Project5

Concurrency Control

Scalable Computing Systems Laboratory
Hanyang University

Milestone1

Lock Manager

Submission

- You should submit your project in a directory structure like this: "your_repo/project5/db_project".

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

5 directories, 15 files
```

- Follow the directory structure shown above.
- You can use different names for the source and header files and add files and directories for sources or headers if necessary.

Lock Manager

- Your database system is not yet supporting the transaction.
- Implement the **transaction** concept that can support ‘**I**solation’ and ‘**C**onsistency’ using your lock manager (lock table).
- Your lock manager should provide:
  - Conflict-serializable schedule for transactions
  - Strict-2PL
  - Deadlock detection (abort the transaction if detected)
  - Record-level locking with Shared(S)/Exclusive(X) mode

Project Specification

- Your library should provide two APIs below for transaction operations.

- ***int trx_begin(void)***
  - Allocate a transaction structure and initialize it.
  - Return a unique **transaction id (>= 1)** if success, otherwise return 0.
  - Note that the transaction id should be unique for each transaction; that is, you need to allocate a transaction id holding a mutex.

- ***int trx_commit(int trx_id)***
  - Clean up the transaction with the given trx_id (transaction id) and its related information that has been used in your lock manager. (Shrinking phase of strict 2PL)
  - Return the completed transaction id if success, otherwise return 0.

Project Specification

- Also, your library should provide two APIs below for database operations that can be wrapped in a transaction.

- ***int db_find(int64_t table_id, int64_t key, char* ret_val, uint16_t * val_size, int trx_id)***
  - Read a value in the table with a matching key for the transaction having ***trx_id***.
  - return 0 (SUCCESS): operation is successfully done, and the transaction can continue the next operation.
  - return non-zero (FAILED): operation is failed (e.g., deadlock detected), and the transaction should be aborted. Note that all tasks that need to be handled (e.g., **releasing the locks that are held by this transaction**, **rollback of previous operations**, etc.) should be completed in db_find().

Project Specification

- Also, your library should provide two APIs below for database operations that can be wrapped in a transaction.

- ***int db_update(int64_t table_id, int64_t key, char* values, uint16_t new_val_size, uint16_t* old_val_size, int trx_id)***
    - Find the matching key and modify the values.
    - If found matching 'key', update the value of the record to 'values' string with its 'new_val_size' and store its size in 'old_val_size'.
