Project6

Transaction Logging & Three-Pass Recovery

Project Hierarchy

- Your project hierarchy should be like this.
  - Your_hconnect_repo
    - project6
      - include/
      - lib/
      - Makefile
      - src/
- If your Makefile doesn’t make libbpt.a library file at the exact path, you’ll get zero score. (your_hconnect_repo/project6/lib/libbpt.a)

Goal

- The goal of this project is to **implement a perfect three-pass recovery algorithm** in your DBMS. (in your lecture 23~25)
  - Analysis pass – Redo pass – Undo pass
  - Consider redo for fast recovery
  - Compensate Log Record in abort / undo pass
  - ‘Undo Next Sequence’ to make progress despite repeated failures

Log Manager

- Implement your own log manager that can support ‘Atomicity’ and ‘Durability’.
- Your log manager should satisfy these properties.
  - No force (REDO) & Steal (UNDO) policy
  - Write Ahead Logging (WAL)
  - Recovery when initializing whole DBMS

- **You just consider in ‘update case’. Transactions only operate find and update.**

Project Specification

- Your library (libbpt.a) should provide those API services.
- Additional Role of Transaction APIs
  - **int trx_begin(void);**
    - Allocate a transaction structure and initialize it.
    - Return a unique transaction id (>=1) if success, otherwise return 0.
    - **You must provide the transaction id by increasing it by 1. (1, 2, 3, 4 ….)**
  - **int trx_commit(int trx_id);**
    - Return the committed transaction if success, otherwise return 0.
    - Clean up the transaction with the given trx_id and its related information that has been used in the lock manager. (Shrinking phase of strict 2PL)
    - **User can get a response once all modifications of the transaction are flushed from log buffer to a log file.**
    - **If the user gets a successful return, that means your database can recover committed transaction after system crash.**
  - **int trx_abort(int trx_id);**
    - Return the aborted transaction id if success, otherwise return 0.
    - **All affected modifications should be canceled and return to the old state.**

Project Specification

- Your library (libbpt.a) should provide those API services.

1. **int init_db (int buf_num, int flag, int log_num, char* log_path, char* logmsg_path);**
   - If success, return 0, Otherwise, return a non-zero value.
   - Do recovery after initialization in this function. (DBMS initialization -> Analysis – Redo – Undo)
   - **Log file will be made using log_path.**
   - **Log message file will be made using logmsg_path.**
   - **flag is needed for recovery test, 0 means normal recovery protocol, 1 means REDO CRASH, 2 means UNDO CRASH.**
   - **log_num is needed for REDO/UNDO CRASH, which means the function must return 0 after the number of logs is processed.**
2. **int open_table (char * pathname);**
   - **We limit the file name format as "DATA[NUM]" (For example, there should be data files named like "DATA1", "DATA2", …)**
   - **Return value that indicates the table id should be NUM. (That means, data file whose file name is "DATA3" has its table id as 3 from now on. And maximum table number is set to 10).**
3. **int db_insert (int table_id, int64_t key, char * value);**
4. **int db_find (int table_id, int64_t key, char* ret_val, int trx_id);**
5. **int db_delete (int table_id, int64_t key);**
6. **int db_update(int table_id, int64_t key, char* value, int trx_id);**
7. **int close_table(int table_id);**
8. **int shutdown_db(void);**

Project Specification

- Your library (libbpt.a) should provide those API services.
  - **int init_db (int buf_num, int flag, int log_num, char* log_path, char* logmsg_path);**
    - Do recovery after database initialization. (DBMS initialization - Analysis – Redo – Undo)
    - **Log file and log message file will be named through log_path and logmsg_path.**
    - **flag is needed for recovery test, 0 means normal recovery protocol, 1 means REDO CRASH, 2 means UNDO CRASH.**
    - **You must flush log buffer and all dirty pages in the buffer pool before return (even if returning for REDO/UNDO CRASH case).**
    - **log_num is needed for REDO/UNDO CRASH, which means the function must return 0 after the number of logs is processed.**

  - **REDO/UNDO CRASH**
    - **When the recovery phase occurs in init_db, you need to make an arbitrary crash(return 0 during recovery) for the recovery test.**
    - **So if the flag of init_db is a non-zero value, you must return 0 after when log_num of logs are processed.**

      (ex) if flag = 1, log_num = 100, return 0 after 100th log redo is processed.

      if flag = 2, log_num = 100, return 0 after 100th log undo is processed.)
    - **We will test your project6 through these parameters.**

Project Specification

- **Log Process Message**
  - You should print message to log message file when your DBMS proceeds logs under this format.
  - Log messages must be written to log message file
  - Analysis Phase:
    - fprintf(fp, “[ANALYSIS] Analysis pass start\n”);
    - fprintf(fp, “[ANALYSIS] Analysis success. Winner: %d %d .., Loser: %d %d ….\n”, winners, losers);
  - Redo(Undo) Phase
    - fprintf(fp, “[REDO(UNDO)] Redo(Undo) pass start\n”);
    - Begin: fprintf(fp, “LSN %lu [BEGIN] Transaction id %d\n”, lsn, trx_id);
    - Update: fprintf(fp, “LSN %lu [UPDATE] Transaction id %d redo(undo) apply\n”, lsn, trx_id);
    - Commit: fprintf(fp, “LSN %lu [COMMIT] Transaction id %d\n”, lsn, trx_id);
    - Rollback: fprintf(fp, “LSN %lu [ROLLBACK] Transaction id %d\n”, lsn, trx_id);
    - Compensate: fprintf(fp, “LSN %lu [CLR] next undo lsn %lu\n”, lsn, next_undo_lsn);
    - Consider-redo: fprintf(fp, “LSN %lu [CONSIDER-REDO] Transaction id %d\n”, lsn, trx_id);
    - fprintf(fp, “[REDO(UNDO)] Redo(Undo) pass end\n”);

