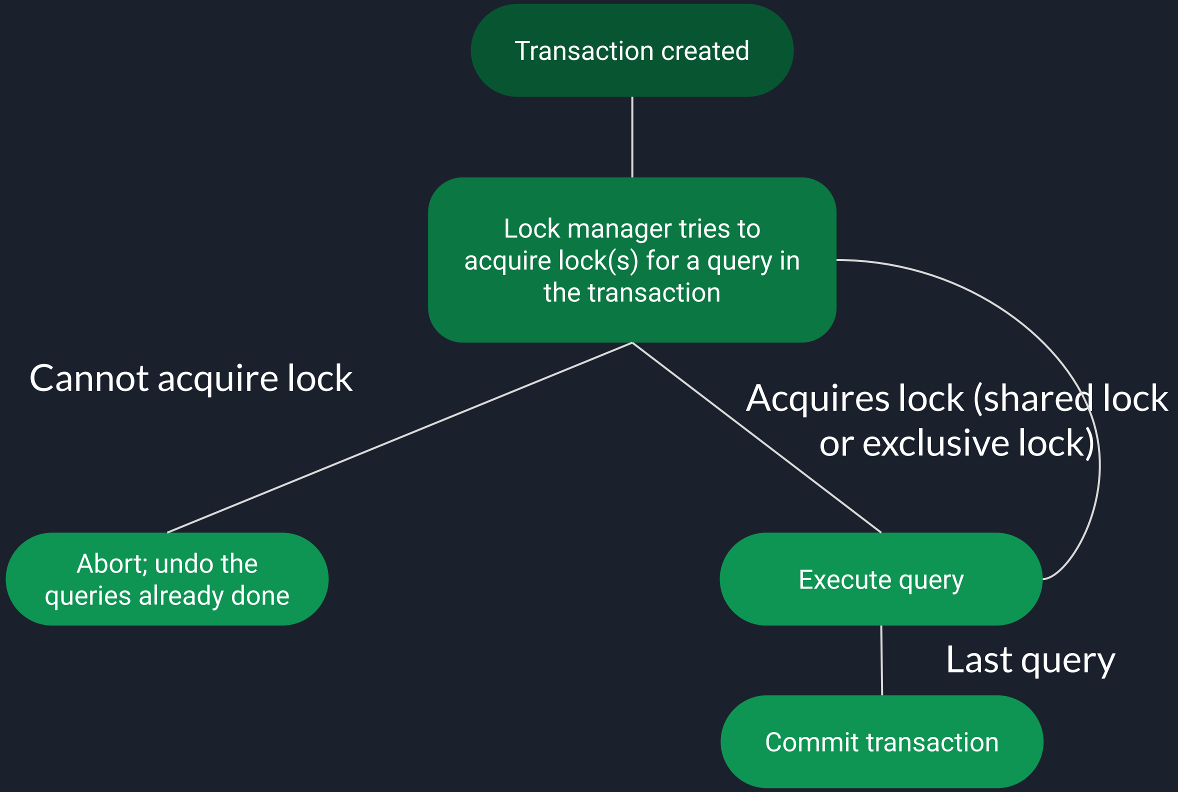
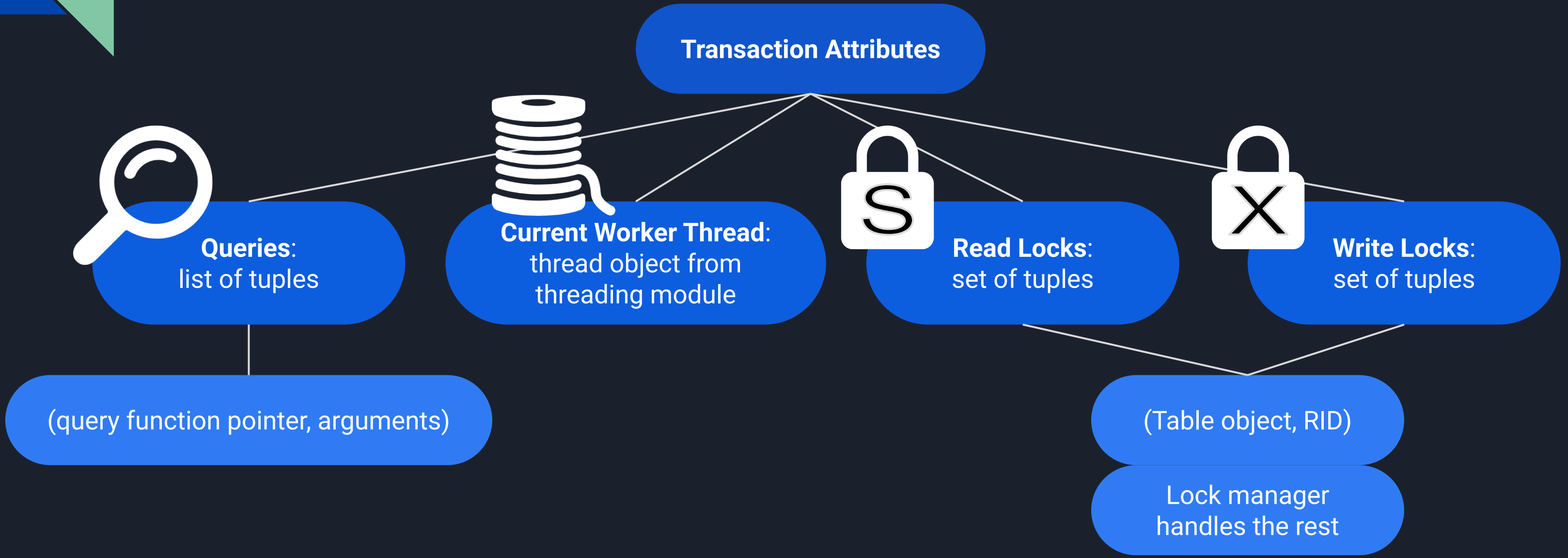
ECS 165A Milestone 3

