Milestone 3

Yingchen Gu, Glenn Chen
Rishika Roy, Kaleb Crans,
Ryan Kim

Overview

Transaction Semantics

Transaction Class

Transaction Worker Class

Atomicity, Consistency, Isolation, Durability (ACID)

Concurrency Control

Locks

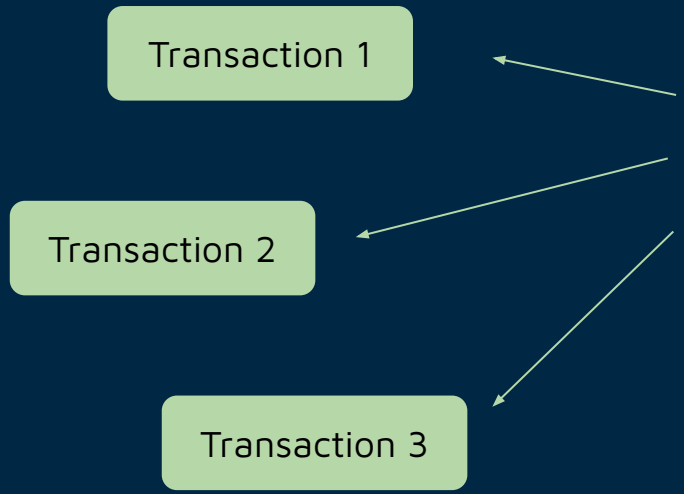
Aborting

Committing

Transaction Semantics

Transaction Class

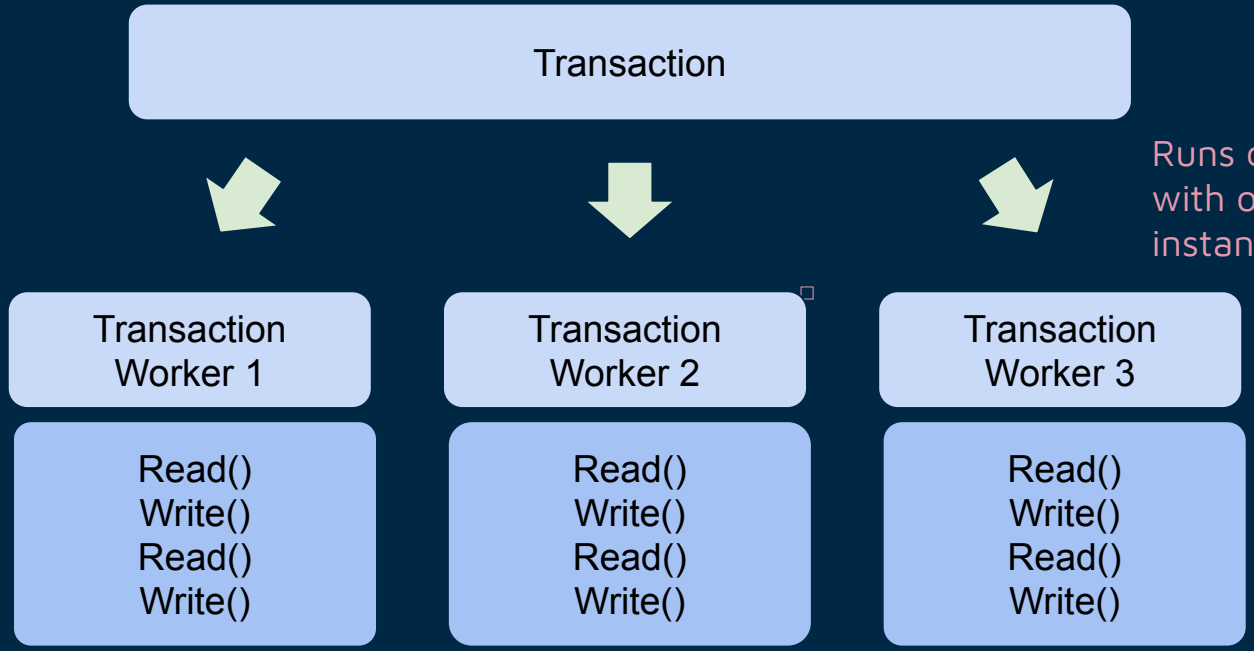
- Contains queries to be executed



Queries get added to the transactions using:

```
def add_query(self, query, table, *args)
        ...
```