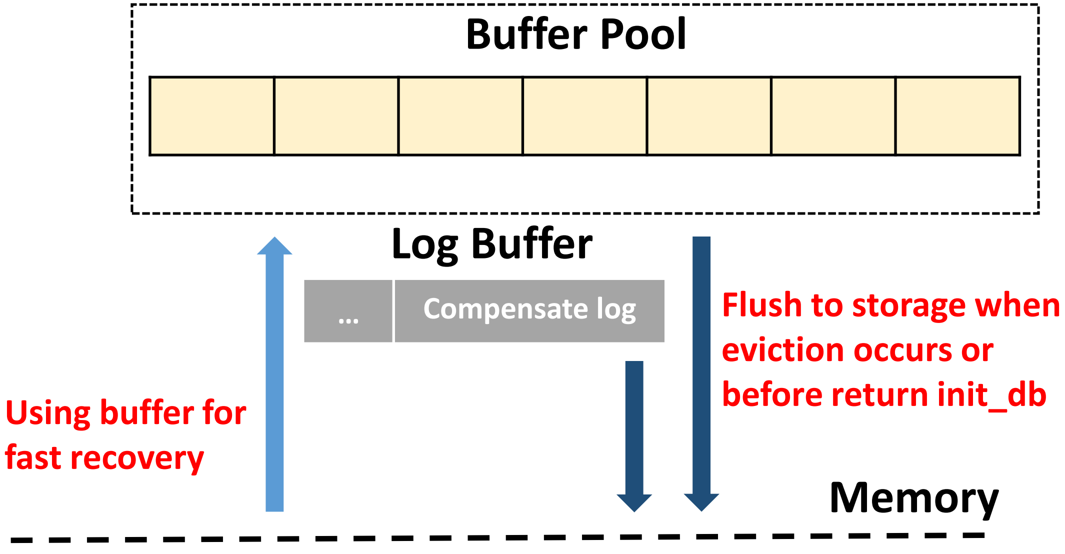
Recovery Test Example

System

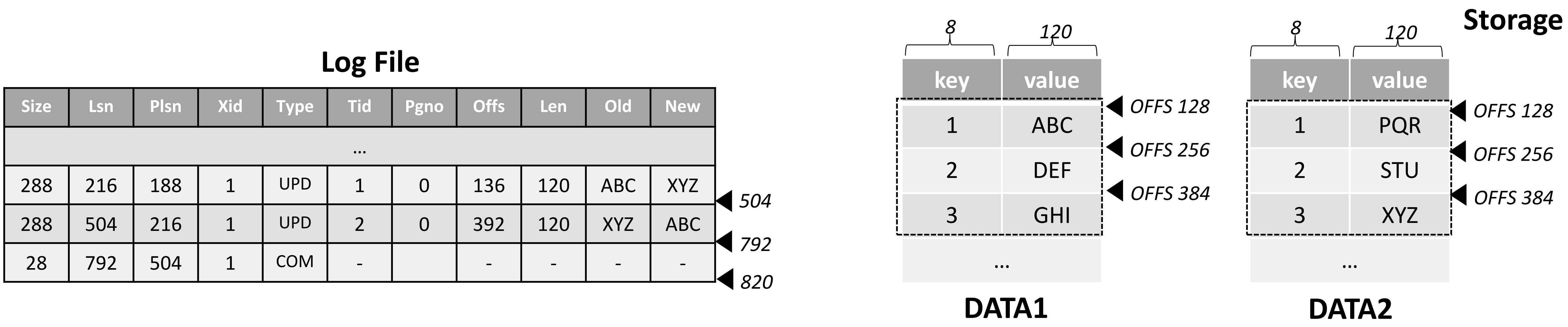
```
int ret = init_db(1000, 1, 100, “logfile.data”, “logmsg.txt”);
```

When your recovered data is stored to disk once, your DBMS must not replay the log.

The log should be processed as consider-redo, and it will make your recovery faster when you restart DBMS after a crash.



Buffer Pool
Log Buffer
...
Compensate log
Using buffer for fast recovery
Flush to storage when eviction occurs or before return init_db
Memory



Log File

Size	Lsn	Plsn	Xid	Type	Tid	Pgno	Offs	Len	Old	New
…										
288	216	188	1	UPD	1	0	136	120	ABC	XYZ
288	504	216	1	UPD	2	0	392	120	XYZ	ABC
28	792	504	1	COM	-		-	-	-	-

DATA1

key	value
1	ABC
2	DEF
3	GHI
…	

DATA2

key	value
1	PQR
2	STU
3	XYZ
…	

Recovery Test Example

We will do recovery test using log message file. Here is a simple example.

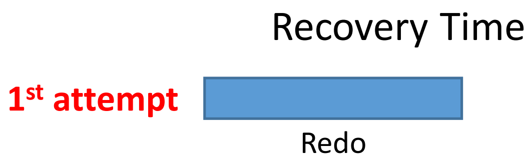
You must print the end offset of the log record whether you use LSN as start offset (LSN + log record size)

You never mind about printing the next undo LSN of compensate log. Print what you choose(start/end offset).

1. REDO CRASH case <init_db(1000, 1, 100, “logfile.data”, “logmsg.txt”);>

```
[ANALYSIS] Analysis pass start
[ANALYSIS] Analysis success. Winner: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20,  Loser: 16 21
[REDO] Redo pass start
LSN 28 [BEGIN] Transaction id 1
LSN 316 [UPDATE] Transaction id 1 redo apply
LSN 344 [COMMIT] Transaction id 1
…
LSN 13132 [UPDATE] Transaction id 17 redo apply                  (99th redo log)
LSN 13160 [COMMIT] Transaction id 17                             (100th redo log)
```

Return init_db 0 (Crash), restart DBMS (continue on next slide)

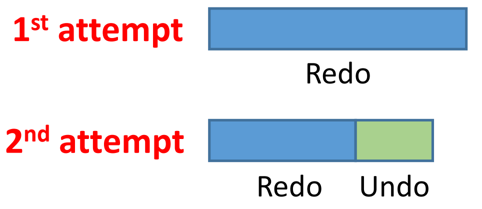


Recovery Test Example

```
2. UNDO CRASH case <init_db(1000, 2, 100, “logfile.data”, “logmsg.txt”);>
[ANALYSIS] Analysis pass start
[ANALYSIS] Analysis success. Winner: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20,  Loser: 16 21
[REDO] Redo pass start
LSN 28 [BEGIN] Transaction id 1
LSN 316 [UPDATE] Transaction id 1 redo apply
LSN 344 [COMMIT] Transaction id 1
…
LSN 13132 [CONSIDER-REDO] Transaction id 17                    (99th redo log)
LSN 13160 [COMMIT] Transaction id 17                           (100th redo log)
…
LSN 72100 [UPDATE] Transaction id 21 redo apply
[REDO] Redo pass end
[UNDO] Undo pass start
LSN 72100 [UPDATE] Transaction id 21 undo apply
…
LSN 21640 [UPDATE] Transaction id 16 undo apply                (100th undo log)
```

