Project6

Transaction Logging & Three-Pass Recovery

Project Hierarchy

- Your project hierarchy should be like this:

  “your_repo/project6/db_project”

```
ktlee20@multicore-36:~/TA/2021_DB/projects_2021$ ls
project1  project2  project3  project4  project5  project6
ktlee20@multicore-36:~/TA/2021_DB/projects_2021$ tree project6/
project6/
└── db_project
    ├── CMakeLists.txt
    ├── db
    │   ├── CMakeLists.txt
    │   ├── include
    │   │   ├── bpt.h
    │   │   ├── buffer.h
    │   │   ├── file.h
    │   │   ├── log.h
    │   │   └── trx.h
    │   └── src
    │       ├── bpt.cc
    │       ├── buffer.cc
    │       ├── file.cc
    │       ├── log.cc
    │       └── trx.cc
    ├── DbConfig.h.in
    ├── main.cc
    └── test
        ├── basic_test.cc
        ├── CMakeLists.txt
        └── file_test.cc

5 directories, 17 files
```

- Follow the directory structure given above.
- You may use different names for the source/header files and add new files/directories if necessary.

Goal

- The goal of this project is to **implement ARIES recovery algorithms** in your DBMS.
  - 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 log manager to support the ‘Atomicity’ and ‘Durability.’
- Your log manager should satisfy these properties.
  - No force (REDO) & Steal (UNDO) policy
  - Write Ahead Logging (WAL)
  - Recovery when initializing the whole DBMS

- **Consider update/find cases only. That is, transactions will only perform db_find() and db_update().**
- **To prevent changes of the b+tree, db_update() function will not change the size of the existing records.**

Project Specification

- Your library (libdb.a) should provide these 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 successful; otherwise, return 0.
    - **You must provide the transaction ids by incrementing it by 1. (1, 2, 3, 4 ….)**
  - **int trx_commit(int trx_id);**
    - Return the committed transaction id if successful; 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)
    - **Users can get a response once all modifications of the transaction are flushed from the log buffer to a log file.**
    - **If the user receives a triumphant return, it means that your database can recover committed transactions even after a system crash.**
  - **int trx_abort(int trx_id);**
    - Return the aborted transaction id if successful; otherwise, return 0.
    - **All modifications should be canceled and be returned to the old state.**

Project Specification

➢Your library (libdb.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.
   - Perform recovery *within* this function, after the initialization phase. (DBMS initialization -> Analysis – Redo – Undo)
   - **Log file will be made using log_path.**
   - **Log message file will be made using logmsg_path.**
   - **flag should be provided for the recovery test, use 0 for normal recovery protocol, 1 for REDO CRASH, 2 for UNDO CRASH.**
   - **log_num is needed for REDO/UNDO CRASH, the function must return 0 after processing the provided number of logs.**
2. **int64_t open_table (char * pathname);**
   - **We limit the file name format to "DATA[NUM]" (e.g., data files should be named like "DATA1", "DATA2", …)**
   - **The return value that indicates the table id should be 'NUM'. (e.g., a data file with the name "DATA3" should have a table id of 3)**
3. **int db_insert (int64_t table_id, int64_t key, char * value, uint16_t val_size);**
4. **int db_find (int64_t table_id, int64_t key, char* ret_val, uint16_t* val_size, int trx_id);**
5. **int db_delete (int64_t table_id, int64_t key);**
6. **int db_update(int64_t table_id, int64_t key, char* value, uint16_t new_val_size, uint16_t* old_val_size, int trx_id);**
7. **int shutdown_db(void);**

Project Specification

➢Your library (libdb.a) should provide these API services.

- **int init_db (int buf_num, int flag, int log_num, char* log_path, char* logmsg_path);**
  - **You must flush log buffer and all dirty pages in the buffer pool before return (even if returning for REDO/UNDO CRASH case).**

- **CRASH DURING REDO or UNDO PHASES**
  - **You must check if your flag and log_num works well. We plan to make arbitrary crashes for the recovery, so make sure that you handle all the cases.**
  - **If the 'flag' given for init_db() is a non-zero value, you must return 0 after a total of 'log_num' logs are processed.**

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

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

Project Specification

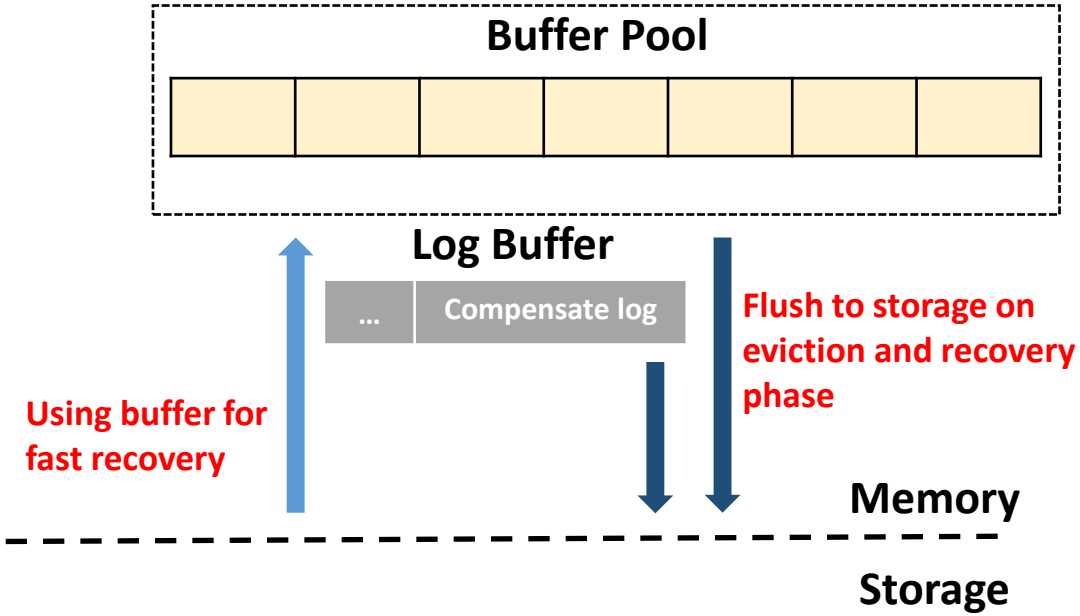
- **Log Process Message**
  - You should print messages under this format to the log message file when your DBMS processes the logs.
  - Log messages must be written to the log message file.
  - Analysis Phase:
    - fprintf(fp, “[ANALYSIS] Analysis pass start\n”);
    - fprintf(fp, “[ANALYSIS] Analysis 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