    - return 0 (SUCCESS): operation is successfully done, and the transaction can continue the next operation.
    - return non-zero (FAILED): operation is failed (e.g., deadlock detected), and the transaction should be aborted. Note that all tasks that need to be handled (e.g., **releasing the locks that are held on this transaction**, **rollback of previous operations**, etc.) should be completed in db_update().

Project Specification

- Your database must provide the following API.

```
#include <stdint.h>

int init_db( int num_buf);
int shutdown_db();
int64_t open_table(char* filename);
int db_insert( int64_t table_id, int64_t key, char* value, uint16_t val_size);
int db_delete( int64_t table_id, int64_t key);

// Project5
int db_find( int64_t table_id, int64_t key, char* ret_val, uint16_t * val_size, int trx_id);
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);
int trx_begin();
int trx_commit( int trx_id);
```

Project Specification

- Note that, in this project, you don't have to support *db_insert()* or *db_delete()* working **in a transaction** that may require structural modifications on b+tree.

- **For the same reason, if db_update() changes the value size, it can change the structure of b+tree, so db_update() does not change the value size of the existing record.**

- **In this project, db_update() does not change the record size. However, the reason for receiving new_val_size as a parameter is to copy only the allowed memory area.**

- We will first populate the database with *db_insert()* or open a sample database file and then run transactions in our test.

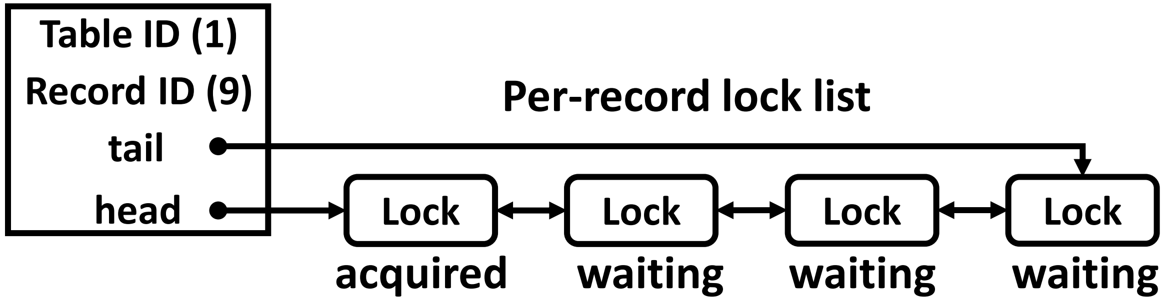
APIs for Lock Table Module

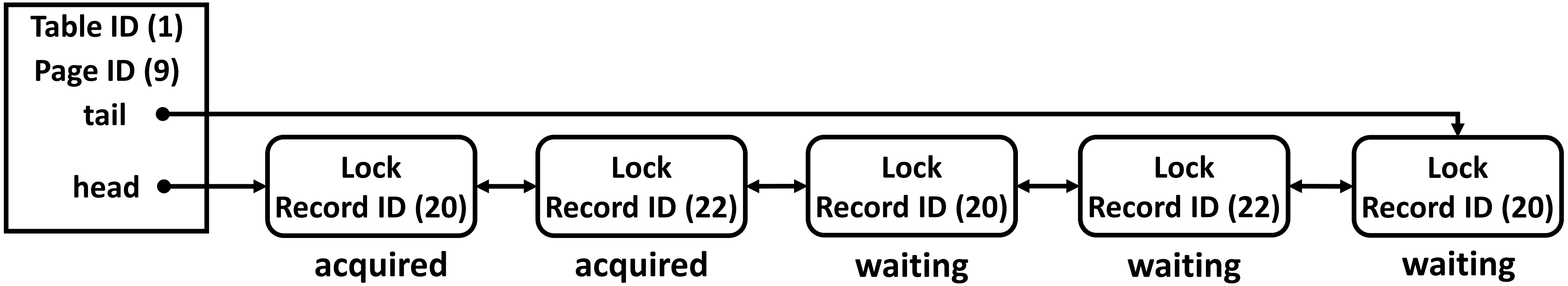
- Your lock module's APIs would like below. Use them appropriately in your database operation functions.
  - ➢ It is accepted to change the APIs (return type, parameters, etc.) of the lock table module if you want.
- ***int init_lock_table(void)***
  - Initialize any data structures required for implementing a lock table, such as a hash table, a lock table latch, etc.
  - If success, return 0. Otherwise, return a non-zero value.
- ***lock_t* lock_acquire(int64_t table_id, pagenum_t page_id, int64_t key, int trx_id, int lock_mode)***
  - Allocate and append a new lock object to the lock list of the record having the page id and the *key.*
    - If there is a predecessor's conflicting lock object in the lock list, **sleep** until the predecessor releases its lock.
    - If there is no predecessor's conflicting lock object, return the address of the new lock object.
  - If an error occurs, return NULL.
  - ***lock_mode: 0 (SHARED) or 1 (EXCLUSIVE)***

APIs for Lock Table Module

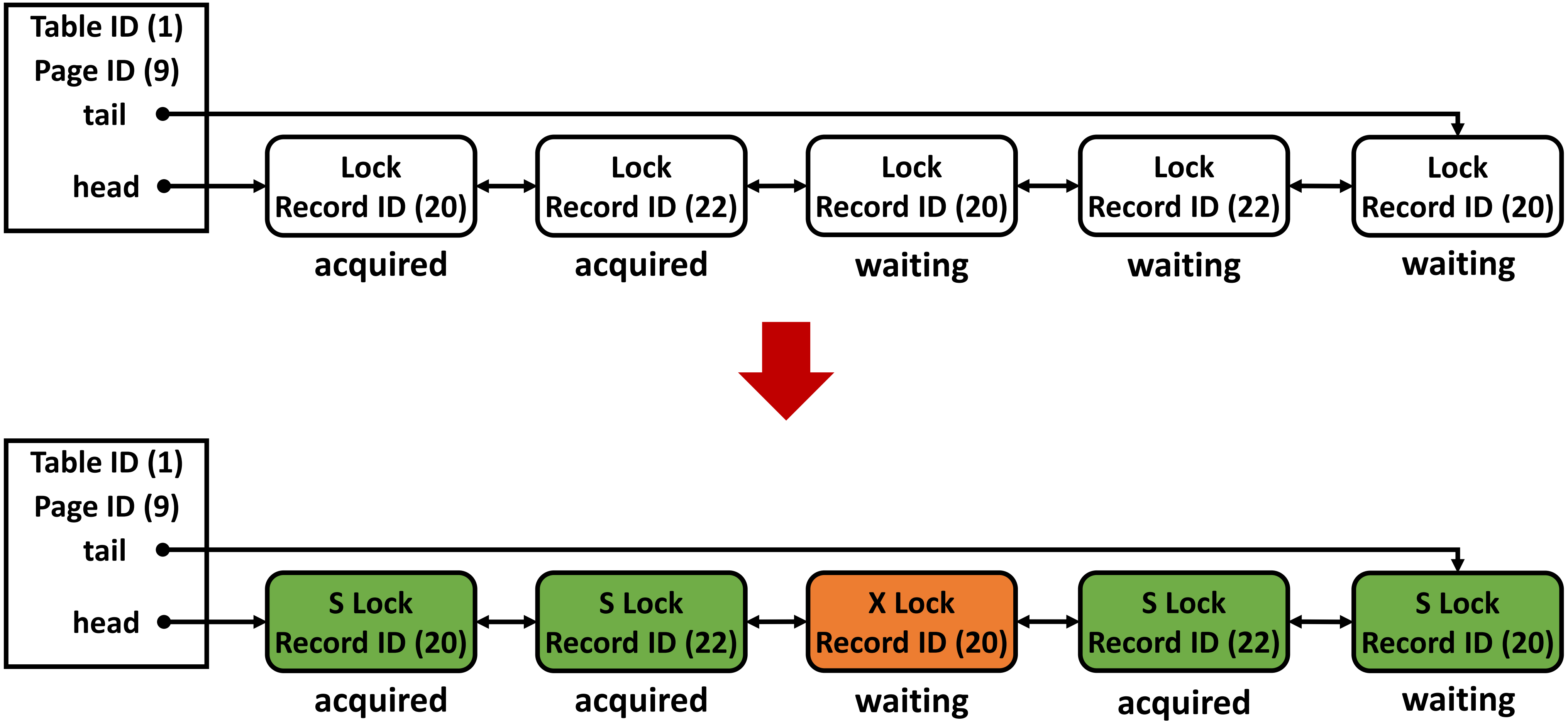
