Project4

Lock Table

Project Hierarchy

- Your project hierarchy should be like this.

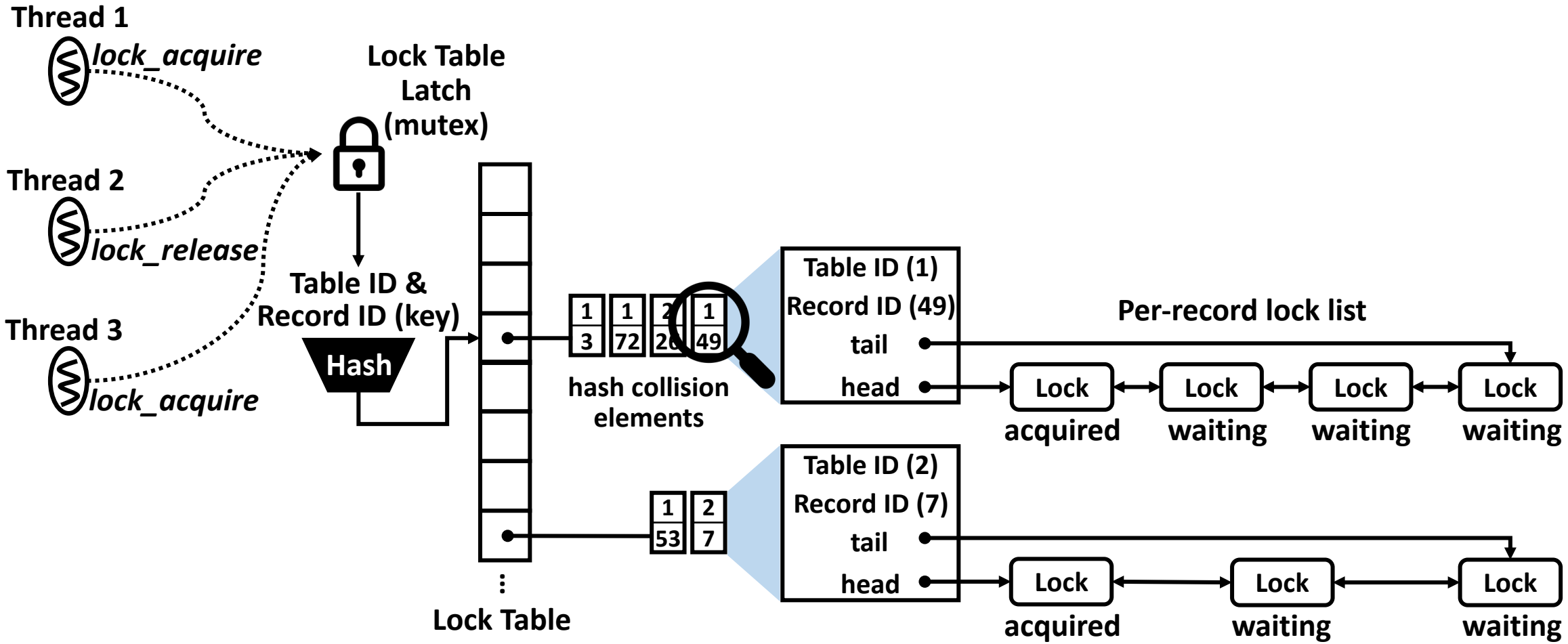
```
kihwang@multicore-96-2:/your_repo/project4 git:(master*) $ tree -a
.
├── .gitignore
├── include
│   └── lock_table.h
├── Makefile
├── src
│   └── lock_table.c
└── unittest
    └── unittest_lock_table.c

3 directories, 5 files
```

- **The output file must be an executable file named *unittest_lock_table*,**
  - ***not a library file as usual.***

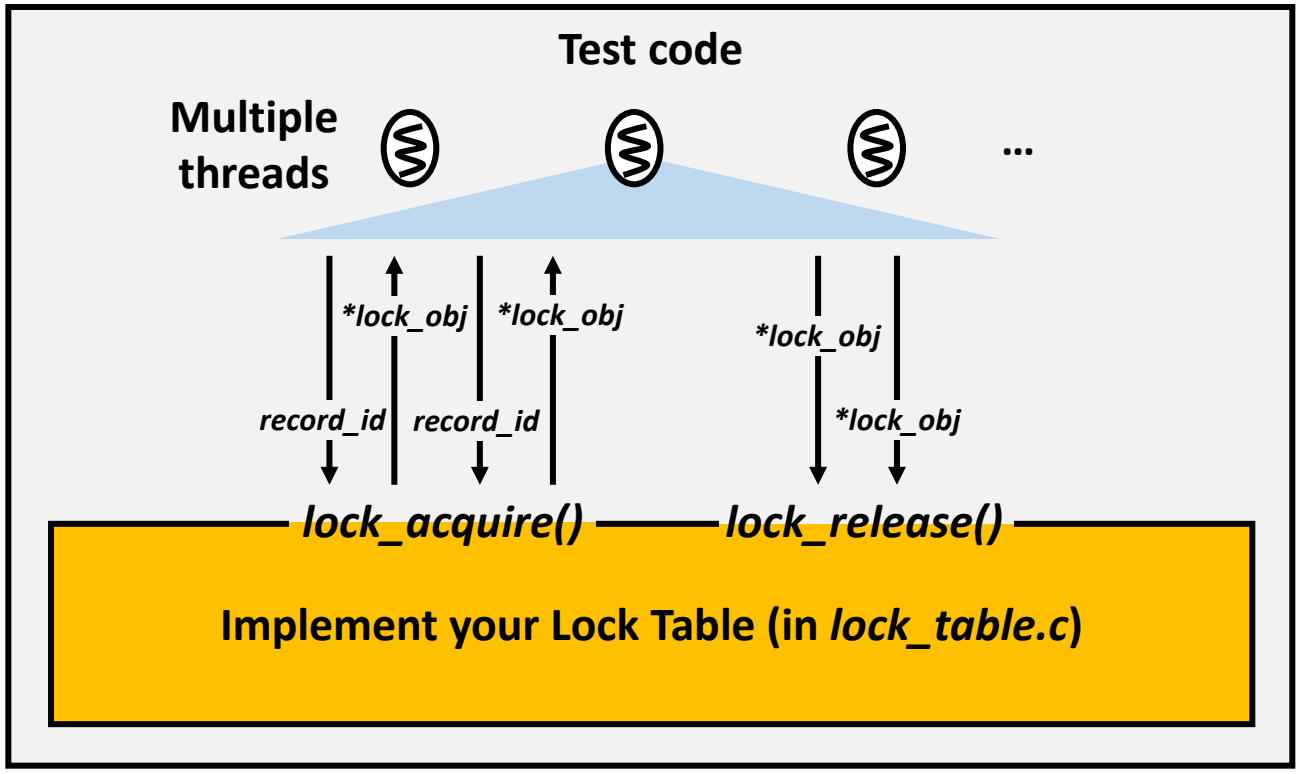
Project Overview

- Your task is to implement a **lock table** module that manages lock objects of multiple threads.
- The module doesn’t need to be compatible with your developing database in this step.
- Instead, the module **should be correctly working with the given test code**.
- This project is a prerequisite step for the next project, Concurrency Control.
- Design your lock table and describe it on hconnect **Wiki** page.

