

Database Management Systems

Lecture 4

Crash Recovery

Recovery - ACID

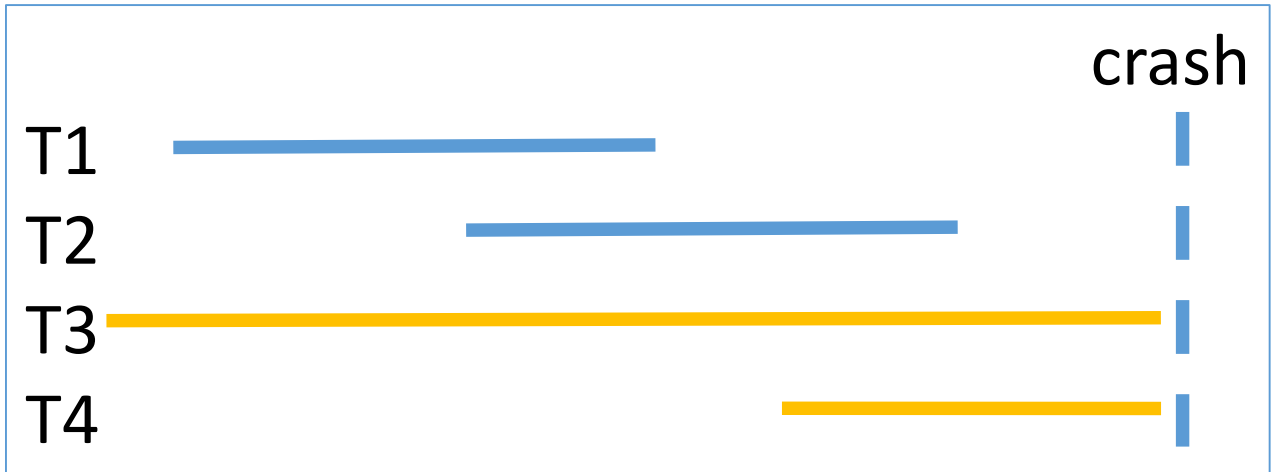
- the Recovery Manager in a DBMS ensures two important properties of transactions
 - atomicity
 - the effects of uncommitted transactions are undone
 - durability
 - the effects of committed transactions survive system crashes

Recovery - ACID

- atomicity
 - transactions can fail for various reasons (rollback)
- durability
 - what happens if the DBMS stops?

- when the system comes back up:

- T1 & T2 – effects must persist
- T3 & T4 - undone (their effects are not persisted)



Transaction Failure - Causes

- system failure
- application error
- action by the Transaction Manager
- self-abort

- system failure
 - hardware failures
 - bugs in the operating system, database system, etc
 - all running transactions terminate
 - contents of internal memory – affected (i.e., lost)
 - contents of external memory – not affected

Transaction Failure - Causes

- system failure
- application error
- action by the Transaction Manager
- self-abort

- application error
 - “bug” => transaction fails
 - e.g., division by 0, infinite loop, etc
 - transaction should be executed again only after the error is corrected

Transaction Failure - Causes

- system failure
 - application error
 - action by the Transaction Manager
 - self-abort
-
- action by the Transaction Manager
 - e.g., deadlock resolution scheme
 - a transaction is chosen as the deadlock victim and terminated
 - the transaction might complete successfully if executed again