- Your lock module's APIs would like below. Use them appropriately in your database operation functions.
  - ➢ It is accepted to change the APIs (return type, parameters, etc.) of the lock table module if you want.
- ***int lock_release(lock_t* lock_obj)***
  - Remove the *lock_obj* from the lock list.
    - If there is a successor's lock waiting for the transaction releasing the lock, **wake up** the successor.
  - If success, return 0. Otherwise, return a non-zero value.

Changing Lock Table





Shared / Exclusive Lock

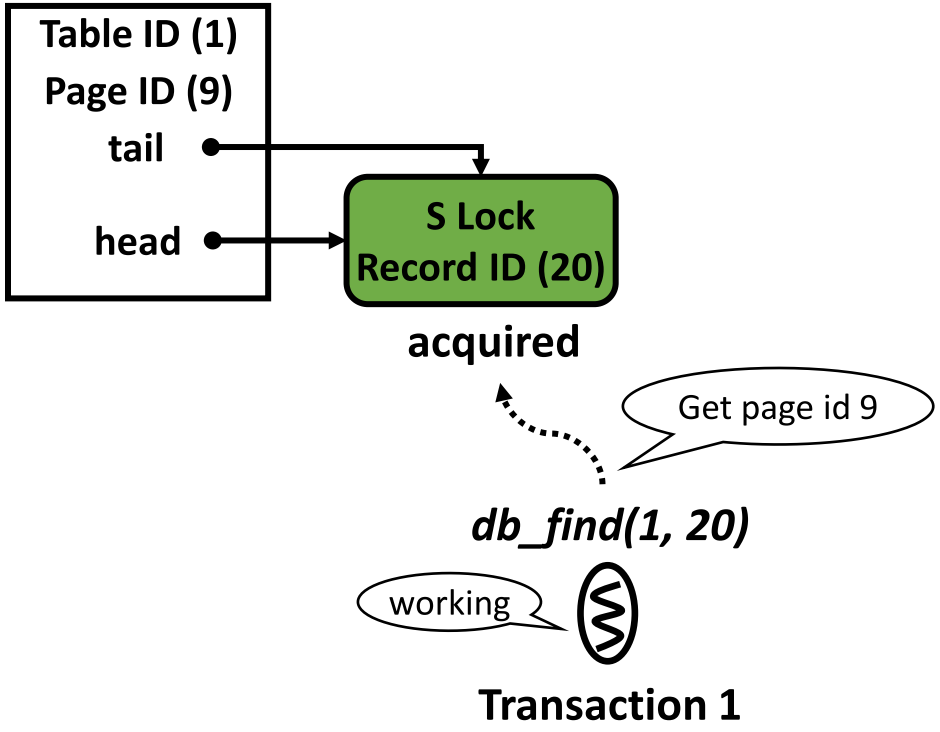


Transaction Example

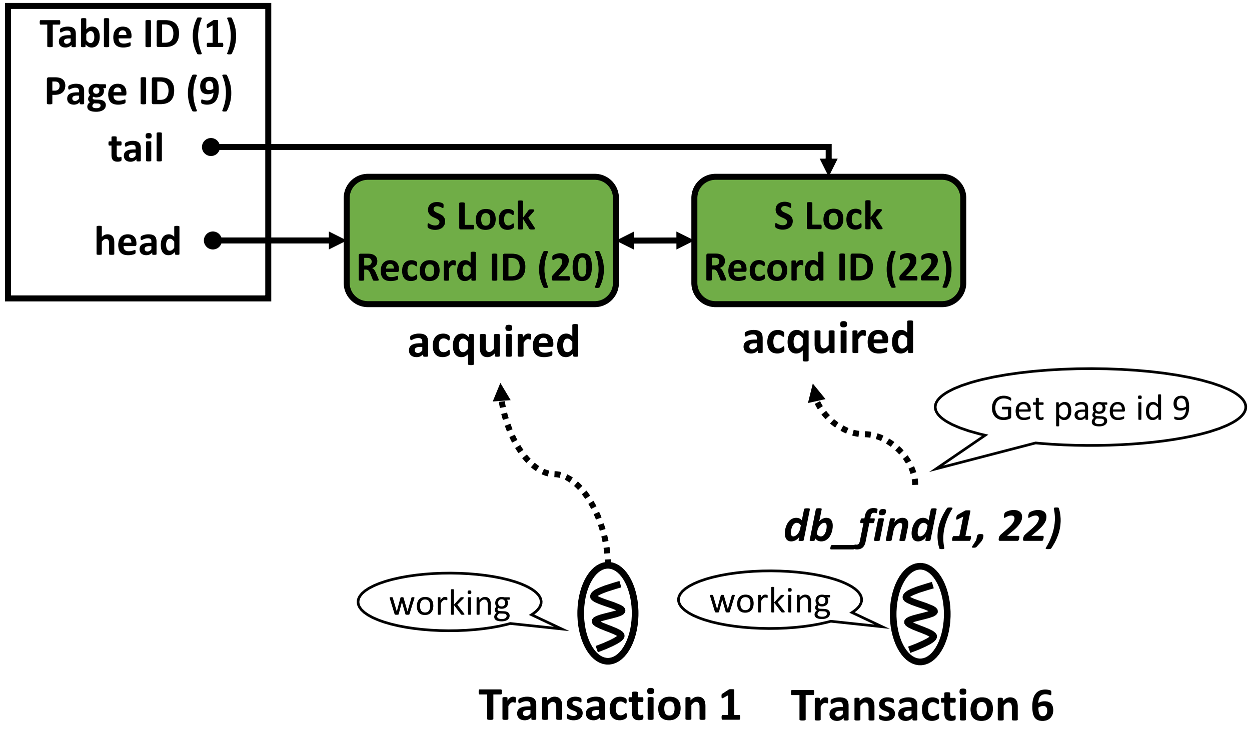
Transaction

```
trx_begin()
…
db_find()
        lock_acquire(SHARED)
db_find()
        lock_acquire(SHARED)
db_update()
        lock_acquire(EXCLUSIVE)
…
trx_commit()
        lock_release()
        lock_release()
        lock_release()
        …
```

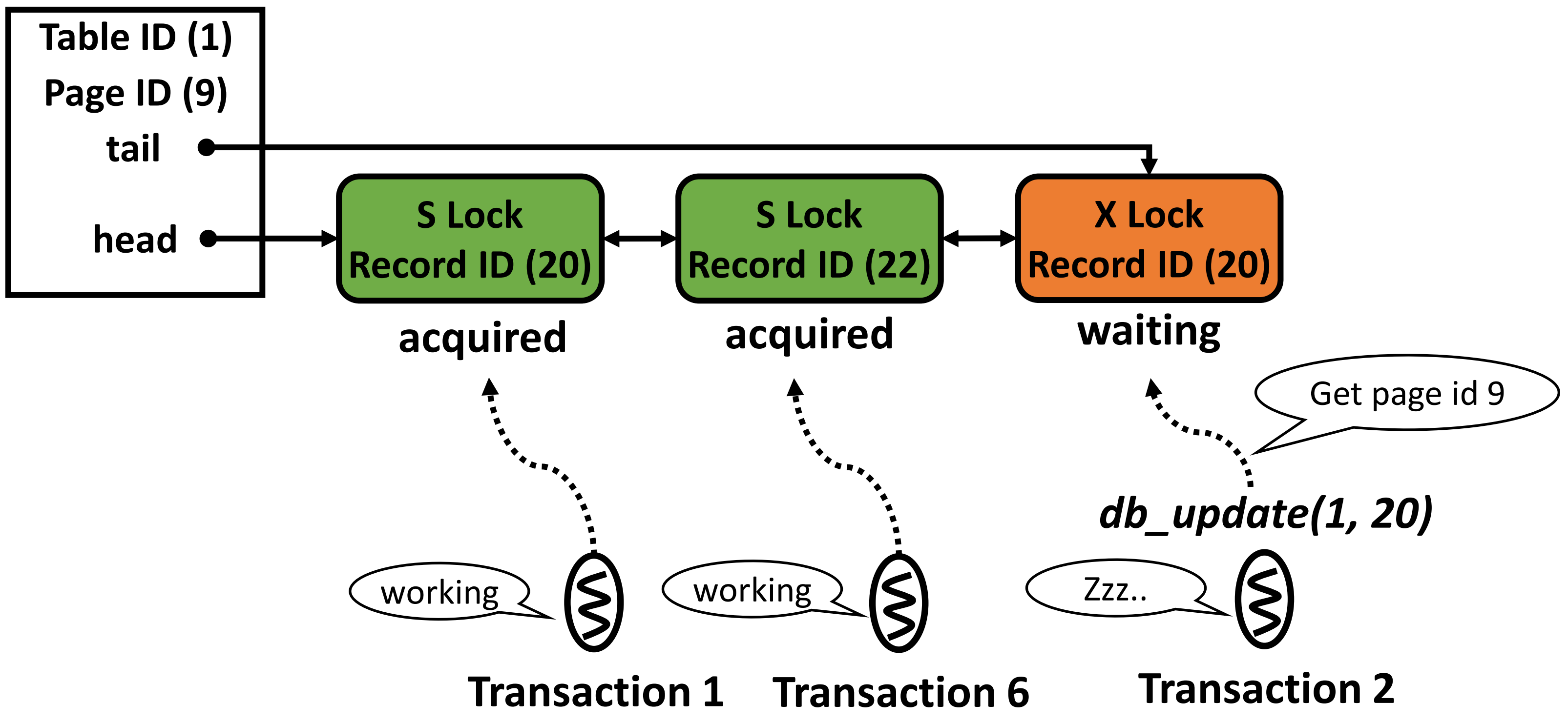
Shared / Exclusive Lock



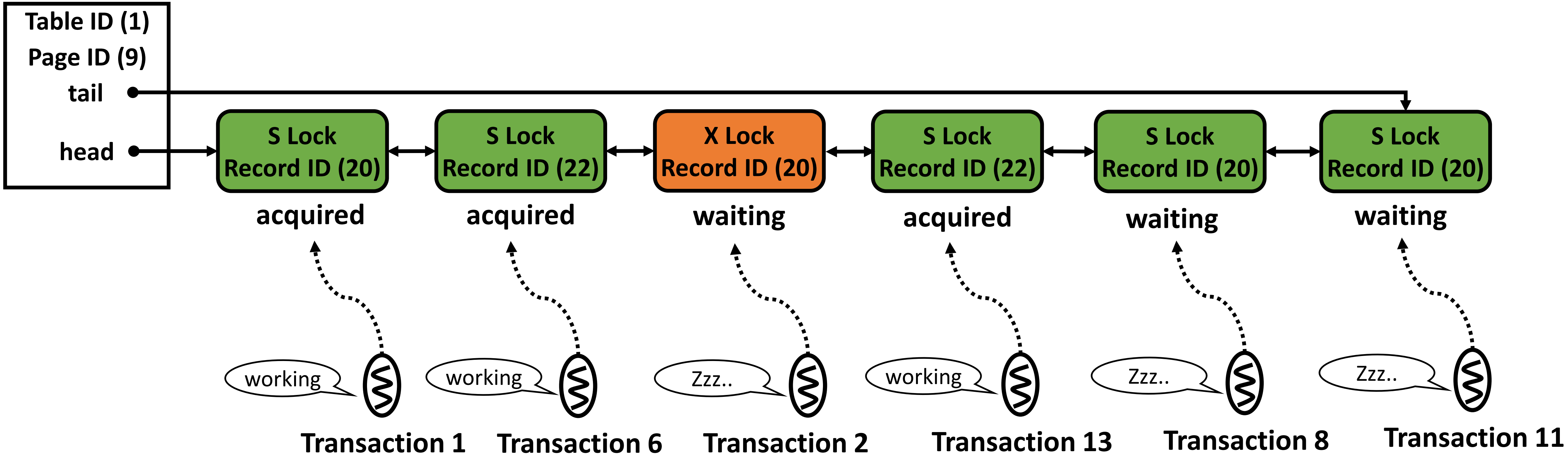
Shared / Exclusive Lock



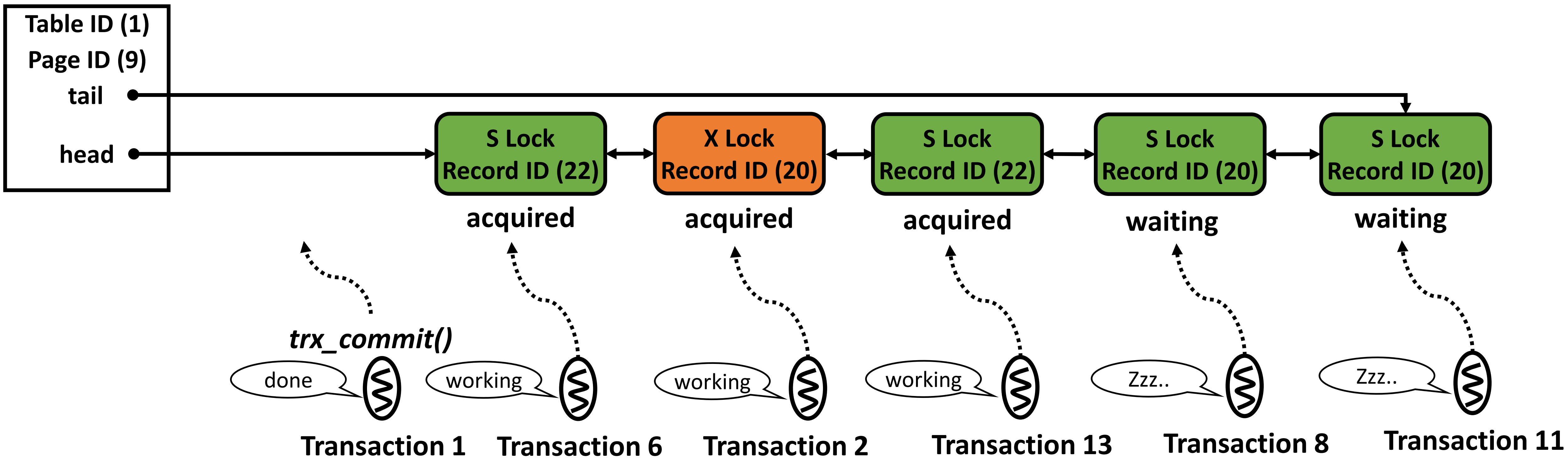
Shared / Exclusive Lock



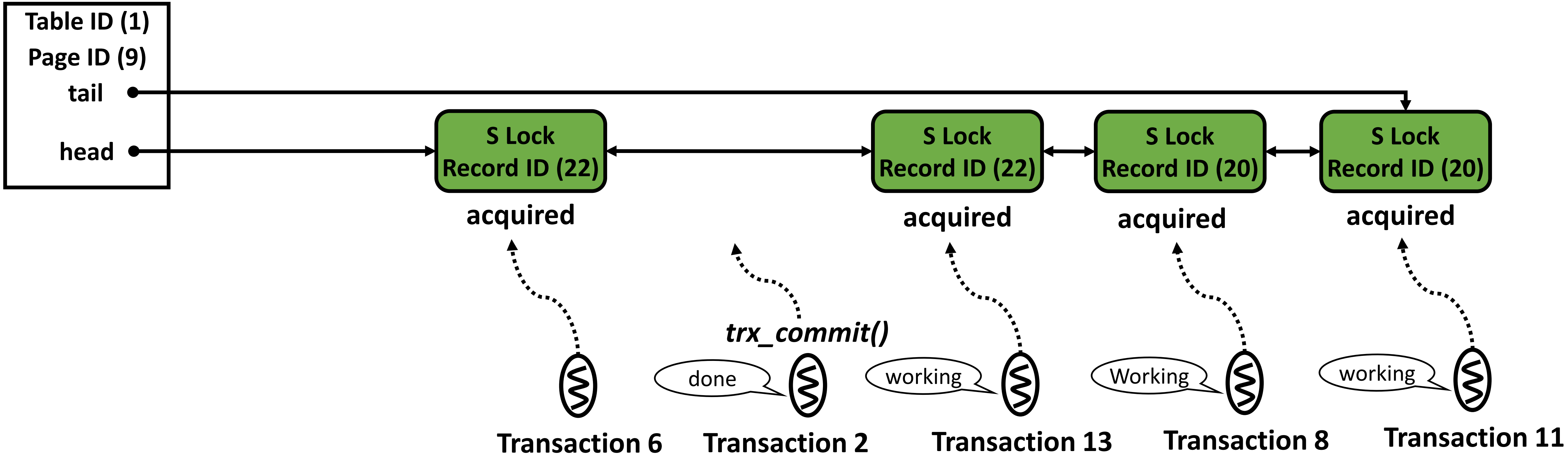
Shared / Exclusive Lock



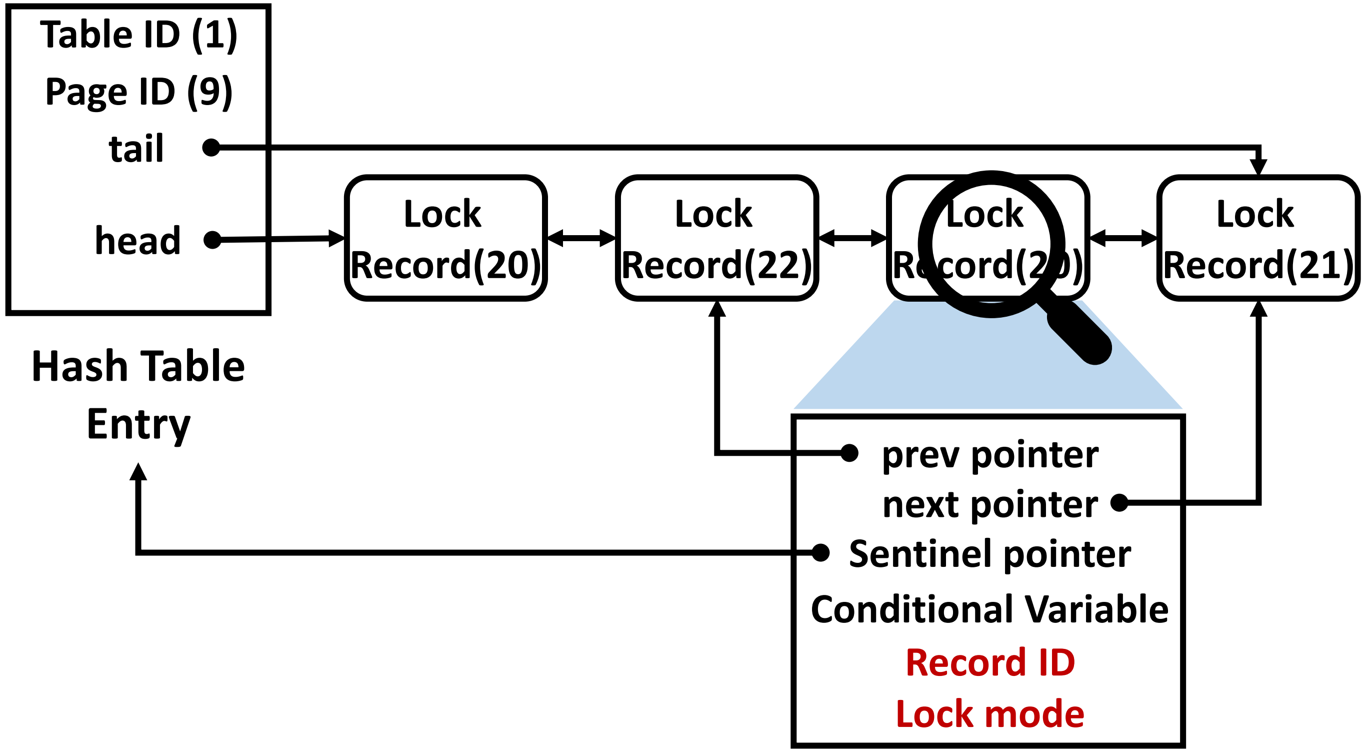
Shared / Exclusive Lock



Shared / Exclusive Lock



Shared / Exclusive Lock



Deadlock

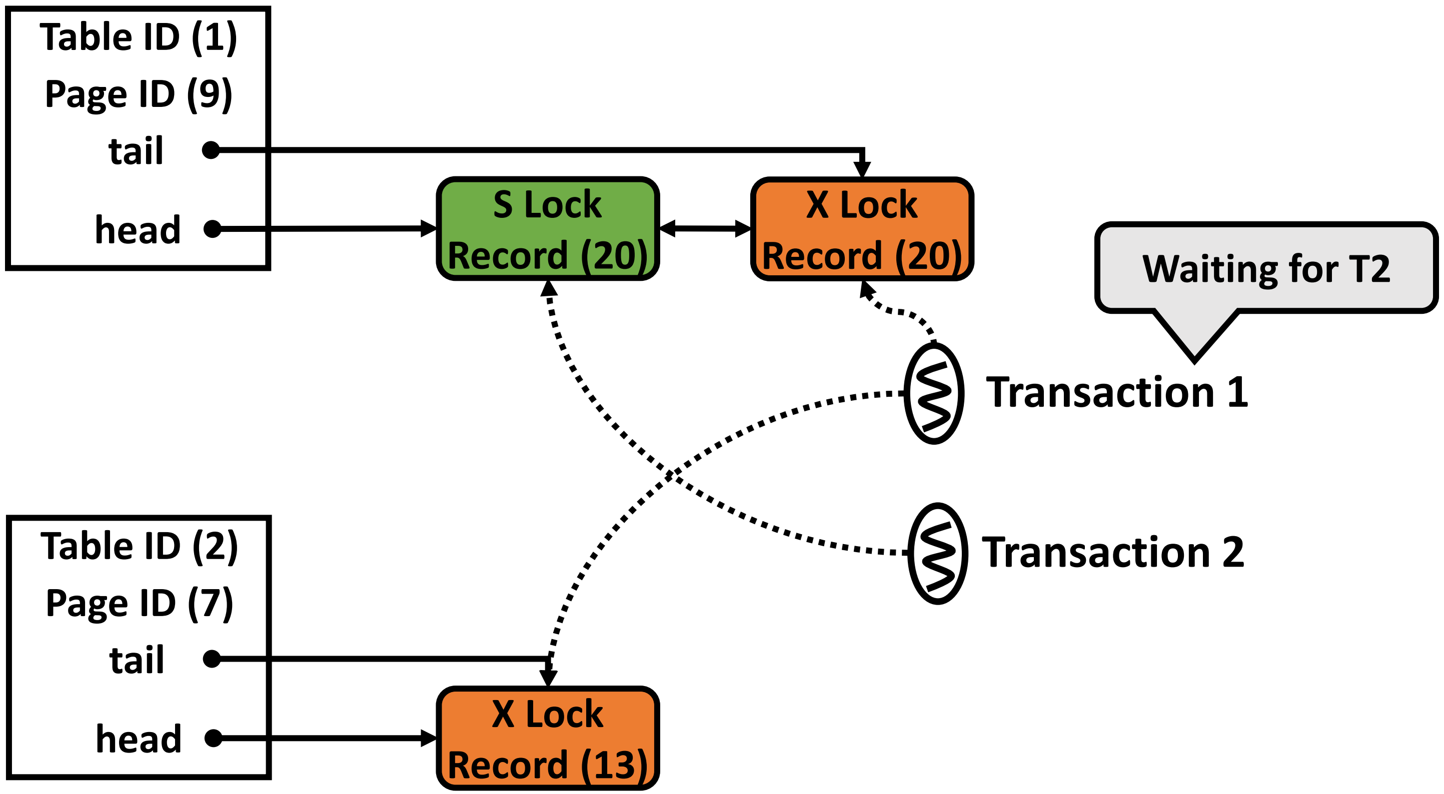


Table ID (1)
Page ID (9)
tail
head
S Lock
Record (20)
X Lock
Record (20)
Waiting for T2
Transaction 1
Transaction 2
Table ID (2)
Page ID (7)
tail
head
X Lock
Record (13)

Deadlock

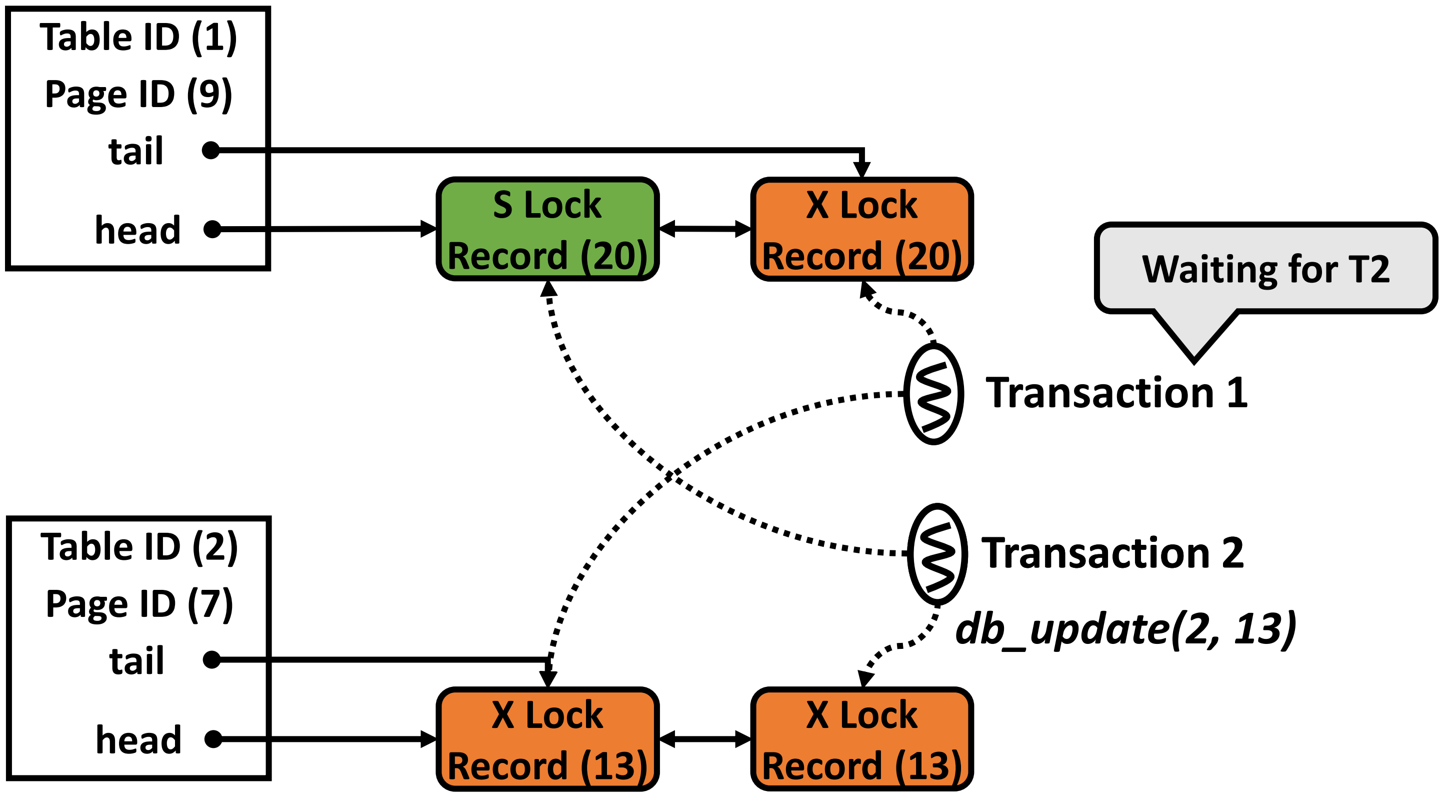


Table ID (1)
Page ID (9)
tail
head
S Lock
Record (20)
X Lock
Record (20)
Waiting for T2
Transaction 1
Transaction 2
db_update(2, 13)
Table ID (2)
Page ID (7)
tail
head
X Lock
Record (13)
X Lock
Record (13)

Deadlock

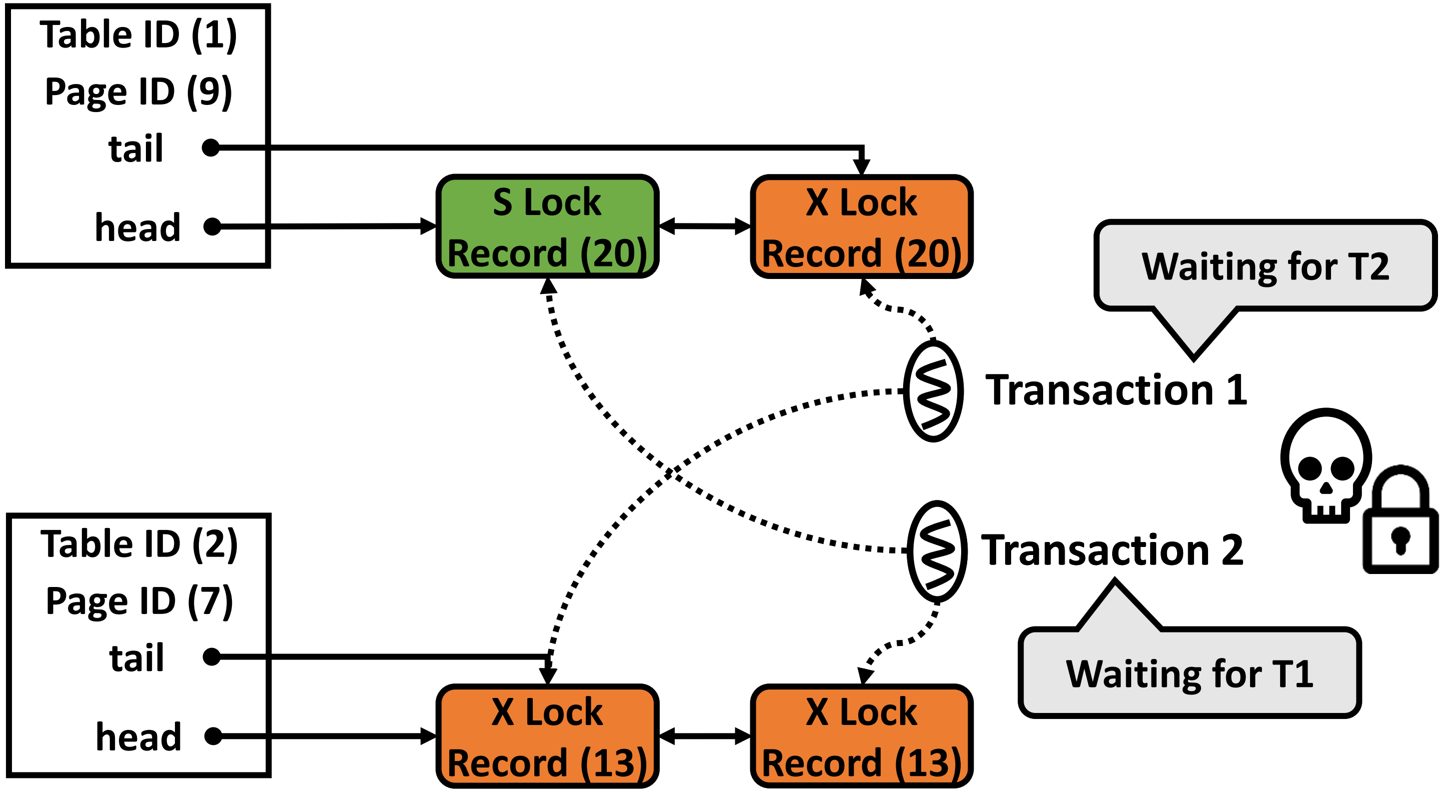


Table ID (1)
Page ID (9)
tail
head
S Lock
Record (20)
X Lock
Record (20)
Waiting for T2
Transaction 1
Transaction 2
Waiting for T1
Table ID (2)
Page ID (7)
tail
head
X Lock
Record (13)
X Lock
Record (13)

Deadlock

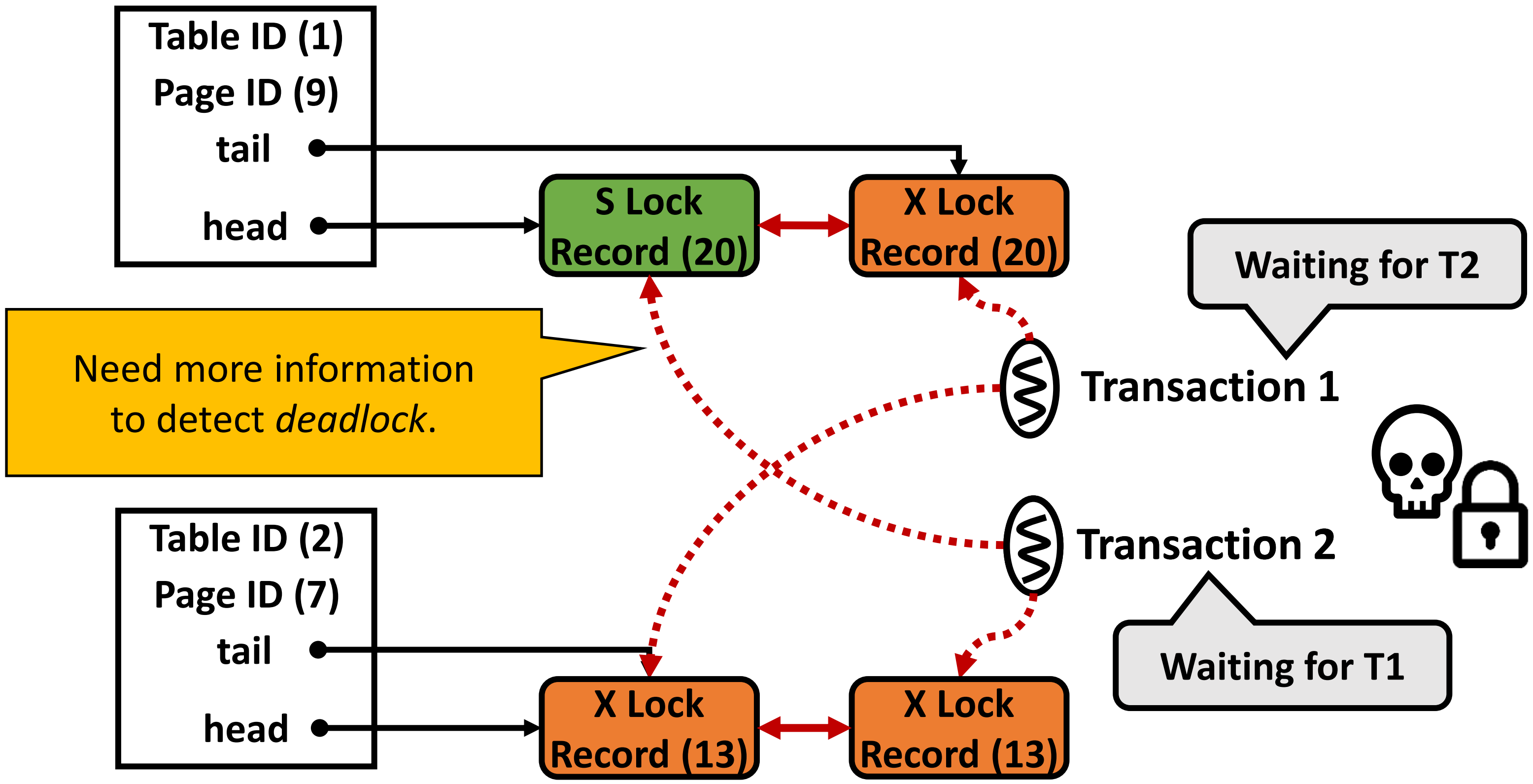
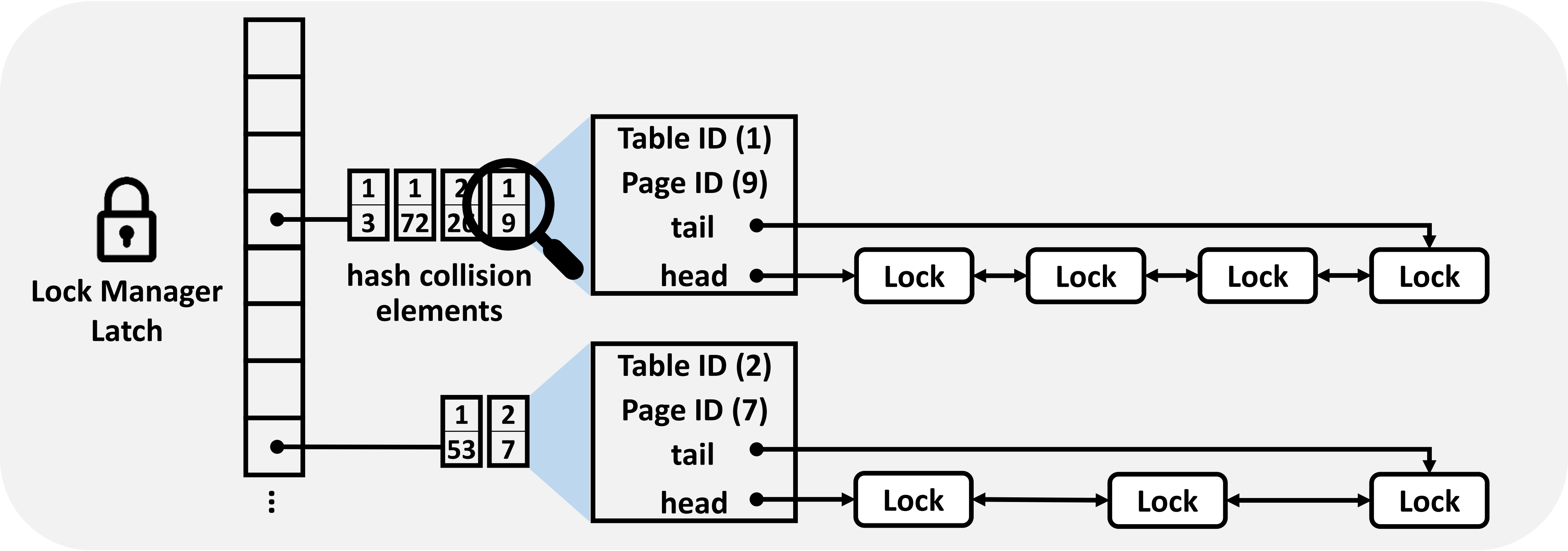
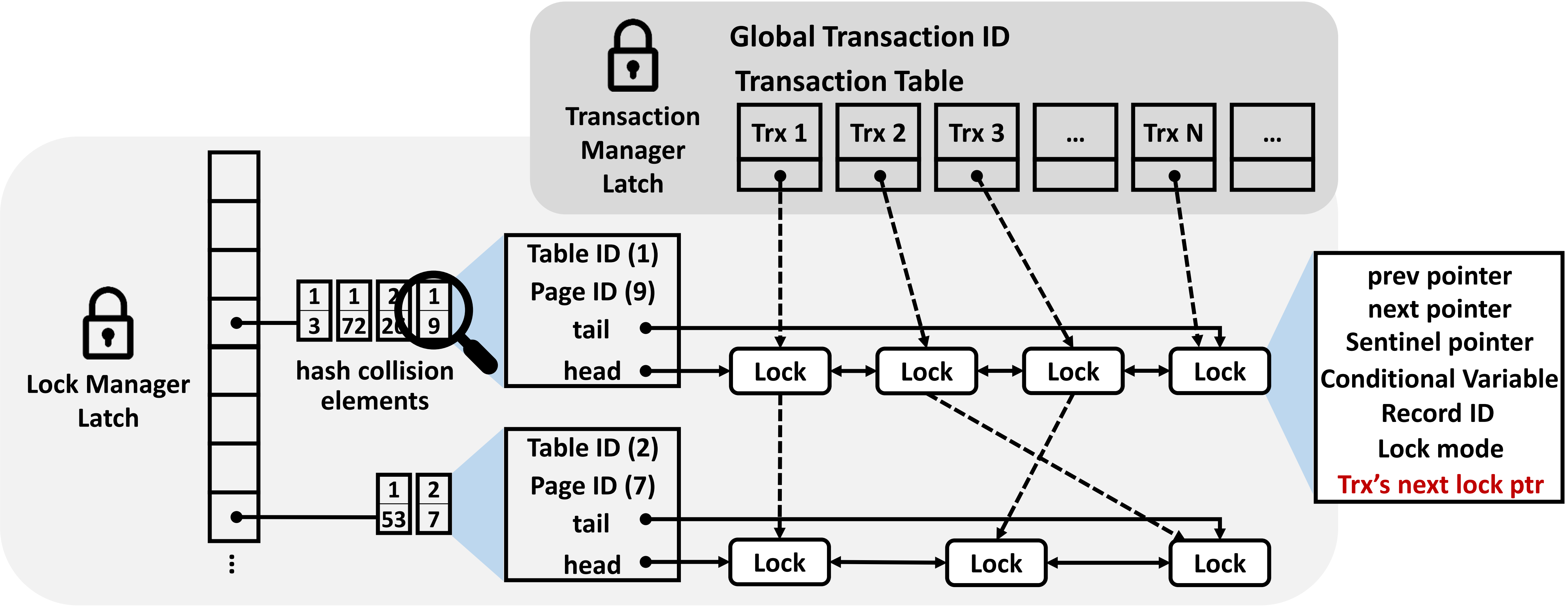


Table ID (1)
Page ID (9)
tail
head
S Lock
Record (20)
X Lock
Record (20)
Waiting for T2
Need more information
to detect *deadlock*.
Transaction 1
Transaction 2
Table ID (2)
Page ID (7)
tail
head
X Lock
Record (13)
X Lock
Record (13)
Waiting for T1

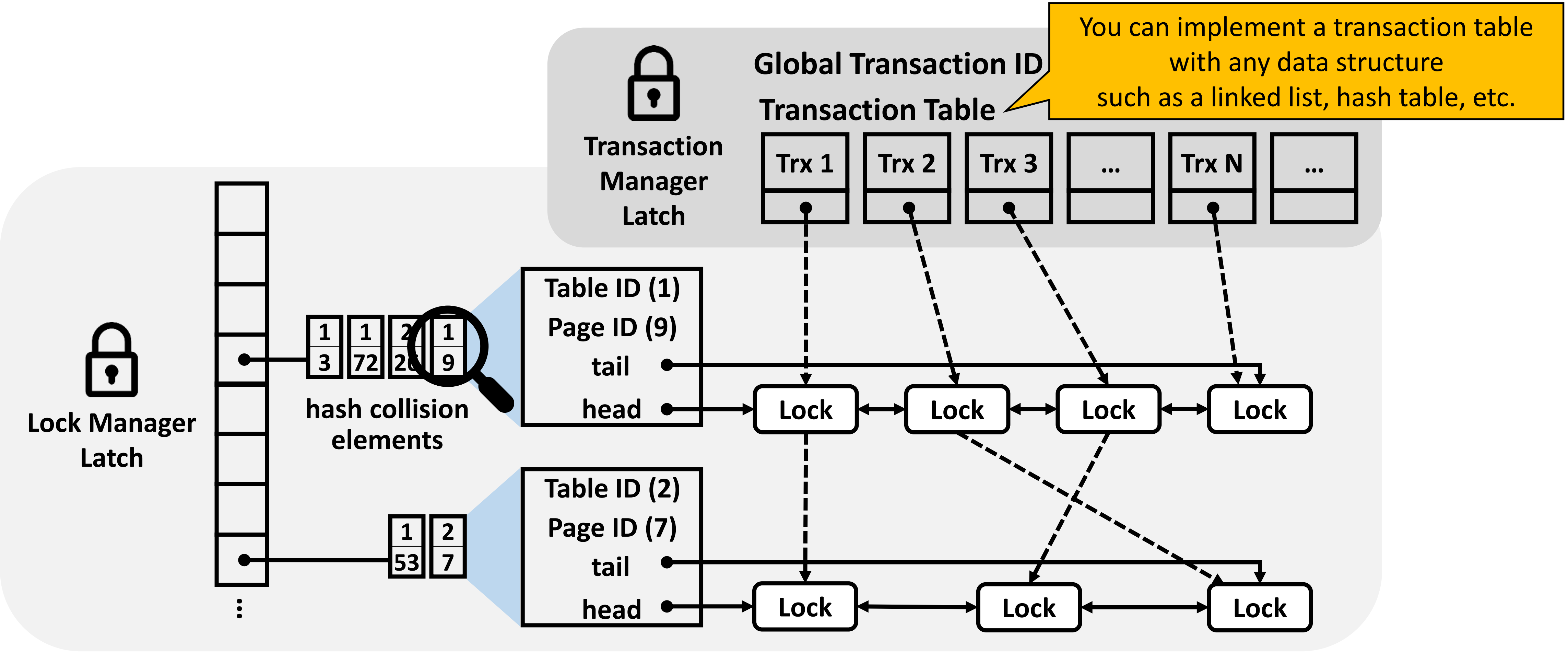
Lock Manager



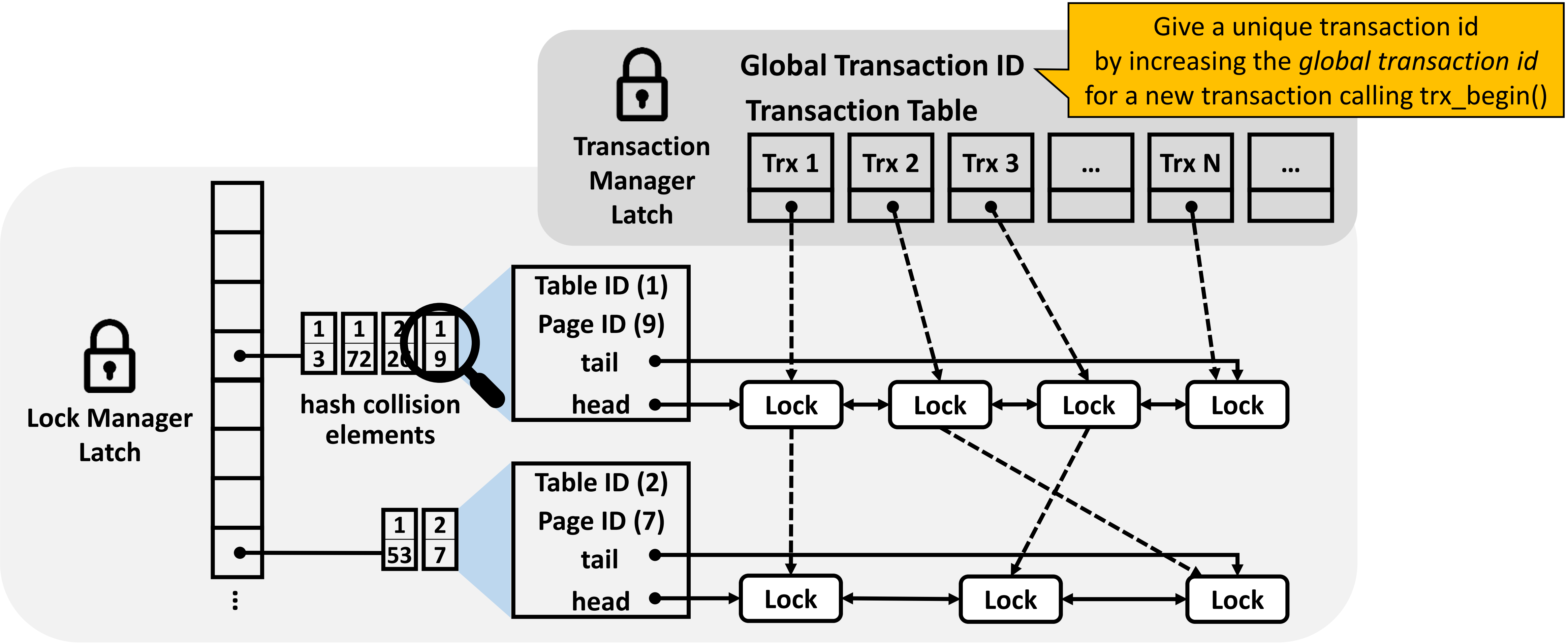
Transaction Manager



Transaction Manager

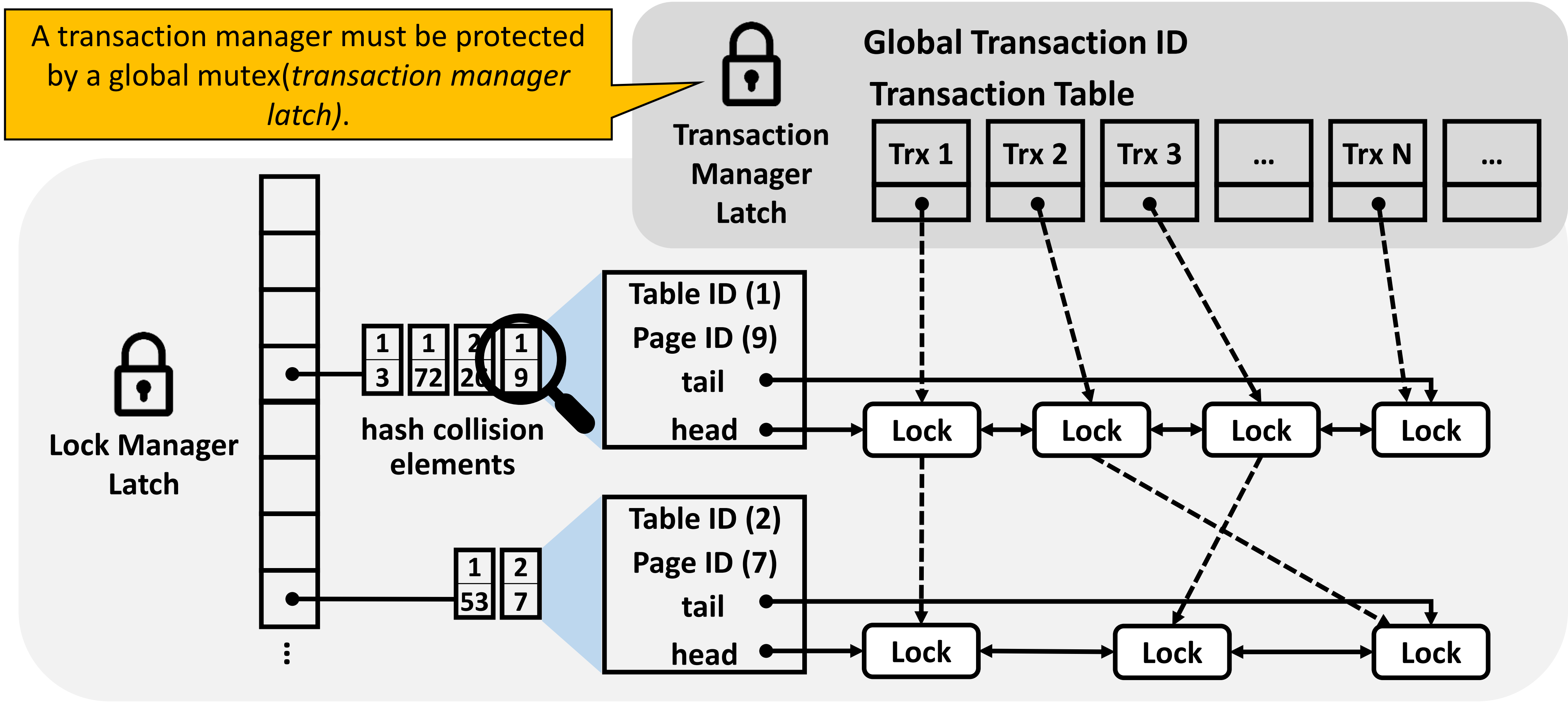


Transaction Manager

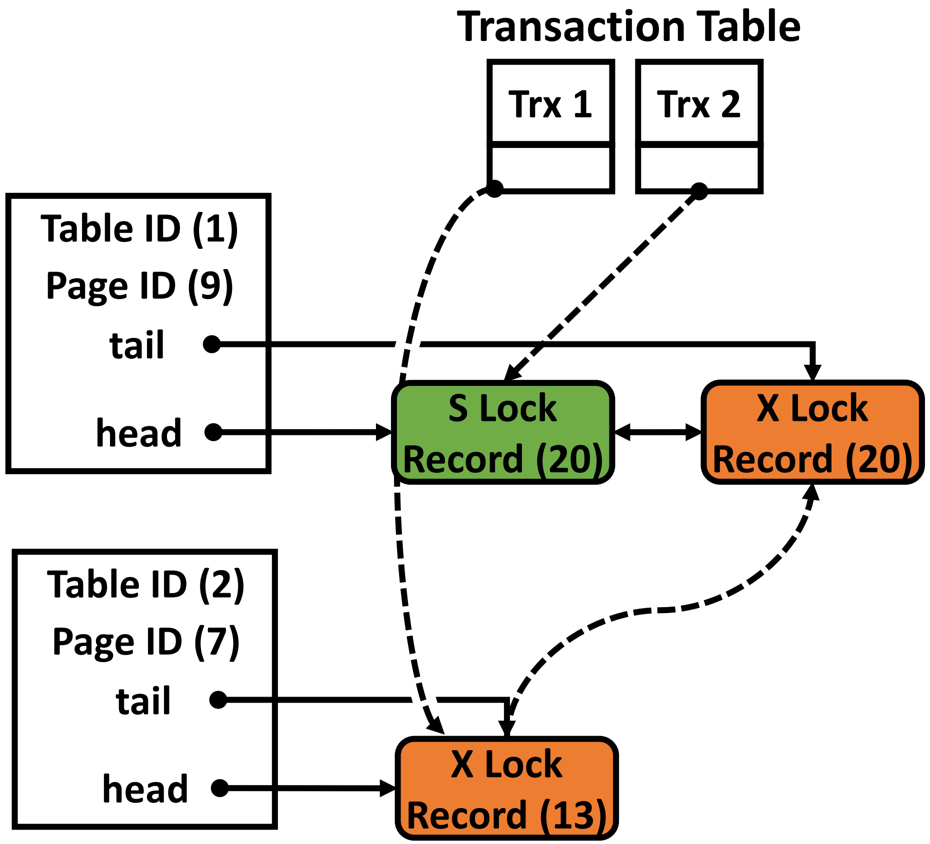


Transaction Manager

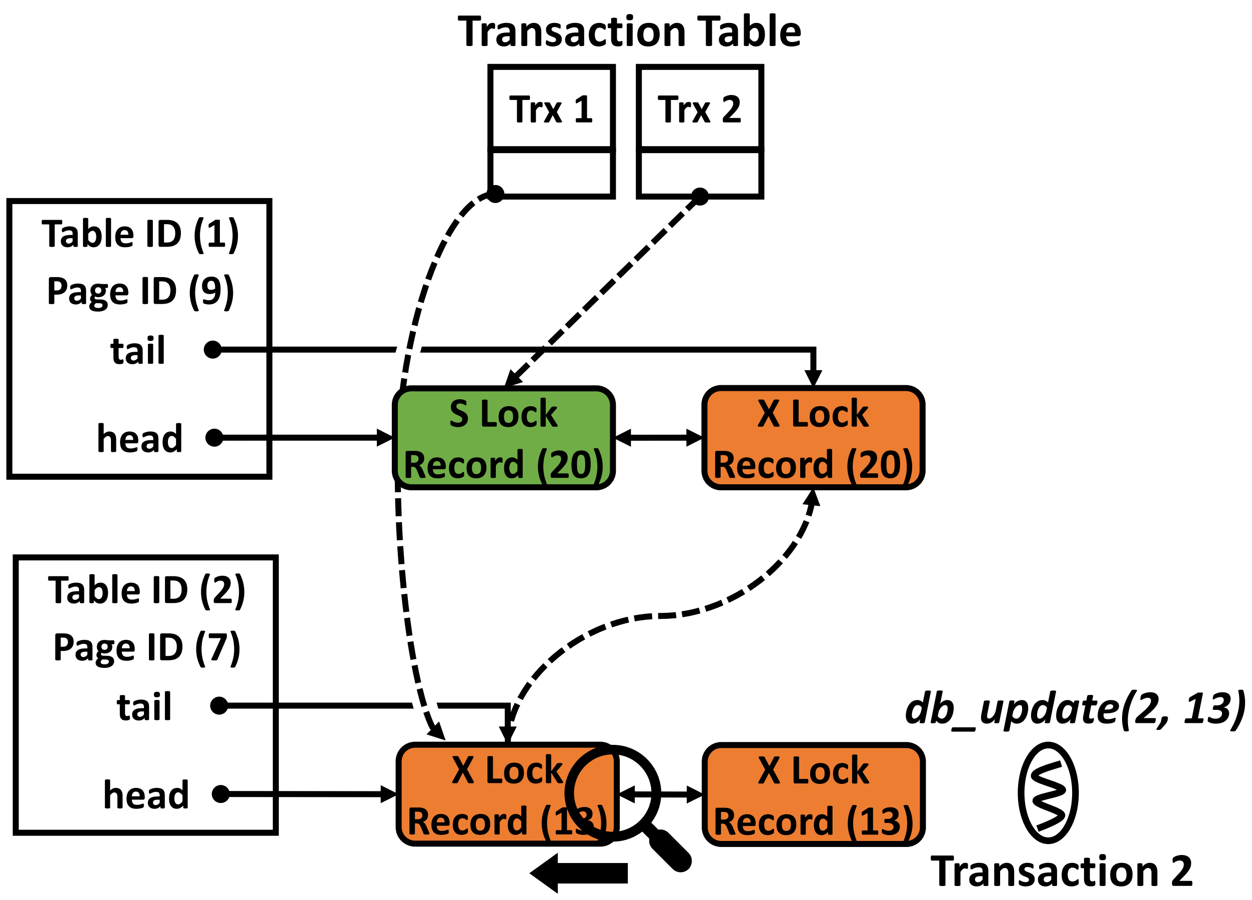
A transaction manager must be protected by a global mutex(*transaction manager latch*).



