L-Store
Milestone 3

ECS 165A

Natheenthorn Teachaurangchit
Michael Shaw
Stuart Feng
Henry Chou

Transaction Semantics

Shared & Exclusive Locks (Granting Conditions)

Lock Management

Concurrency Control

No-Wait 2PL Policy

Forward Looking

Post-M3 Database Program

Transaction Semantics

Shared & Exclusive Locks

Conditions of granting different locks:		Initially Holds	
		Shared	Exclusive
Request For	Shared	Grant	No Grant
	Exclusive	No Grant	No Grant

Shared Locks:

1. Select
2. Sum

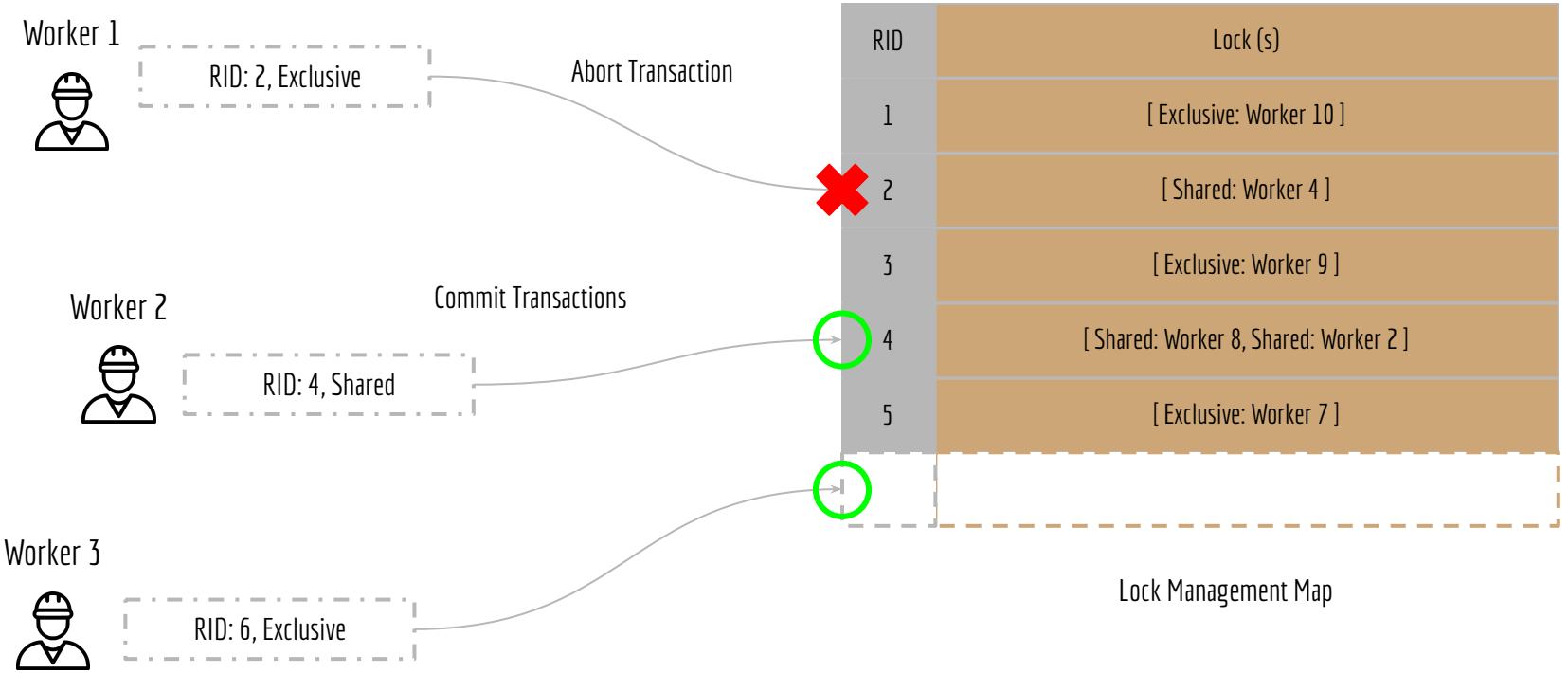
Exclusive Locks:

1. Insert
2. Delete
3. Update

* One exception if the same thread tried to exchange its shared lock for an exclusive lock, we will grant the request