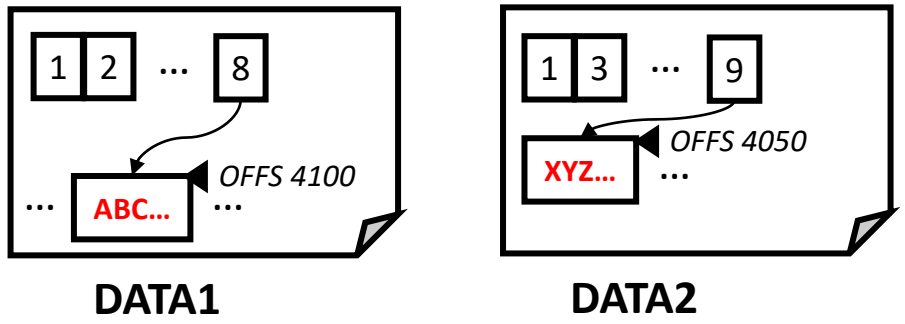
```
int ret = init_db(1000, 1, 100, “logfile.data”, “logmsg.txt”);
```

When your data is successfully recovered and stored on disk once, your DBMS should not replay that log. In short, the corresponding logs should be treated as a consider-redo for faster recovery.



Log File

Size	Lsn	Plsn	Xid	Type	Tid	Pgno	Offs	Len	Old	New	
...											
168	264	236	1	UPD	1	0	4100	60	ABC...	XYZ...	*432*
168	432	264	1	UPD	2	0	4050	60	XYZ...	ABC...	*600*
28	600	432	1	COM	-		-	-	-	-	*628*

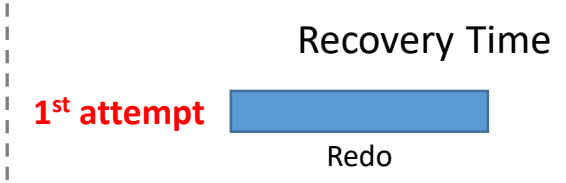


Recovery Test Example

We verify your recovery process using the log message file.
Here is a simple example.

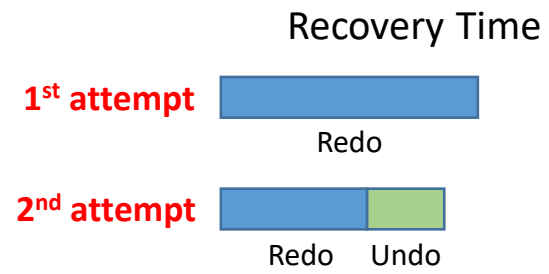
```
1. REDO CRASH case <init_db(1000, 1, 100, “logfile.data”, “logmsg.txt”);>
```

[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)



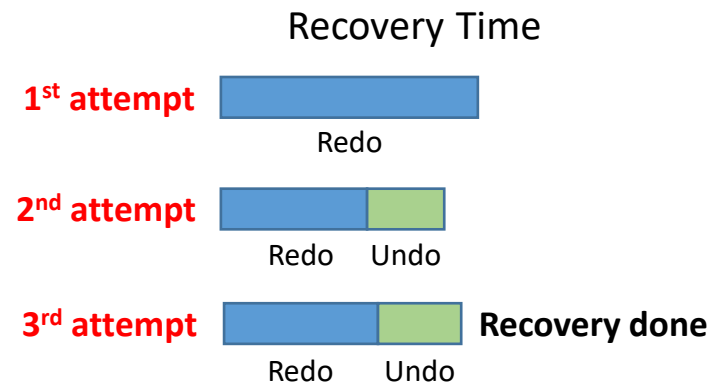
Recovery Test Example

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)



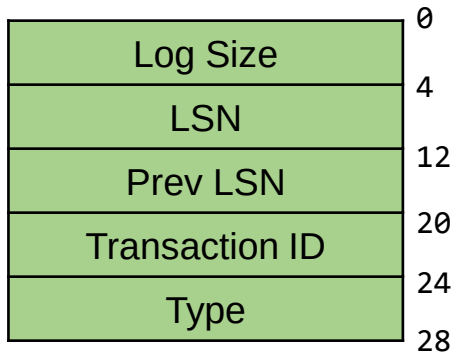
Recovery Test Example

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(…

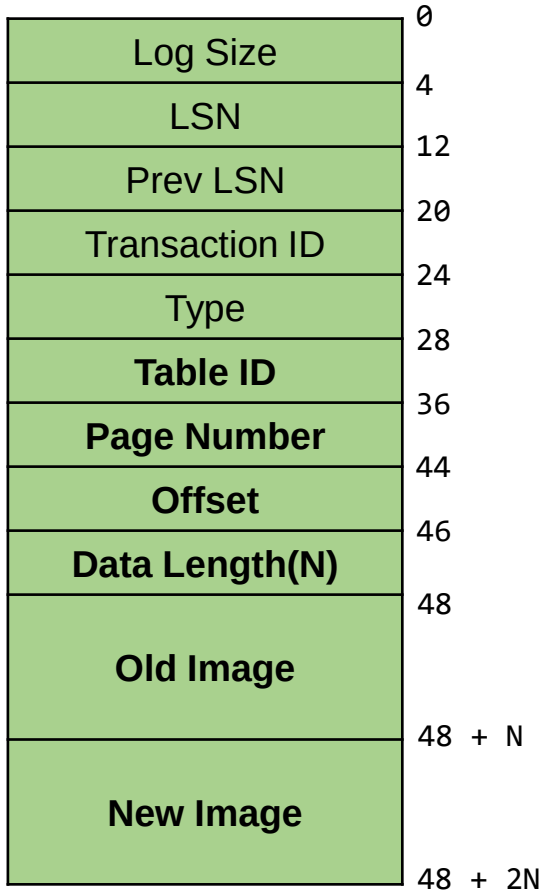


Log Record Format (using LSN as start offset)