Return init_db 0 (Crash), restart DBMS (continue on next slide)

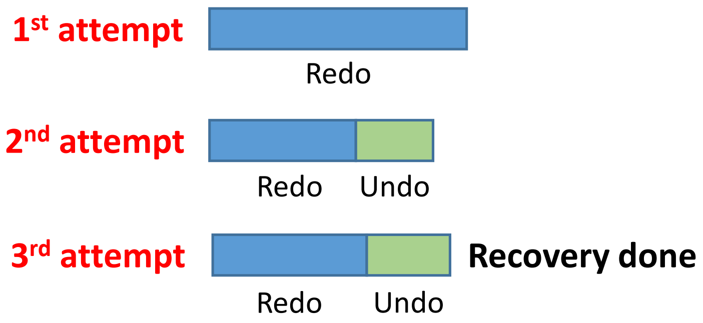
Recovery Time



Recovery Test Example

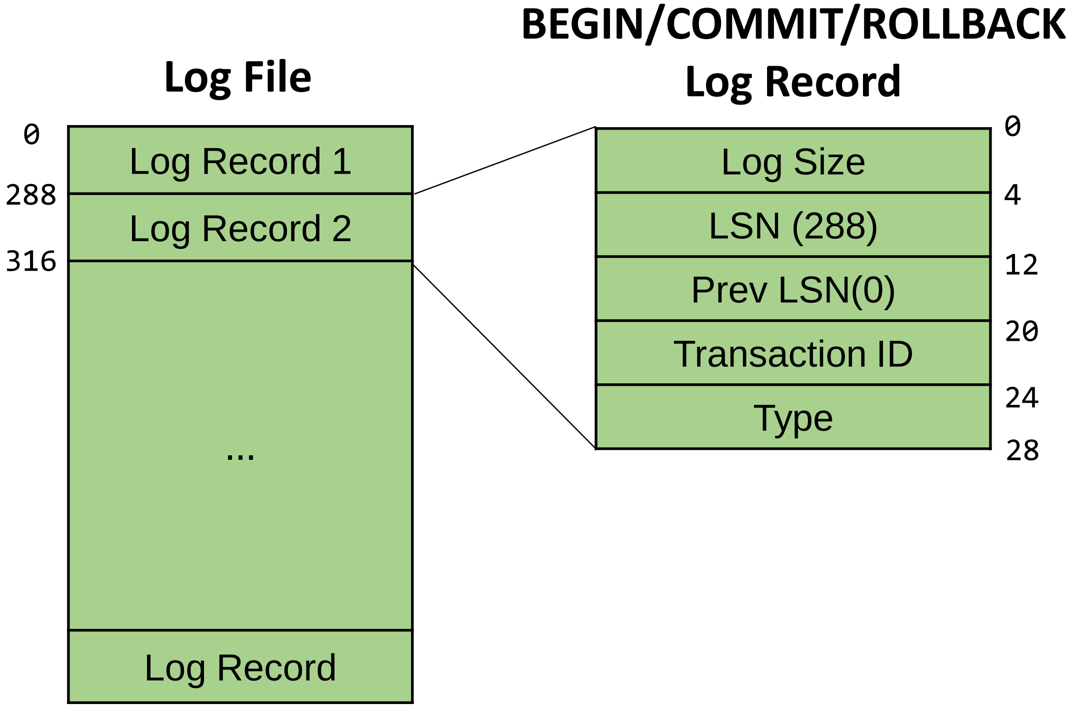
```
3. NORMAL RECOVERY case <init_db(1000, 0, 0, “logfile.data”, “logmsg.txt”);>
[ANALYSIS] Analysis pass start
[ANALYSIS] Analysis success. Winner: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20,  Loser: 16 21
[REDO] Redo pass start
LSN 28 [BEGIN] Transaction id 1
…
LSN 72100 [CONSIDER-REDO] Transaction id 21
LSN 78220 [CONSIDER-REDO] Transaction id 21
…
LSN 80236 [CONSIDER-REDO] Transaction id 16
[REDO] Redo pass end
[UNDO] Undo pass start
LSN 21360 [UPDATE] Transaction id 16 undo apply
…
LSN 18714 [UPDATE] Transaction id 16 undo apply
[UNDO] Undo pass end
All recovery phase is done. Then check recovered data.
open_table(…
```