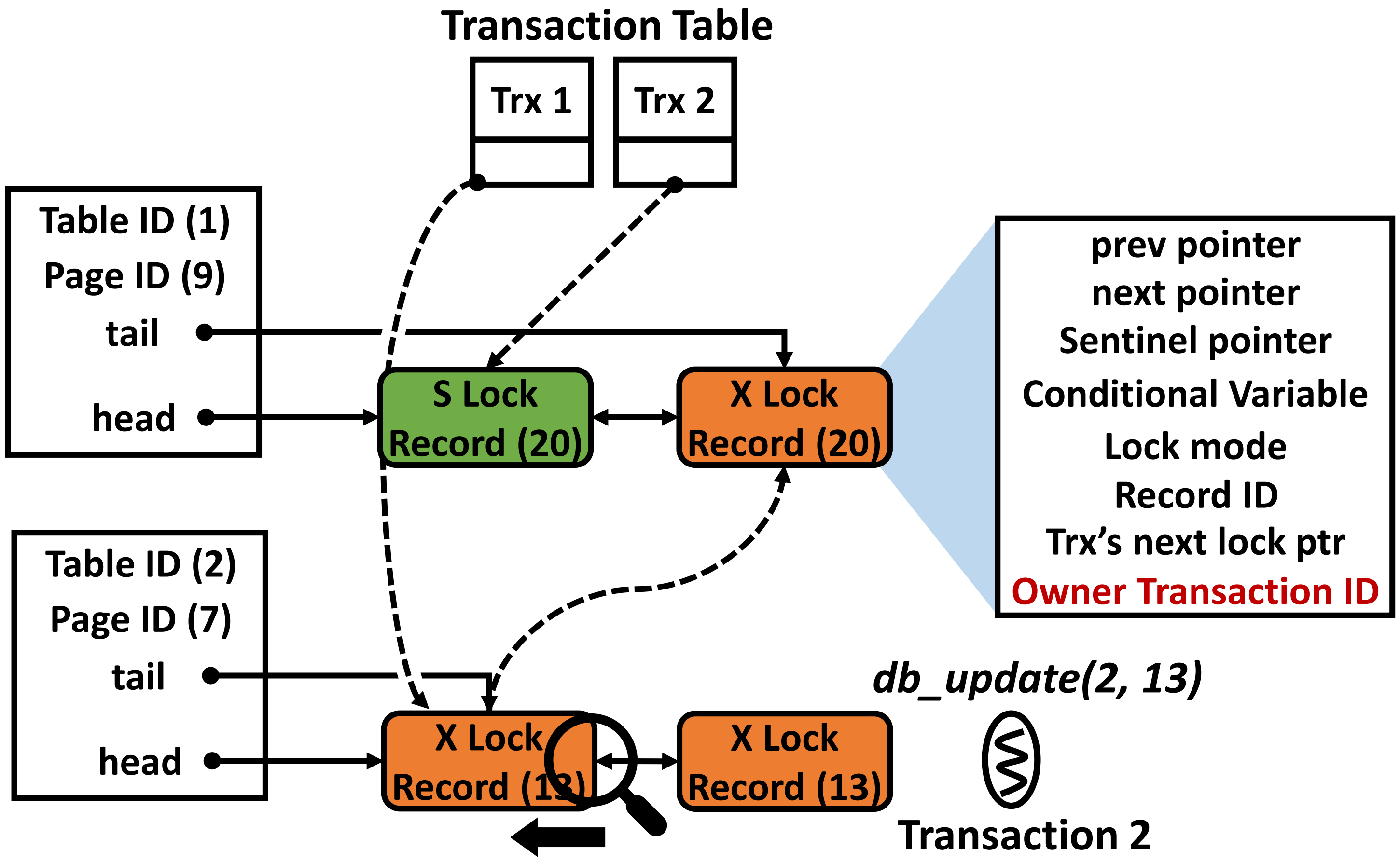
Deadlock Detection



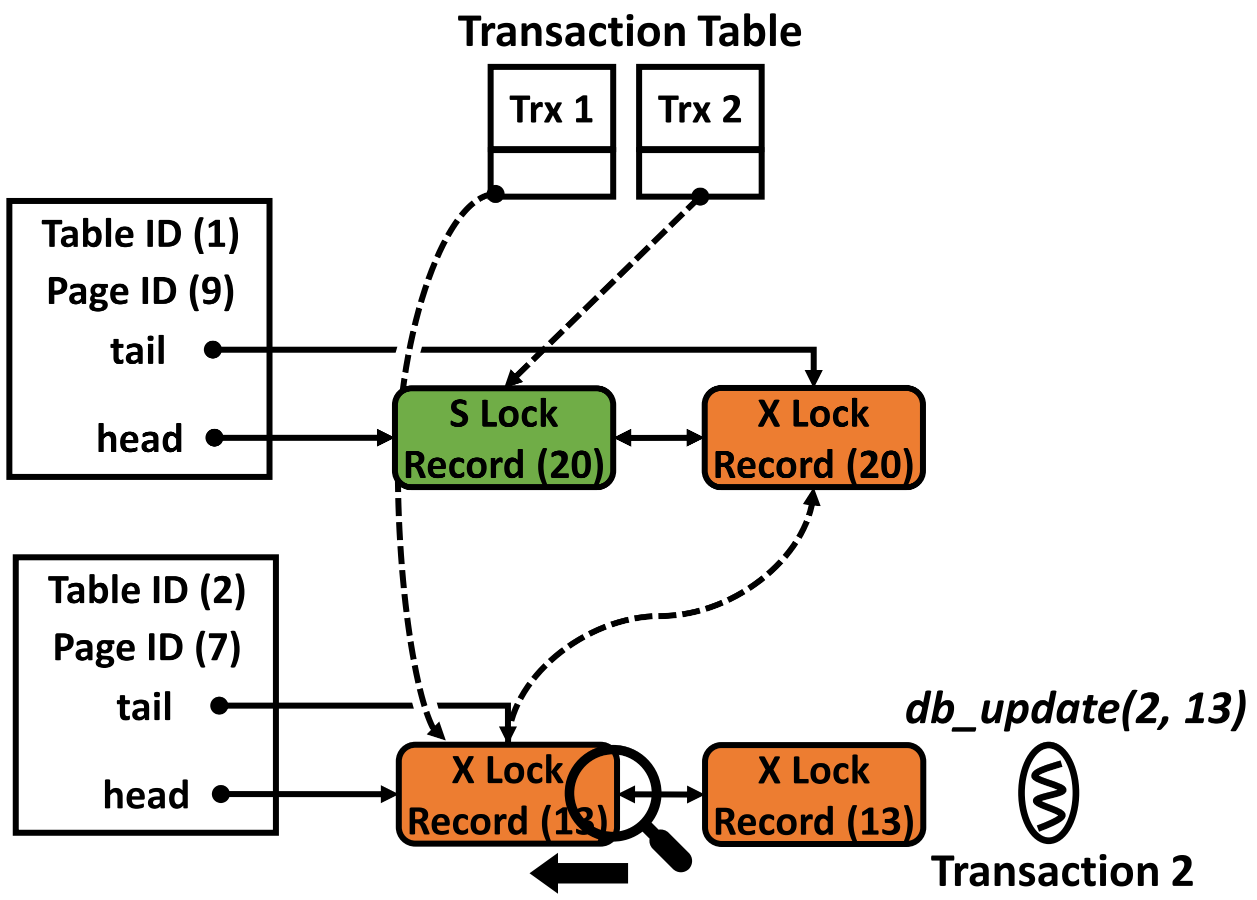
Deadlock Detection



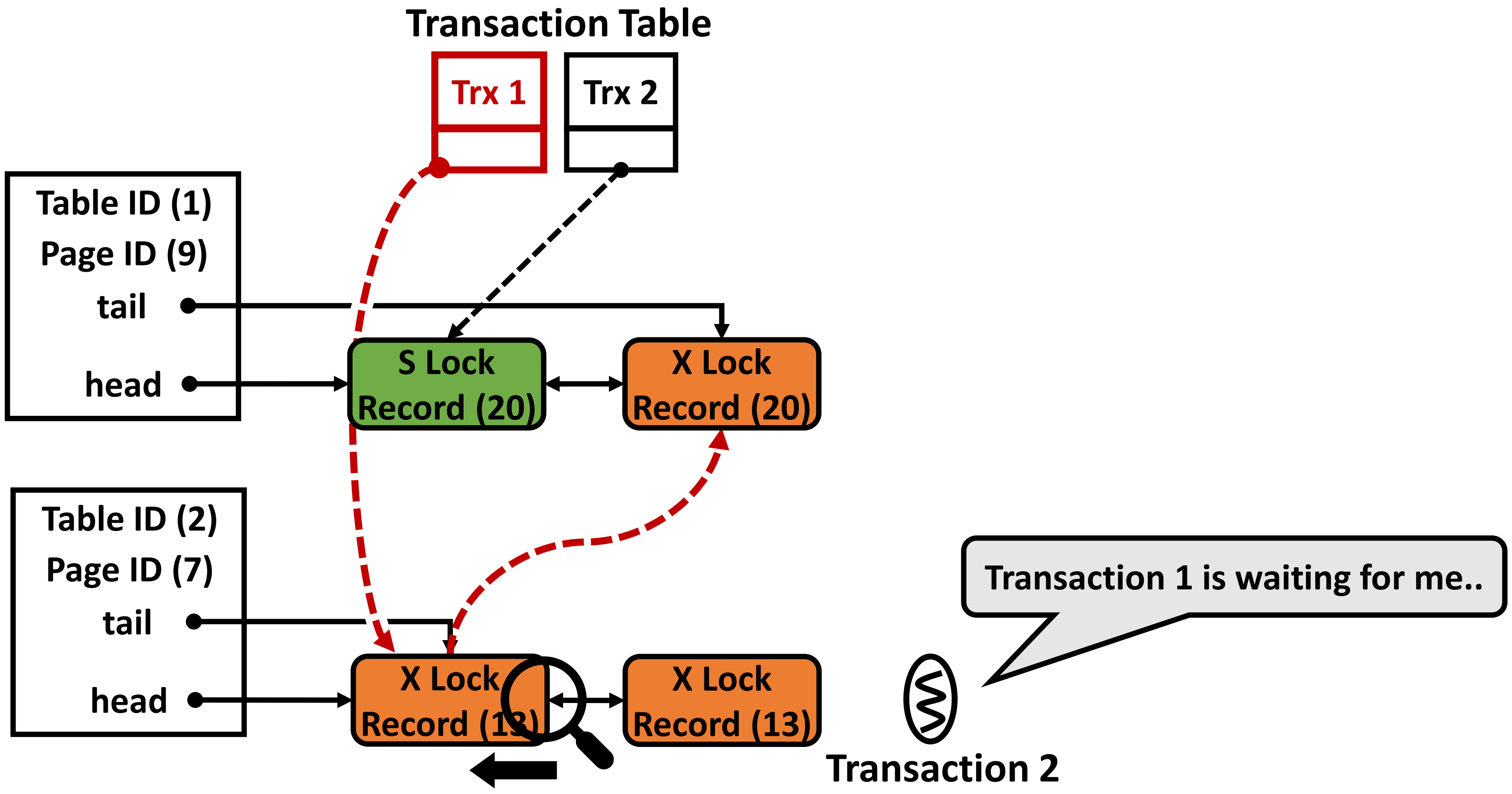
Deadlock Detection



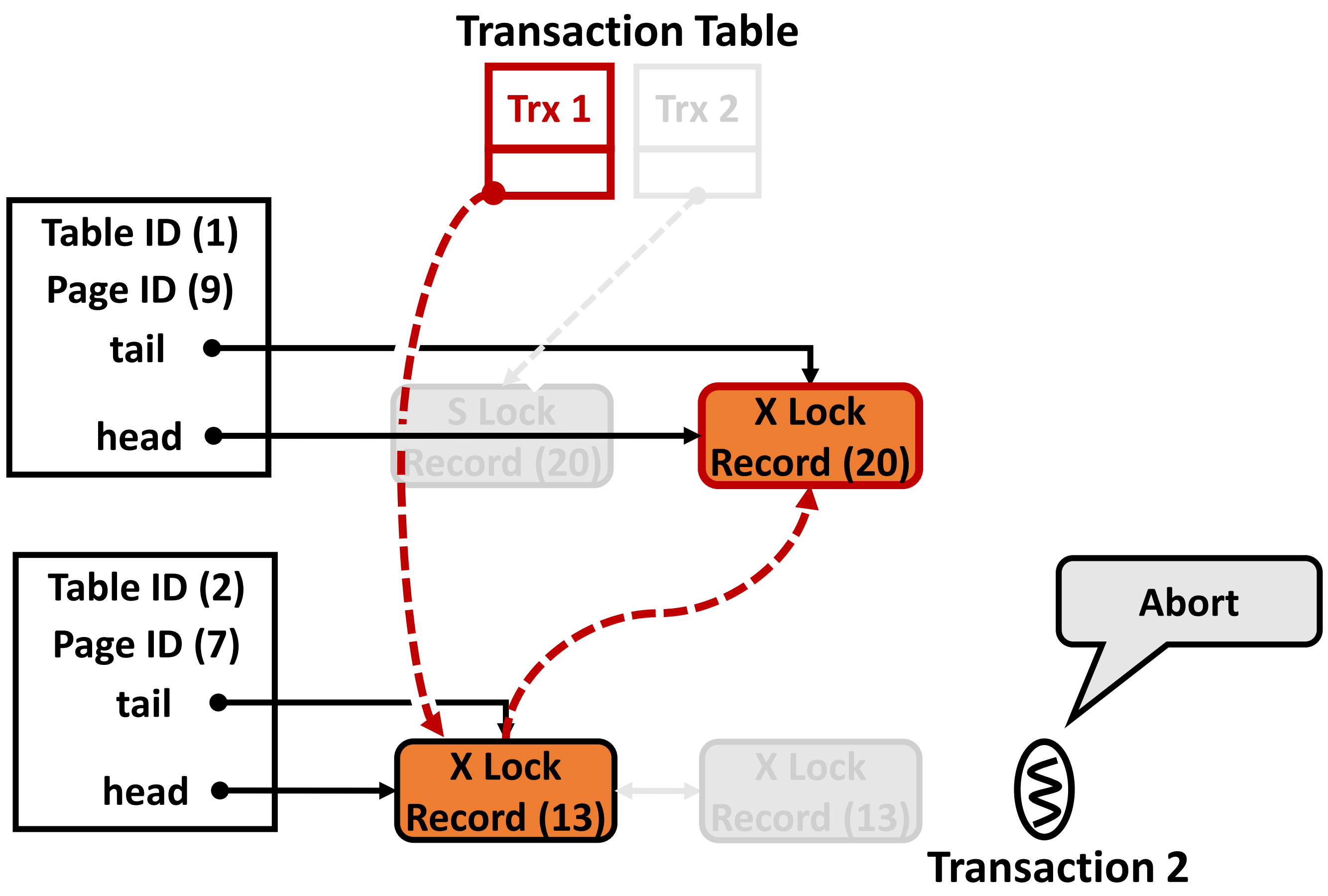
Deadlock Detection



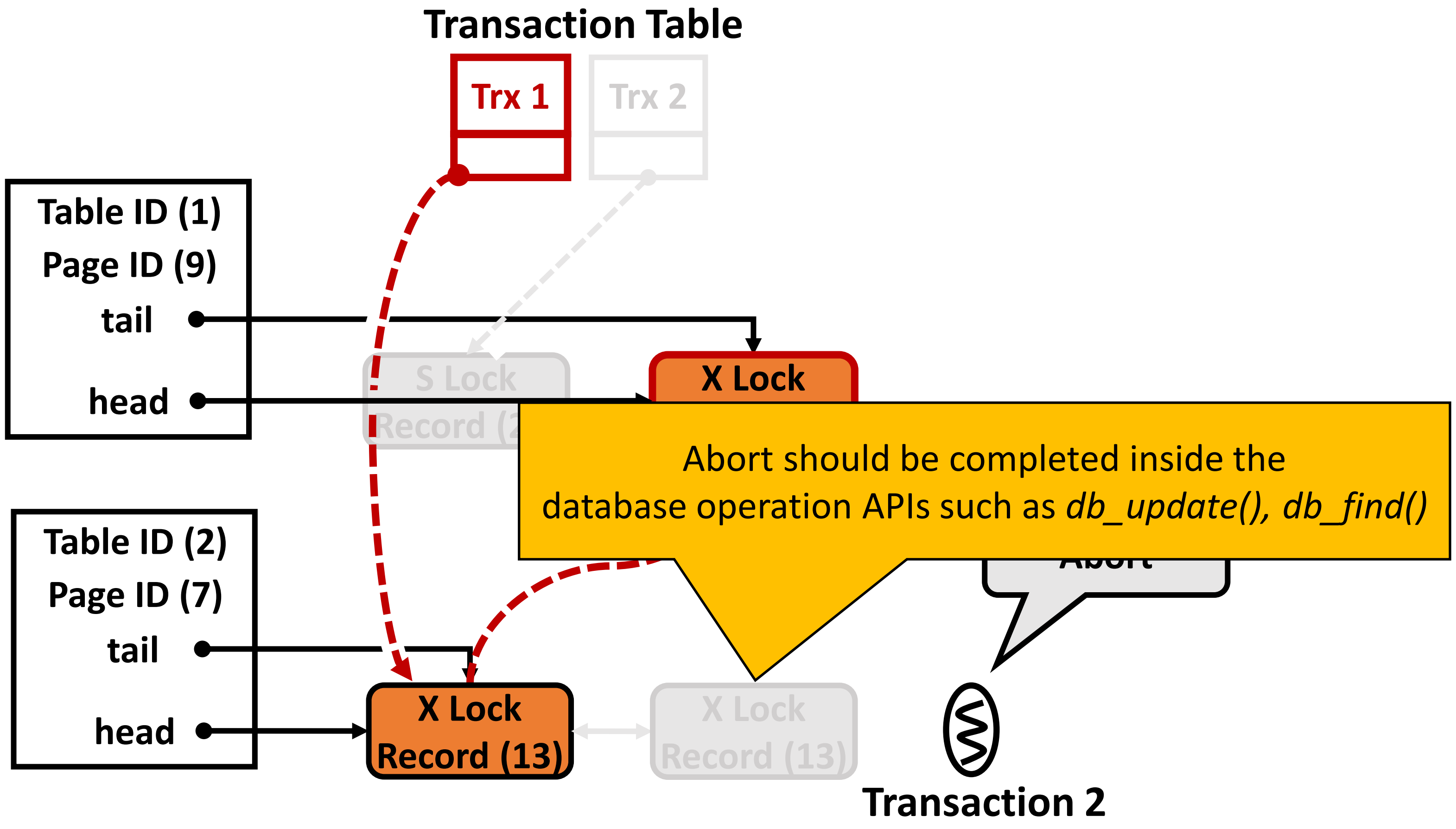
Deadlock Detection



Deadlock Detection

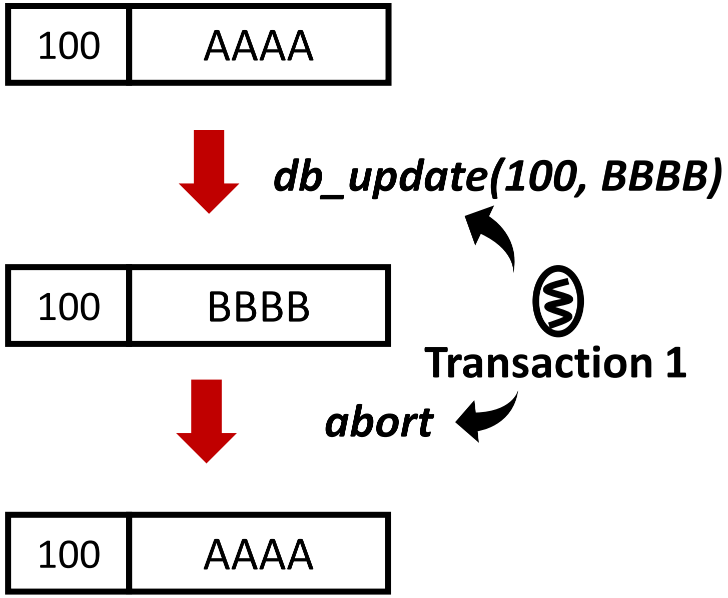


Deadlock Detection

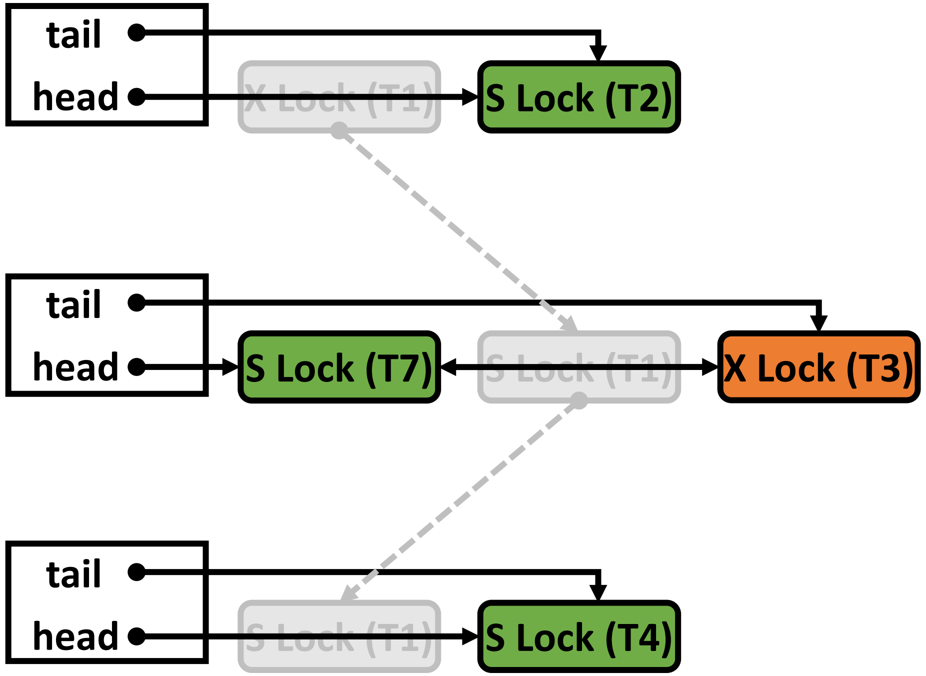


Transaction Abort

1.