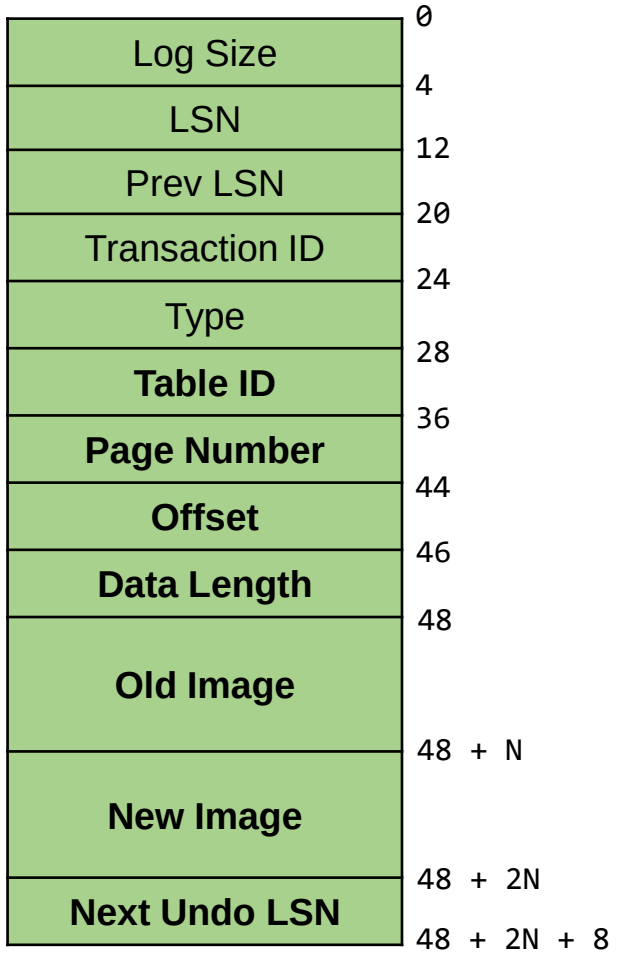
BEGIN/COMMIT/ROLLBACK Log Record



UPDATE Log Record

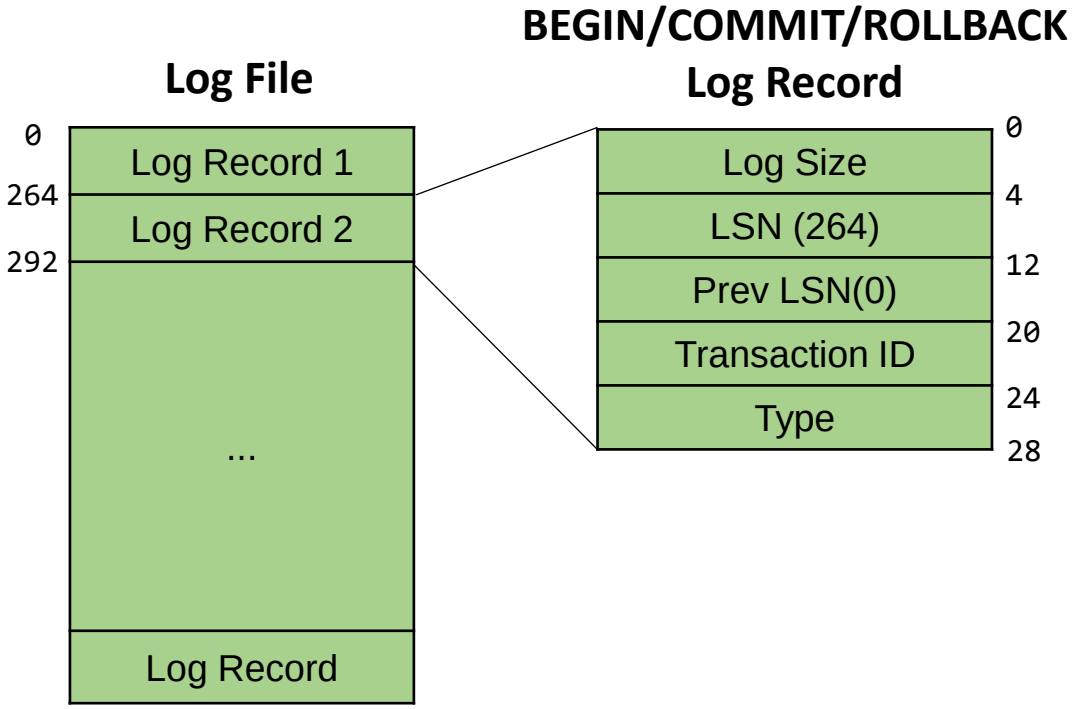


COMPENSATE Log Record



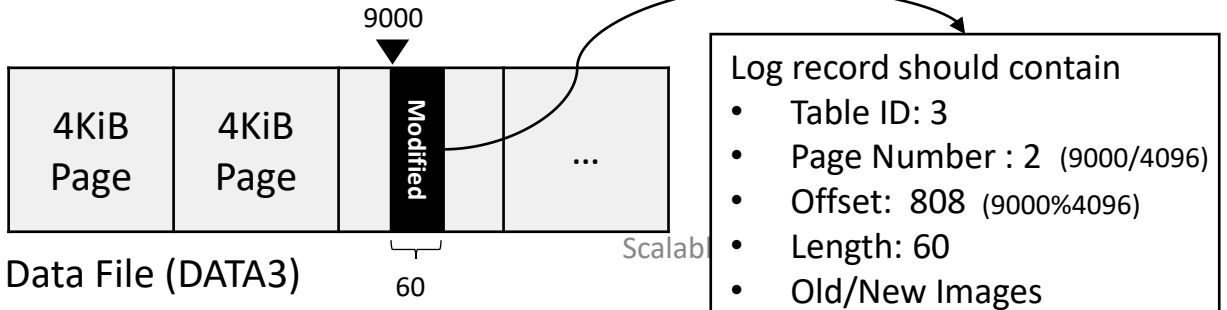
Log File