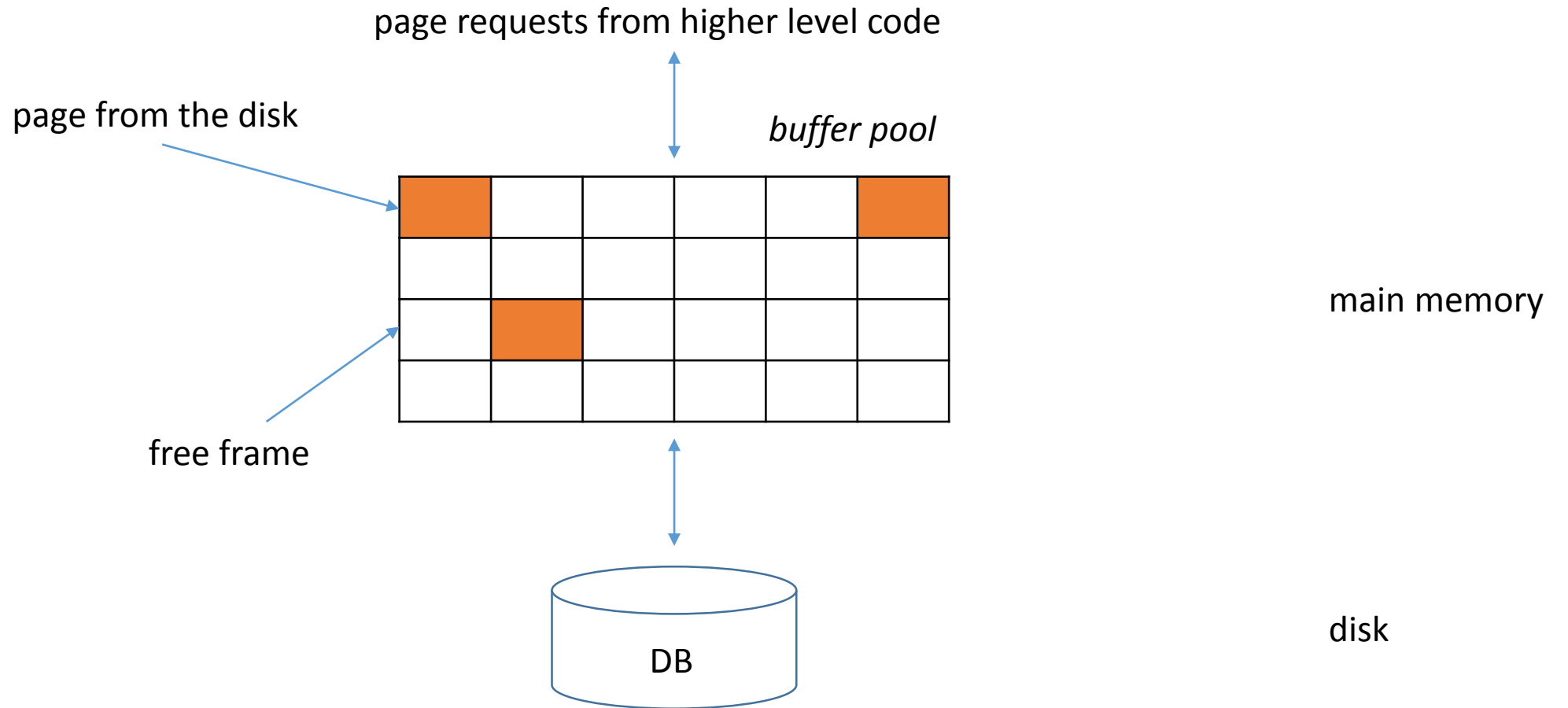
Transaction Failure - Causes

- system failure
- application error
- action by the Transaction Manager
- self-abort
- self-abort
 - based on some computations, a transaction can decide to terminate and undo its actions
 - special statements, e.g., *ABORT*, *ROLLBACK*
 - can be seen as a special case of *action by the Transaction Manager*

Normal Execution

- transaction
 - reads/writes of database objects
- read database object O
 - bring O from the disk into a frame in the Buffer Pool (BP)
 - copy O's value into a program variable
- write database object O
 - modify an in-memory copy of O (in the BP)
 - write the in-memory copy to disk

Buffer Management*



*see the *Databases* course in the 1st semester

Buffer Management

- higher level layer L in the DBMS:
 - asks the BM for page P
 - if P is not in the BP, the BM brings it into a frame F in the BP
 - when P is no longer needed:
 - L releases P
 - the BM is notified, so F can be reused
 - if P has been modified, the BM is notified and propagates the changes in the BP to the disk

Buffer Management

- BM maintains 2 variables for every frame F
 - pin_count
 - number of current users (requested the page in F but haven't released it yet)
 - only frames with pin_count = 0 can be chosen as replacement frames
 - dirty
 - boolean value indicating whether the page in F has been changed since it was brought into F

Buffer Management

- initially, $\text{pin_count} = 0$, $\text{dirty} = \text{off}$, $\forall F \in \text{BP}$
- L asks for a page P; the BM:
 1. checks whether page P is in a frame F in BP; if so, $\text{pin_count}(F)++$;
otherwise:
 - a. BM chooses a frame FR for replacement
 - if the BP contains multiple frames with $\text{pin_count} = 0$, one frame is chosen according to the BM's replacement policy
 - $\text{pin_count}(\text{FR})++$;
 - b. if $\text{dirty}(\text{FR}) = \text{on}$, BM writes the page in FR to disk
 - c. BM reads page P in frame FR
 2. the BM returns L the address of the BP frame that contains P

Buffer Management

- replacement policies
 - *Least Recently Used (LRU)*
 - *Most Recently Used (MRU)*
 - *random*
 - ...