Undo all modified records by the transaction

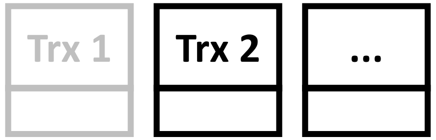


2.
Release all acquired lock objects



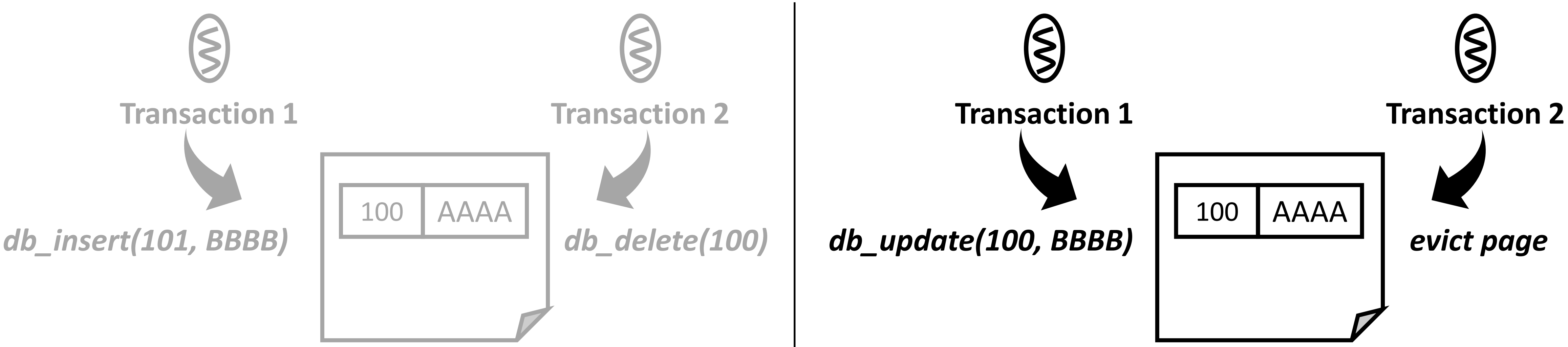
3.
Remove the transaction table entry

Transaction Table



Buffer Manager Issues

- Your buffer manager must correctly work under concurrent accesses by multiple transactions.

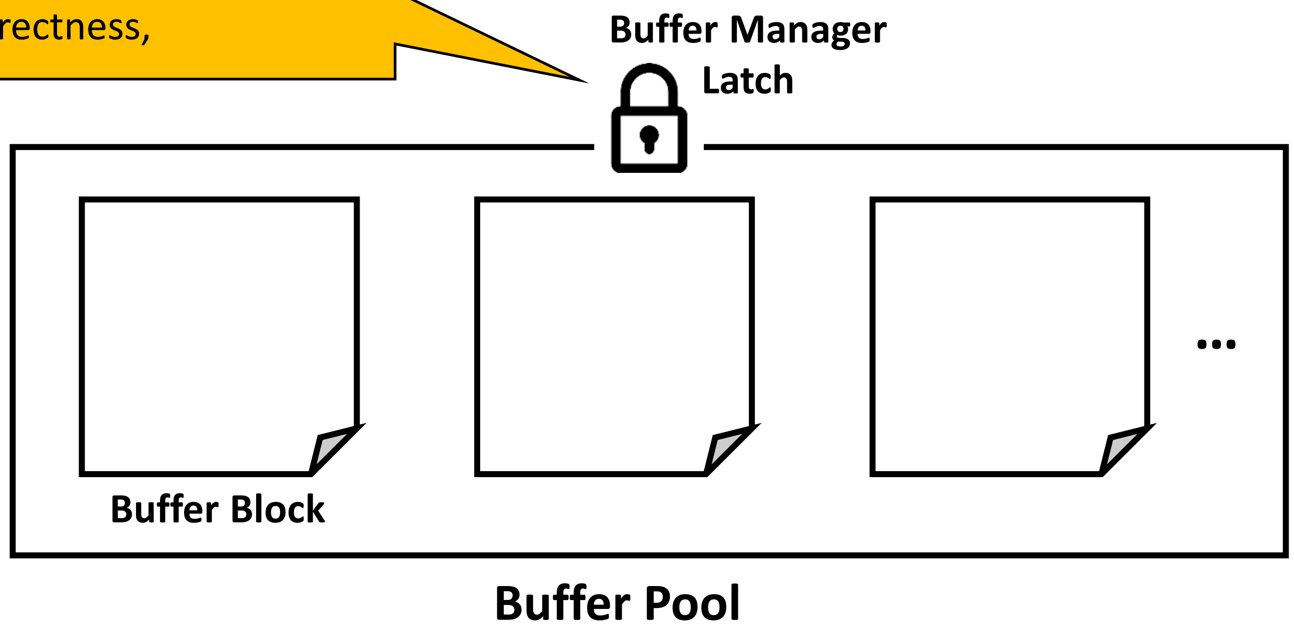


We do not have to consider concurrent db_delete() or db_insert() operations in this project.

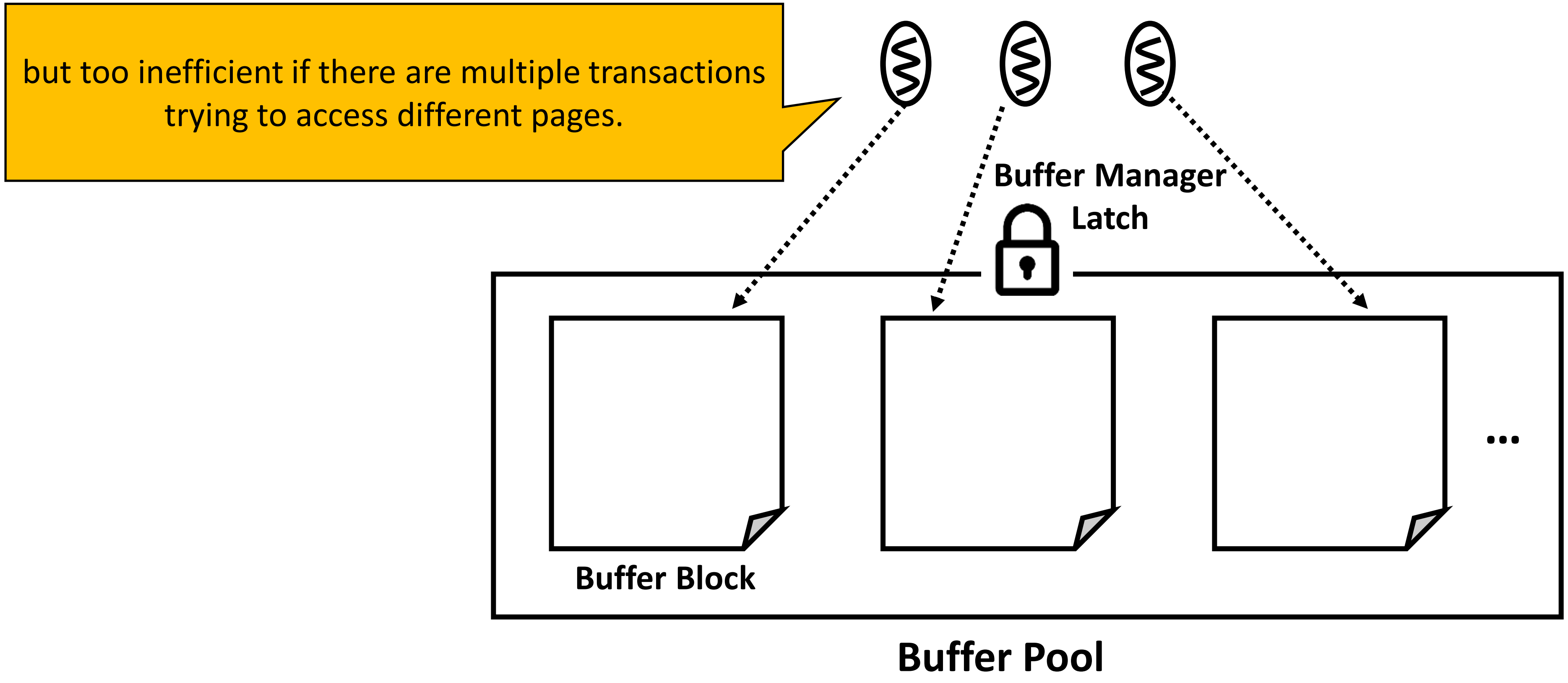
However, we still have to deal with concurrent accesses by multiple transactions.

Buffer Manager Issues

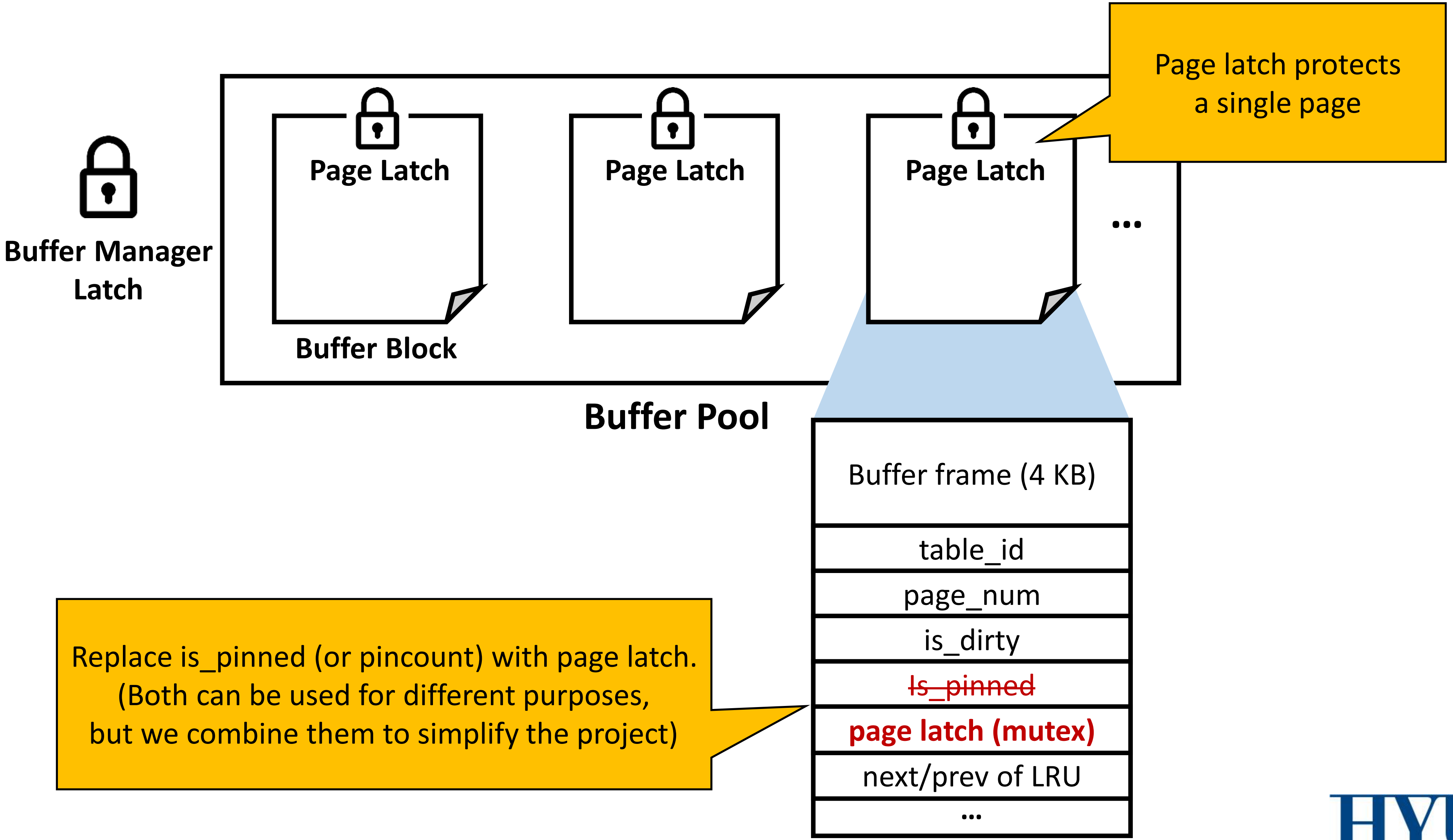
It is enough to safely protect the entire buffer pool
by using a global **buffer manager latch**
for satisfying correctness,



Buffer Manager Issues



Page Latch

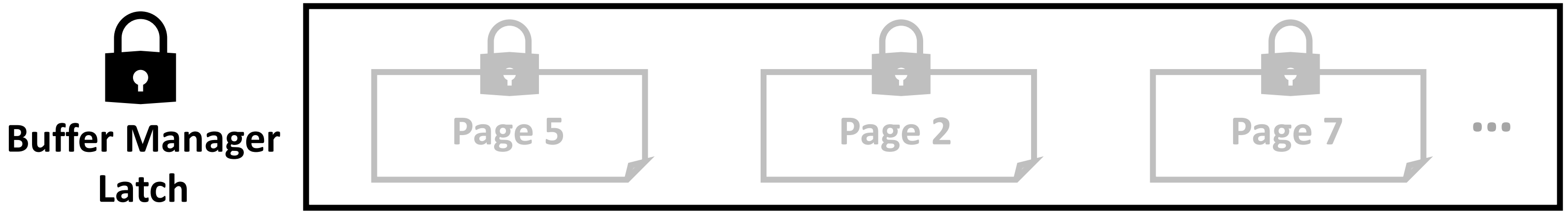

Page latch protects a single page
Buffer Manager Latch
Page Latch
Page Latch
Page Latch
…
Buffer Block
Buffer Pool
Buffer frame (4 KB)
table_id
page_num
is_dirty
~~Is_pinned~~
page latch (mutex)
next/prev of LRU
…
Replace is_pinned (or pincount) with page latch.