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

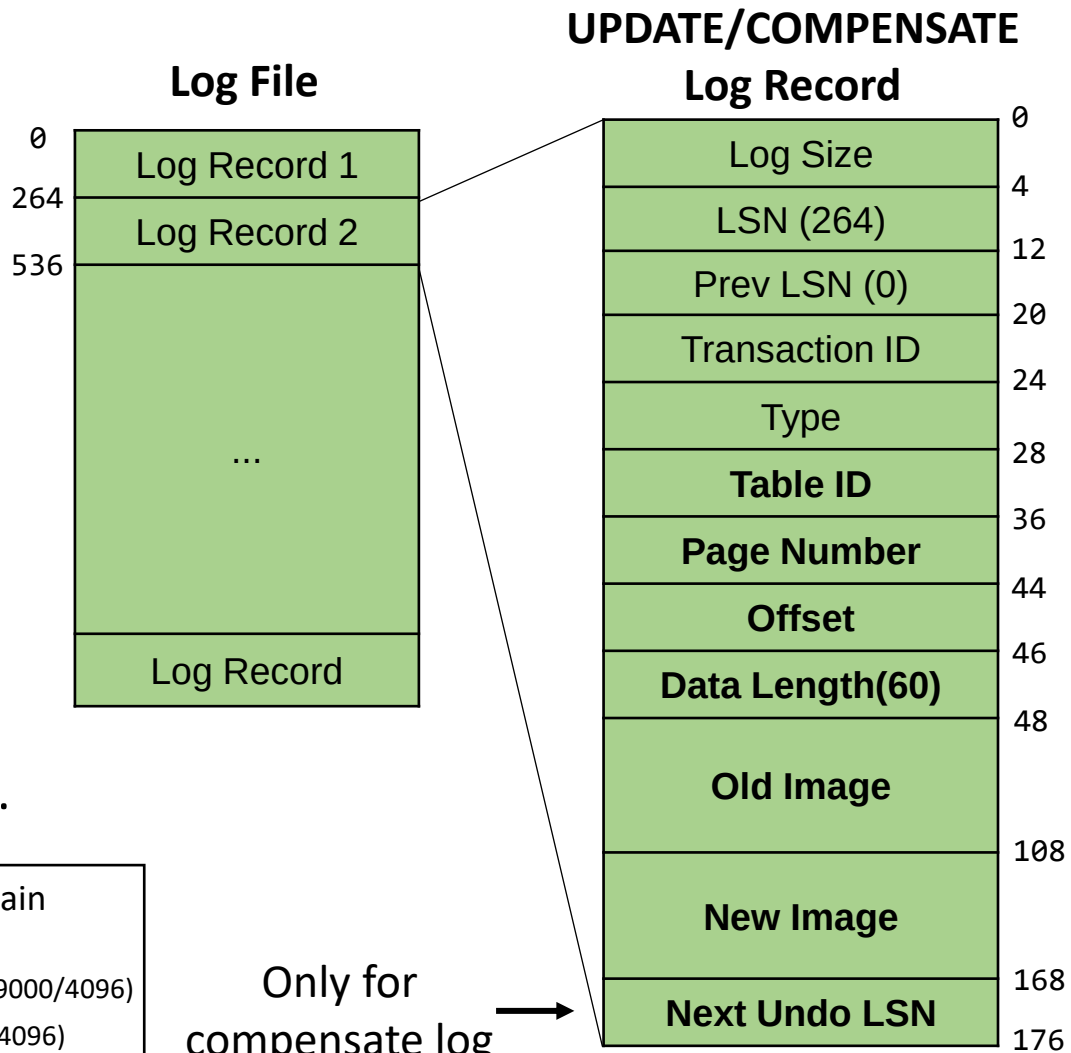


Log File

- Log file is a sequence of log records.
- Log record consists 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.

