ECS 165A Milestone 3

Aadvika Ahuja, Manvi Nawani, Veda Periwal, Neerja Natu, Rahul Lakshmanan, Vibha Raju

Outline

1. Transactions
2. Concurrency Control

1. Transactions

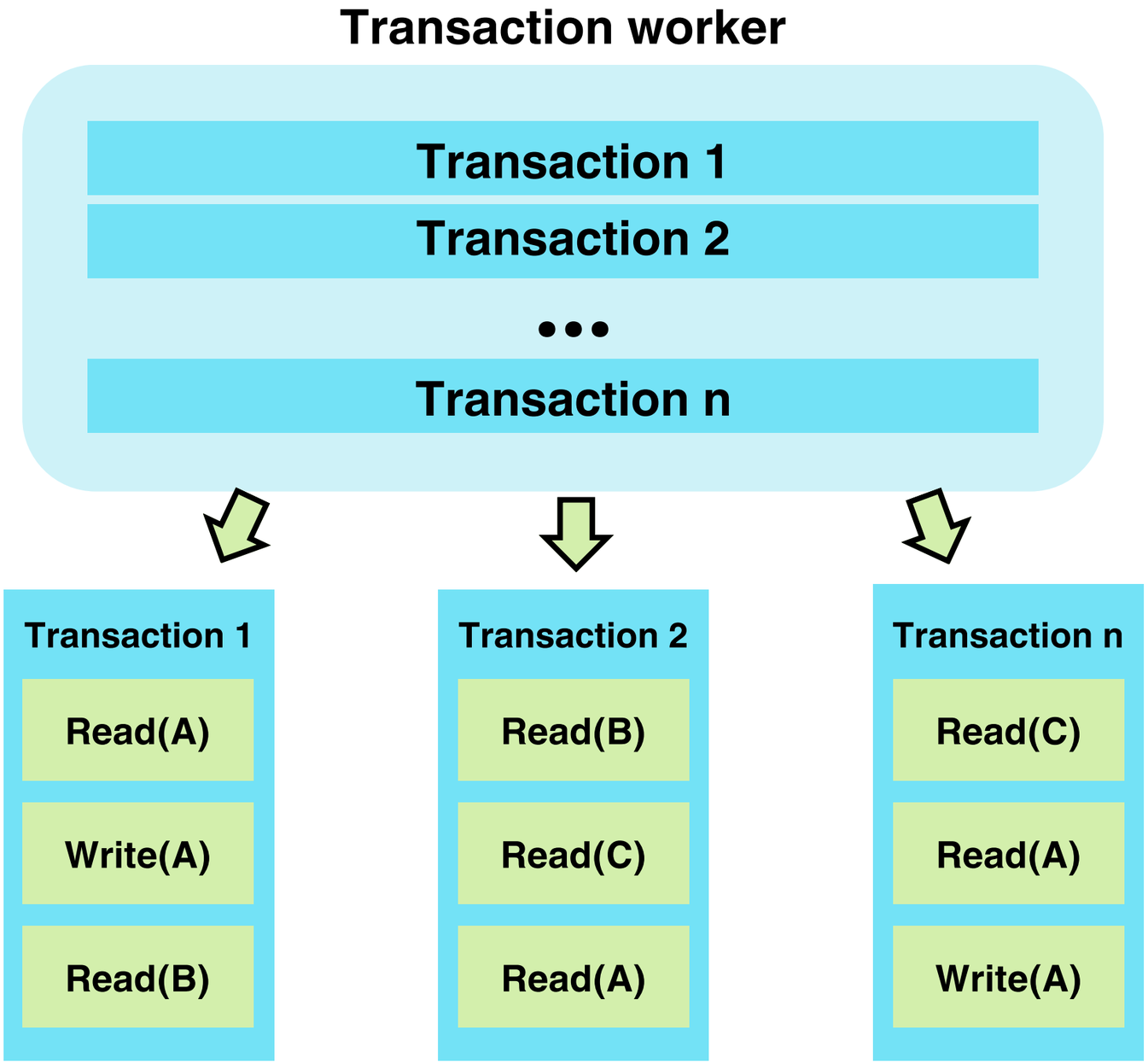
Transaction and Transaction Worker

Transaction:

