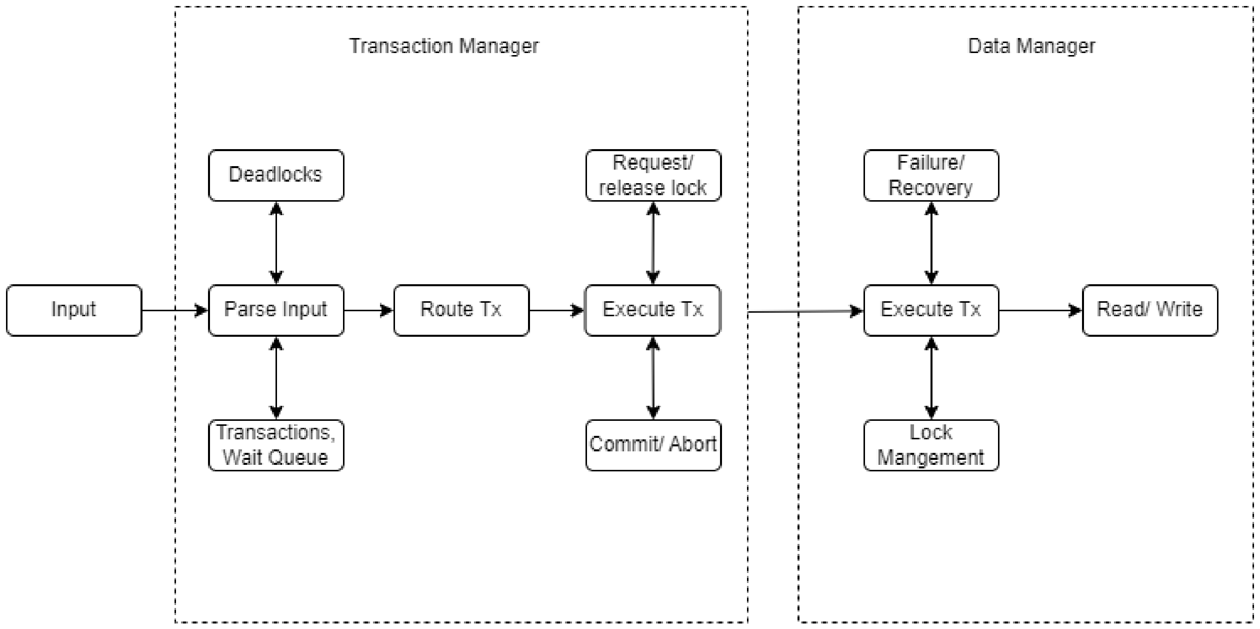
Design Document: Replicated Concurrency control and Recovery

Teammates: Deep Mehta (dnm7500), Sarvani Nadiminty (sn2884)

Flow Diagram:



Module Signatures:

```
# Deep Mehta
class Transaction:
    def __init__(self, id):
        self.id = id
        self.data = {}
        self.locks = {}
        self.RO_flag = False
        self.start_time = time.time()
        self.sites_accessed = []  # Compare these during commit -if s fails, abort Ts
    def read(self, var, sites):
        """ Read from site s """
        ...
    def write(self, var, sites):
        """ Write/ Save x in transaction T """
        ...
    def request_lock(self):
        """ Request for a new lock """
        ...
    def release_lock(self):
        """ Release locks on end """
        ...
    def commit(self):
        """ Validate and commit all updated variables into all up_sites """
        ...
    def abort(self):
        """ Release locks and abort T """
        ...
```

```python
from data_manager import DataManager
from transaction import Transaction

# Deep Mehta
class TransactionManager:
    def __init__(self):
        self.transactions = [...]  # T1, T2
        self.wait_queue = [...]
        self.dm_handler = DataManager()
    def input_parser(self):
        """ Read inputs one by one execute them """
        ...
    def begin_transaction(self, tx):
        """ Create a Transaction node and add it to the list """
        transaction = Transaction(tx)
        self.transactions.append(transaction)
    def execute_transaction(self, tx):
        """ Execute transaction tx """
        ...
    def end_transaction(self, tx):
        """" Commit - if any, and delete tx from list """
        ...
    def routing(self, var):
        """ Find the site to work with for T """
        if var.id % 2 == 1:
            sites = [(1 + var.id) % 10]
        else:
            sites = self.dm_handler.up_sites
        return sites
    def deadlock_cycle(self):
        """ Somehow check for a deadlock - maybe DFS on graph """
        ...
    def printer(self, message):
        """ Print whatever you want """
        print(message)
    def dump(self):
        """ Get all variables from all sites and dump """
        ...
```

```python
# Sarvani Nadiminty
class Site:
    def __init__(self, id):
        self.id = id
        self.data = {}  # list of variables in the site
        self.status = 1  # 1 for up, 0 for down
        self.locks = {}
    def read_data(self):
        """ Returns data on var x in the site """
        ...
    def write_data(self):
        """ Commit data into the file/ storage """
        ...
    def failure(self):
        """ Simulate a site failure """
        # Make all replicated variables unavailable
        ...
    def recovery(self):
        """ Recover a site from failure """
        ...
    def dump(self):
        """ Returns the data in the site s """
        return self.data
```

```python
from sites import Site

# Sarvani Nadiminty
class DataManager:
    def __init__(self):
        self.sites = [Site(i+1) for i in range(10)]
        self.up_sites = []
        self.locks = {}  # store all locks on all vars in all sites
        self.RO_cache = {}  # dictionary of RO data
    def read(self):
        """ Validate tx, locks and read if allowed """
        ...
    def write(self):
        """ Validate tx, locks and write if allowed """
        # Update RO_cache accordingly
        ...
    def set_lock(self):
        """" Update lock status on site s for var x """
        ...
    def read_lock_status(self):
        """ Return lock status  """
        ...
    def handle_failure(self):
        """ Simulate failure in site s """
        ...
    def handle_recovery(self):
        """
            Simulate recovery in site s;
            Update committed values for replicated data
        """
        ...
    def dump(self):
        """ Get all variables from all sites and dump """
        ...
```