Recovery Time



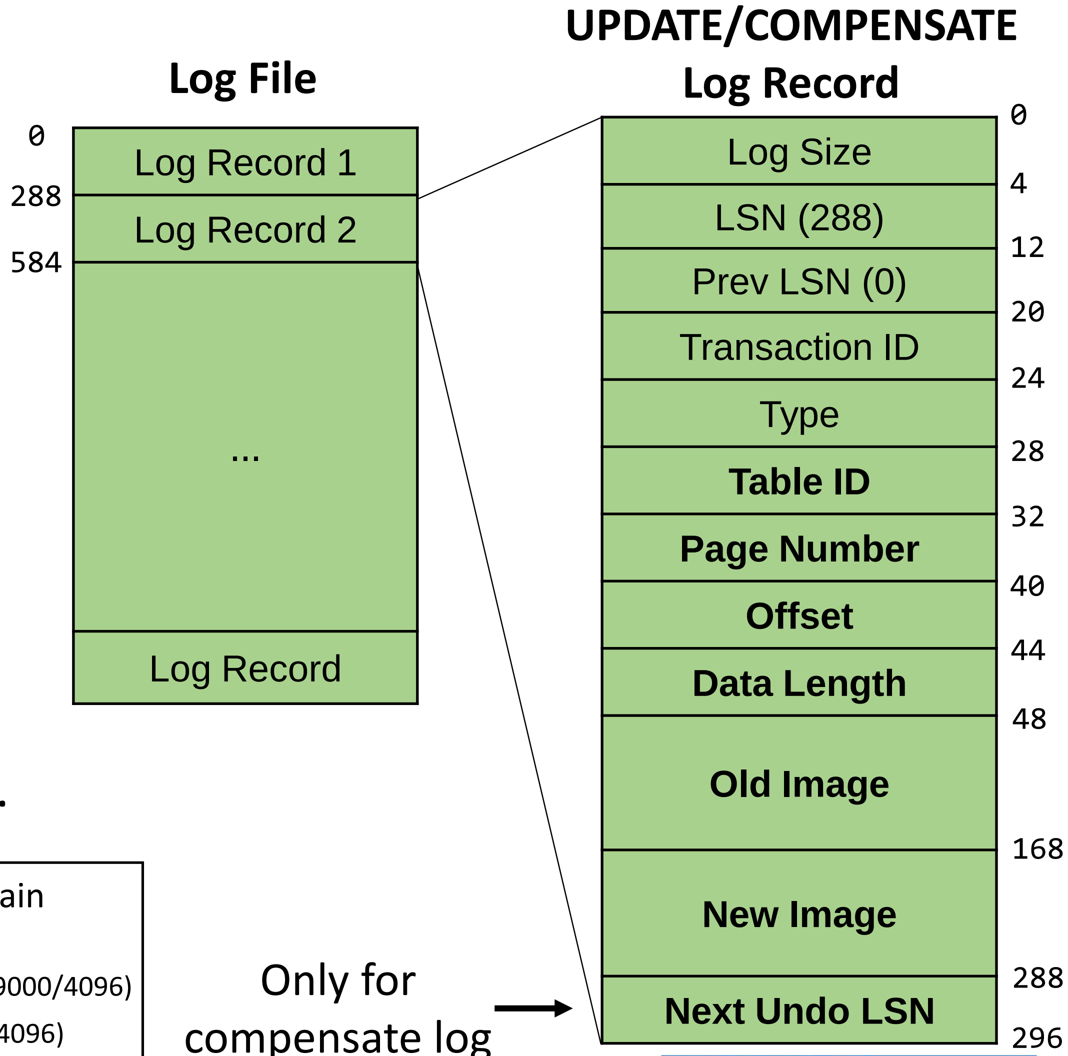
Log File

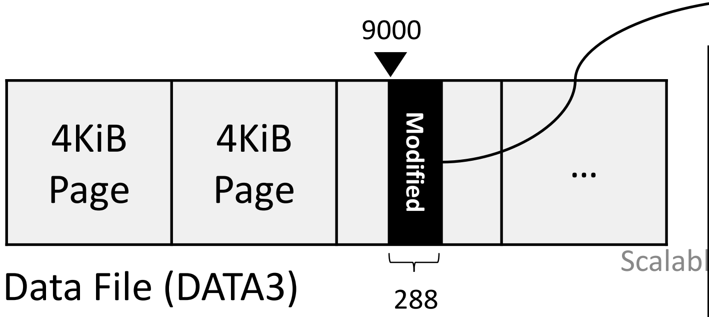
- Log file is a sequence of log records.
- Log record is consisted of
  - LSN: Start/end offset of a current log record.
  - Prev LSN: LSN of the previous log record written by same Transaction ID.
  - Transaction ID: Indicates the transaction that triggers the current log record.
  - Type: The type of current log record.
    - BEGIN (0)
    - UPDATE (1)
    - COMMIT (2)
    - ROLLBACK (3)
    - COMPENSATE (4)



Log File

- Log file is a sequence of log records.
- Log record is consisted of
  - Table ID: Indicates the data file. (data file name should be like “DATA[Table ID]”)
  - Page Number: Page that contains the modified area.
  - Offset: Start offset of the modified area within a page.
  - Data Length: The length of the modified area.
  - Old Image: Old contents of the modified area.
  - New Image: New contents of the modified area.
  - Next Undo LSN: Next undo point for compensate log.



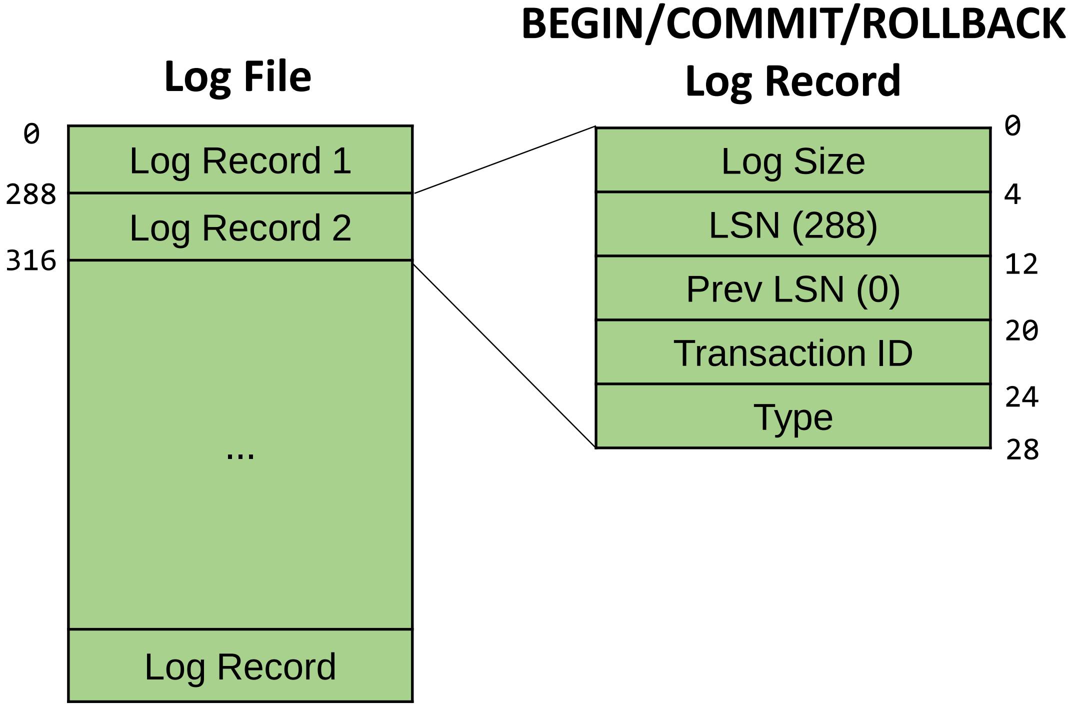


Data File (DATA3)

Log record should contain
- Table ID: 3
- Page Number : 2 (9000/4096)
- Offset: 808 (9000%4096)
- Length: 120
- Old/New Images