- A collection of multiple queries/operations to be executed

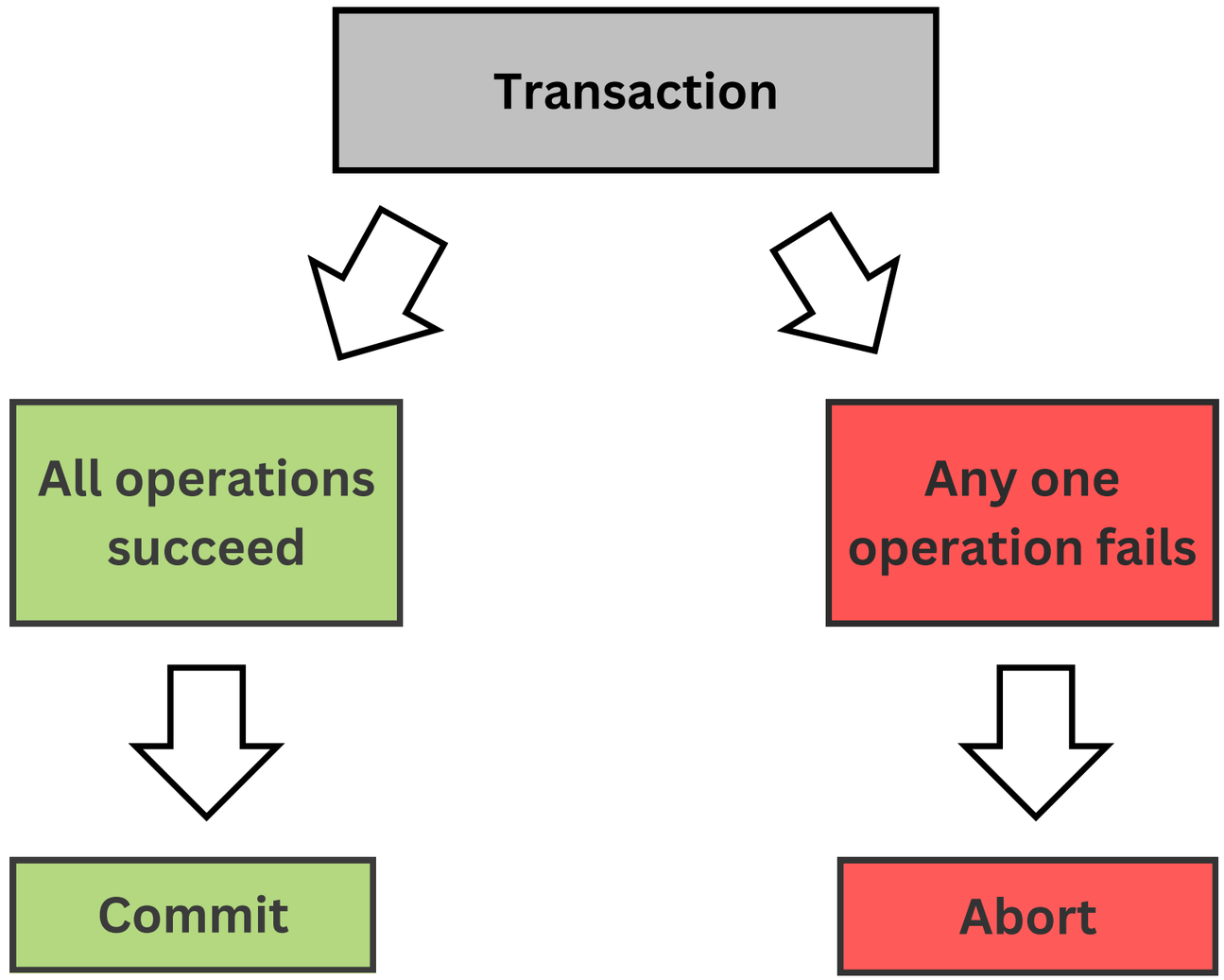
Transaction worker:

- List of transactions
- Contains status of each transaction
  - *True* for committed transaction
  - *False* for aborted transaction



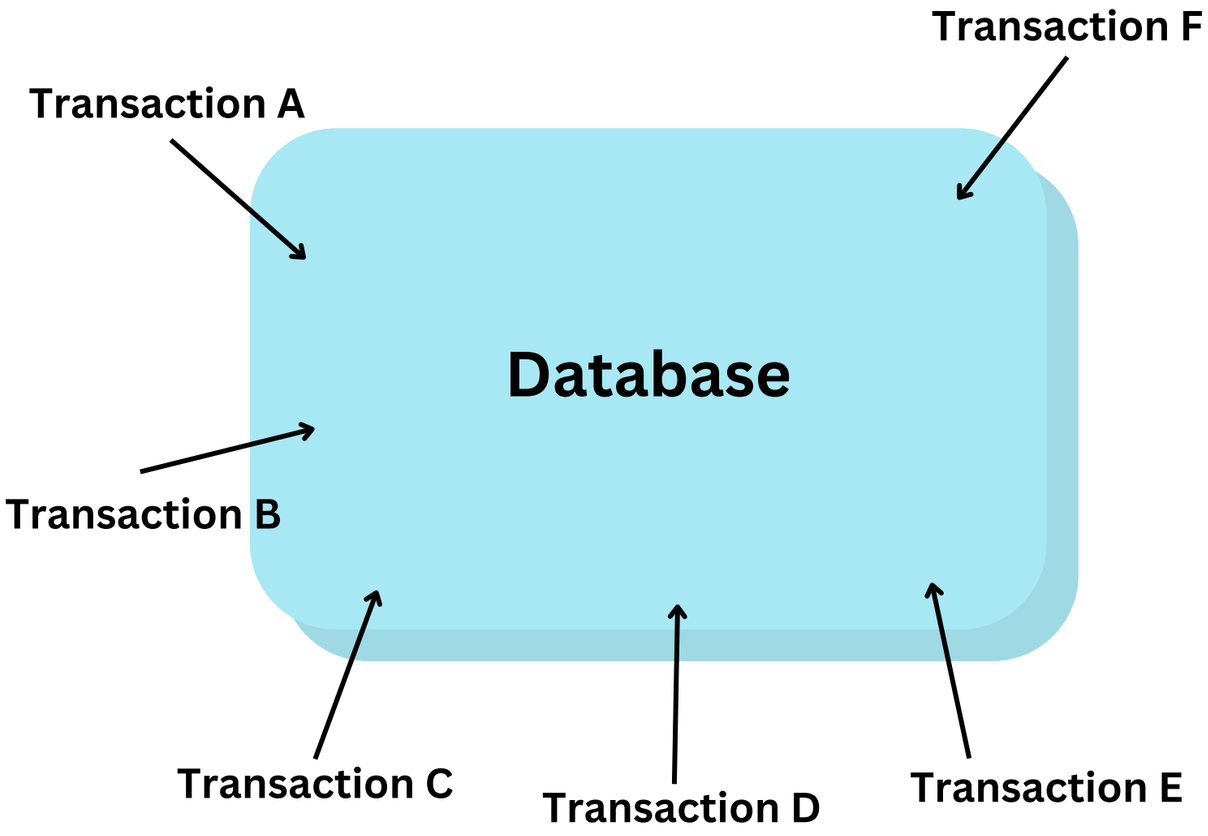
Atomicity

ALL operations of a transaction are either completed entirely or no operations at all