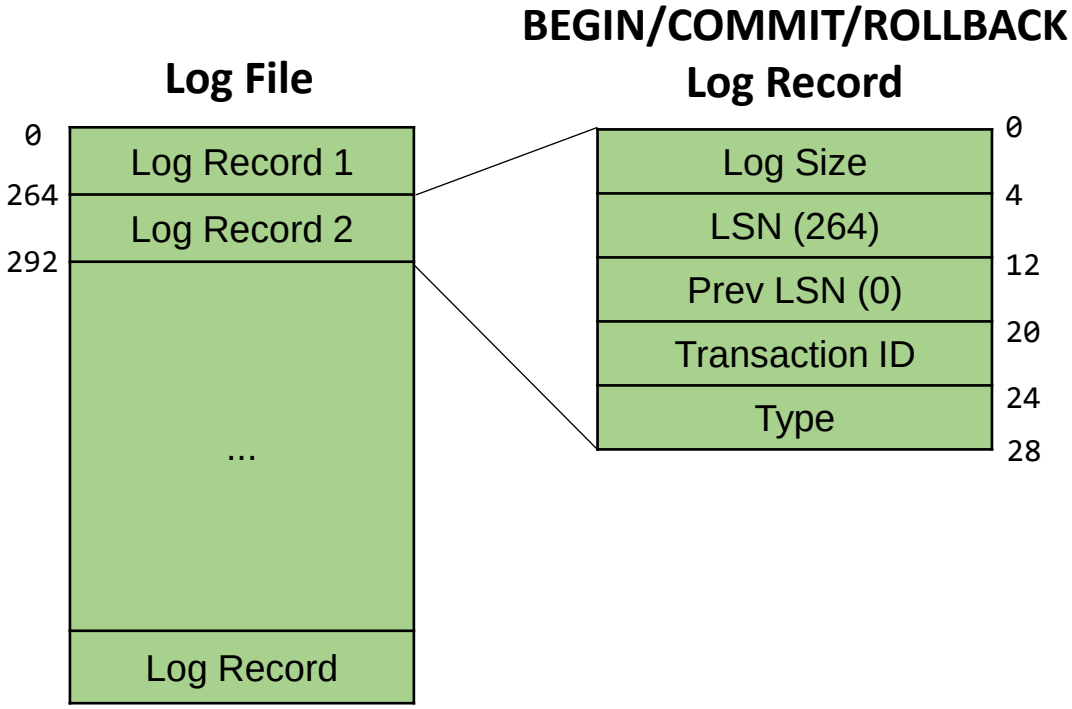


Log record should contain
- Table ID: 3
- Page Number : 2 (9000/4096)
- Offset: 808 (9000%4096)
- Length: 60
- Old/New Images



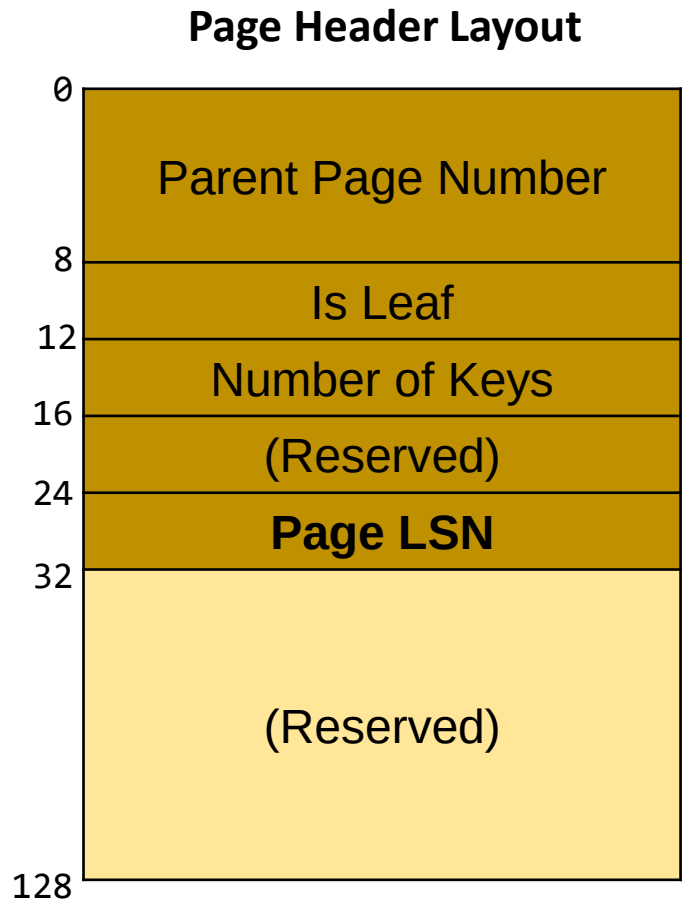
Log File

- Log file is a sequence of log records.
- Your Log Record should have the sizes:
  - BEGIN/COMMIT/ROLLBACK : 28 Byte
  - UPDATE : `48 + 2 * (Data length)` Byte
  - COMPENSATE : `56 + 2 * (Data length)` Byte