Overall Architecture



Overall Architecture



Lock Table APIs

int init_lock_table(void)

- Initialize any data structures required for implementing lock table, such as hash table, lock table latch, etc.
- If success, return 0. Otherwise, return non-zero value.

lock_t* lock_acquire(int table_id, int64_t key)

- Allocate and append a new lock object to the lock list of the record having the *key.*
  - If there is a predecessor's lock object in the lock list, **sleep** until the predecessor to release its lock.
  - If there is no predecessor's lock object, return the address of the new lock object.
- If an error occurs, return NULL.

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 thread releasing the lock, **wake up** the successor.
- If success, return 0. Otherwise, return non-zero value.

Lock Table APIs

```
lock_t* lock_acquire(int table_id, int64_t key)
{
        pthread_mutex_lock(&lock_table_latch);

        …

        pthread_mutex_unlock(&lock_table_latch);
        return …
}
```

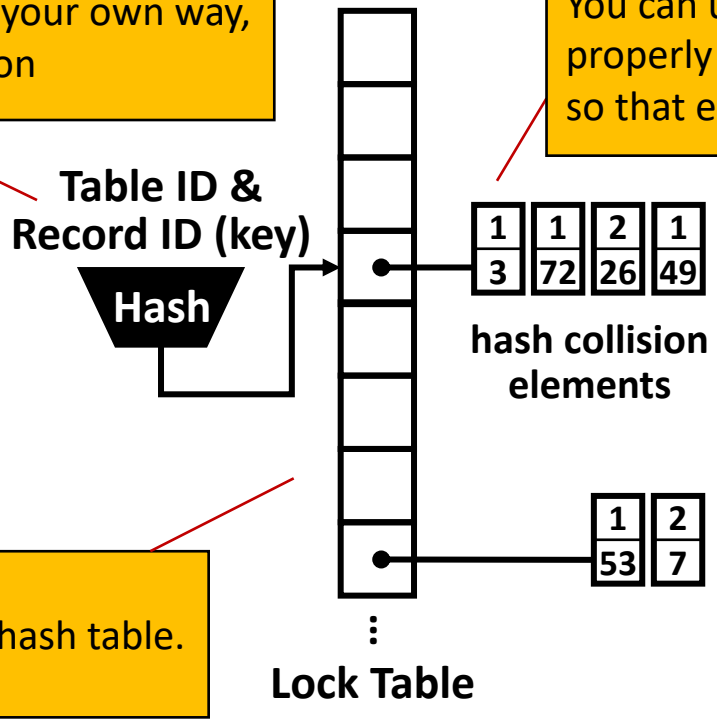
```
int lock_release(lock_t* lock_obj)
{
        pthread_mutex_lock(&lock_table_latch);

        …

        pthread_mutex_unlock(&lock_table_latch);
        return …
}
```

Protect the acquire and release function as a ***critical section*** so that only one thread should be able to access the lock table at a time.

Hash Table

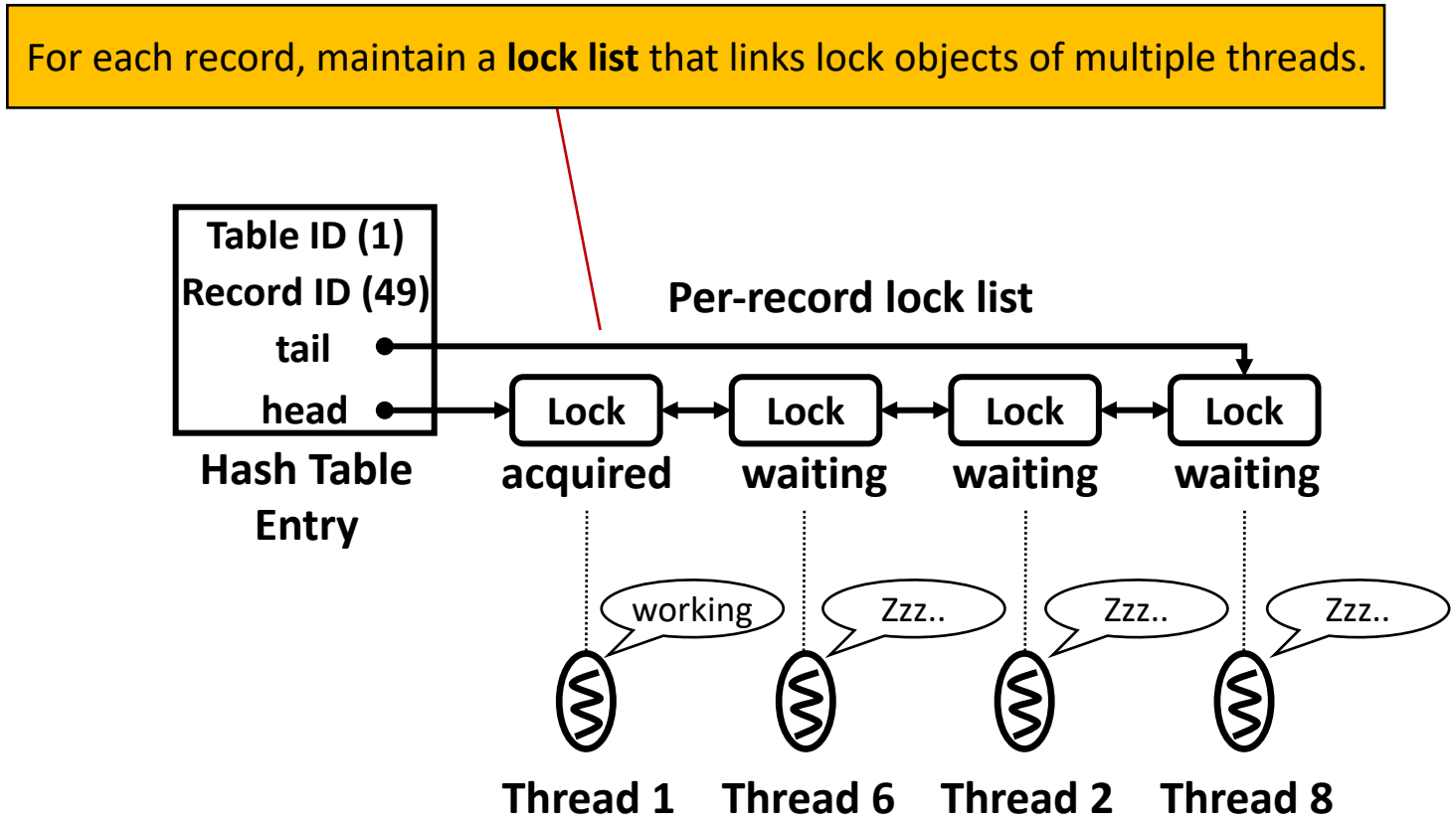
Combine *table id* and *record id (key)* in your own way, and use it as an input of the hash function

You can use any hash table (custom, stl, open-sourced, …) that properly **resolve hash collision**,
so that each <table id & record id> is uniquely distinguishable.