(Both can be used for different purposes,
but we combine them to simplify the project)

Page Latch

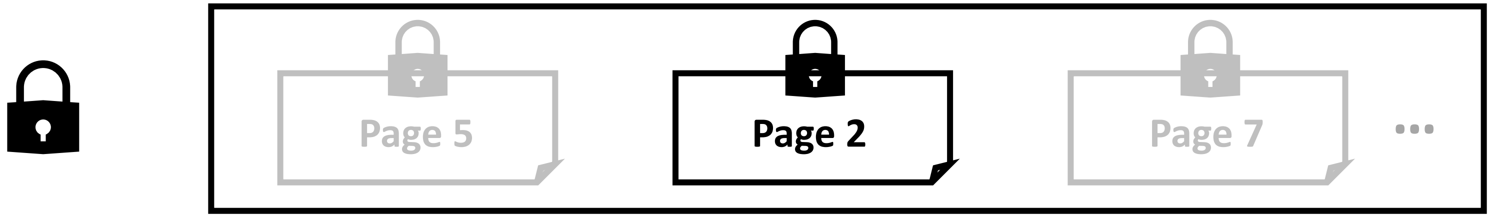
Transaction

Try to access page 2

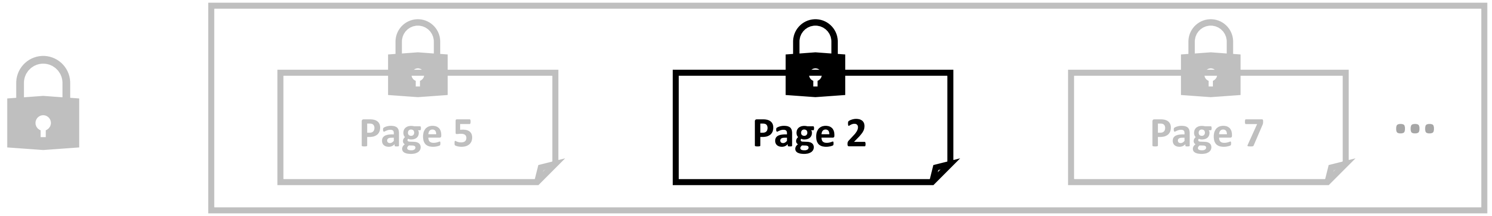
1. Acquire the buffer manager latch



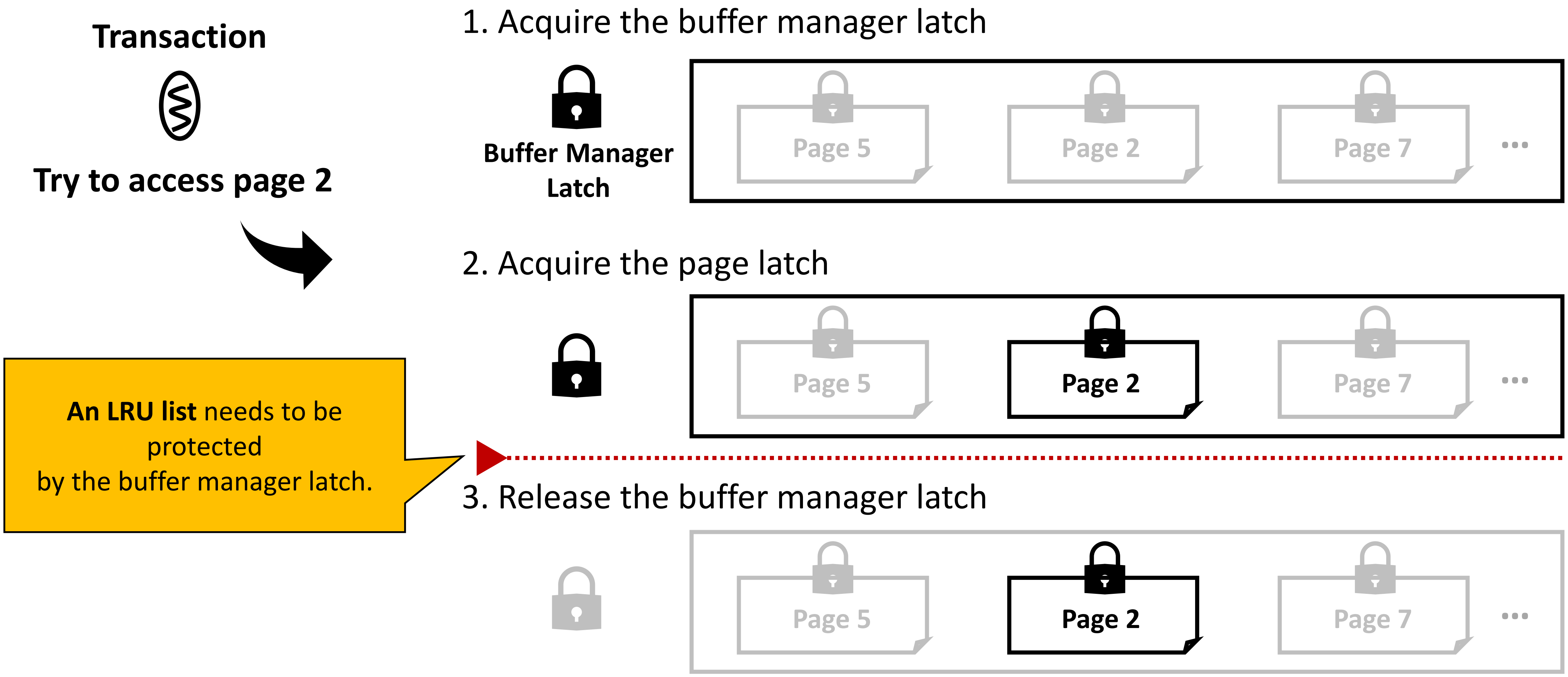
2. Acquire the page latch



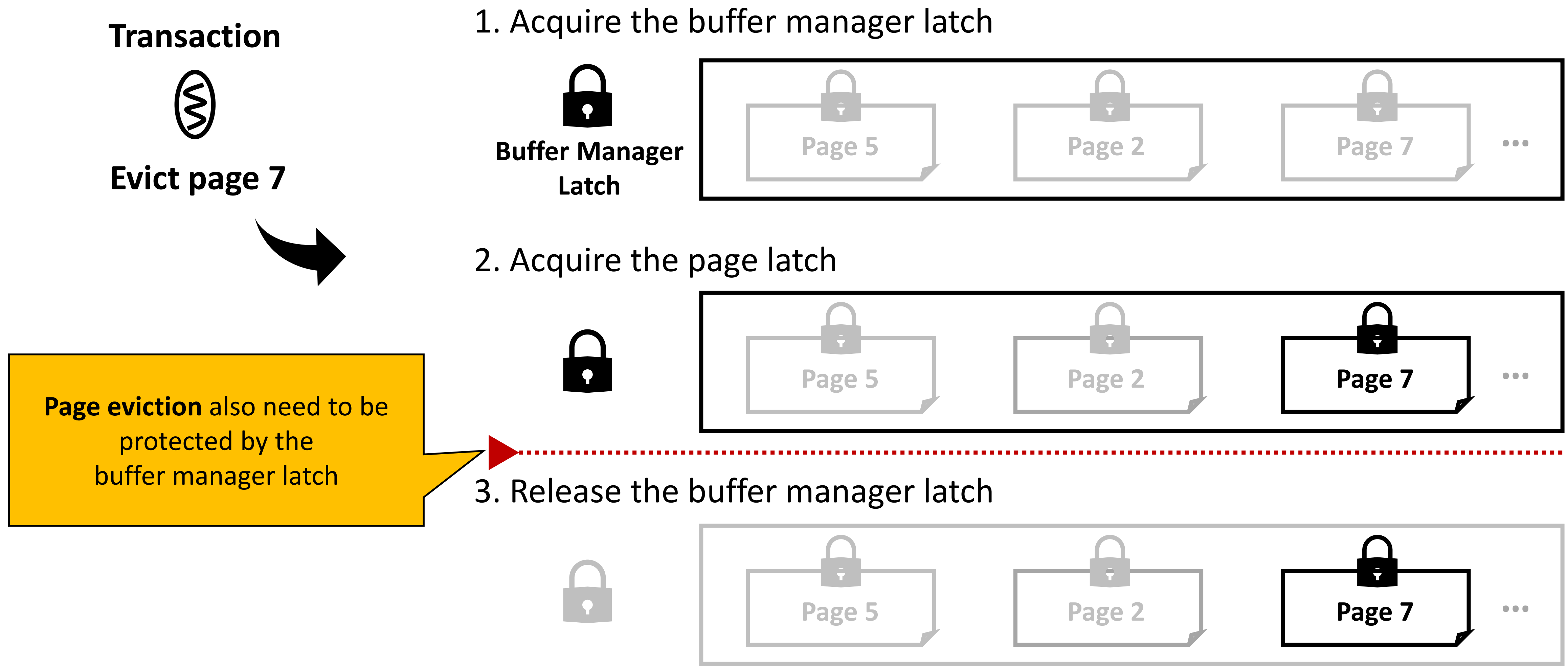
3. Release the buffer manager latch



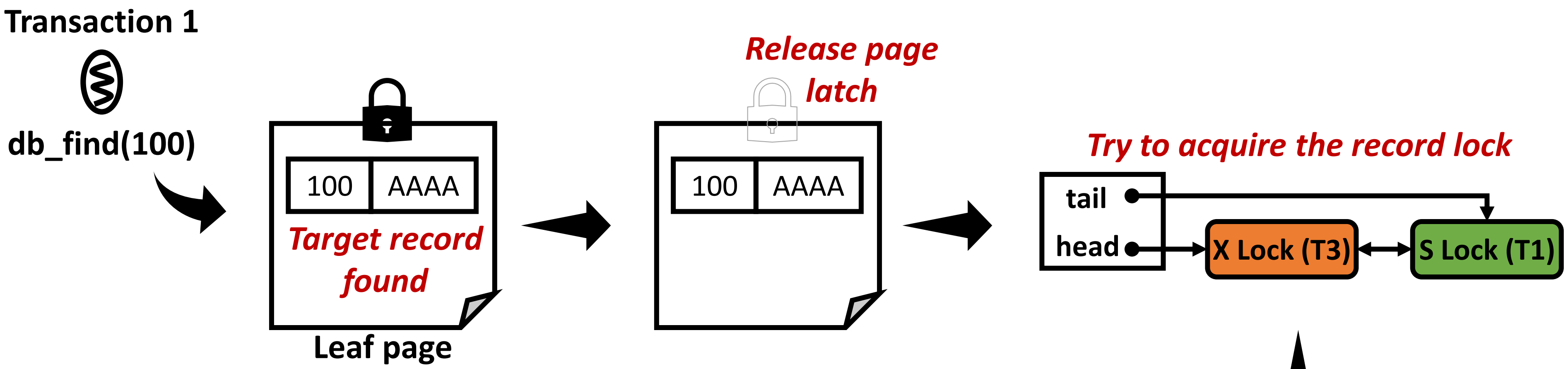
Page Latch

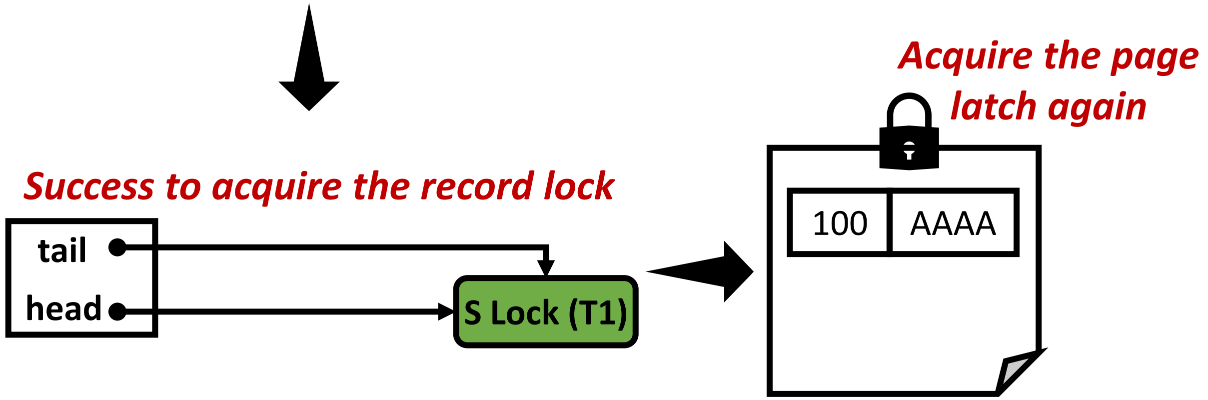

Transaction
Try to access page 2
An LRU list needs to be protected by the buffer manager latch.
1. Acquire the buffer manager latch
Buffer Manager Latch
Page 5
Page 2
Page 7
…
2. Acquire the page latch
Page 5
Page 2
Page 7
…
3. Release the buffer manager latch
Page 5
Page 2
Page 7
…

Page Latch


Transaction
Evict page 7
1. Acquire the buffer manager latch
Buffer Manager
Latch
Page 5
Page 2
Page 7
...
2. Acquire the page latch
Page 5
Page 2
Page 7
...
Page eviction also need to be
protected by the
buffer manager latch
3. Release the buffer manager latch
Page 5
Page 2
Page 7
...

Page Latch & Record Lock

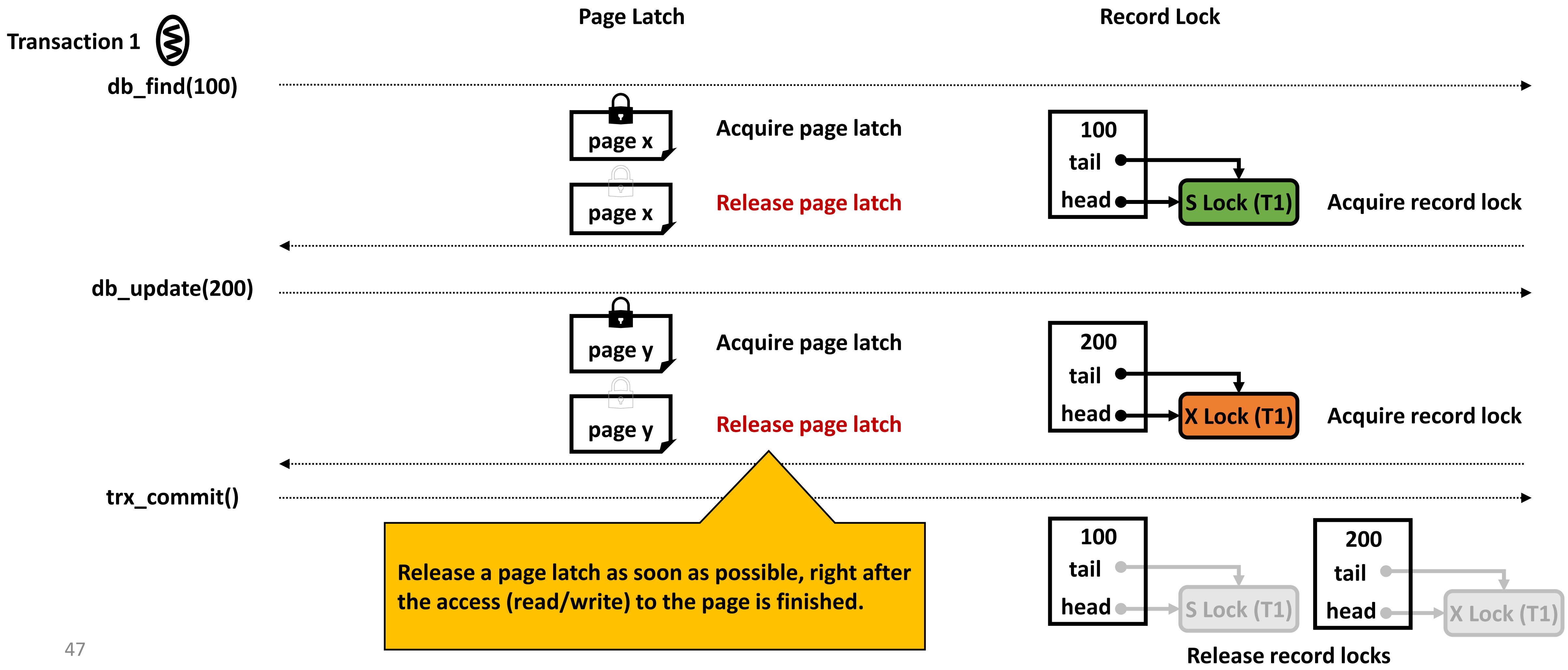




After a transaction finds the leaf page containing the target record,
release the page latch before acquiring the record lock.
Otherwise, no one can access the page until the transaction acquires the record lock.
Even worse, a deadlock could occur.
After the transaction success to acquire the record lock, acquire the page latch again for doing the actual operation. (find / update)

Page Latch & Record Lock

- Page latch duration vs Record lock duration



Wiki

- Your wiki should contain descriptions about
  - lock mode (shared & exclusive),
  - deadlock detection,
  - abort and rollback,
  - and whatever you want to describe.

Milestone2

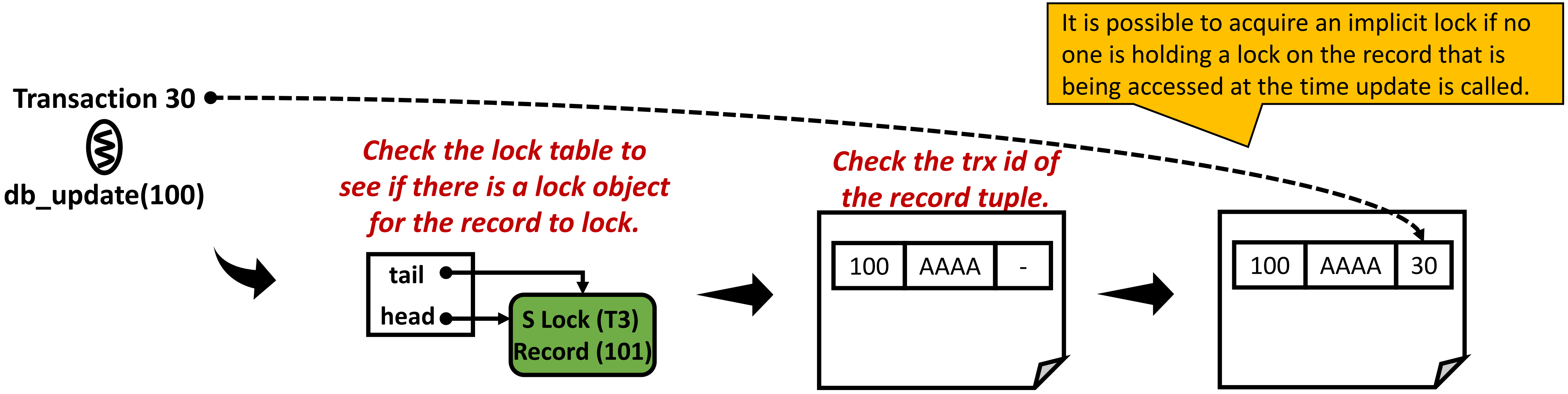
Two Optimization Techniques for the Lock Manager

Optimizing Your Lock Manager

- The currant lock manager supports **textbook definitions of concurrency control** faithfully.
- In some workloads, your lock manager would face **space overhead**, i.e., when a transaction **updates or reads 1 billion rows from a huge table**.
- The above case would definitely incur the following problems:
  - It increases **space overhead** by allocating 1 billion lock objects in the table.
  - It increases **time complexity** by dynamically allocating memory when acquiring a lock.
- To mitigate the fundamental issue, you have to implement two optimization techniques in milestone2.
  1. **Implicit locking** - optimization for reducing space overhead for exclusive locks
  2. **Lock compression** – optimization for reducing space overhead for shared locks

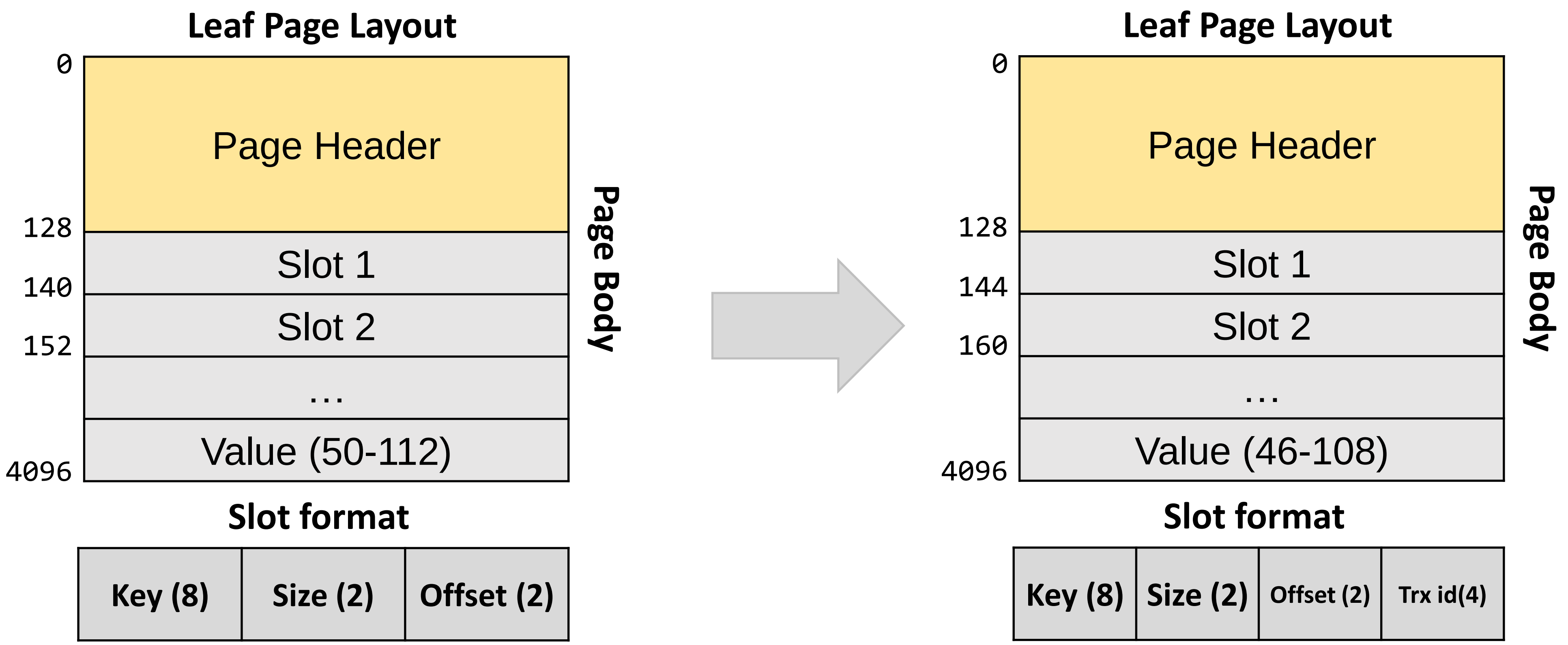
Implicit Locking

- **"Implicit" locking** enables a transaction to acquire an "exclusive lock" by simply writing a transaction id in a record **without "explicitly" inserting a lock object into the lock hash table**, if the transaction knows that it is the first transaction who can safely hold an exclusive lock on the corresponding record **because no transaction currently accesses the record**.
- **Converting an implicit lock to an explicit one should be done by other conflicting transaction when it detects that the owner of the implicit lock is still alive.**
  - **To this end, a transaction needs to check whether trx_id in a record is still alive by looking up the transaction table.**



Implicit Locking

- To add the trx id field to the tuple slot, the layout of the leaf page must be modified.