Aditi Bali, Jie Lyu, Jessica Ma, Nathan Chan, William Shih

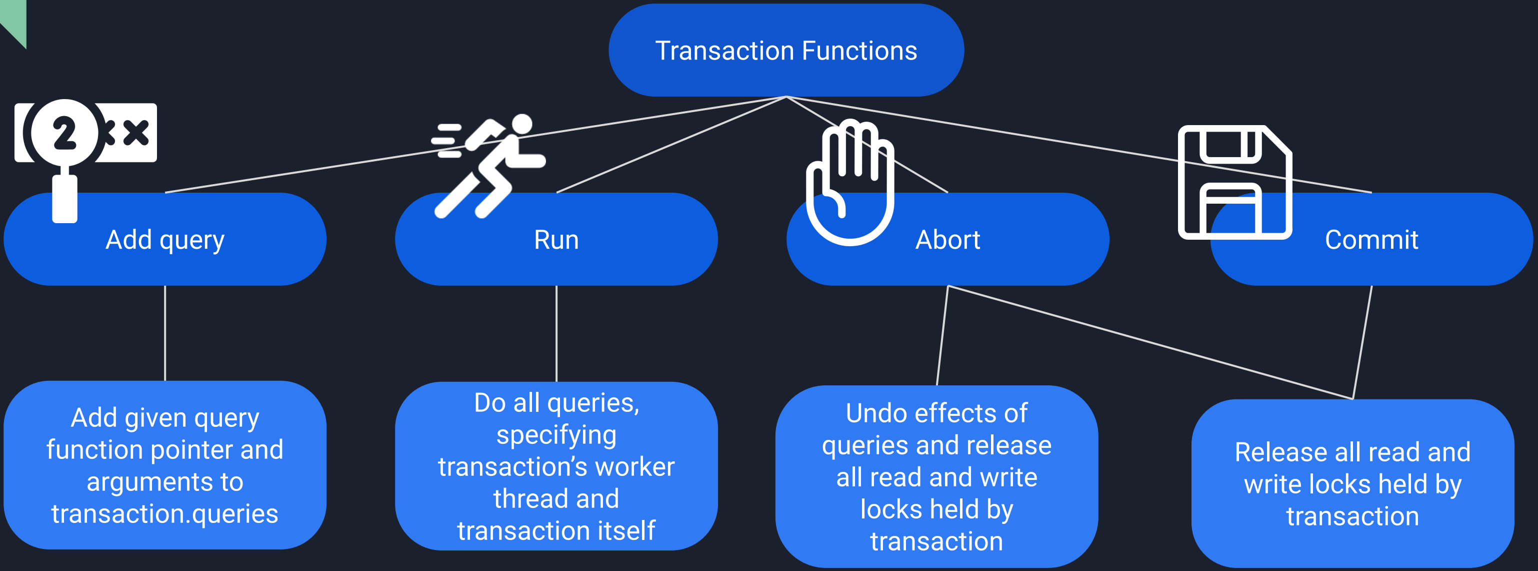
Process Overview (2PL)