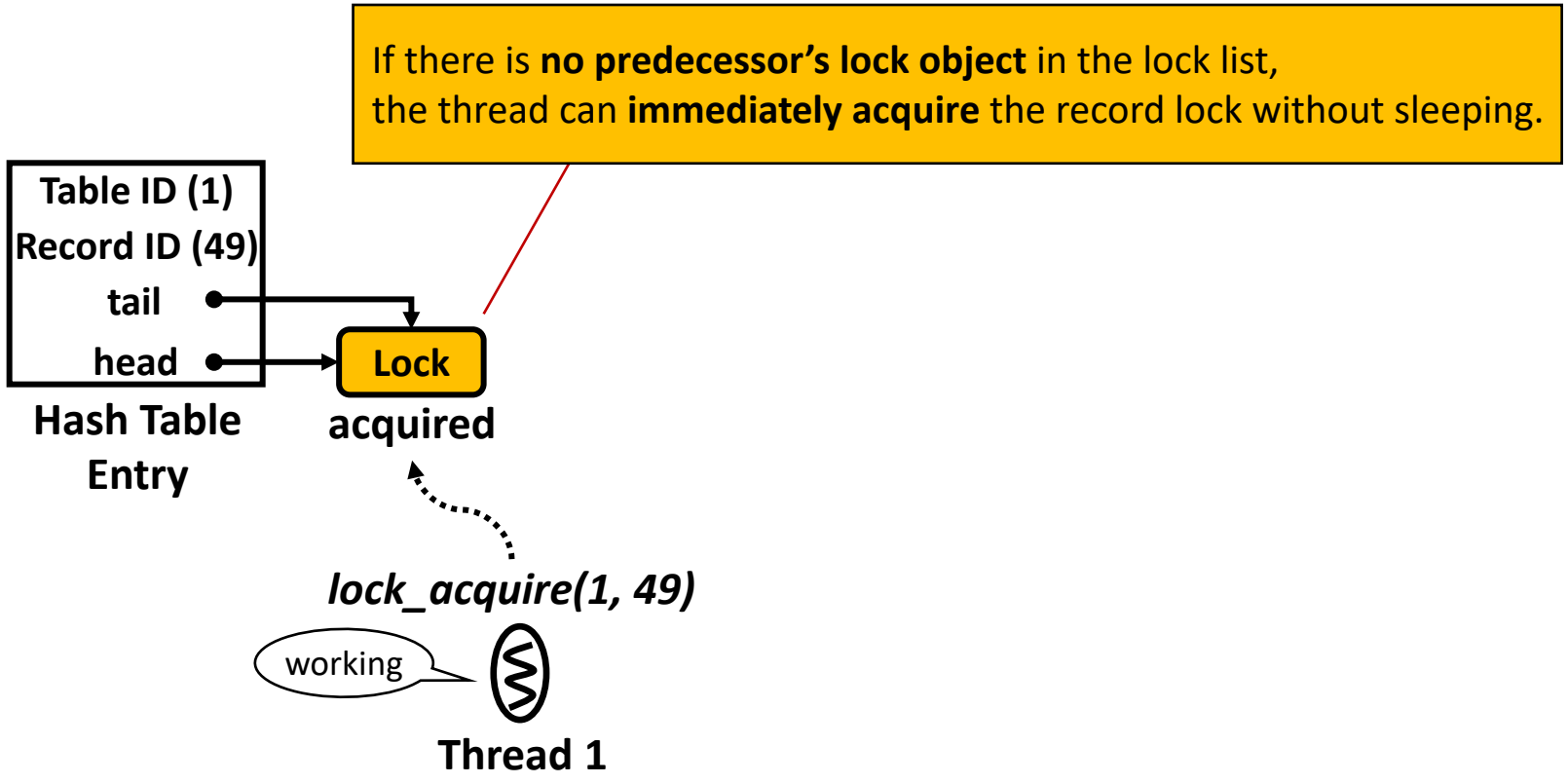


There is **no limitation of the size** of the hash table.

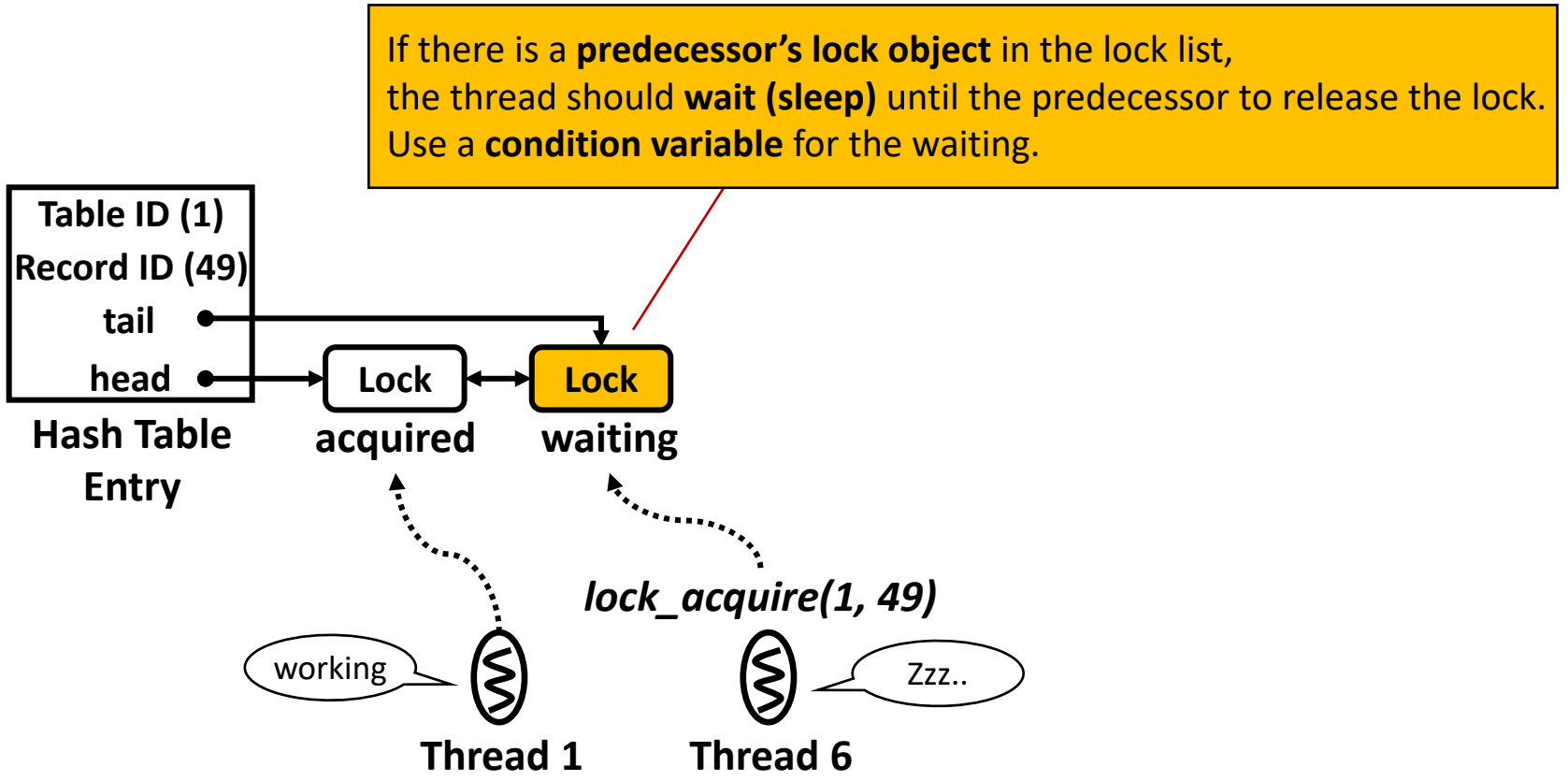
Lock List


For each record, maintain a **lock list** that links lock objects of multiple threads.
Table ID (1)
Record ID (49)
tail
head
Hash Table Entry
Per-record lock list
Lock
Lock
Lock
Lock
acquired
waiting
waiting
waiting
working
Zzz..
Zzz..
Zzz..
Thread 1
Thread 6
Thread 2
Thread 8