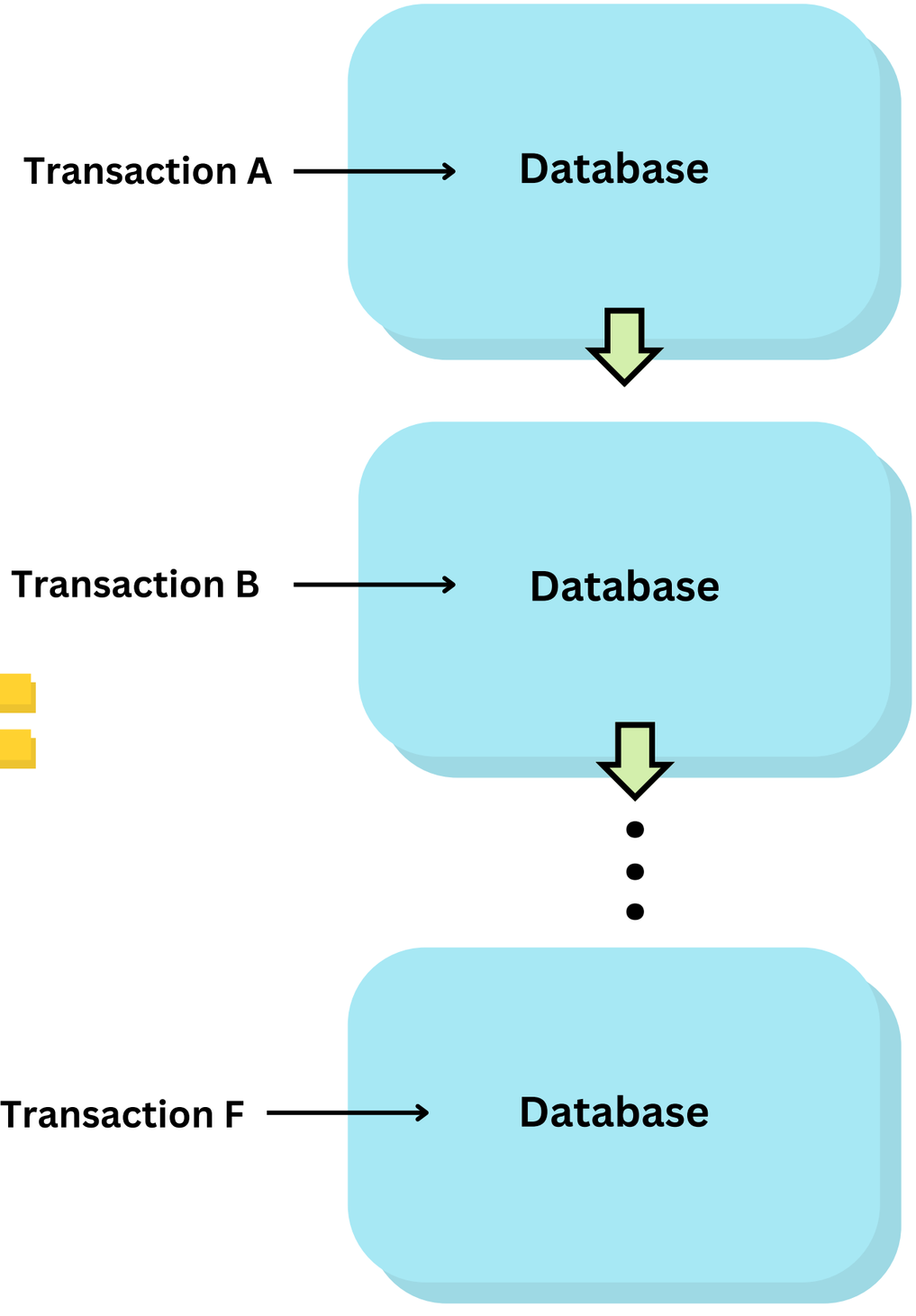
Isolation

Concurrent Execution of Transactions

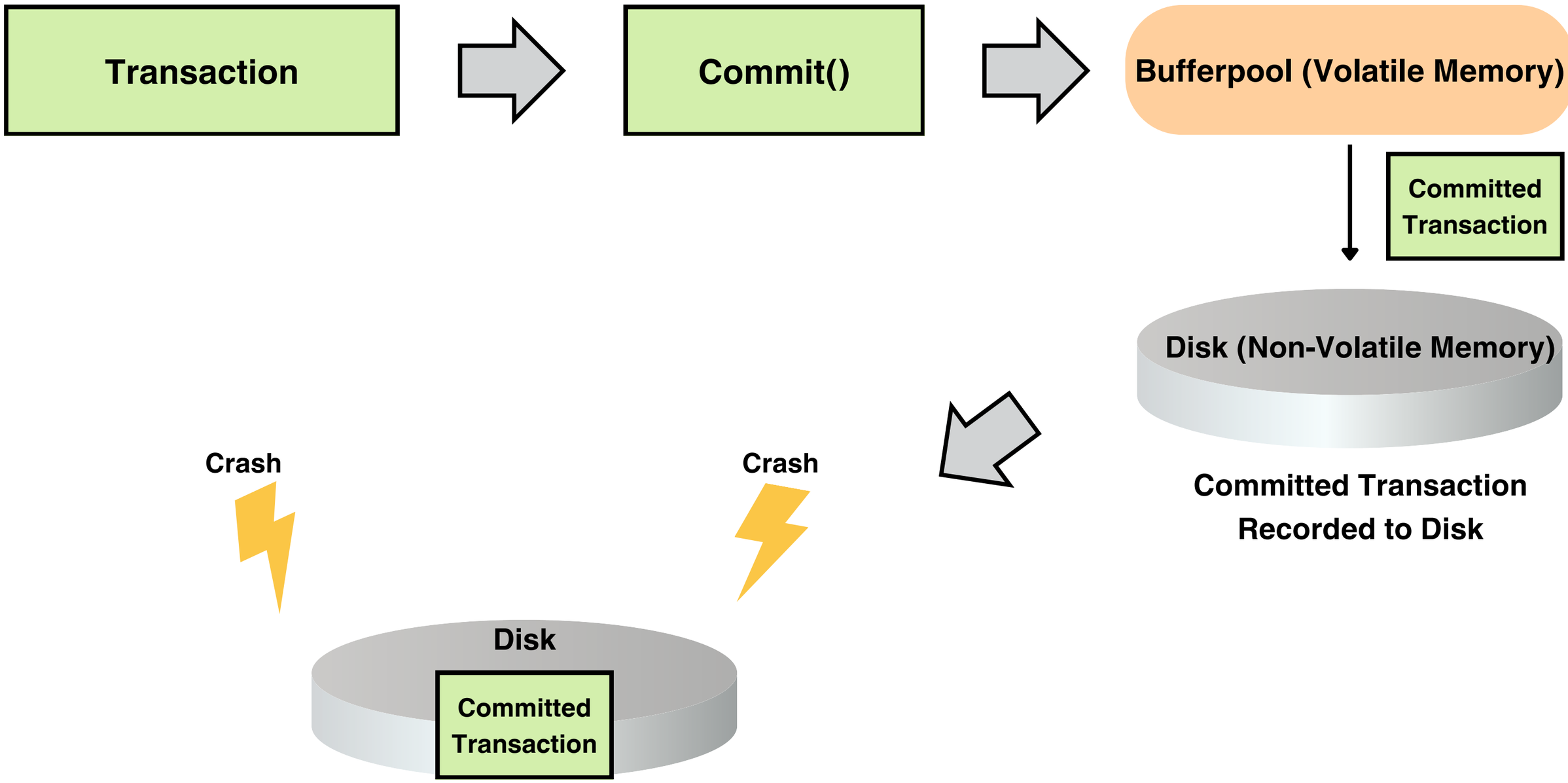


=

Serial Execution



Durability



Committed transaction stays in non-volatile memory even after crash

2. Concurrency Control

Shared and Exclusive Locks

Requests For \ Initially Holds	Shared	Exclusive
Shared	Grant	Don't Grant
Exclusive	Don't Grant	Don't Grant