Transaction created
Lock manager tries to acquire lock(s) for a query in the transaction
Cannot acquire lock
Acquires lock (shared lock or exclusive lock)
Abort; undo the queries already done
Execute query
Last query
Commit transaction

class Transaction (attributes)


Transaction Attributes
Queries: list of tuples
Current Worker Thread: thread object from threading module
Read Locks: set of tuples
Write Locks: set of tuples
(query function pointer, arguments)
(Table object, RID)
Lock manager handles the rest
S
X

`class Transaction` (functions)



Transaction Functions

Add query

Add given query function pointer and arguments to transaction.queries

Run

Do all queries, specifying transaction’s worker thread and transaction itself

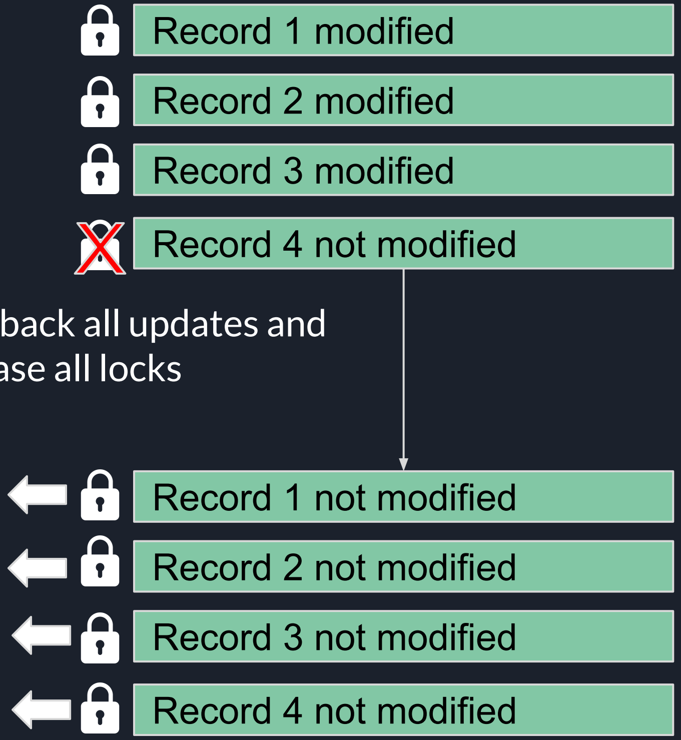
Abort

Undo effects of queries and release all read and write locks held by transaction

Commit

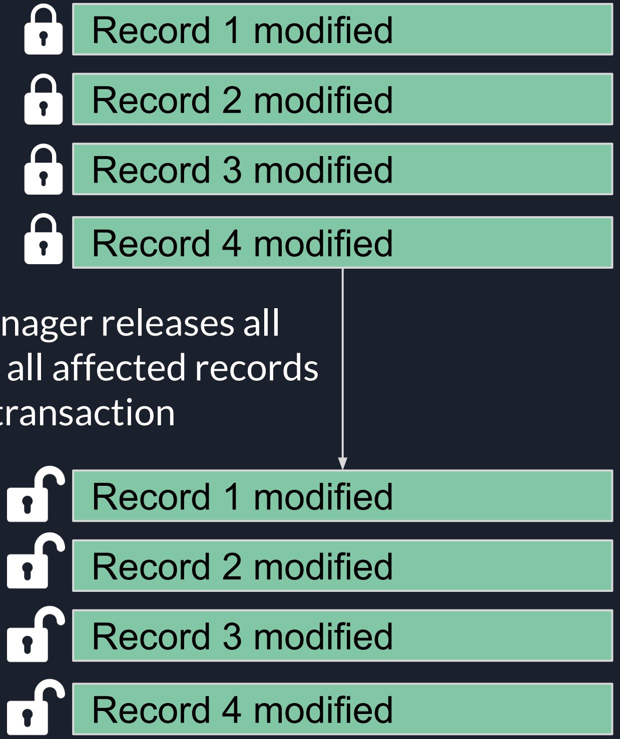
Release all read and write locks held by transaction

