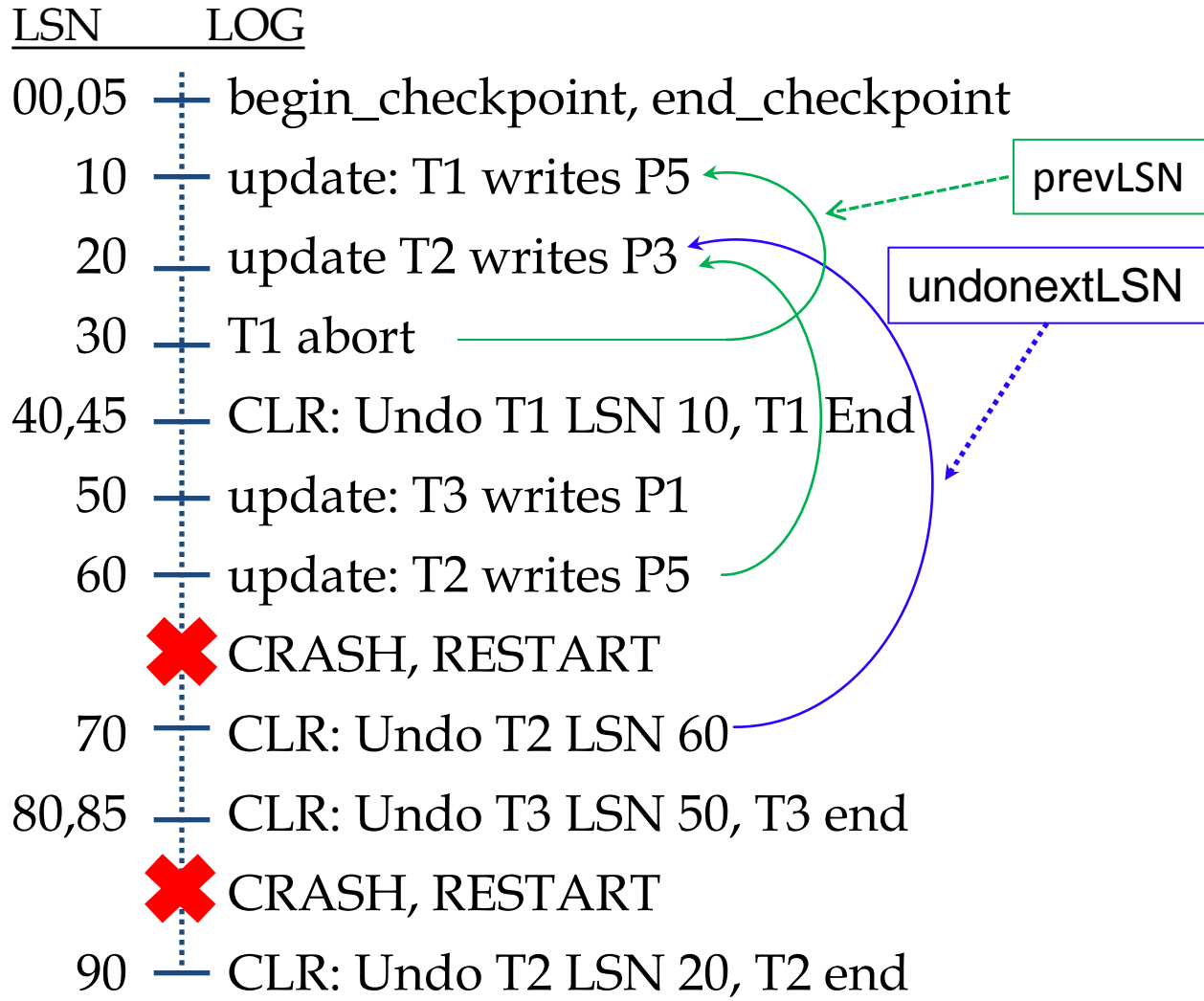
Undo the update

Write a CLR

Add prevLSN to ToUndo

Until **ToUndo** is empty

# An Example



# Additional Crash Issues

- What happens if the system crashes while “Restart” is in the Analysis phase?
  - All the work done is lost!
  - On a second Restart, the Analysis phase starts afresh
- What happens if the system crashes while “Restart” is in the Redo phase?
  - Restart starts again with the Analysis phase then the Redo phase
  - But, some of the changes made during Redo may have been written to disk
    - The update log records that were done the first time around will not be redone a second time (why?)

# Summary

- **Recovery Manager** guarantees Atomicity & Durability
- WAL is used to allow STEAL/NO-FORCE without sacrificing correctness
- LSNs identify log records; linked into backwards chains per transaction (via prevLSN)
- pageLSNs allow comparisons of data pages and log records

# Summary

- **Checkpointing:** A quick way to limit the amount of log to scan on recovery
- Recovery works in 3 phases:
  - **Analysis:** Forward from checkpoint
  - **Redo:** Forward from oldest recLSN
  - **Undo:** Backward from end to first LSN of oldest transaction alive at crash
- Upon Undo, write CLR
- Redo “repeats history”: Simplifies the logic!



# Next Class

The NoSQL Movement!