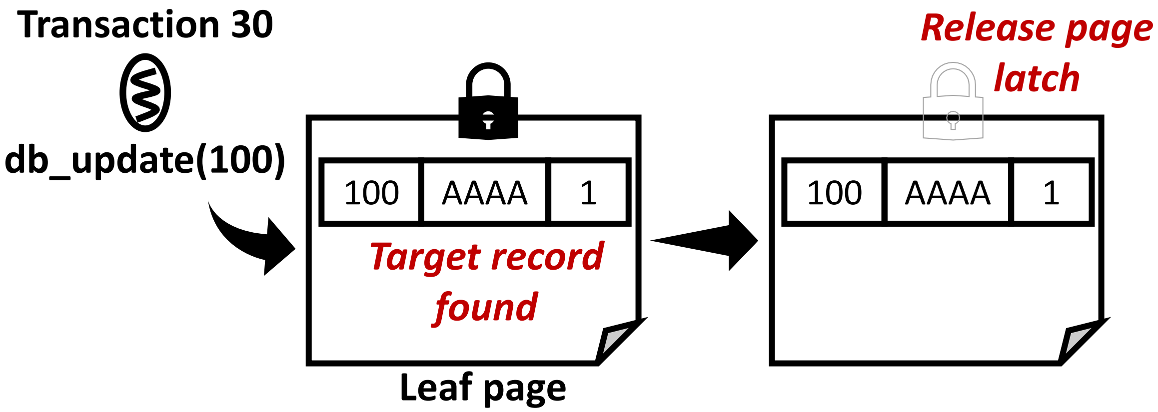


Implicit Locking

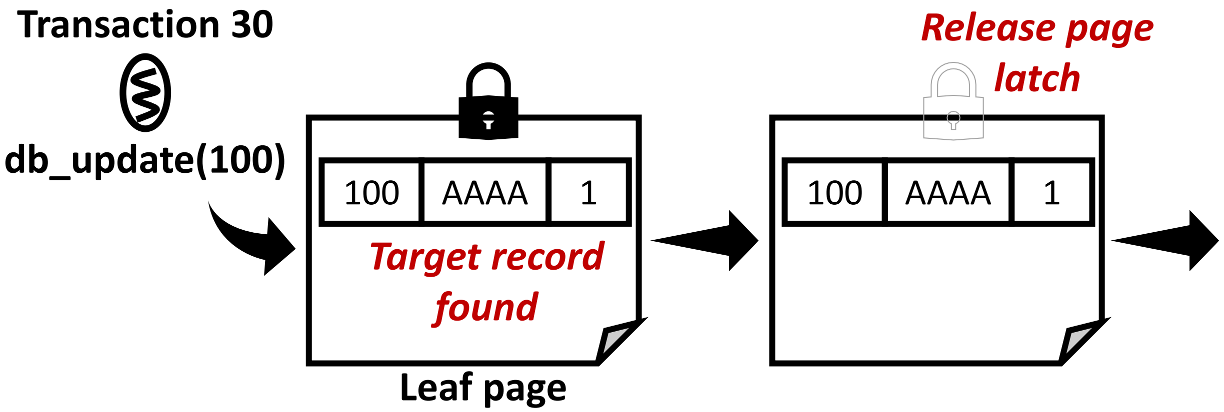
Transaction 30

db_update(100)

Implicit Locking

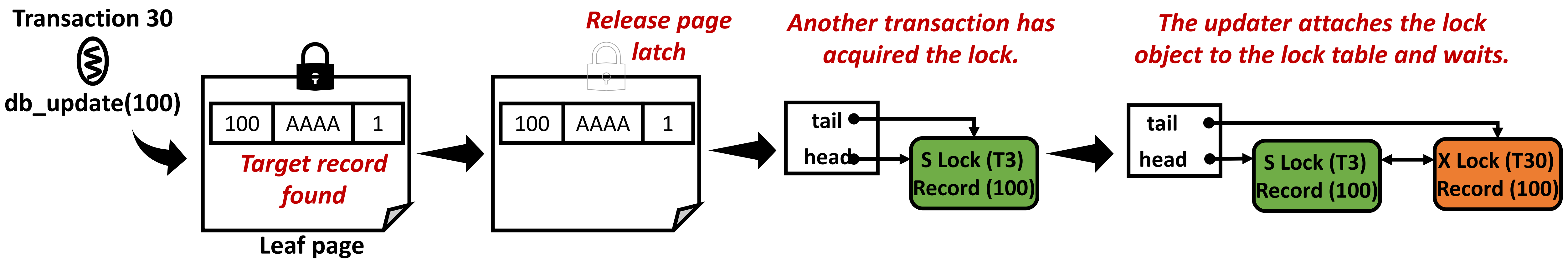


Implicit Locking

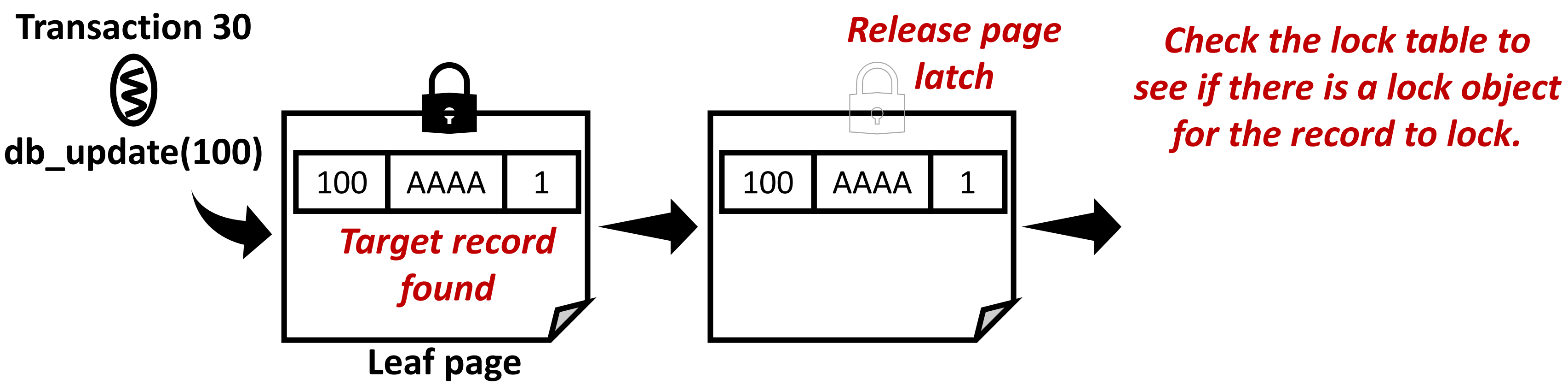


Check the lock table to see if there is a lock object for the record to lock.

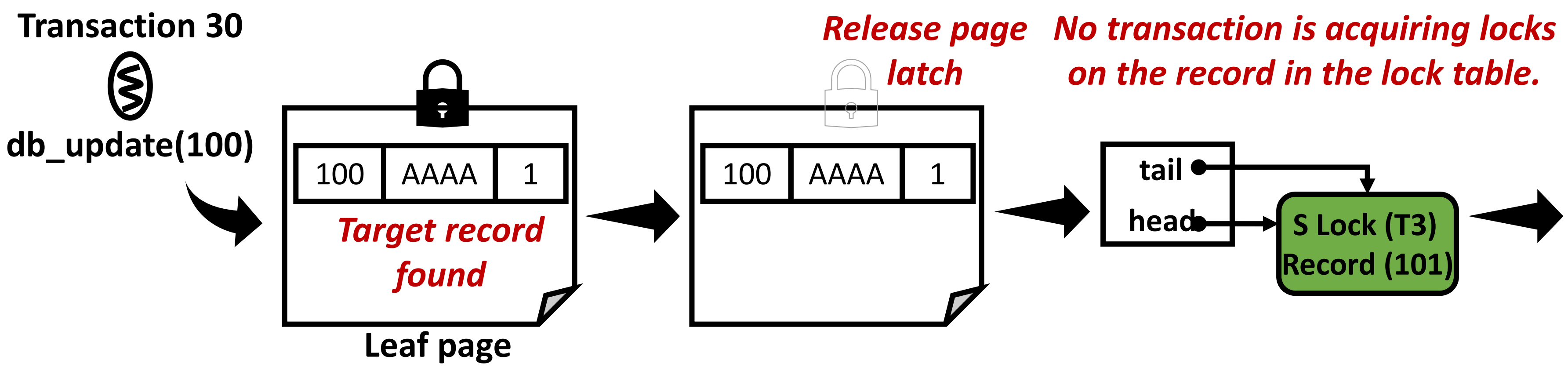
Implicit Locking



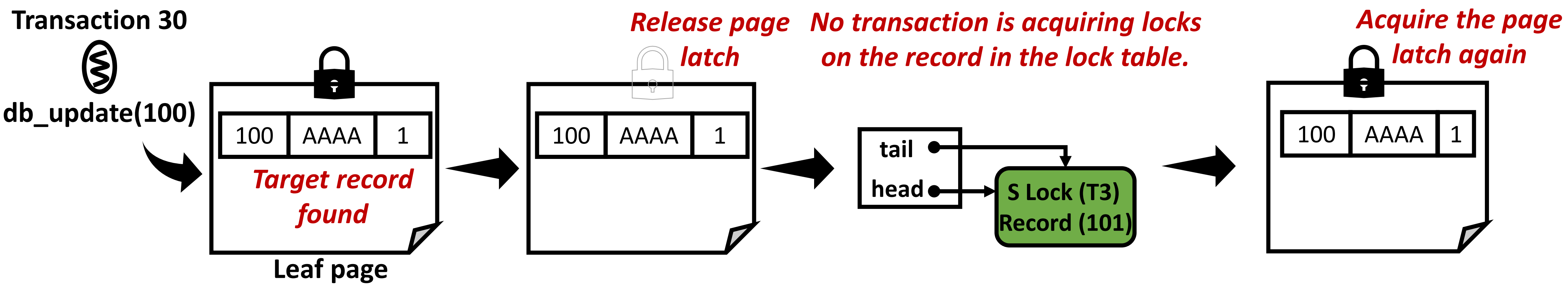
Implicit Locking



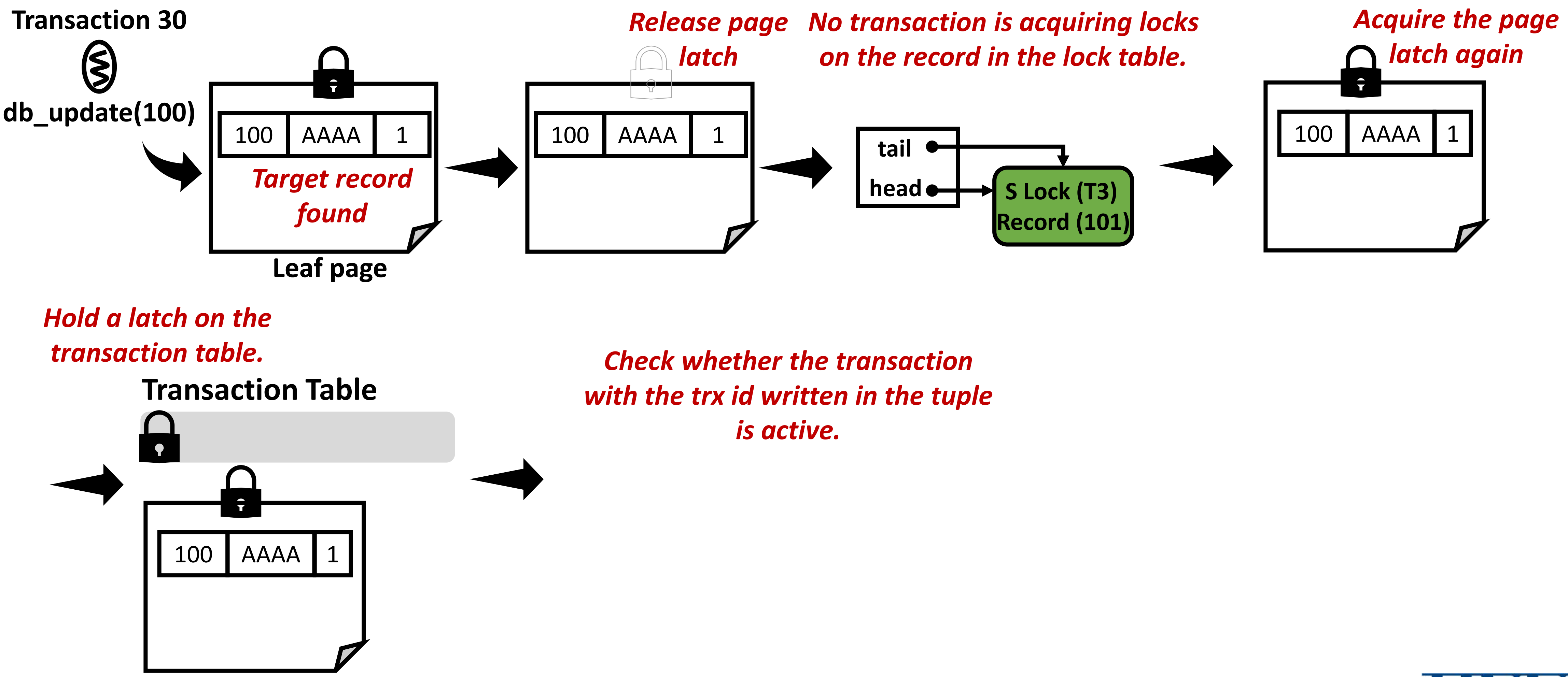
Implicit Locking



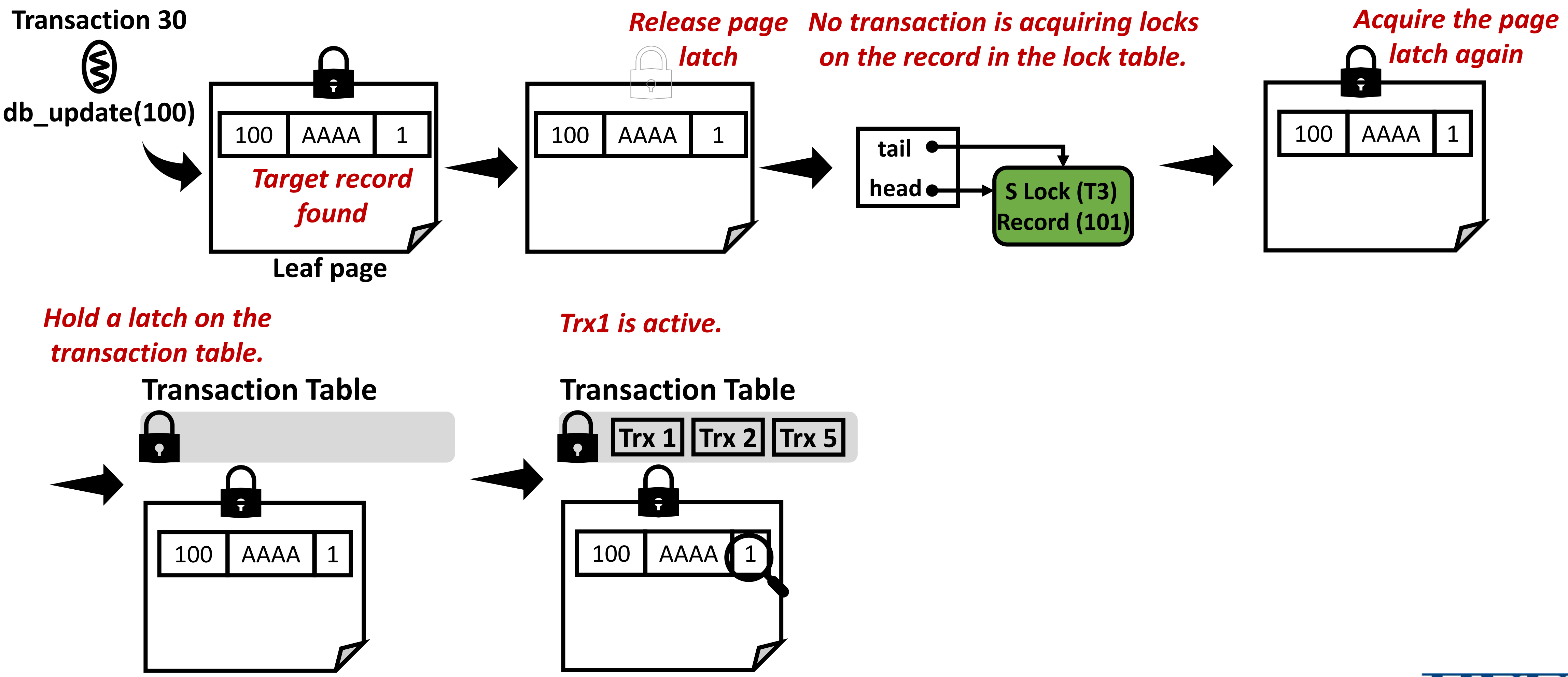
Implicit Locking



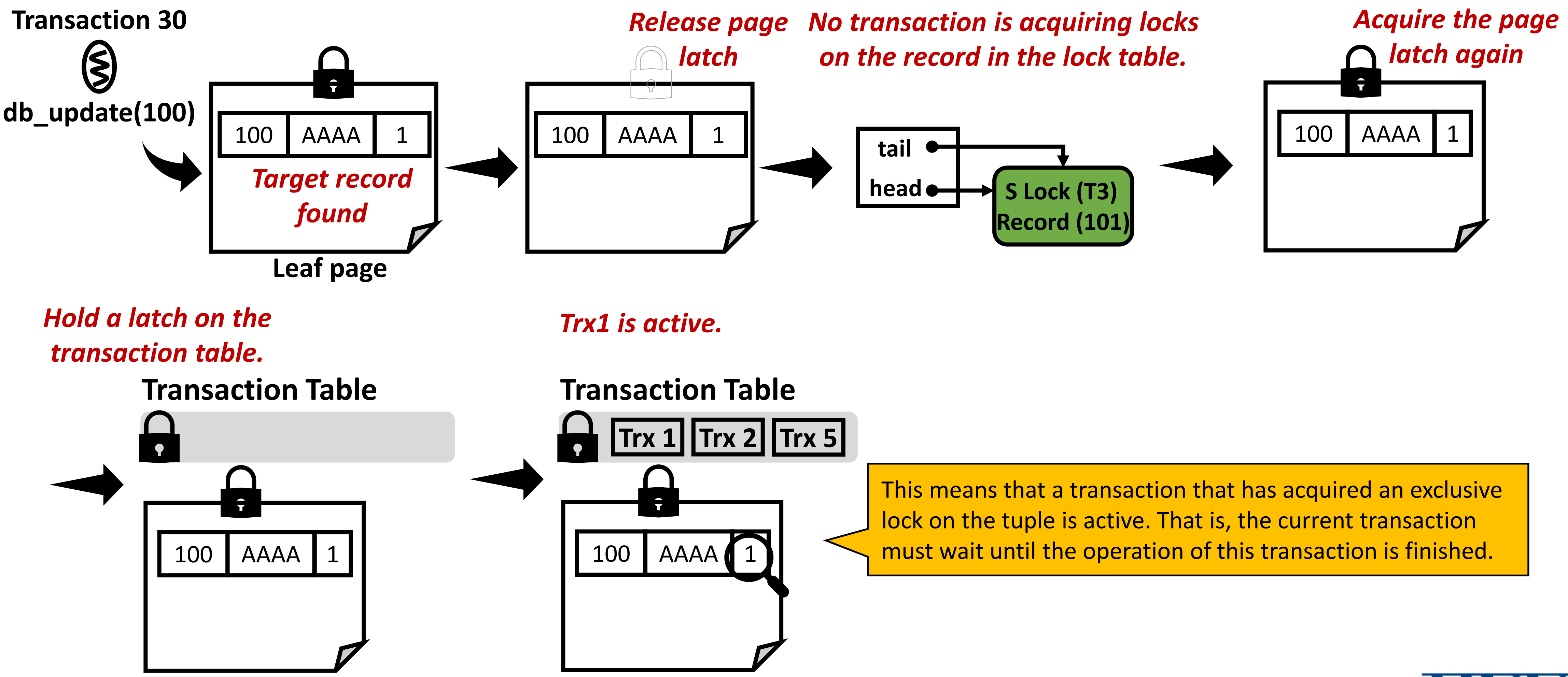
Implicit Locking



Implicit Locking

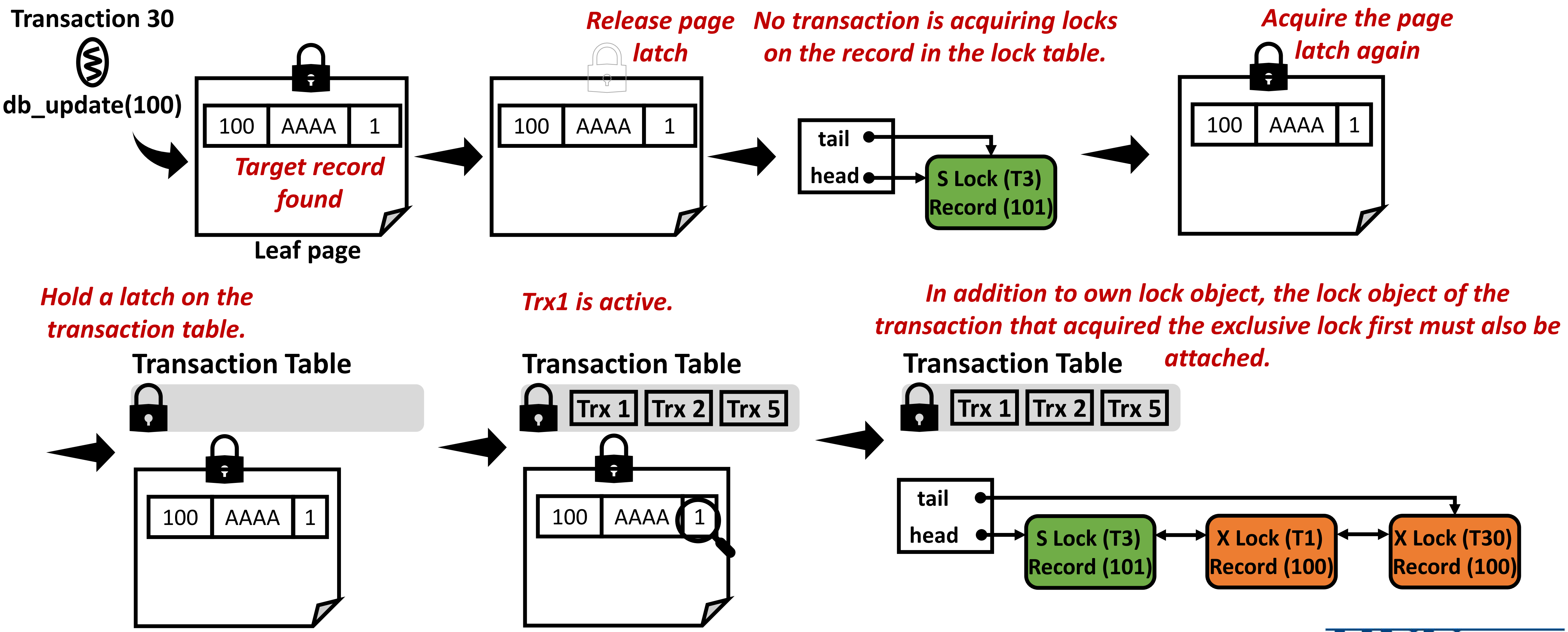


Implicit Locking

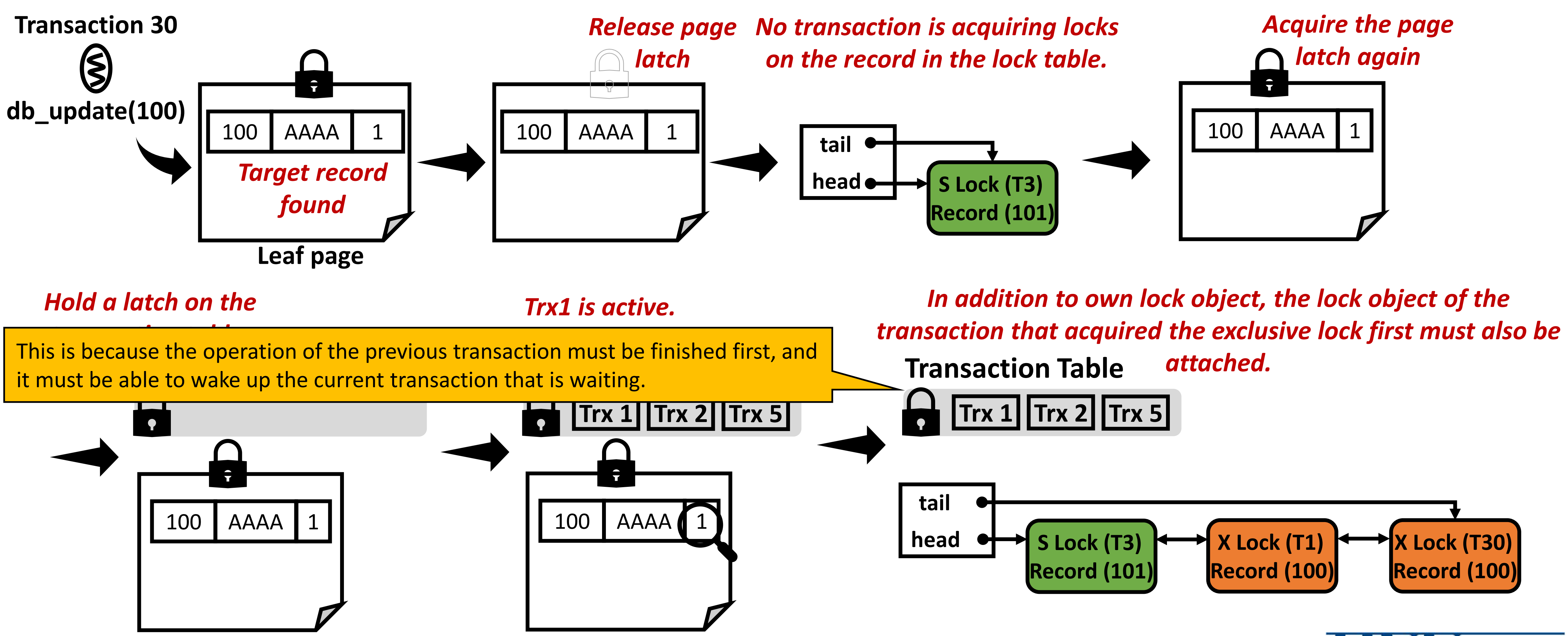


This means that a transaction that has acquired an exclusive lock on the tuple is active. That is, the current transaction must wait until the operation of this transaction is finished.