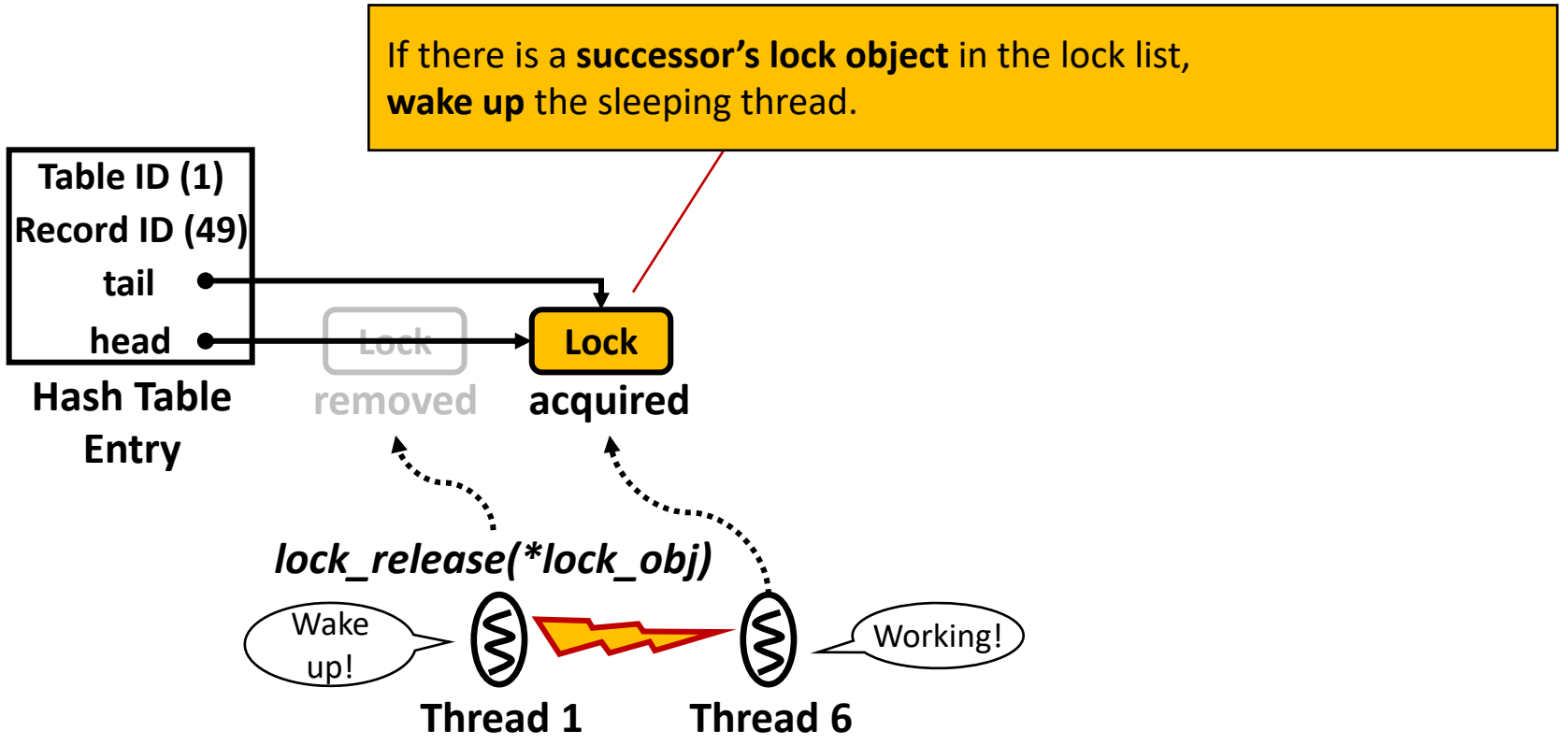
lock_acquire()



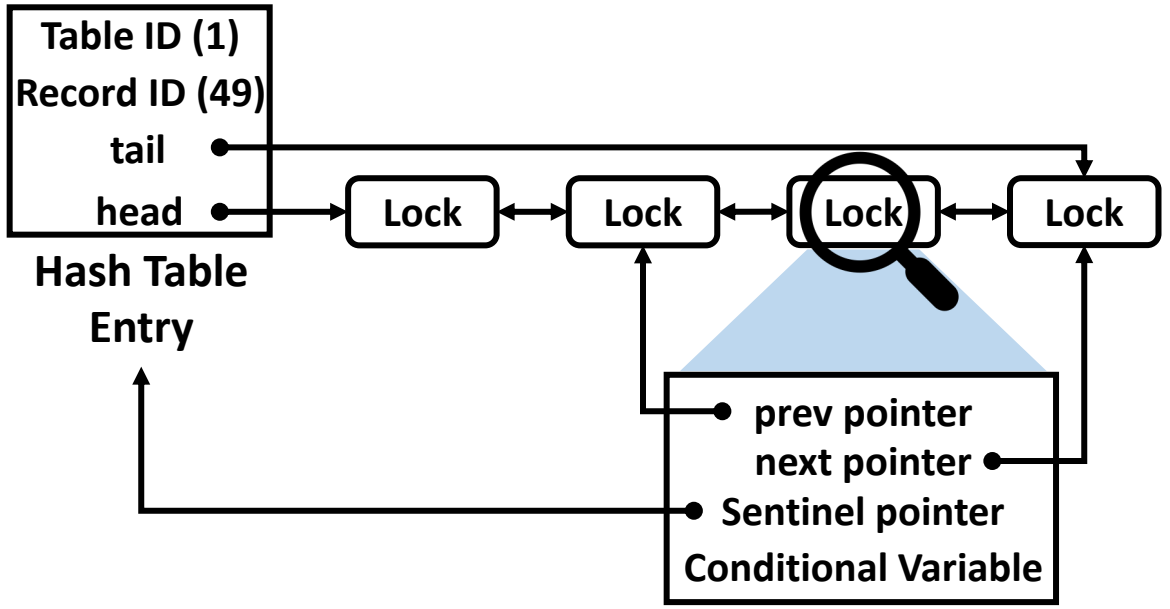
lock_acquire()



lock_release()



Lock Object



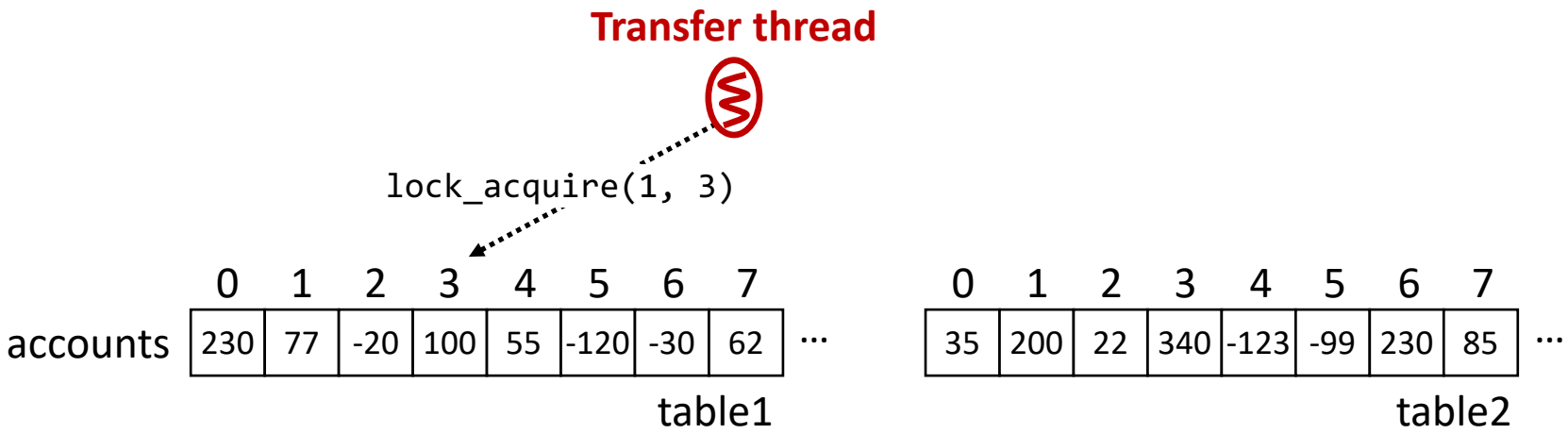
Test Code

- The given test code will
  - call *init_lock_table()* API function,
  - create multiple threads each of which
    - repeatedly acquire and release multiple record locks by calling *lock_acquire()*, *lock_release()*.