Abort



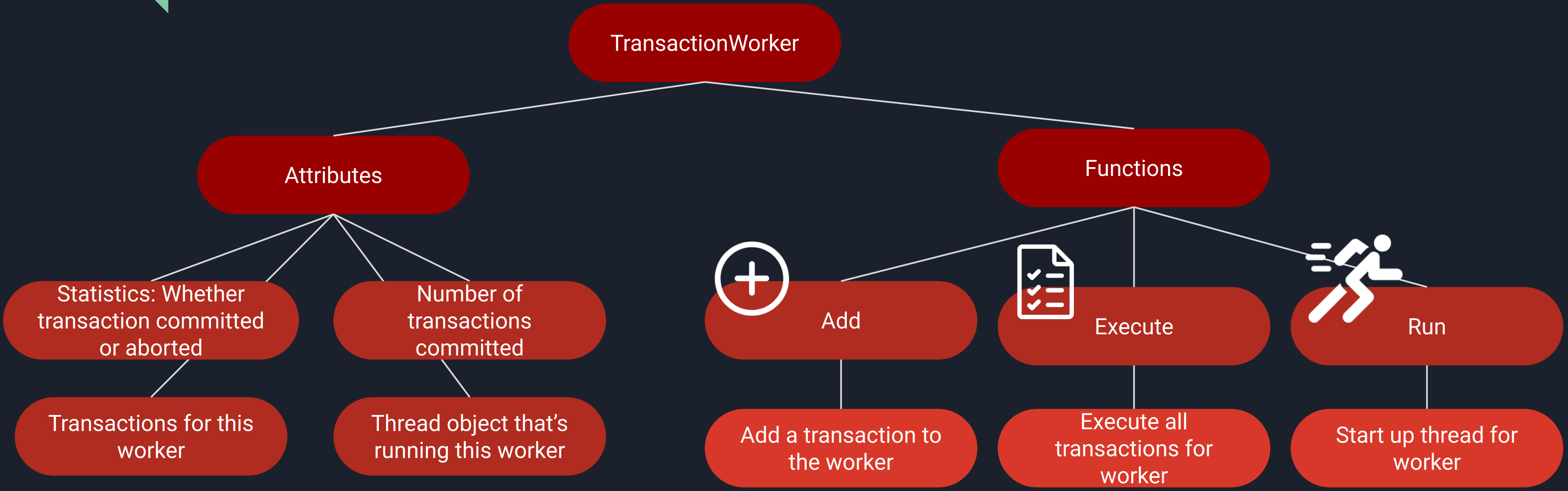
Roll back all updates and release all locks