Writing Objects

- transaction T changes object O (in the BP)
- *steal* approach
 - T's changes can be written to disk while T is in progress
 - i.e., when another transaction T2 needs a page and the BM chooses as a replacement frame the frame that contains O's changes
- *no-steal* approach
 - T's changes cannot be written to disk before T commits
- *no-steal* approach
 - advantage
 - changes of aborted transactions don't have to be undone
 - drawback
 - assumption: all pages modified by active transactions can fit in the BP

Writing Objects

- transaction T changes object O (in the BP)
- *force* approach
 - T's changes are immediately forced to disk when T commits
- *no-force* approach
 - T's changes are not forced to disk when T commits
- *force* approach
 - advantage
 - actions of committed transactions don't have to be redone
 - drawback
 - can result in excessive I/O
- *steal, no-force* approach – used by most systems

ARIES

- recovery algorithm
- steal, no-force
- three phases
 - analysis
 - redo
 - undo
- fundamental principle - Write-Ahead Logging
 - a change to an object O is first recorded in the log (e.g., in log record LR)
 - LR must be written to disk before the change to O is written to disk

ARIES

- analysis
 - determine active transactions at the time of the crash
 - determine dirty pages, i.e., pages in BP whose changes have not been written to disk
- redo
 - reapply all changes (starting from a certain record in the log), i.e., bring the DB to the state it was in when the crash occurred
- undo
 - undo changes of uncommitted transactions

ARIES

- example

LSN	Log
1	T1 writes P1
2	T2 writes P2
3	T2 commit
4	T2 end
5	T3 writes P3
6	T3 writes P2
crash & restart	

- analysis

- active transactions at crash time: T1, T3 (to be undone)
- committed transactions: T2 (its effects must persist)
- potentially dirty pages: P1, P2, P3

ARIES

- example

LSN	Log
1	T1 writes P1
2	T2 writes P2
3	T2 commit
4	T2 end
5	T3 writes P3
6	T3 writes P2
crash & restart	

- redo
 - reapply all changes in order (1, 2, ...)

ARIES

- example

LSN	Log
1	T1 writes P1
2	T2 writes P2
3	T2 commit
4	T2 end
5	T3 writes P3
6	T3 writes P2
crash & restart	

- undo
 - undo changes of T1 and T3 in reverse order (6, 5, ...)

The Log

- journal – history of actions executed by the DBMS
- records are added to the end of the log
- *stable storage*
 - ensures the “durability” of the log
 - keep ≥ 2 copies of the log on different disks (locations)
- *log tail*
 - most recent fragment of the log
 - kept in main memory
 - periodically forced to stable storage

The Log

- Log Sequence Number (LSN)
 - unique id for every log record
 - monotonically increasing
 - e.g., address of 1st byte of log record
- pageLSN on page P
 - the LSN of the most recent log record that describes a change to P
- log record – fields
 - prevLSN – a transaction's log records are maintained as a linked list
 - transID – id of the corresponding transaction
 - type – type of the log record

The Log

- each of the following actions results in a log record being written
 - update page
 - commit
 - abort
 - end
 - undo an update
- update page P
 - add an *update type* log record ULR to the log tail (with LSN_{ULR})
 - $pageLSN(P) := LSN_{ULR}$
- transaction T commits
 - add a *commit type* log record CoLR to the log tail
 - force log tail to stable storage (including CoLR)

The Log

- actions
 - transaction T commits
 - complete subsequent actions (remove T from transaction table)
 - transaction T aborts
 - add an *abort type* log record to the log
 - initiate Undo for T
 - transaction T ends
 - T commits / aborts - complete required actions
 - add an *end type* log record to the log
- obs. committed transaction – a transaction whose commit log record has been written to stable storage