Lock Management

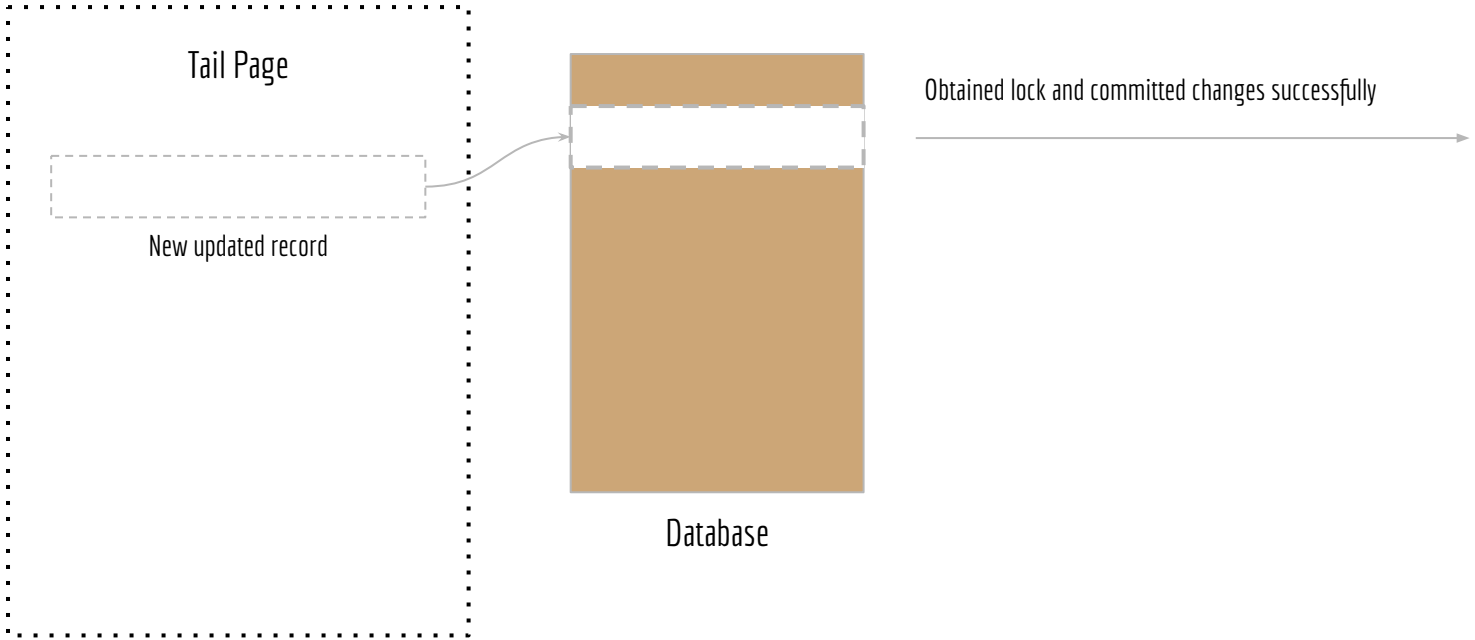
Using Python Dictionary to keep track of all the locks on different RIDs



Concurrency Control

Strict No-Wait 2PL Policy

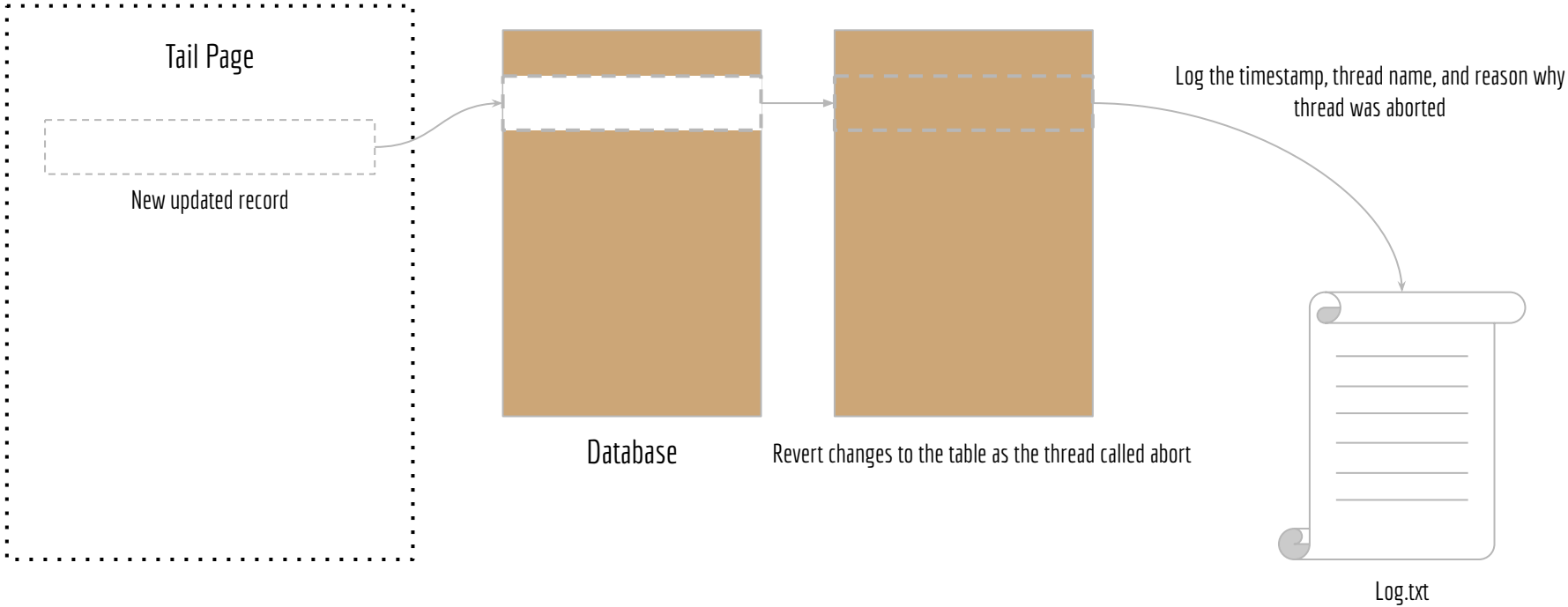
Scenario 1: The thread was able to obtain the lock successfully