Commit

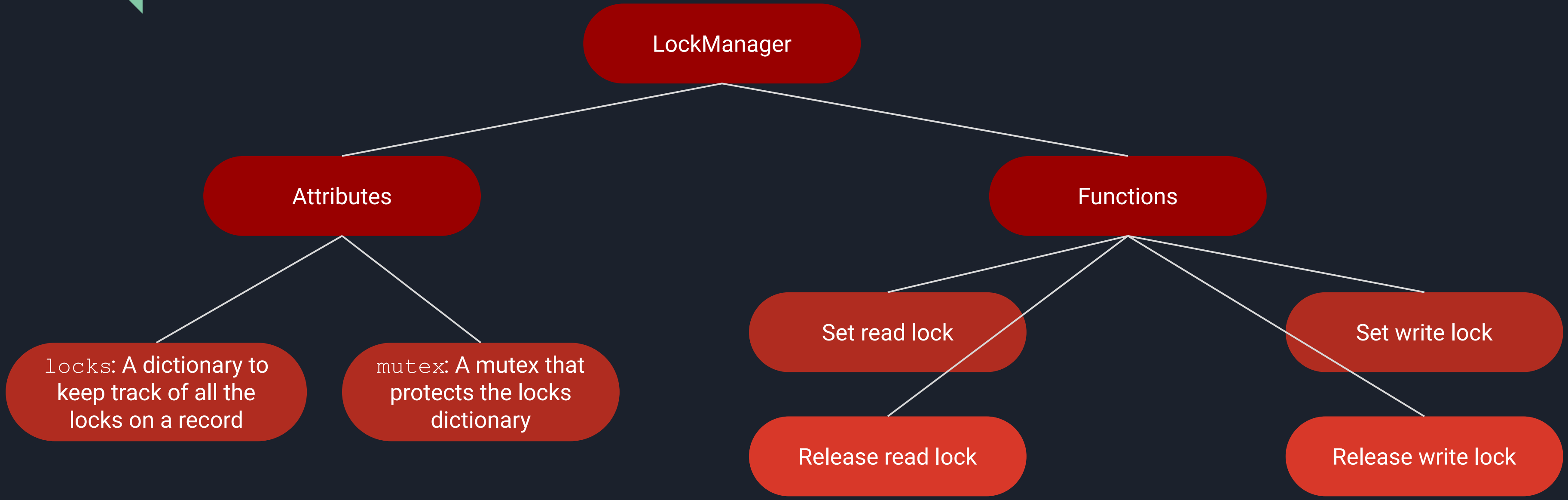


Lock manager releases all locks on all affected records for this transaction

class TransactionWorker


TransactionWorker
Attributes
Functions
Statistics: Whether transaction committed or aborted
Number of transactions committed
Transactions for this worker
Thread object that’s running this worker
Add
Execute
Run
Add a transaction to the worker
Execute all transactions for worker
Start up thread for worker

class LockManager


LockManager
Attributes
Functions
locks: A dictionary to keep track of all the locks on a record
mutex: A mutex that protects the locks dictionary
Set read lock
Release read lock
Set write lock
Release write lock