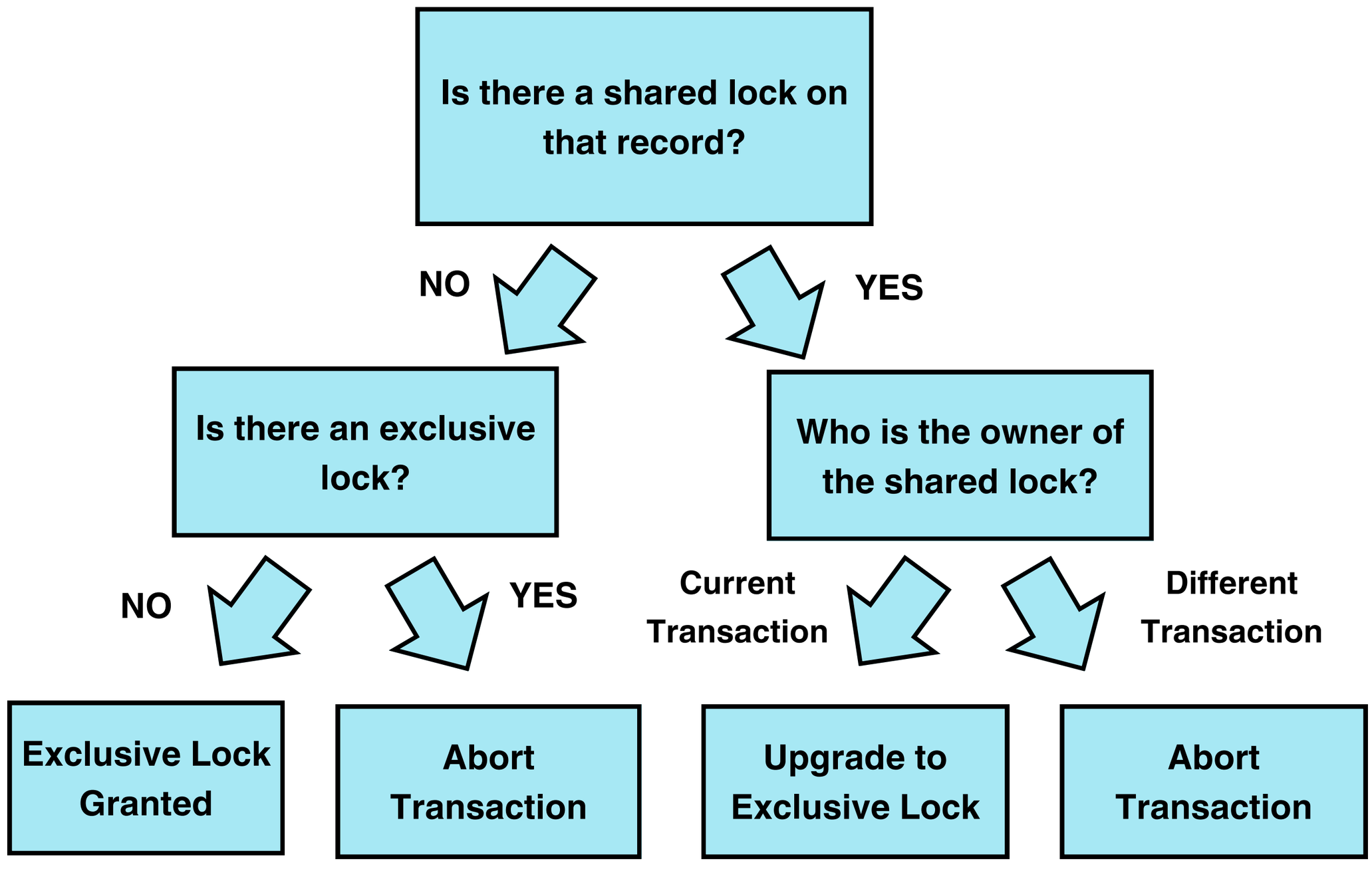
<u>Shared Lock *(reader lock)*</u>:

- Select
- Sum

<u>Exclusive Lock *(writer lock)*</u>:

- Insert
- Update
- Delete

Exclusive Locks and Upgrades



Lock Management

Motivation:

No 2 transactions should access the same resource that creates WR, WW, or RW conflict

Implementation:

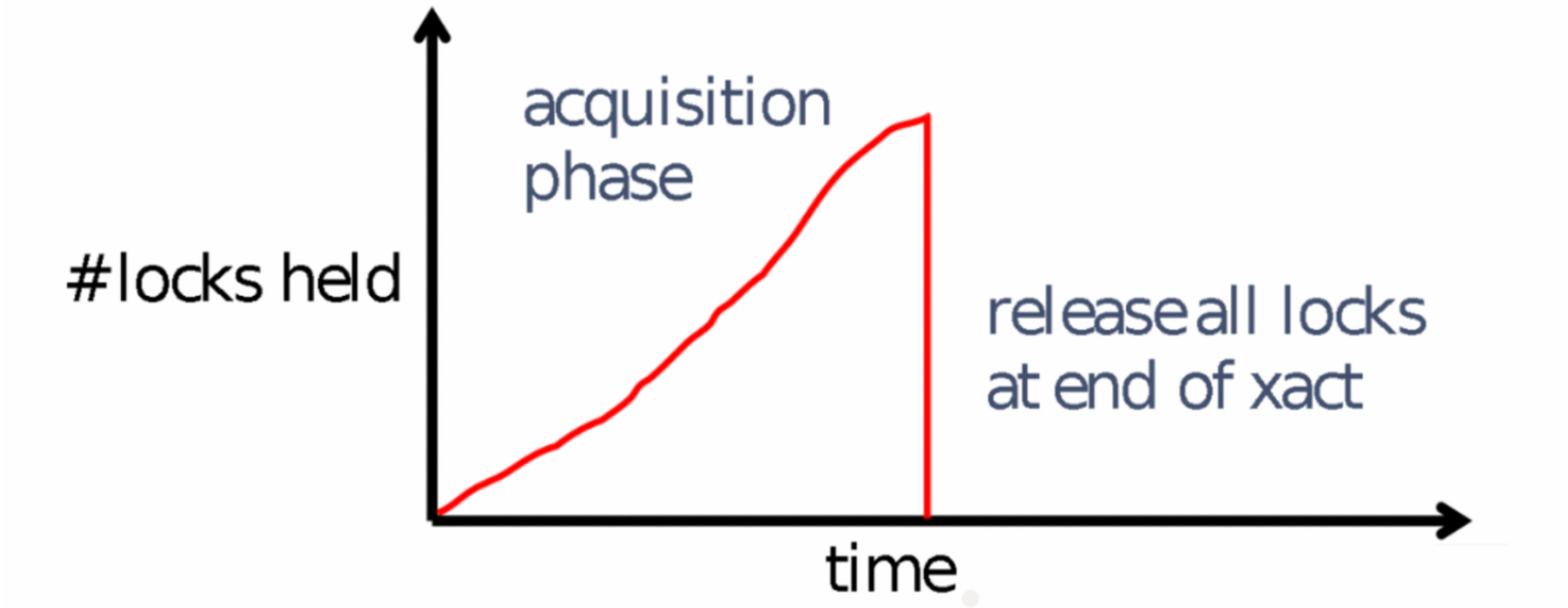
Locks are present on *record level* only

For each transaction,

- **Dictionary of keys mapped to RWLock objects**
- **Set of keys that have write (exclusive) lock**
- **Set of keys that have read (shared) lock**