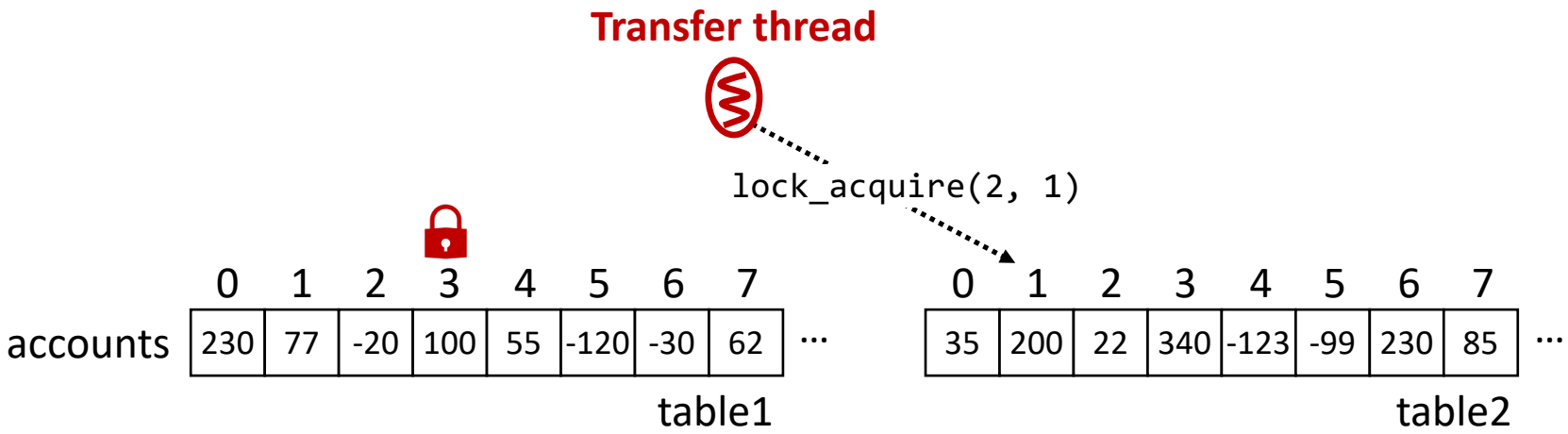
- The test code will safely schedule the operations avoiding deadlock, so you **don't have to deal with the deadlock problem** in this project.

- Analyze the test code as much as you want.

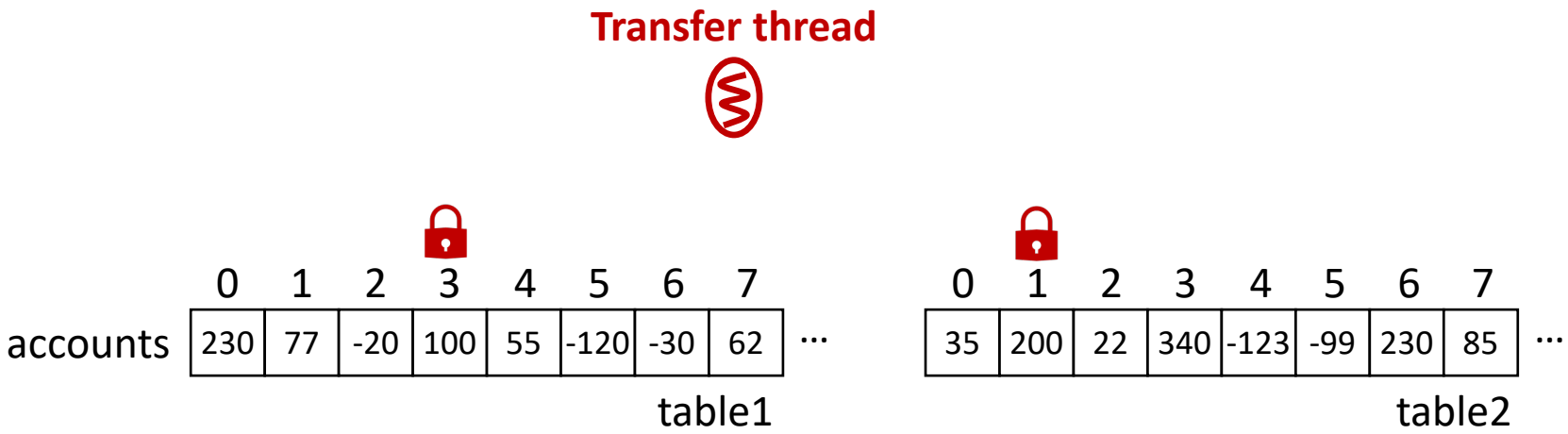
Test Code


Transfer thread
lock_acquire(1, 3)
accounts
0 1 2 3 4 5 6 7
230 77 -20 100 55 -120 -30 62
table1
0 1 2 3 4 5 6 7
35 200 22 340 -123 -99 230 85
table2

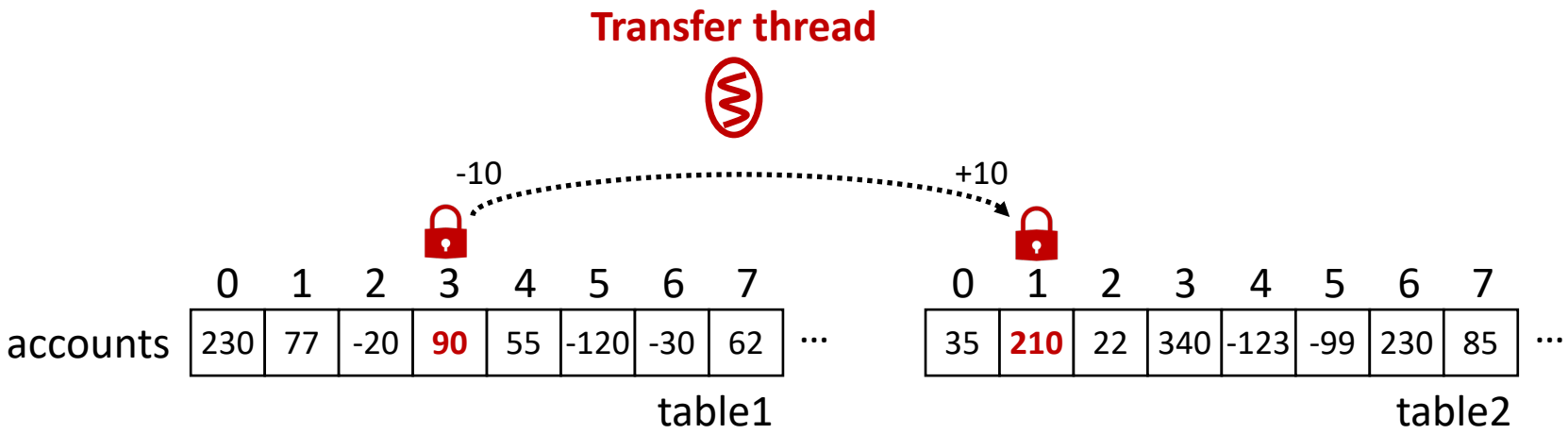
Test Code


Transfer thread
lock_acquire(2, 1)
accounts
0 1 2 3 4 5 6 7
230 77 -20 100 55 -120 -30 62 ...
table1
0 1 2 3 4 5 6 7
35 200 22 340 -123 -99 230 85 ...
table2