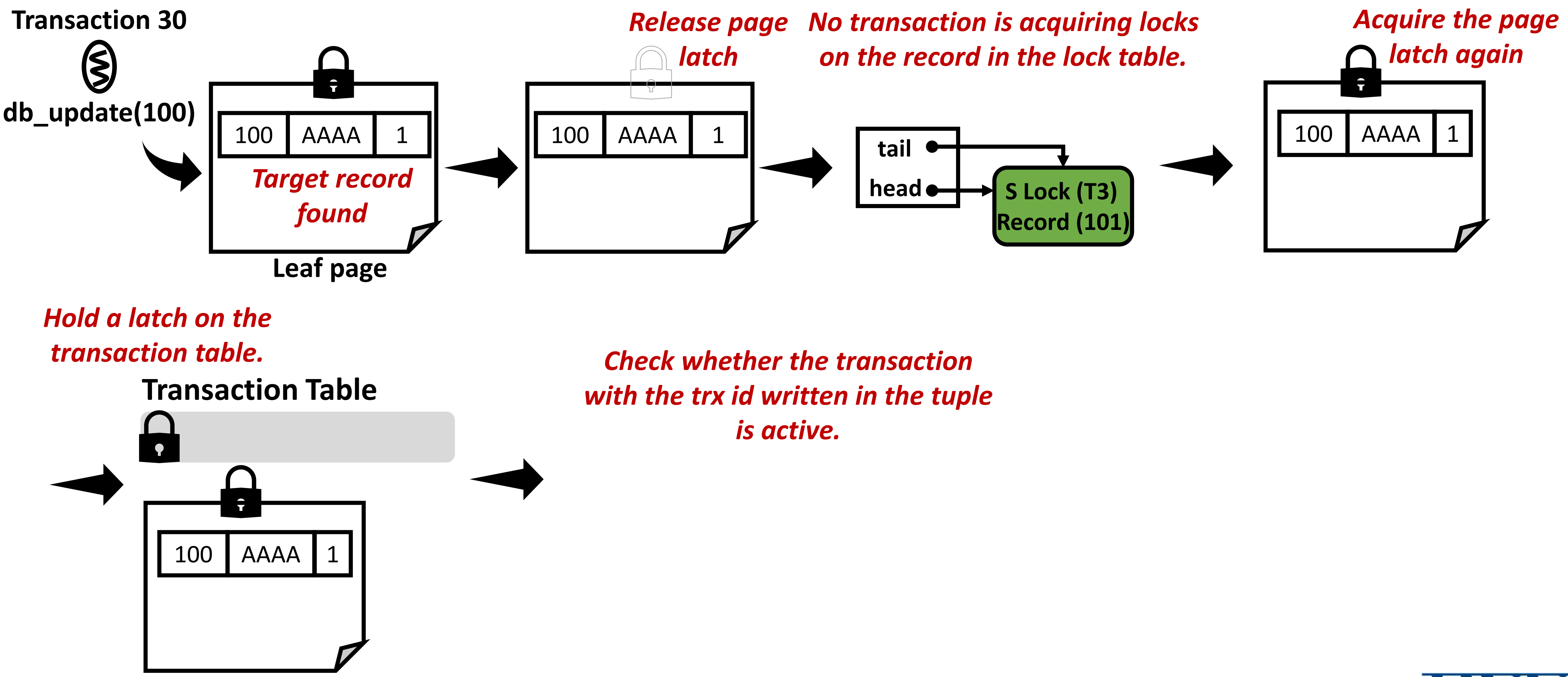
Implicit Locking



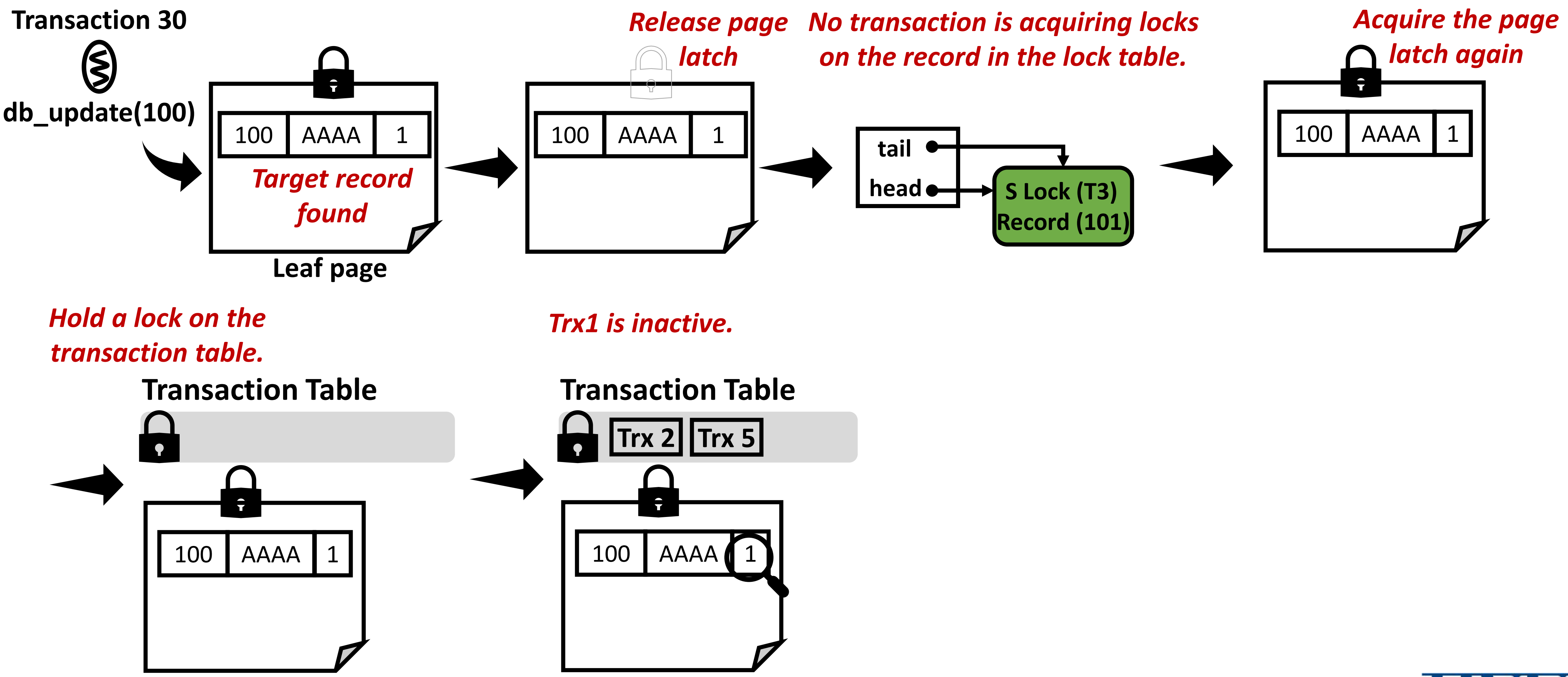
Implicit Locking



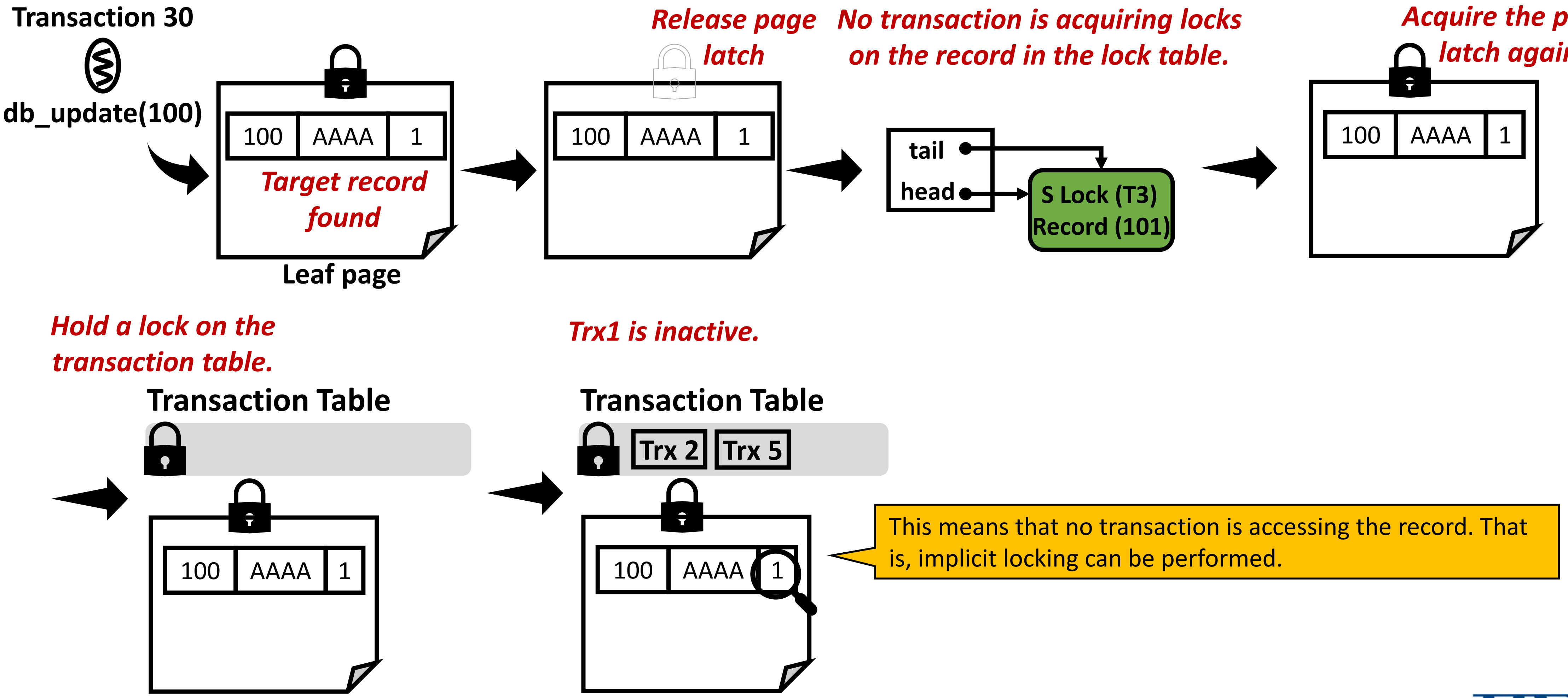
Implicit Locking



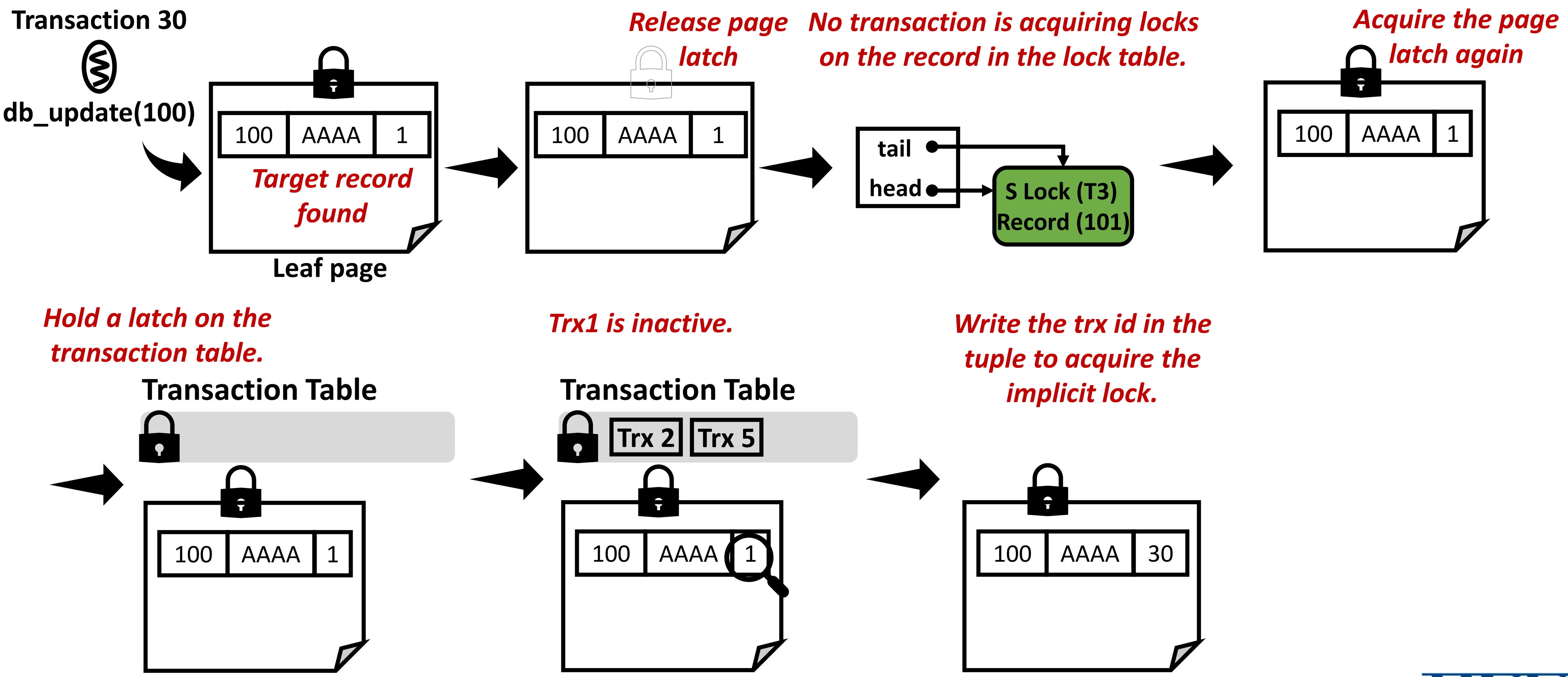
Implicit Locking



Implicit Locking

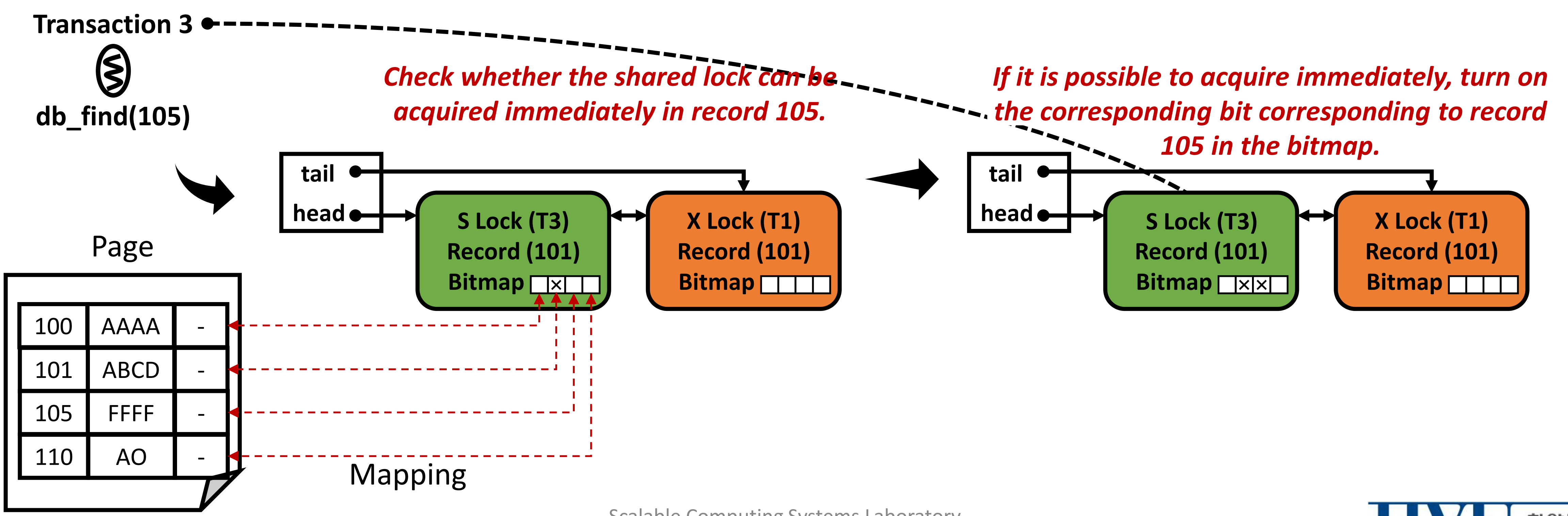


Implicit Locking



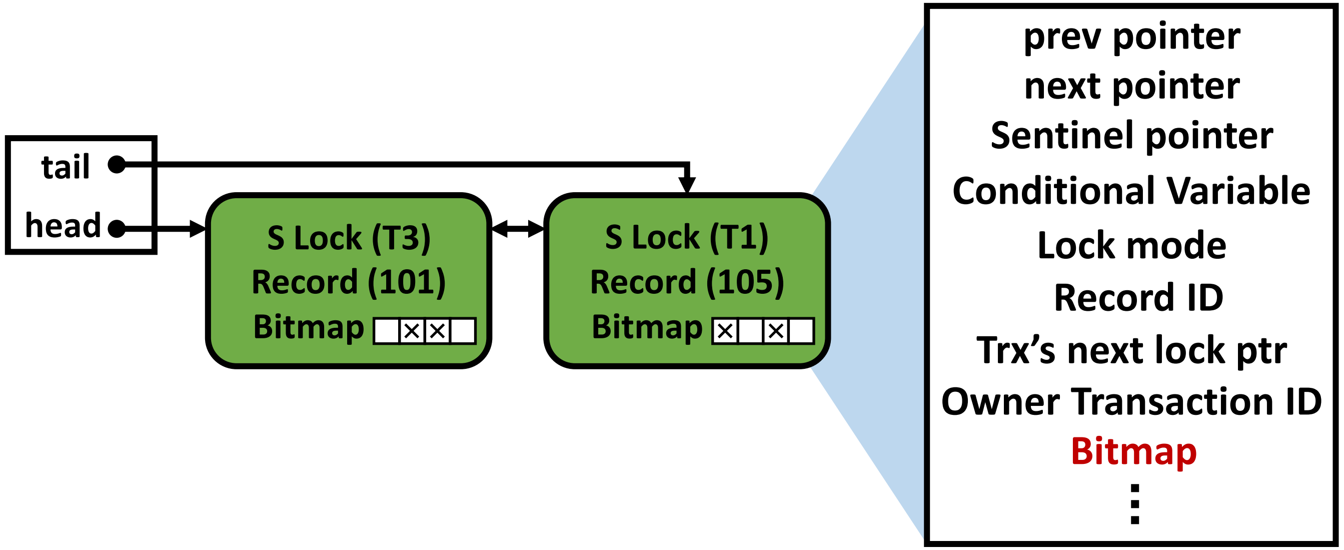
Lock Compression

- Lock "compression" is the optimization technique **to express multiple locks having the same type (e.g., SHARED) held by the same transaction on records in the same database page**.
  - Representing multiple locks can be done using lock bitmap (i.e., **1 bit for each lock in the page**)
- In the above situation, lock compression helps to use the lock object of an existing record.



Locking Compression

- To use a bitmap, a field for bitmap must be entered in the lock object.



Wiki

- Your wiki should contain descriptions about
  - Implicit locking
  - Locking compression

Submission

- Implement transaction locking manager and submit a report about your design and implementation on Wiki.
  - Deadline: Dec 06 11:59pm
- We will only score your commit before the deadline, and your submission after the deadline will not be accepted.
- When building your source codes by using CMake and make, the library file must be made in the lib directory.

IMPORTANT NOTES

- Plagiarism is **STRICTLY FORBIDDEN** as mentioned before
  - We take this issue seriously, and we'll make sure that the student caught plagiarizing will pay the most significant price that we can give within the university's authority.

Thank you