Log File

- Log file is a sequence of log records.
- Your Log Record should have sizes of below
  - BEGIN/COMMIT/ROLLBACK : 28 Byte
  - UPDATE : 288 Byte
  - COMPENSATE : 296 Byte



Log Record Format (using LSN as start offset)

BEGIN/COMMIT/ROLLBACK Log Record

Log Size	0
LSN	4
Prev LSN	12
Transaction ID	20
Type	24
	28

UPDATE Log Record

Log Size	0
LSN	4
Prev LSN	12
Transaction ID	20
Type	24
Table ID	28
Page Number	32
Offset	40
Data Length	44
Old Image	48
New Image	168
	288

COMPENSATE Log Record

Log Size	0
LSN	4
Prev LSN	12
Transaction ID	20
Type	24
Table ID	28
Page Number	32
Offset	40
Data Length	44
Old Image	48
New Image	168
Next Undo LSN	288
	296

Log Record Format (using LSN as end offset)

BEGIN/COMMIT/ROLLBACK
Log Record

LSN	0	8
Prev LSN	8	16
Transaction ID	16	20
Type	20	24
Log Size	24	28

UPDATE
Log Record

LSN	0	8
Prev LSN	8	16
Transaction ID	16	20
Type	20	24
Table ID	24	28
Page Number	28	36
Offset	36	40
Data Length	40	44
Old Image	44	164
New Image	164	284
Log Size	284	288

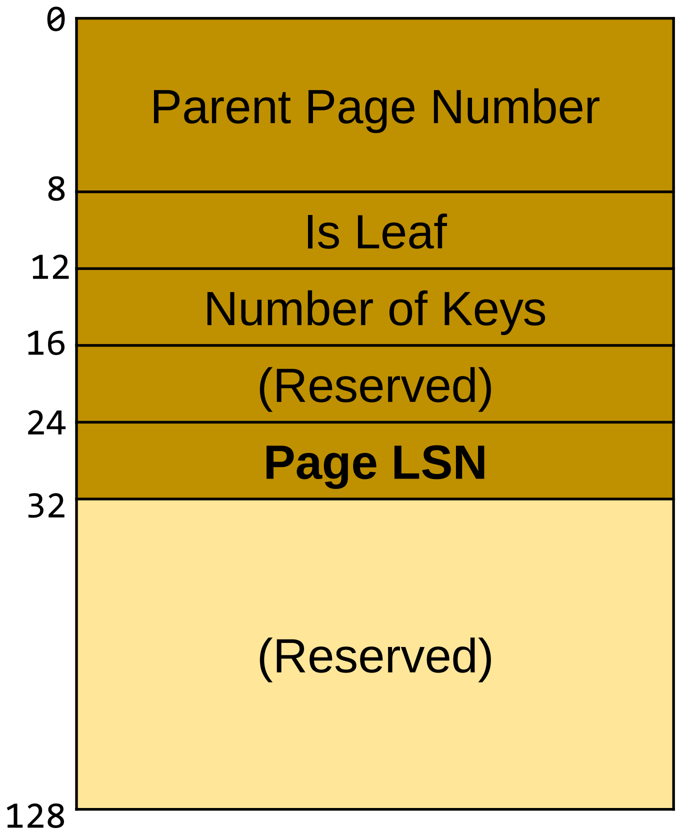
COMPENSATE
Log Record

LSN	0	8
Prev LSN	8	16
Transaction ID	16	20
Type	20	24
Table ID	24	28
Page Number	28	36
Offset	36	40
Data Length	40	44
Old Image	44	164
New Image	164	284
Next Undo LSN	284	292
Log Size	292	296

Page Header Layout

- You should maintain a page LSN information from every on-disk image.
- The page LSN indicates the last updated version of this on-disk page.
- Maintain the page LSN value (8 bytes) located at the page header structure starting from byte offset 24.
- Every page including the header page should maintain that field.

Page Header Layout



0

Parent Page Number

8

Is Leaf

12

Number of Keys

16

(Reserved)

24

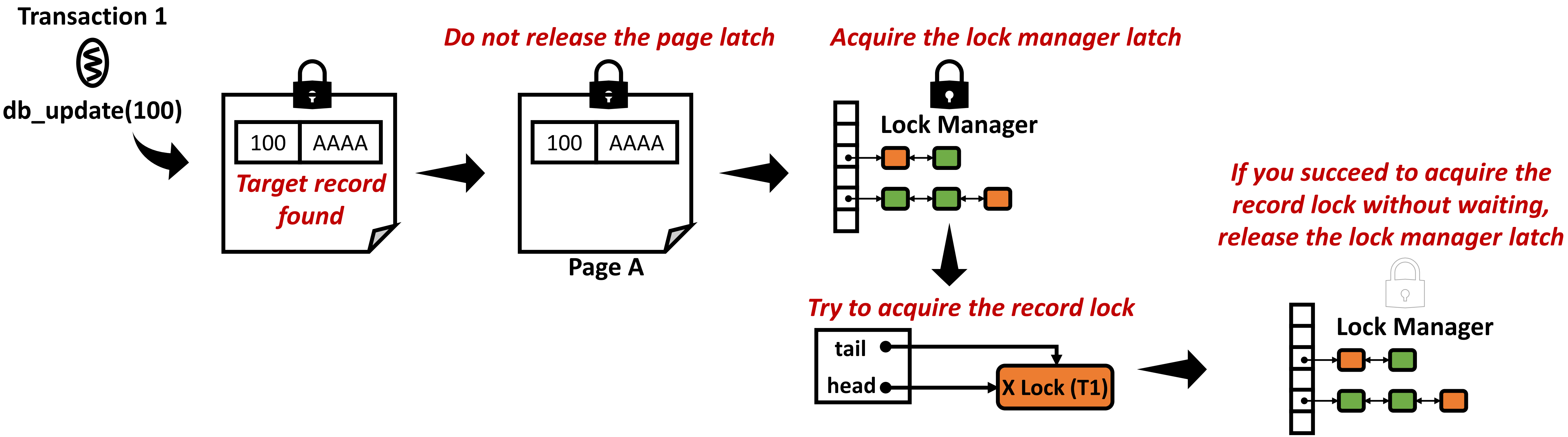
Page LSN

32

(Reserved)