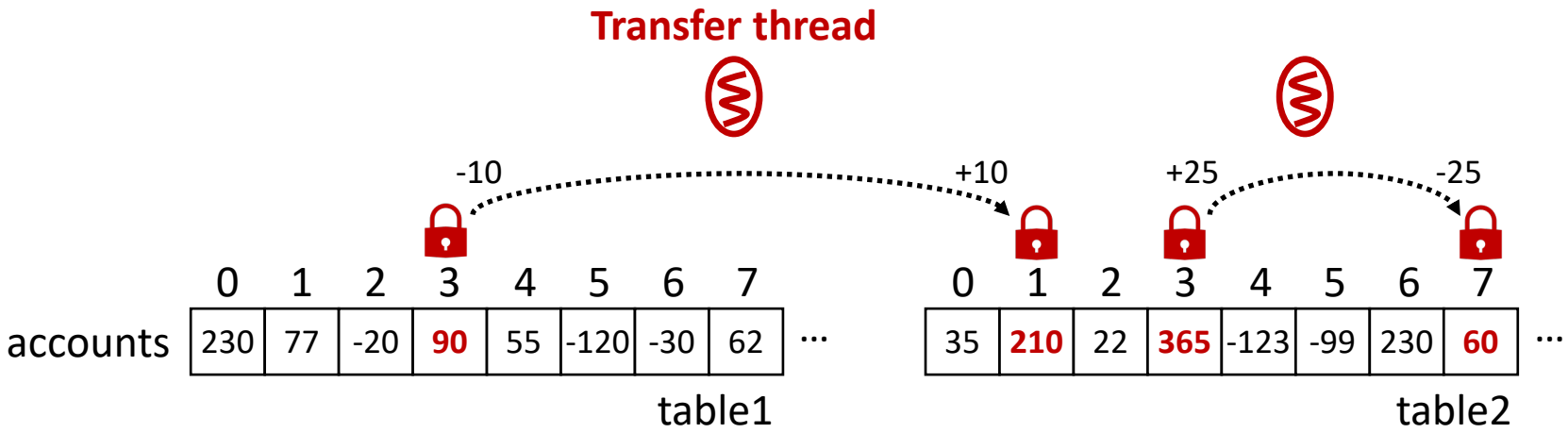
Test Code


Transfer thread
accounts
0 1 2 3 4 5 6 7
230 77 -20 100 55 -120 -30 62
...
table1
0 1 2 3 4 5 6 7
35 200 22 340 -123 -99 230 85
...
table2

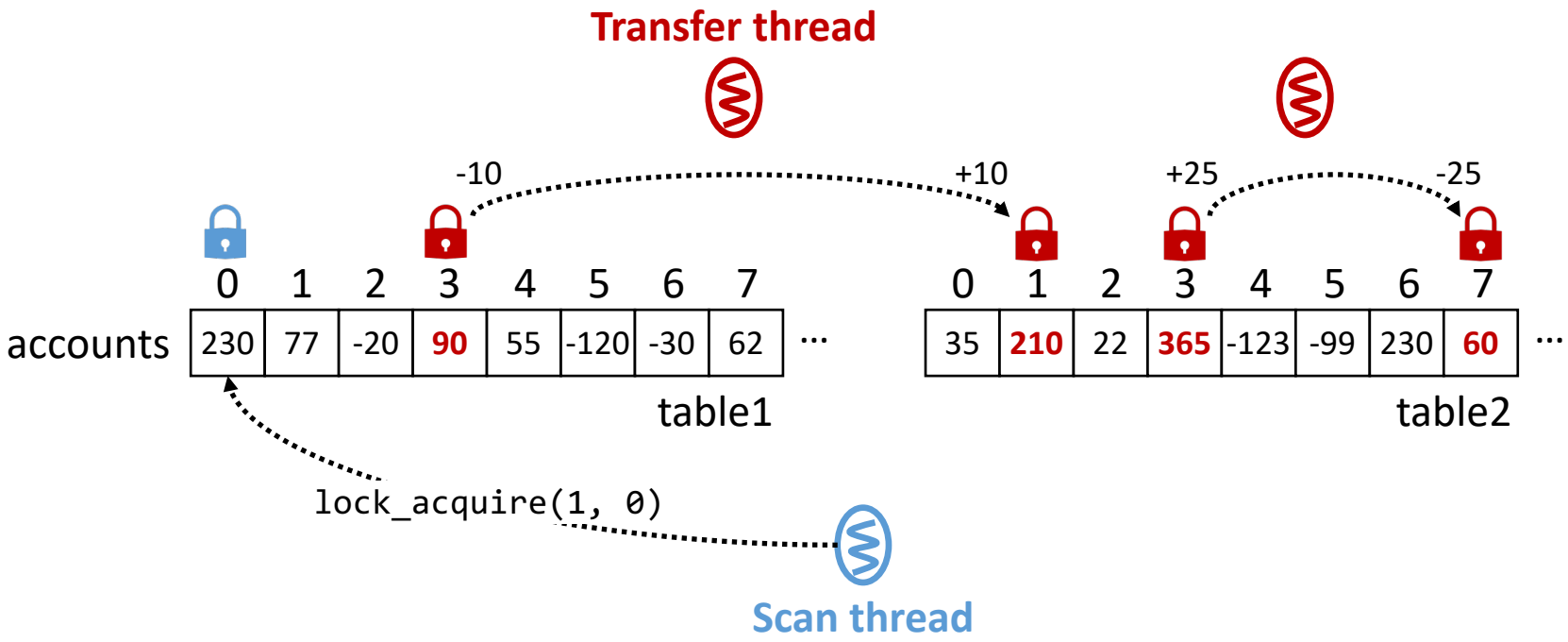
Test Code


Transfer thread
-10
+10
0 1 2 3 4 5 6 7
accounts
230 77 -20 90 55 -120 -30 62 ...
table1
0 1 2 3 4 5 6 7
35 210 22 340 -123 -99 230 85 ...
table2