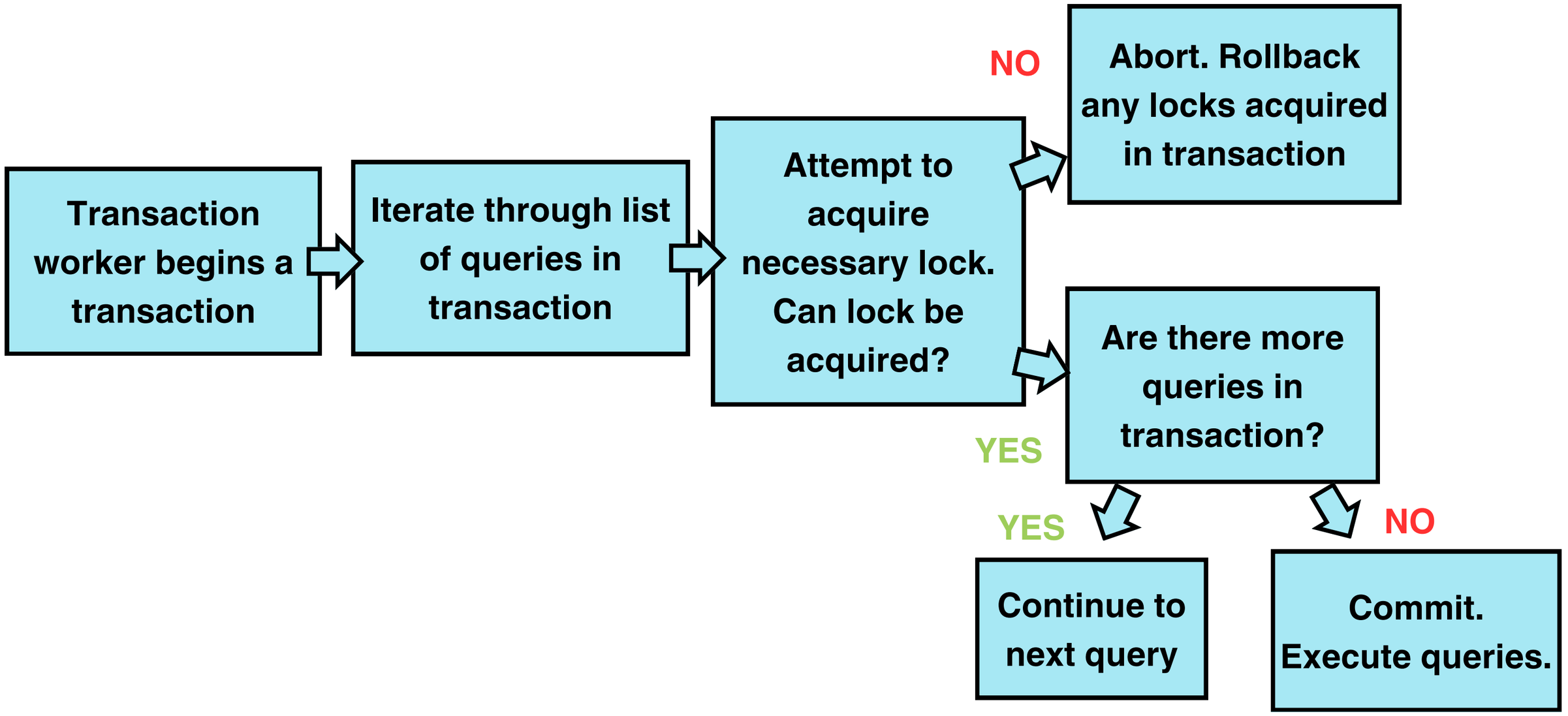
We use Python Threading, Lock(), Acquire(), and Release()

Concurrency Using Strict 2PL Policy



- **No wait policy aborts transaction if we fail to acquire lock**
- **All acquired locks are released <u>together</u> once the transaction is complete**

Commits and Aborts



Commits

Step 1: Acquire ALL the required locks

Step 2: Execute all queries

Step 3: When the transaction is ready to commit:

- **Write all changes to disk**
- **After the changes are written to disk, release ALL of its locks**

Step 4: Committed transactions return *true*

Aborts commonly occur when:

- failure to acquire locks (most common)

- DB crashes

- Power Failures

Aborts

Step 1: Attempt to acquire ALL the required locks

Step 2: When failure to acquire a lock, release all previously acquired locks by this transaction

Step 3: Aborted transactions return *false*

Step 4: Reattempt the transaction

Conclusion

Additional Testing and Improvements to M3 are still being explored implemented :)

Improvements for the future: Using QueCC instead of 2PL for better concurrency, using a different programming language to achieve true parallelism

Takeaways from the entire project: How to effectively communicate within a group, collaborative efforts during the coding process, technical skills related to columnar database, debugging techniques, performance testing

A very special thank you to Professor Sadoghi and the TAs for this learning opportunity, constant support and constructive feedback.