128

Update Latch Sequence with Log Manager



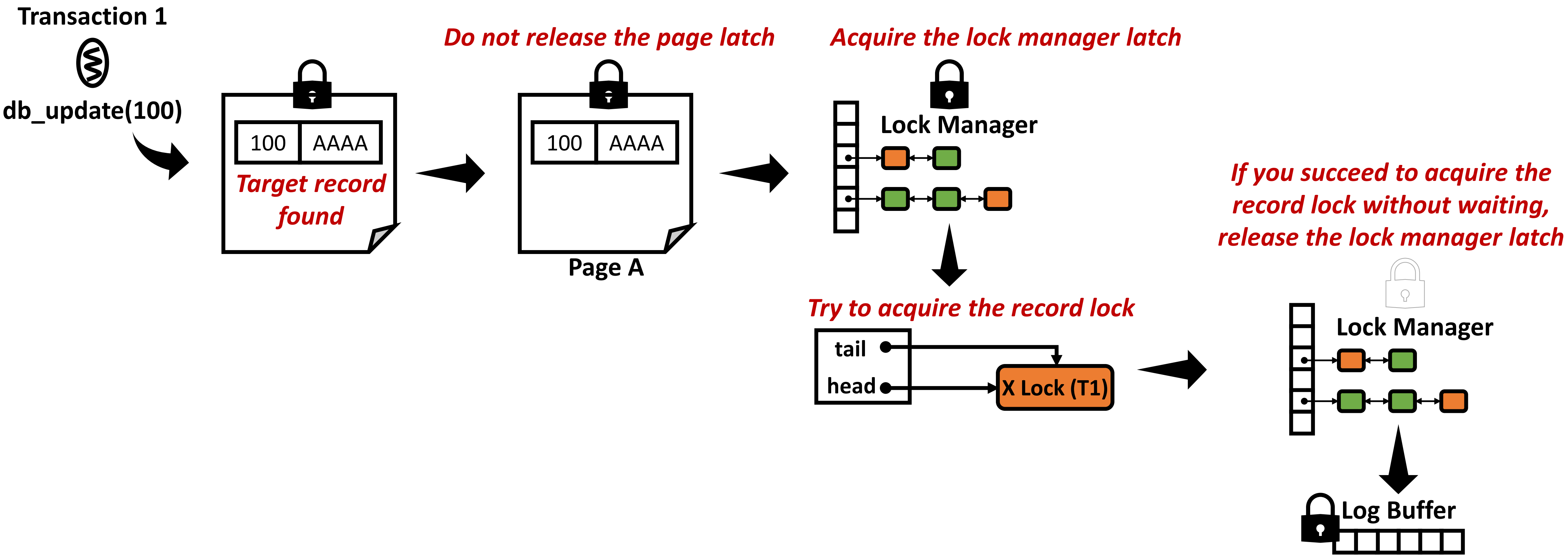
Do not release the page latch

Acquire the lock manager latch

Try to acquire the record lock

If you succeed to acquire the record lock without waiting, release the lock manager latch

Update Latch Sequence with Log Manager



Target record found

Do not release the page latch

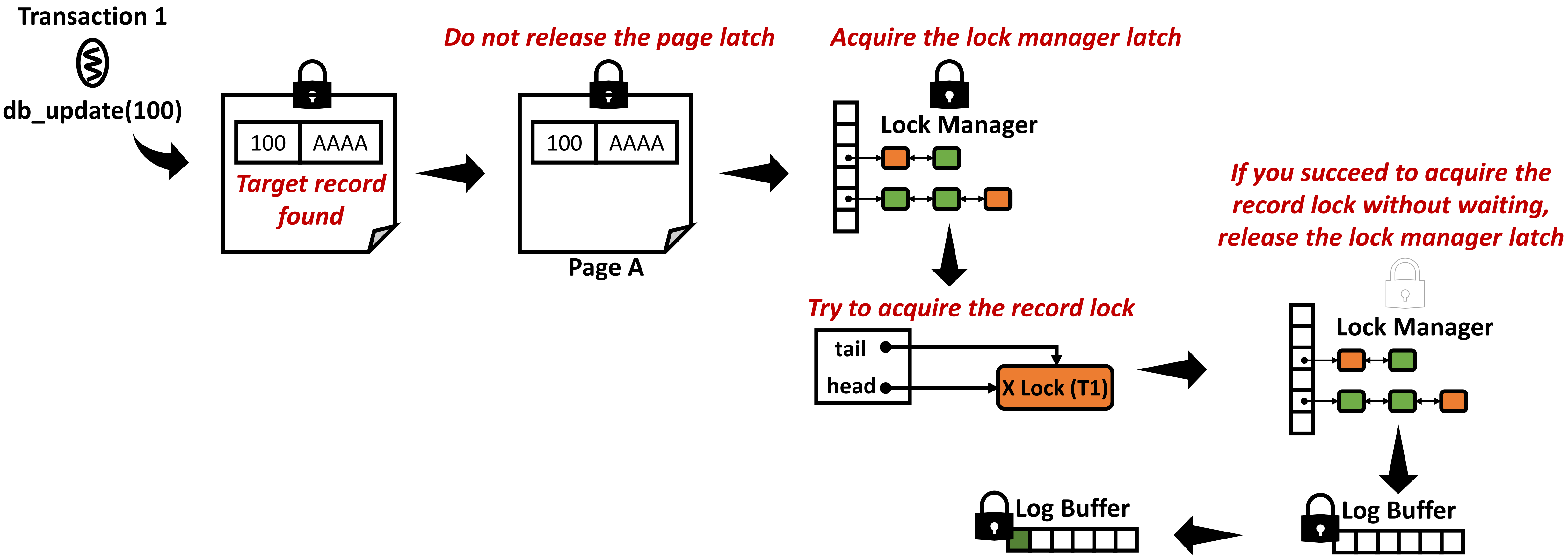
Acquire the lock manager latch

Try to acquire the record lock

If you succeed to acquire the record lock without waiting, release the lock manager latch