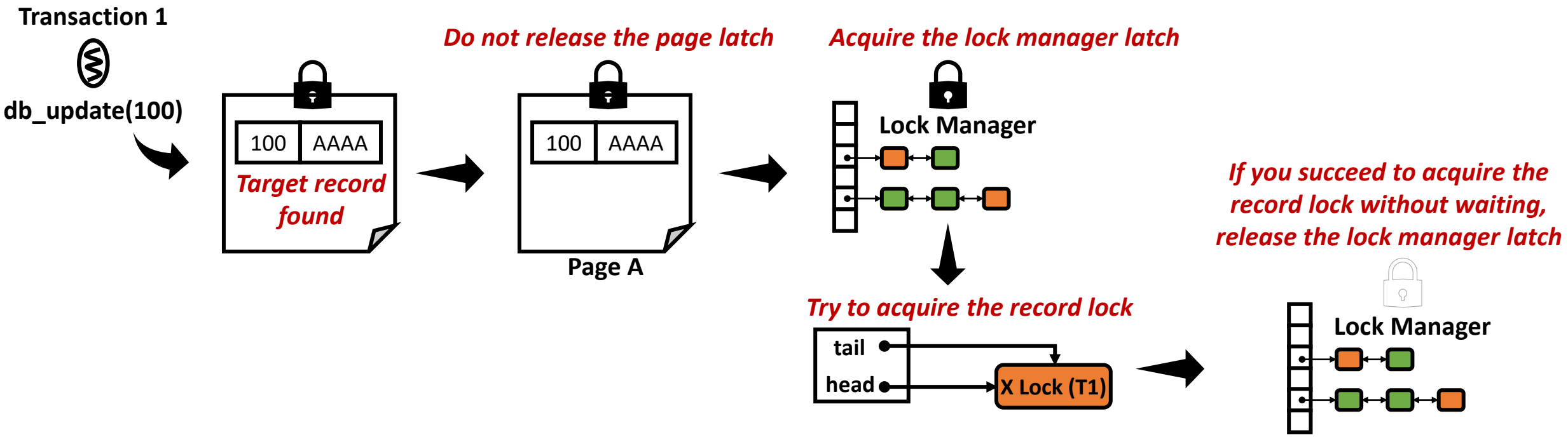


Page Header Layout

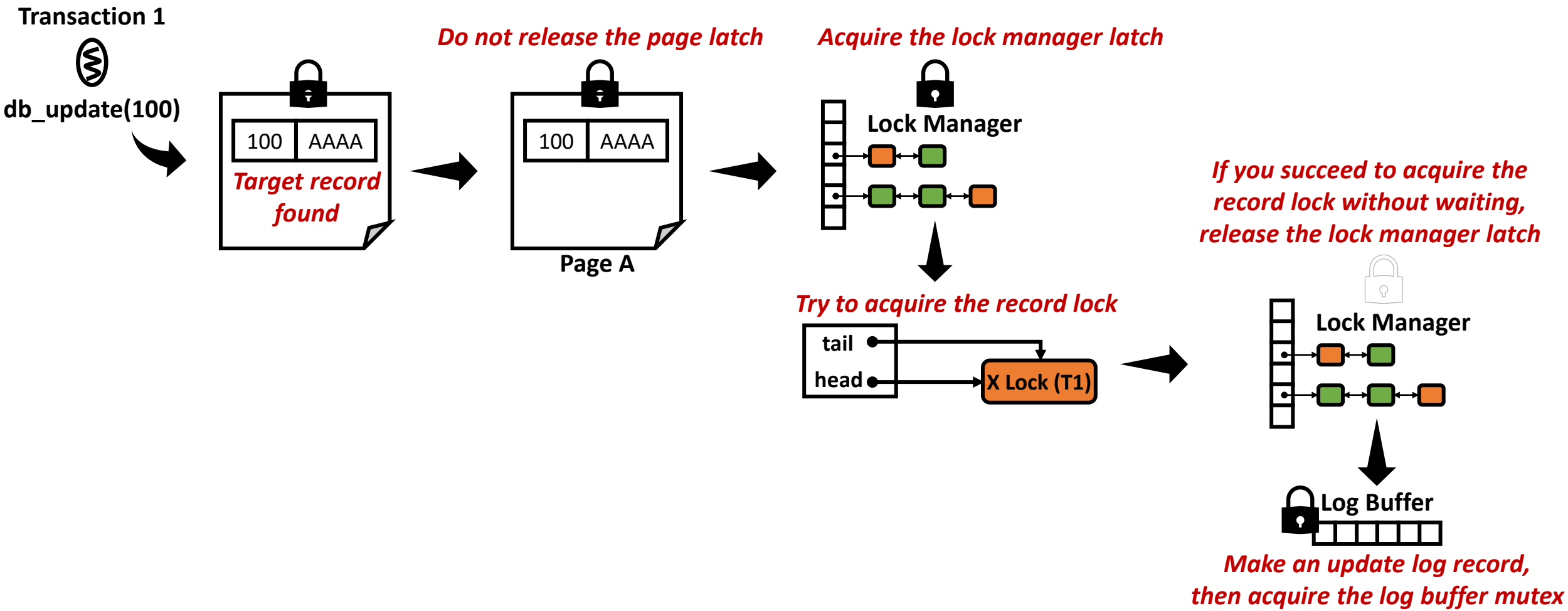
- You should maintain a page LSN information from every on-disk image.
- The page LSN indicates the last updated version of this on-disk page.
- Maintain the page LSN value (8 bytes) located at the page header structure starting from byte offset 24.
- Every page, including the header page, should maintain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