*Note: Rollback is only for insert, update, and delete. For read and sum, the thread will simply abort

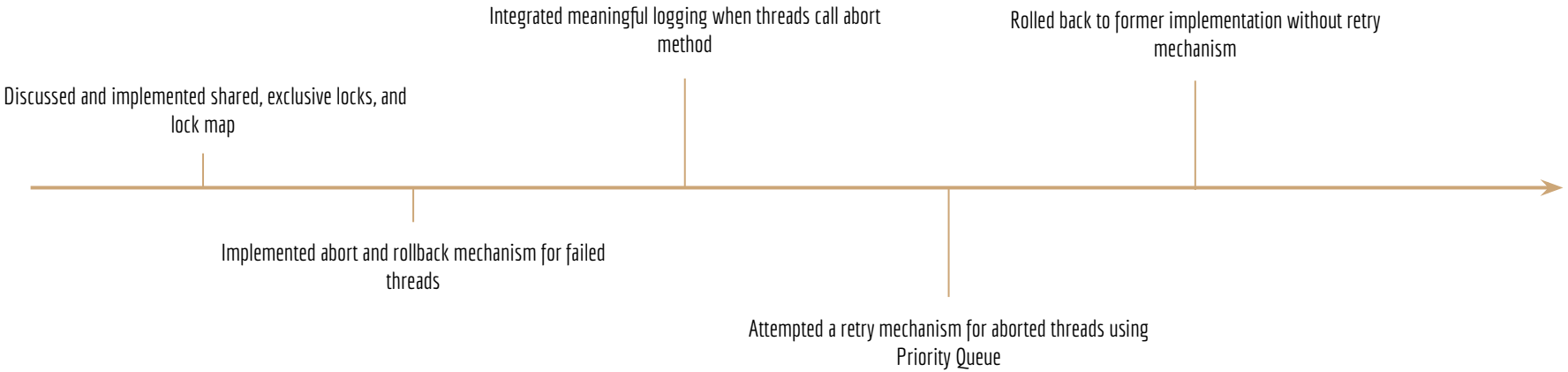
Strict No-Wait 2PL Policy

Scenario 2: The thread was not able to obtain the lock and has to abort



*Note: Rollback is only for insert, update, and delete. For read and sum, the thread will simply abort

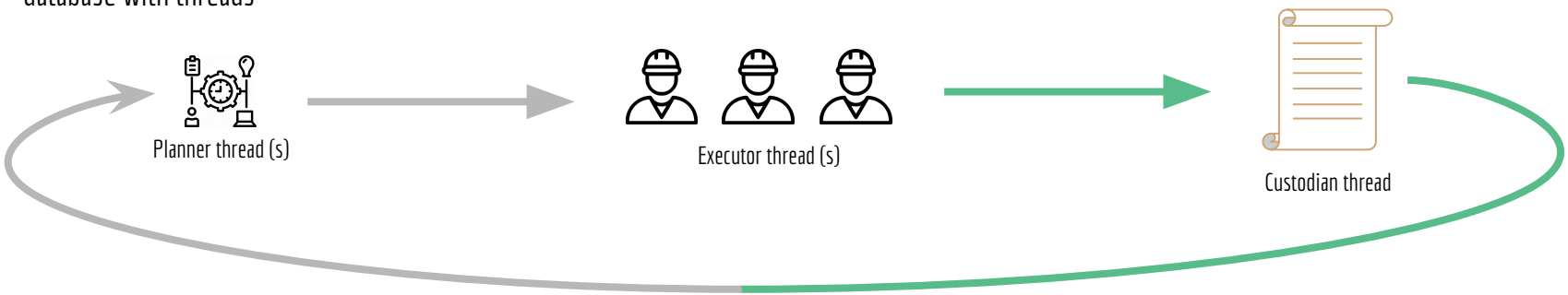
Roadmap


Discussed and implemented shared, exclusive locks, and lock map
Implemented abort and rollback mechanism for failed threads
Integrated meaningful logging when threads call abort method
Attempted a retry mechanism for aborted threads using Priority Queue
Rolled back to former implementation without retry mechanism

Forward Looking

Post-M3 Database Program

Attempted a second version of Milestone 3 as a prototype, which replicates the Consumer-Producer model to resemble the real world use of database with threads



Basic Idea:

- Create a transaction queue that stores all the aborted thread so that we can re-execute them
- Create layers of queues to effectively re-arrange threads based on transactions to enhance performance

Thank you!