The Log

- actions
 - undoing an update
 - i.e., when the change described in an update log record is undone
 - add a *compensation log record* (CLR) to the log

The Log


- update log record
 - fields
 - pageID
 - id of changed page
 - length
 - change – length (in bytes)
 - offset
 - change – offset
 - before-image
 - value before the change
 - after-image
 - value after the change
 - can be used to undo / redo the change

The Log

- compensation log record
 - let U be an update log record describing an update of transaction T
 - let C be the compensation log record for U, i.e., C describes the action taken to undo the changes described by U
 - C has a field named undoNextLSN
 - the LSN of the next log record to be undone for T
 - set to the value of prevLSN in U

The Log

- example



prevLSN	transID	type	pageID	length	offset	before-image	after-image
	T10	update	P100	2	10	AB	CD
	T15	update	P2	2	10	YW	ZA
	T15	update	P100	2	9	EC	YW
	T10	update	P10	2	10	JH	AB

log

- undo T10's update to P10

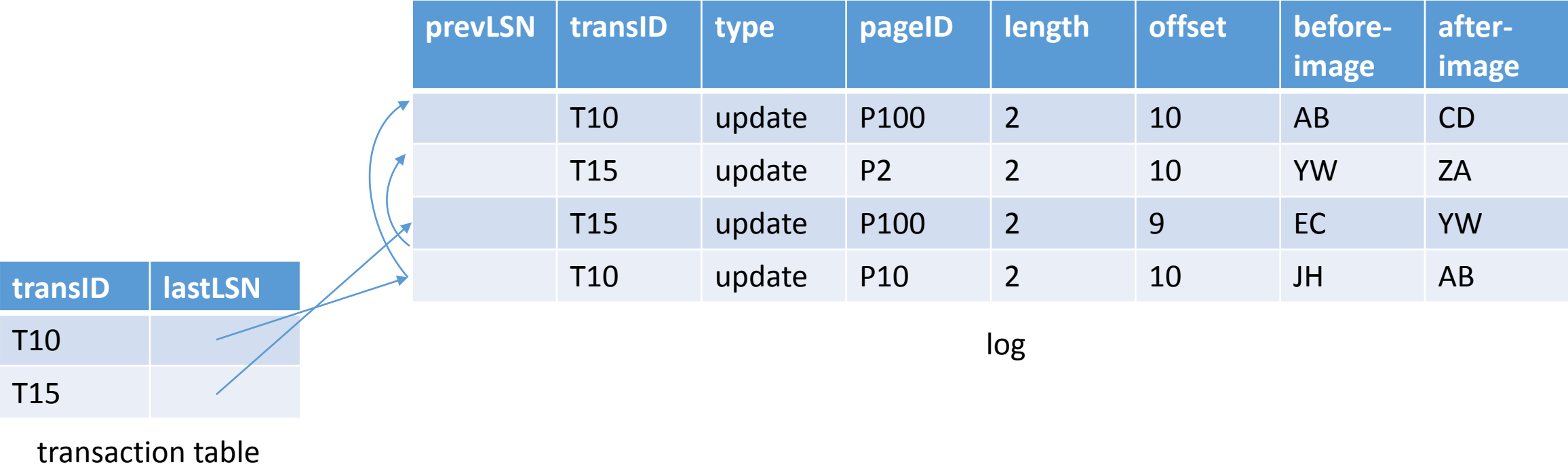
=> CLR with *transID* = T10, *pageID* = P10, *length* = 2, *offset* = 10, *before-image* = JH and *undoNextLSN* = LSN of 1st log record (i.e., the next record that is to be undone for transaction T10)

The Transaction Table and the Dirty Page Table

- important information for the recovery process
- transaction table
 - 1 entry / active transaction
 - fields
 - transID
 - lastLSN
 - LSN of the most recent log record for the transaction
 - status
 - in progress / committed / aborted

The Transaction Table and the Dirty Page Table

- example (the status is not displayed)