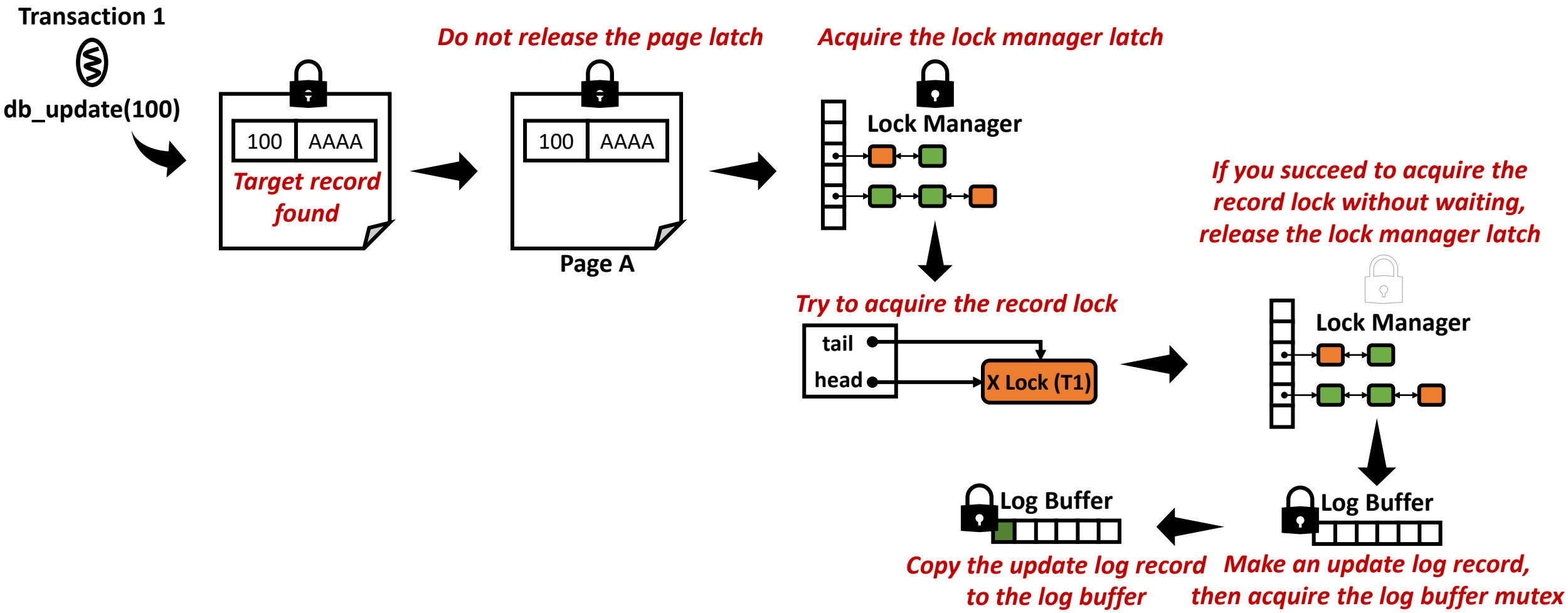
Update Latch Sequence with Log Manager



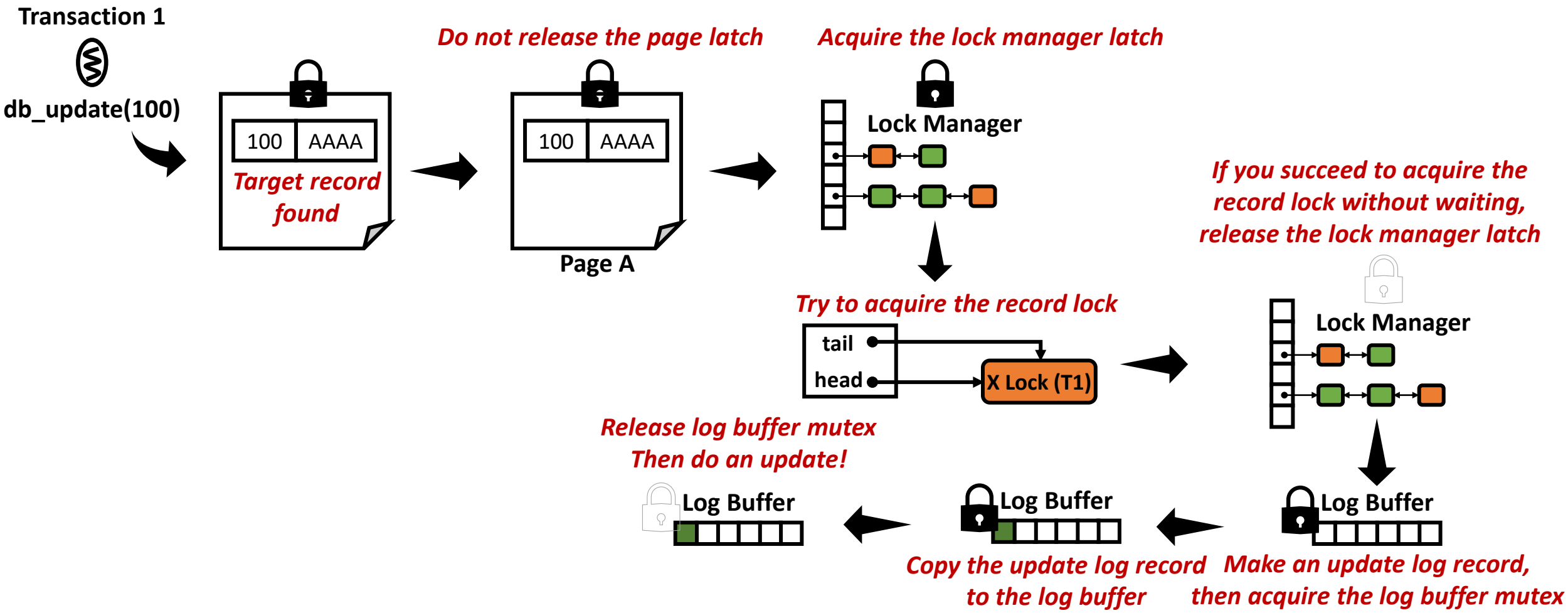
Update Latch Sequence with Log Manager



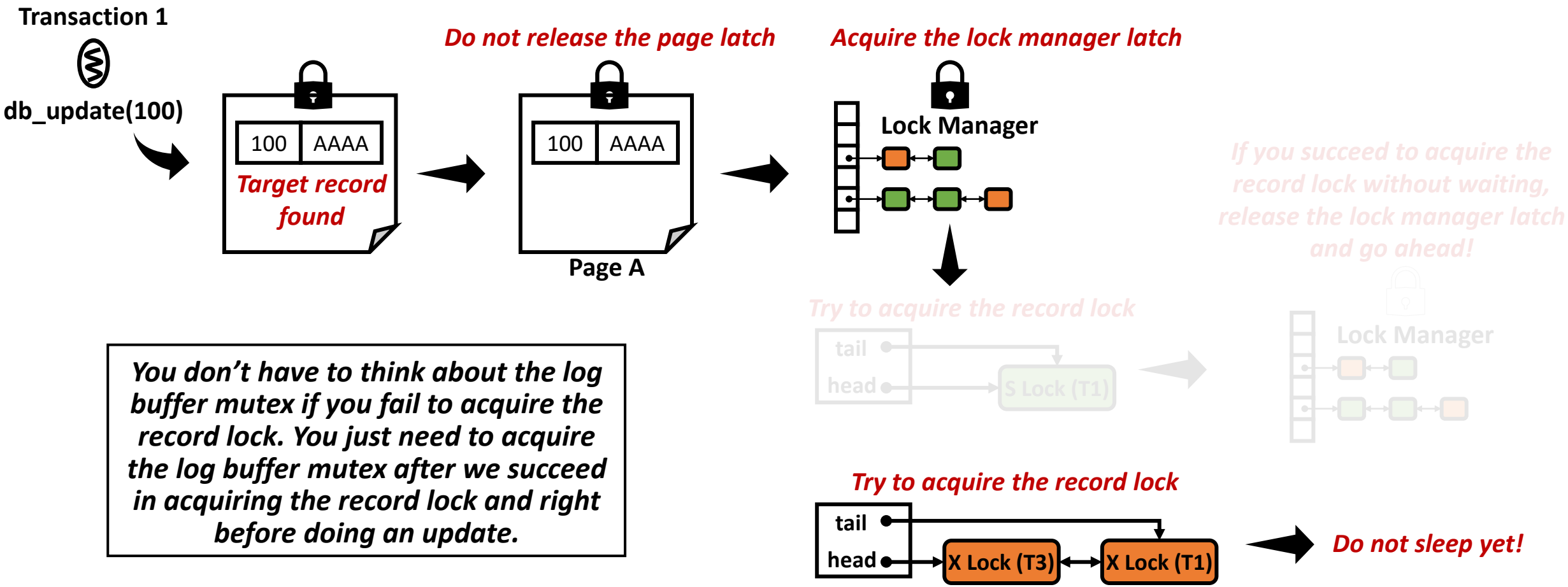
Update Latch Sequence with Log Manager



Update Latch Sequence with Log Manager



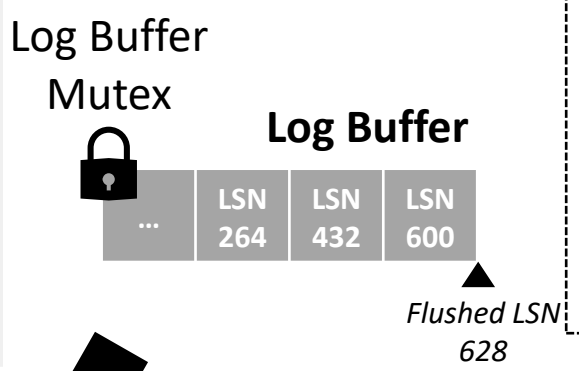
Update Latch Sequence with Log Manager

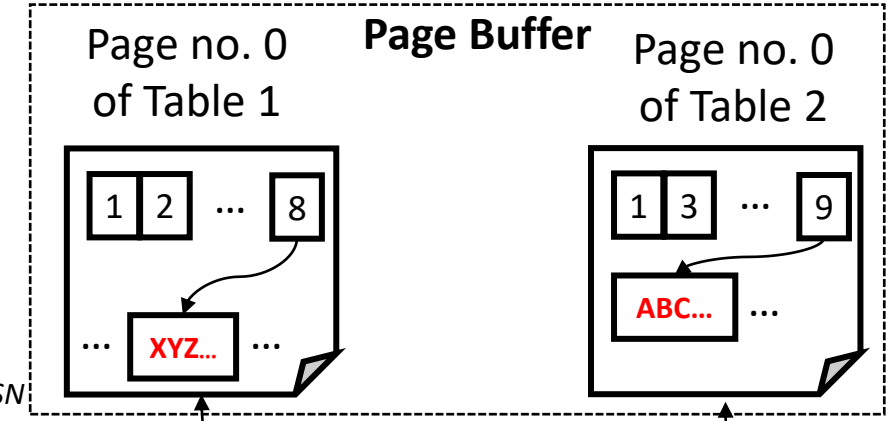


Project Example

Transaction

```
int trx_id = trx_begin();
uint16_t o_size;
…
db_update(1, 8, “XYZ…”, 60, &o_size,
trx_id);
db_update(2, 9, “ABC…”, 60, &o_size,
trx_id);
trx_commit(trx_id);
```





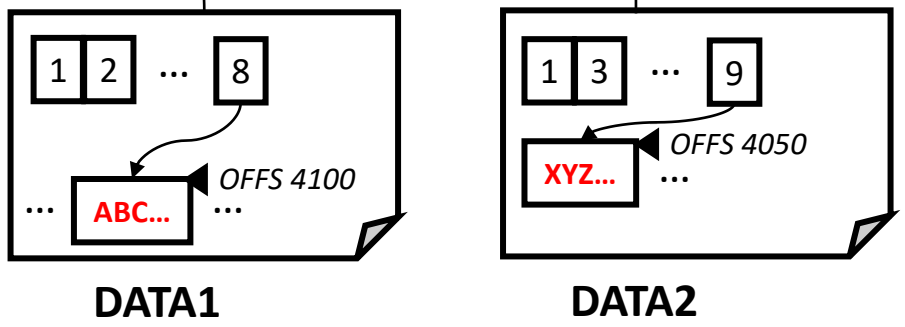
Memory

Storage

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

Log File

Size	Lsn	Plsn	Xid	Type	Tid	Pgno	Offs	Len	Old	New	
…											
168	264	236	1	UPD	1	0	4100	60	ABC…	XYZ…	432
168	432	264	1	UPD	2	0	4050	60	XYZ…	ABC…	600
28	600	432	1	COM	-		-	-	-	-	628



Project Specification

- You should implement ARIES-based recovery that you learned from the lecture but,
  - We don’t have to consider double writes since we assume that torn page writes would never occur.
  - Checkpoint is not considered for this project.

Project Specification

- We will check the correctness of the recovery by executing concurrent transactions and triggering a system crash such as 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();
  uint16_t o_size;
  db_update(1, 3, “XYZ…”, 60, &o_size, trx_id);
  trx_commit(trx_id);
  int new_id = trx_begin();
  db_update(1, 2, “XXX…”, 60, &o_size, 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 should contain the following contents
    - Design/Implementation details on Analysis/Redo/Undo Pass
    - Simple test result
    - Other details worth mentioning
- Deadline: 12/19 23:59
- We will score your project based on the latest commit before the deadline. Any submission after the deadline will not be accepted.

Thank you