Make update log record then acquire log buffer mutex

Update Latch Sequence with Log Manager



Target record found

Do not release the page latch

Acquire the lock manager latch

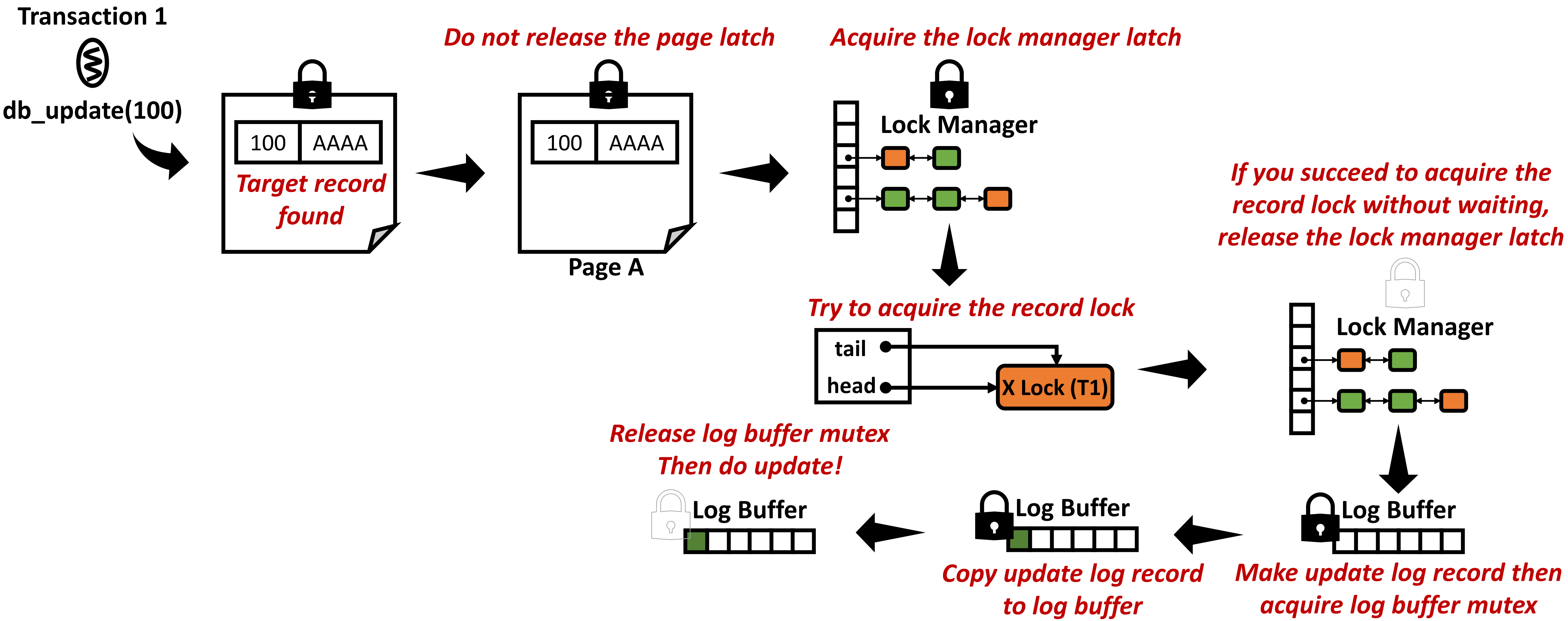
Try to acquire the record lock

If you succeed to acquire the record lock without waiting, release the lock manager latch

Make update log record then acquire log buffer mutex

Copy update log record to log buffer

Update Latch Sequence with Log Manager



Target record found

Do not release the page latch

Acquire the lock manager latch

Try to acquire the record lock

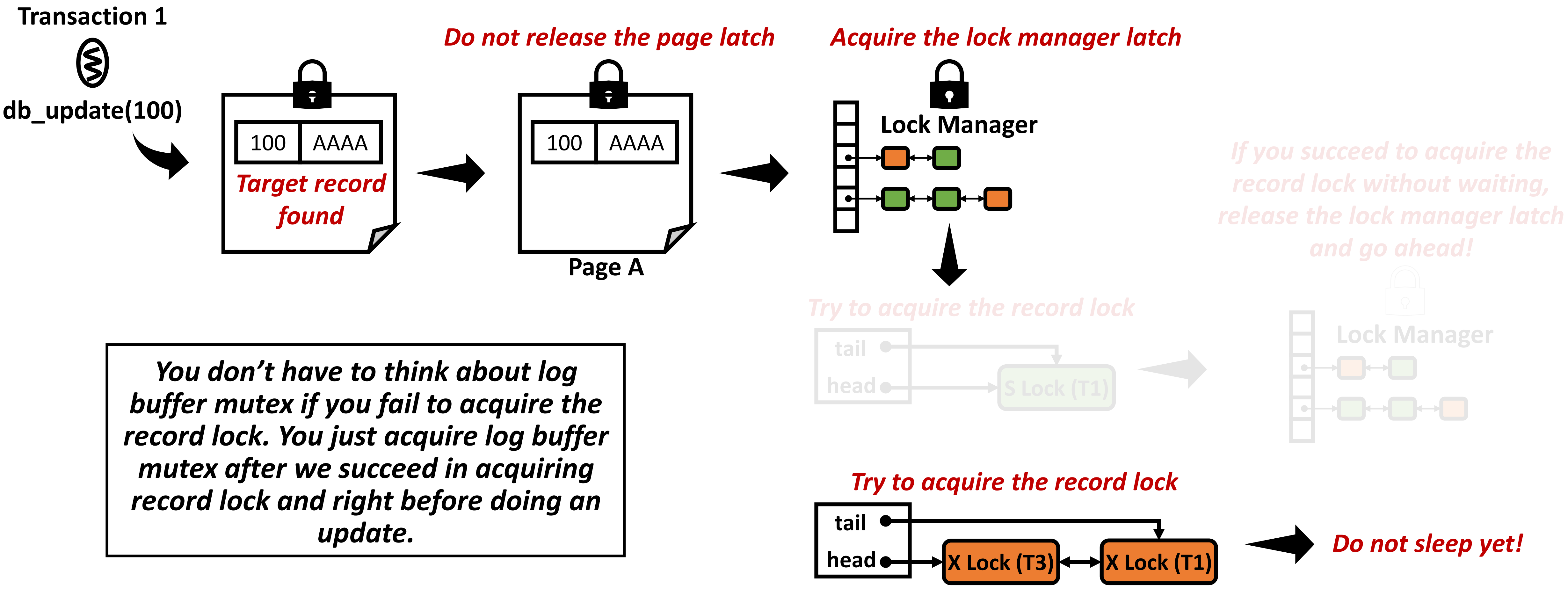
If you succeed to acquire the record lock without waiting, release the lock manager latch

Make update log record then acquire log buffer mutex

Copy update log record to log buffer

Release log buffer mutex
Then do update!