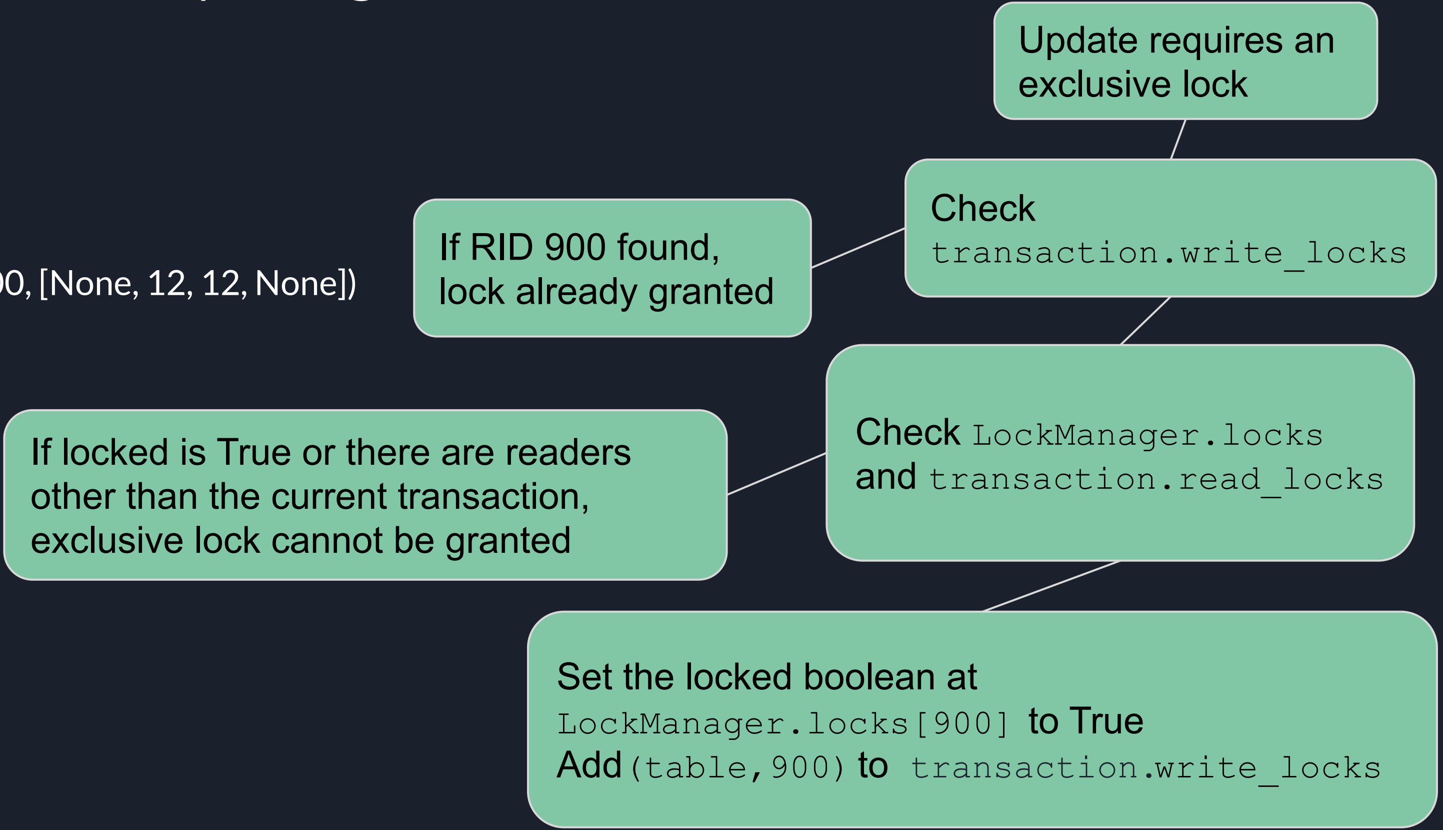
Lock Management

- `LockManager.locks`: {RID: status}
  - Status: num readers, locked
    - num readers: the number of transactions using an S lock on this RID
    - locked: if the X lock of this RID is taken
- `Transaction.read_locks`, `Transaction.write_locks` (table, RID)
  - These are stored in a Transaction object because queries in the same transaction shouldn't be restricted by one another's locks

- S locks: select and sum
- X locks: insert, delete, update
- When acquiring a lock for a query, first check `read_locks` and `write_locks` in its transaction. If the RID to be locked isn't found, check with the lock manager for conflicts.
- When a transaction finishes (abort/commit), we remove all the locks used by this transaction from `LockManager.locks`.

Example: acquiring a lock

- query. update(900, [None, 12, 12, None])



Update requires an exclusive lock

Check `transaction.write_locks`

If RID 900 found, lock already granted

Check `LockManager.locks` and `transaction.read_locks`

If locked is True or there are readers other than the current transaction, exclusive lock cannot be granted

Set the locked boolean at `LockManager.locks[900]` to True
Add `(table,900)` to `transaction.write_locks`

Latches

- `table.id_lock`
  - Protect RID/TID assigning
- `index.locks`
- `Bufferpool.page_locks`
- `LockManager.mutex`

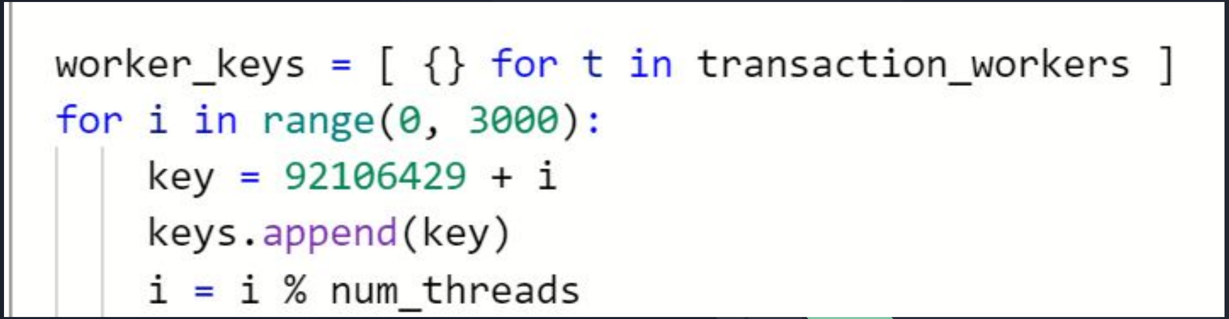
- Since all the read and writes are done in the bf, there's no need to put latches on data files in disk.

- Locks are released when a transaction finishes, but latches are released as soon as the read/write access is finished.
- If a transaction encounters a latch, it will wait till the latch is released to access the protected items instead of aborting.