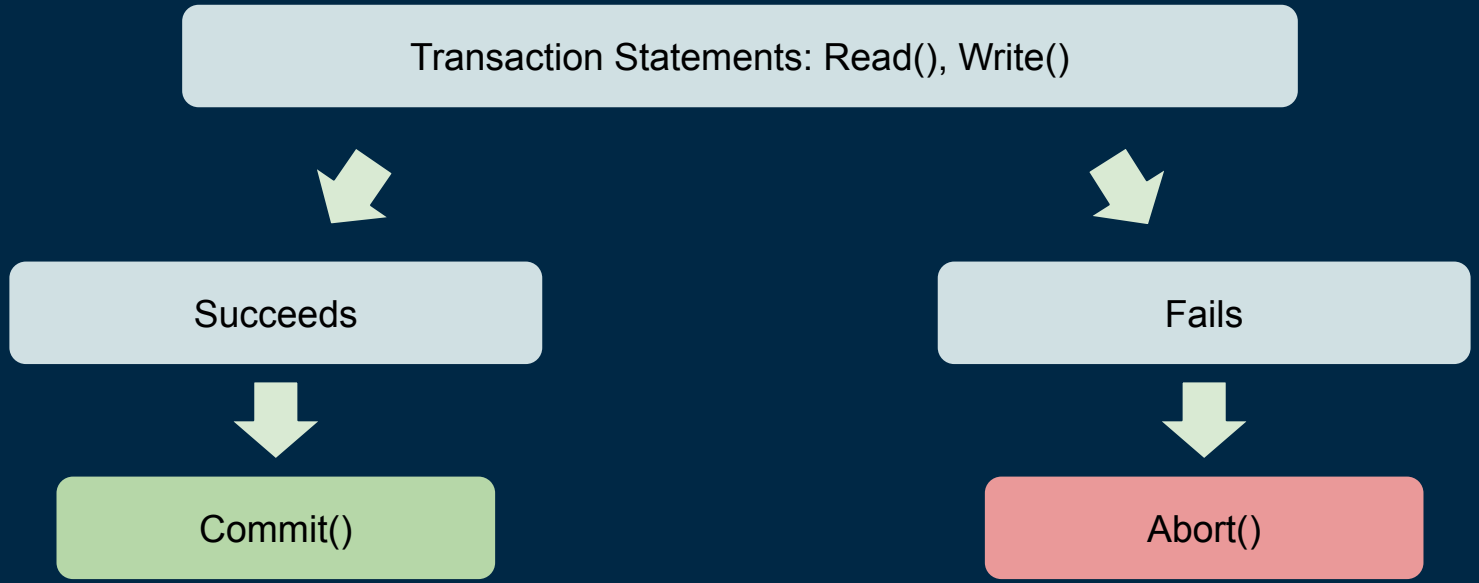
Transaction Worker Class



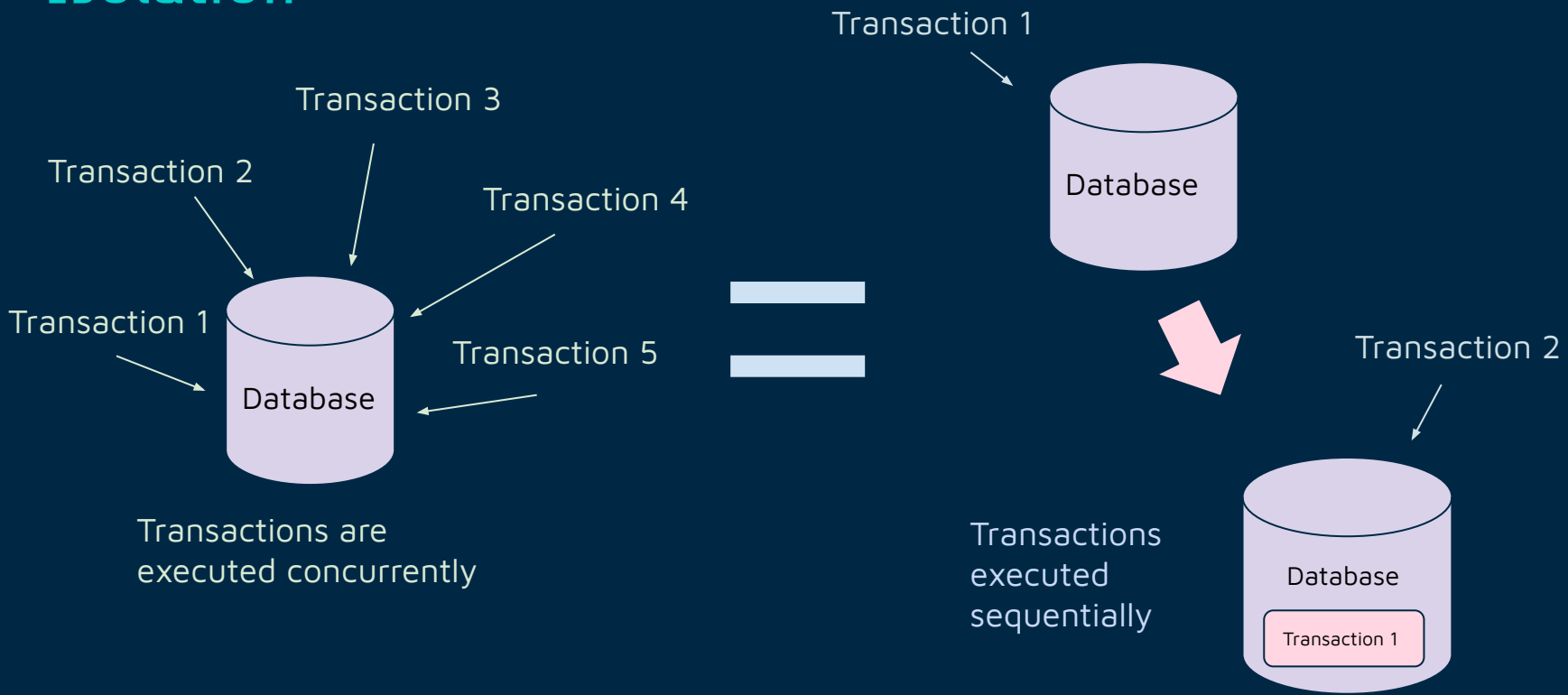
Runs concurrently with other worker instances