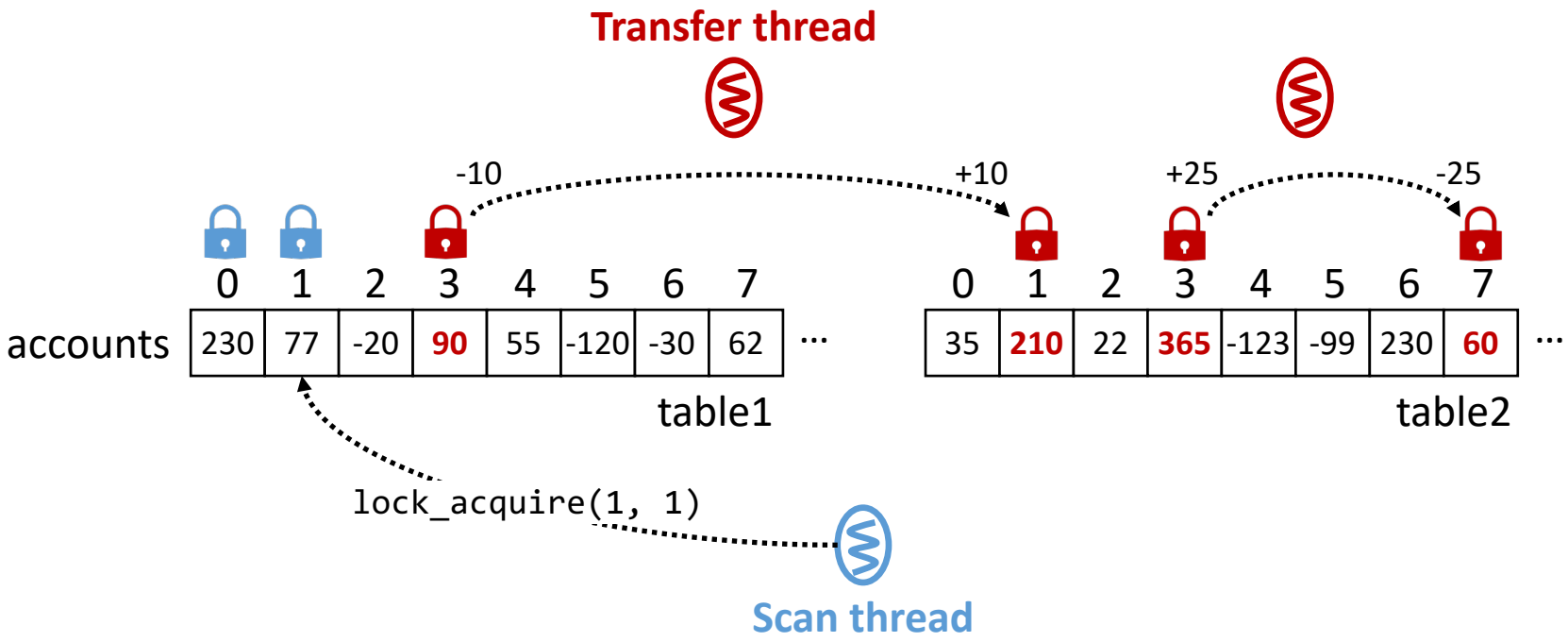
Test Code



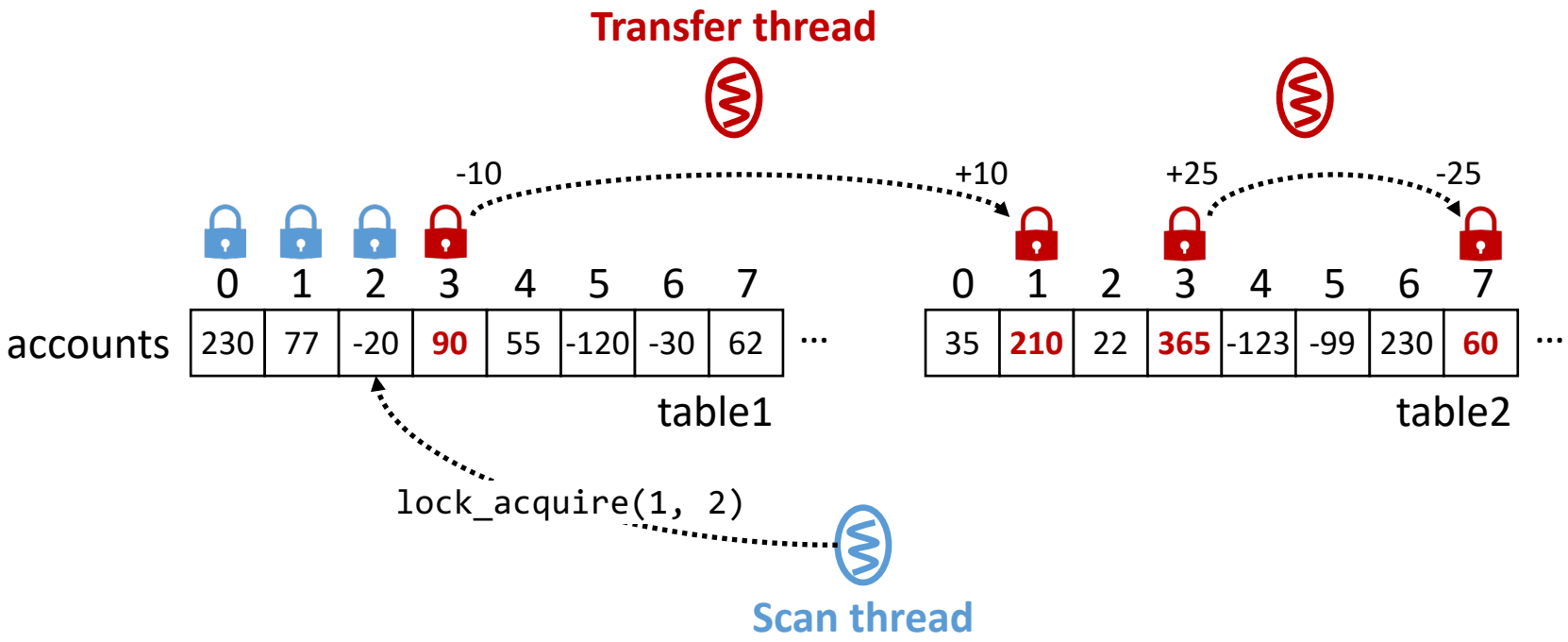
Test Code



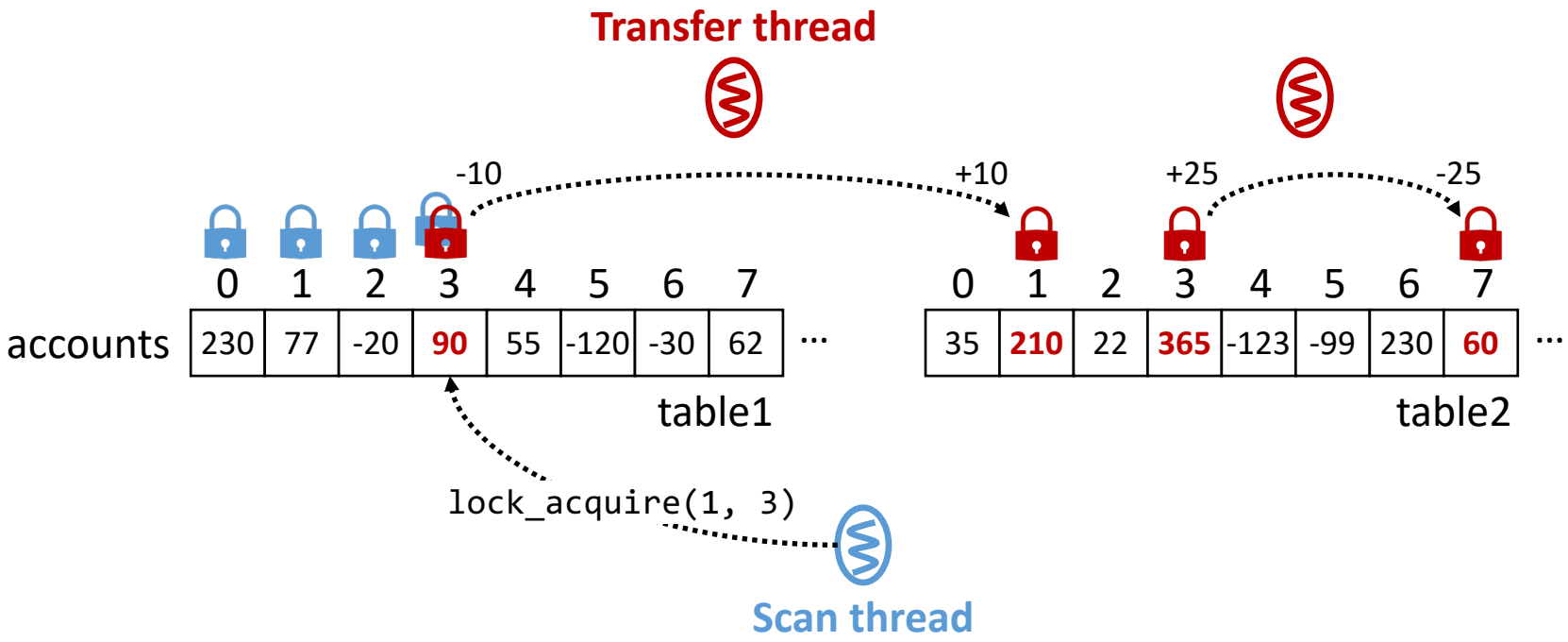
Test Code



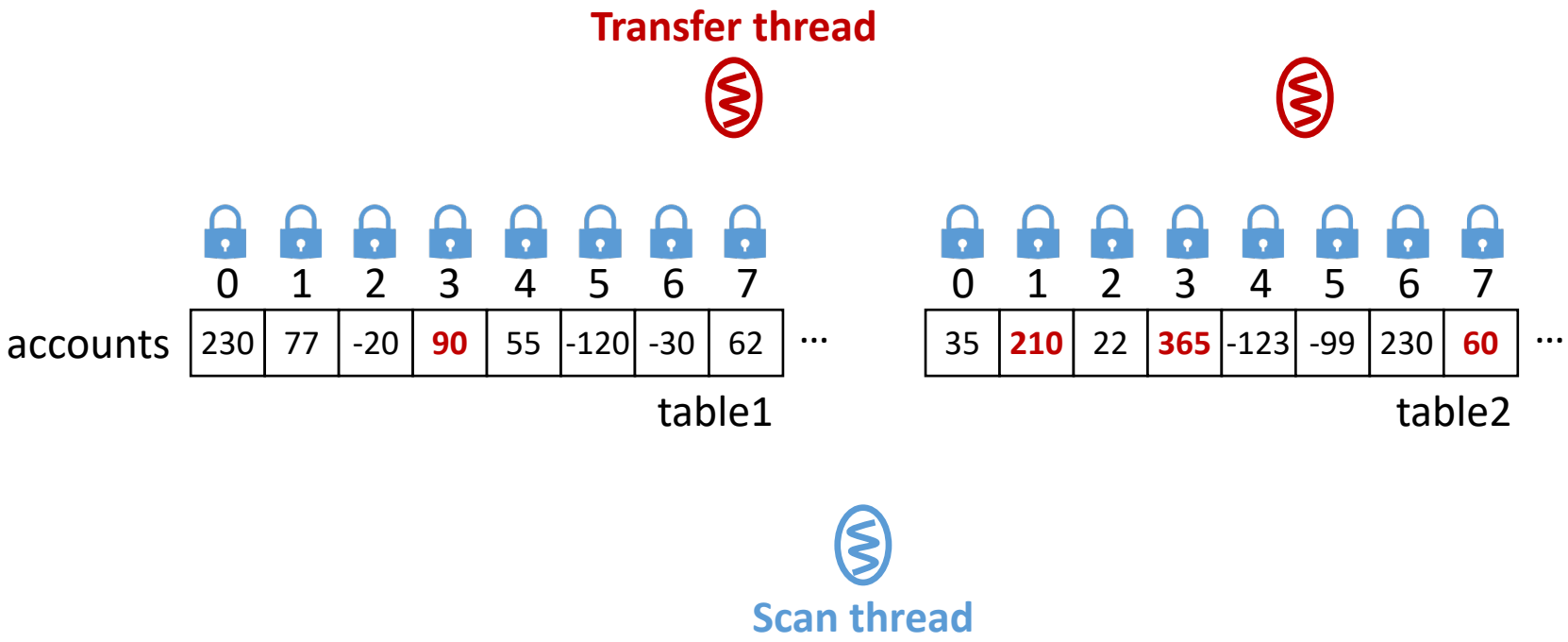
Test Code



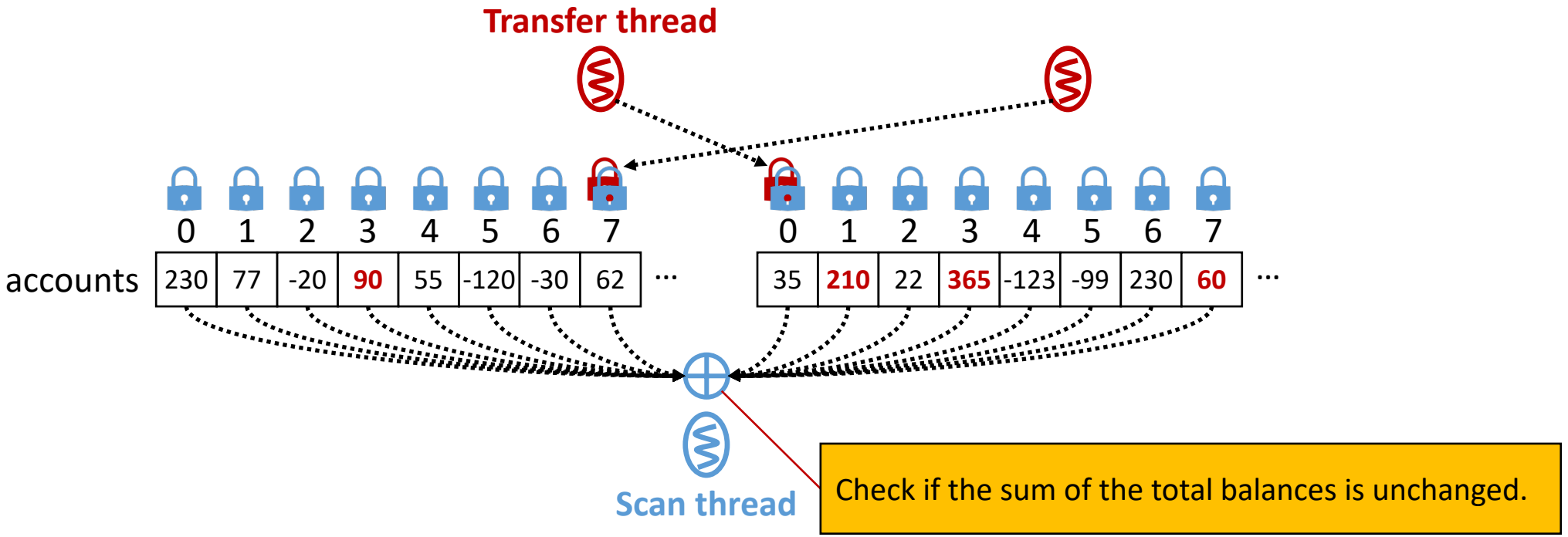
Test Code



Test Code



Test Code



Deadline & Regulations

- Deadline: **Nov 15 11:59pm**
- We'll only score your commit before the deadline, and your submission after the deadline will not be accepted.
- You must follow the given project hierarchy and the path of the "unittest_lock_table" executable file, otherwise, you cannot get a score.

Thank you