Update Latch Sequence with Log Manager



Target record found

Do not release the page latch

Acquire the lock manager latch

Try to acquire the record lock

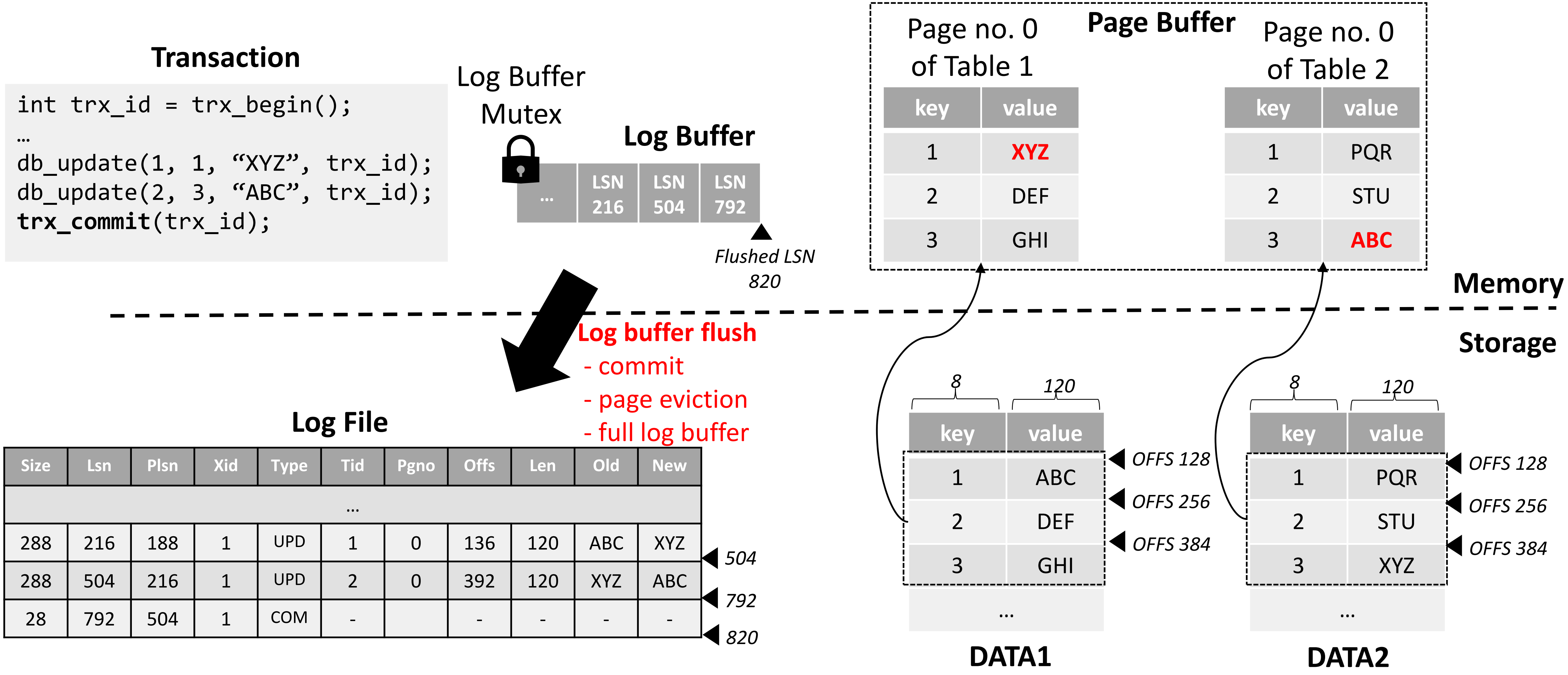
If you succeed to acquire the record lock without waiting, release the lock manager latch and go ahead!

Try to acquire the record lock

Do not sleep yet!

You don’t have to think about log buffer mutex if you fail to acquire the record lock. You just acquire log buffer mutex after we succeed in acquiring record lock and right before doing an update.

Project Example



Transaction

```
int trx_id = trx_begin();
…
db_update(1, 1, “XYZ”, trx_id);
db_update(2, 3, “ABC”, trx_id);
trx_commit(trx_id);
```

Log buffer flush
- commit
- page eviction
- full log buffer

Page Buffer

Page no. 0 of Table 1

key	value
1	XYZ
2	DEF
3	GHI

Page no. 0 of Table 2

key	value
1	PQR
2	STU
3	ABC

Log File

Size	Lsn	Plsn	Xid	Type	Tid	Pgno	Offs	Len	Old	New
…										
288	216	188	1	UPD	1	0	136	120	ABC	XYZ
288	504	216	1	UPD	2	0	392	120	XYZ	ABC
28	792	504	1	COM	-		-	-	-	-

DATA1

key	value
1	ABC
2	DEF
3	GHI
…	

DATA2

key	value
1	PQR
2	STU
3	XYZ
…	

Project Specification

- You should implement ARIES based recovery that you learned from the lecture but,
  - we don’t have to consider double write since we assume that torn page write would not occur.
  - Checkpoint is not considered from this project.

Project Specification

- We will check the correctness of recovery by executing concurrent transactions and triggering system crash like below.

Example case)

```
int ret = init_db(1000, 1, 100, “logfile.data”, “logmsg.txt”);
exit() // system crash
```

or

```
void *thread_func(void *data) {
  int trx_id = trx_begin();
  db_update(1, 3, “XYZ”, trx_id);
  trx_commit(trx_id);
  int new_id = trx_begin();
  db_update(1, 2, “XXX”, new_id);
  exit() // system crash
}

int main (int argc, char** argv){
  int ret = init_db(1000, 0, 0, “logfile.data”, “logmsg.txt”);
  pthread_create(…
  …
}
```

Submission

- **Wiki Requirement**
  - Your Gitlab Wiki must be written under this guideline
    - What is your work for Analysis Pass?
    - What is your work for Redo Pass?
    - What is your work for Undo Pass?
    - Simple test result
- Deadline: 12/17 23:59
- We will only score your commit before the deadline, and your submission after the deadline will not be accepted.

Thank you