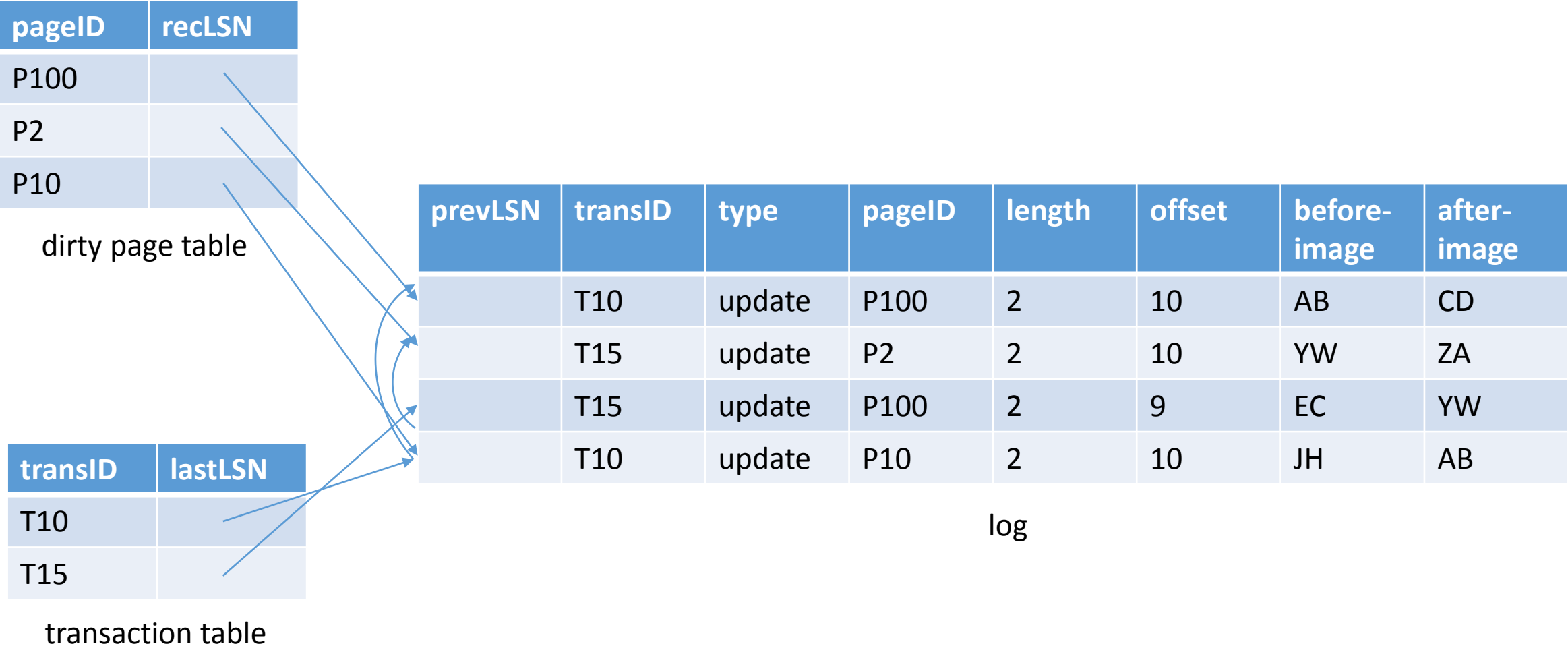


The Transaction Table and the Dirty Page Table

- dirty page table
 - 1 entry / dirty page in the Buffer Pool
 - fields
 - pageID
 - recLSN
 - the LSN of the 1st log record that dirtied the page

The Transaction Table and the Dirty Page Table

- example



Checkpointing

- objective
 - reduce the amount of work performed by the system when it comes back up after a crash
- checkpoint
 - snapshot of the state of the DBMS
 - taken periodically
- 3 steps
 - write a begin_checkpoint record (it indicates when the checkpoint starts; let its LSN be LSN_{BCK})
 - write an end_checkpoint record
 - it includes the current Transaction Table and the current Dirty Page Table

Checkpointing

- 3 steps
 - after the end_checkpoint record is written to stable storage
 - write a master record to a known place on stable storage
 - it includes LSN_{BCK}
- crash -> restart -> system looks for the most recent checkpoint
- normal execution begins with a checkpoint with an empty Transaction Table and an empty Dirty Page Table

References

- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [Le99] LEVENE, M., LOIZOU, G., A Guided Tour of Relational Databases and Beyond, Springer, 1999
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007,
<http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html>
- [Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010, <http://codex.cs.yale.edu/avi/db-book/>
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems,
<http://infolab.stanford.edu/~ullman/fcdb.html>