Creates a thread and runs the transactions as a thread

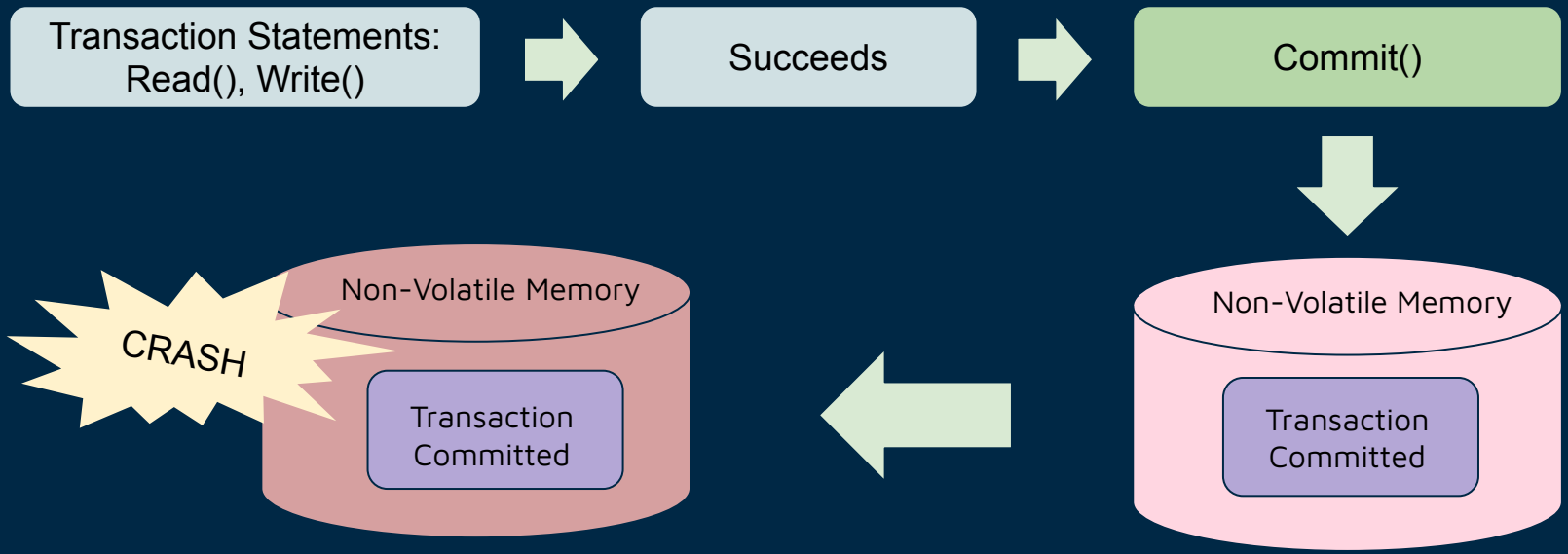
Atomicity


Transaction Statements: Read(), Write()
Succeeds
Fails
Commit()
Abort()
Writes to Disk
Everything is left unchanged
Actions are logged to make sure changes can be undone if aborted

Isolation


Transaction 3
Transaction 2
Transaction 4
Transaction 1
Transaction 5
Database
Transactions are executed concurrently
Transaction 1
Database
Transaction 2
Transactions executed sequentially
Database
Transaction 1

Durability



Once a transaction is committed, it will stay in the non-volatile memory

The committed transaction stays in there even if the system fails

Concurrency Control

Mutually Exclusive Locks

Use Lock object from threading module in Python

Lock.acquire() - thread waits if lock is taken, proceeds if lock is available

Lock.release() - thread releases the lock making it available to be acquired again

Prevent race conditions in critical sections (e.g. creating new page ranges, updating pin counts, etc.)

How it looks in code:

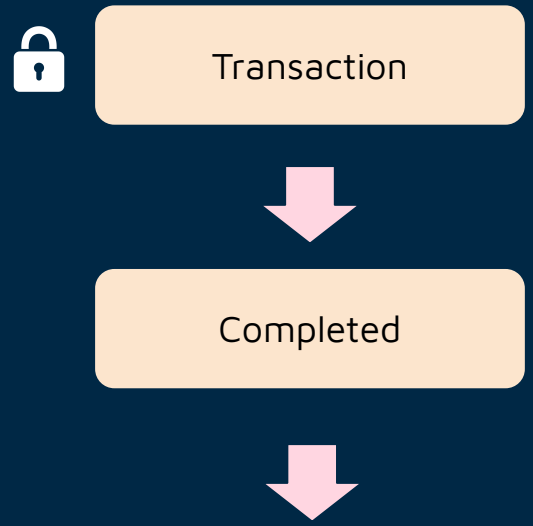
```
lock.acquire()

// Critical section

...

// only one thread can be running
// here at a time

...


// End of critical section

lock.release()
```

2PL – Strict 2 Phase Locking

2PL

- Only allows serializable schedules
- To read:
  - Obtain an shared lock
- To write:
  - Obtain an exclusive lock

No more locks can be acquired after a lock has been released



Locks on Records

Choose lock granularity on records to help minimize conflicts

HashMap using RIDs as keys and the list of locks on the record as values

Values of the HashMap contain the transaction holding the lock, and the type of lock

Shared lock (s) - requested when a transaction wants to read a record

Exclusive lock (x) - requested when a transaction wants to write a record

RID	[(Transaction, Type)]
1	(1, ‘x’)
2	(1, ‘s’), (3, ‘s’)
3	(1, ‘s’), (2, ‘s’), (3, ‘s’)
4	(3, ‘x’)

Aborting

Strict 2PL - transaction aborts if at any point it is unable to acquire any necessary locks

Abort process

- Set the RIDs of all records added by the transaction to be DELETED_RID_VALUE
- Release all locks belonging to the transaction

RID	[(Transaction, Type)]
1	(1, ‘x’)
2	(1, ‘s’), (3, ‘s’)
3	(1, ‘s’), (2, ‘s’), (3, ‘s’)
4	(3, ‘x’)

Transaction 3 already has exclusive lock on RID 4

Transaction 4 aborts

Committing

If the transaction is able to acquire locks on all the records it needs, it commits

Commit process

- Write the changes to disk
- Release all locks belonging to the transaction

Transaction	RIDs Read or Written
1	1, 2, 3
2	3
3	2, 3, 4

RID	[(Transaction, Type)]
1	(1, ‘x’)
2	(1, ‘s’), (3, ‘s’)
3	(1, ‘s’), (2, ‘s’), (3, ‘s’)
4	(3, ‘x’)

Thank You!