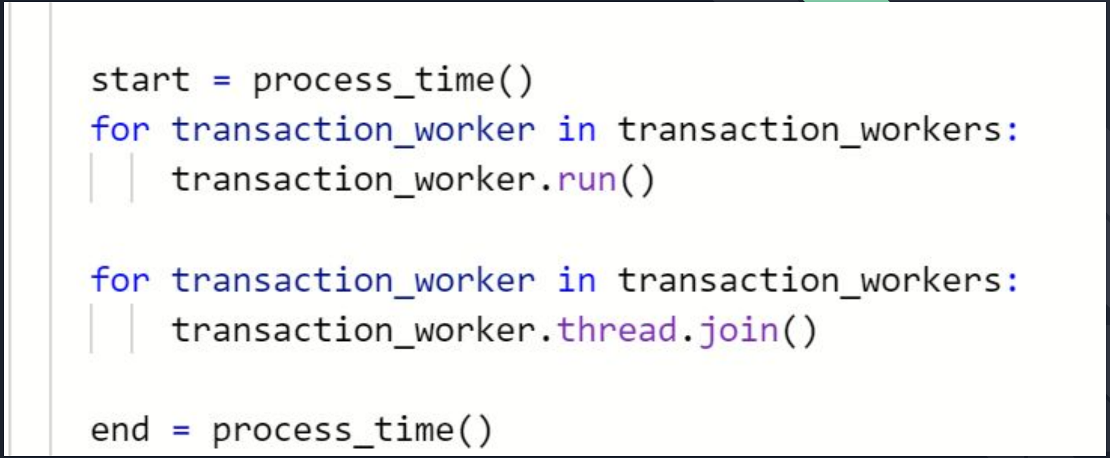
Merge Update

- Before only merges after closing the database
- Now also merges after a fixed number of updates in a page range
  - The first n-1 page ranges can be merged
  - If there is only 1 page range, merge won't happen while database is open
  - Merges all page ranges when closing database however
- 3 Lists to keep track for merge - One data point per page range
  - Page directory list contains the number of merges
    - A new folder created for every merge, numbered sequentially from 0
  - Update list contains the number of updates per page range, updated to zero after merge, initialized at 0
  - TID max list contains the TID cutoff for the indirection column, initialized at new TID

Performance Testing



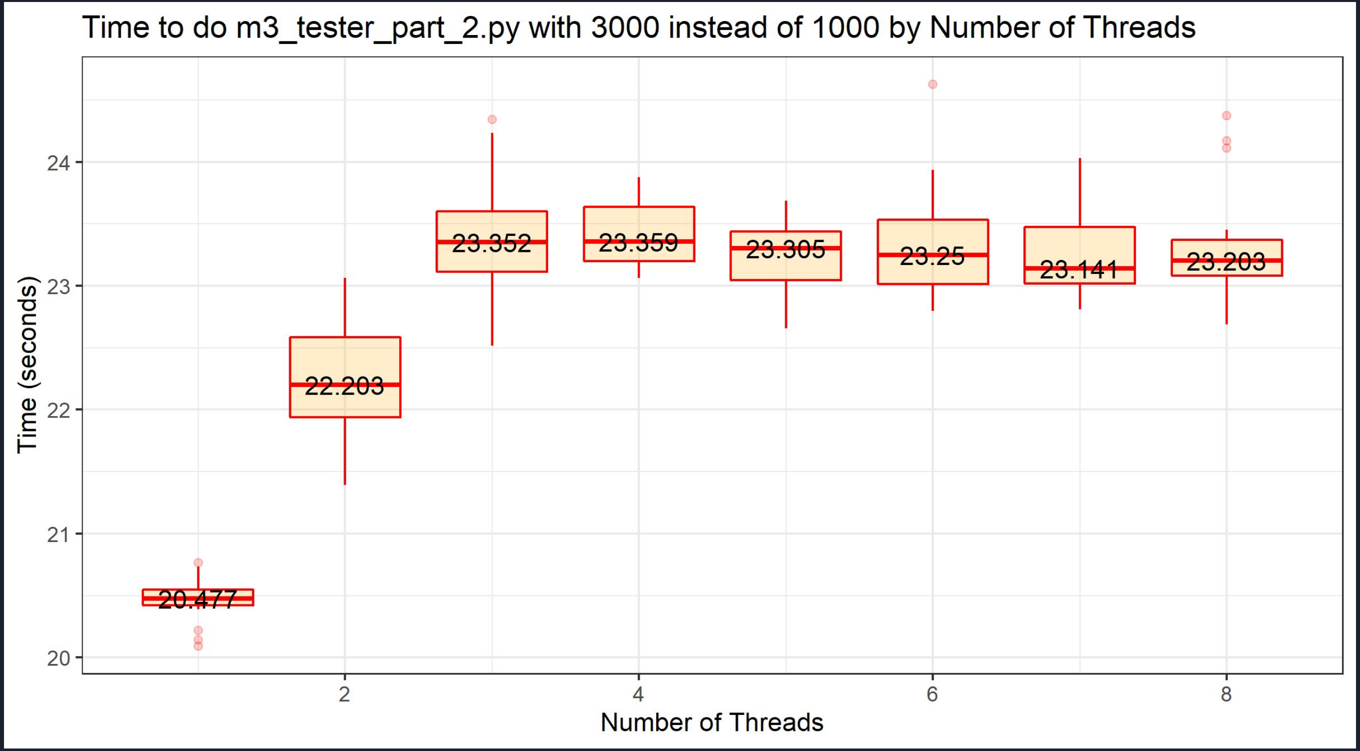
```
worker_keys = [ {} for t in transaction_workers ]
for i in range(0, 3000):
    key = 92106429 + i
    keys.append(key)
    i = i % num_threads
```



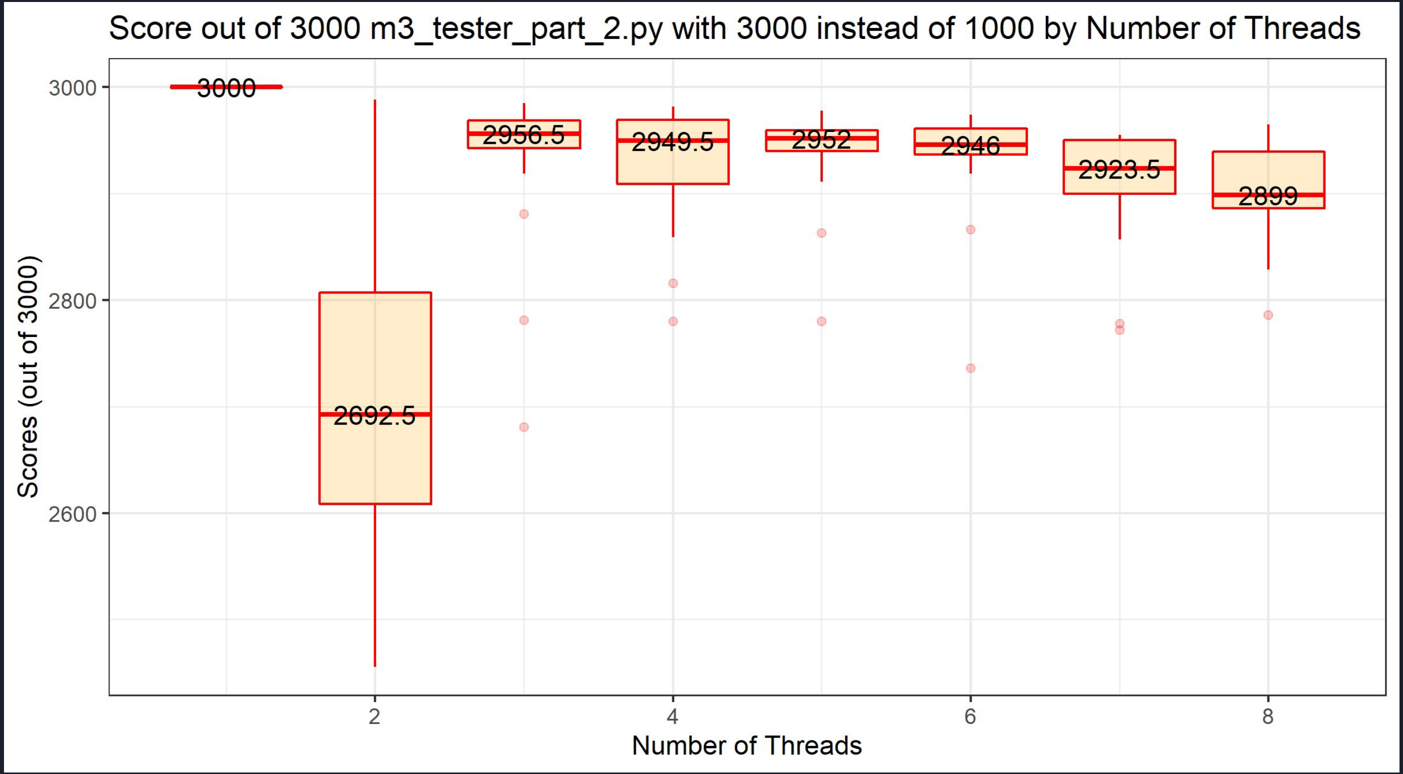
```
start = process_time()
for transaction_worker in transaction_workers:
    transaction_worker.run()

for transaction_worker in transaction_workers:
    transaction_worker.thread.join()

end = process_time()
```

Time to do m3_tester_part_2 by Number of Threads

- Expected to see a concave function, but stabilizes at 23.5 seconds with 3-8 threads
- Side note: The performance testing is done on a slightly different version of the program
  - New version gets higher scores



Score out of 3000 by Number of Threads



Brief Demo:
Threads and Abort