# CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

## CONCURRENCY CONTROL

Lecture B

## TRANSACTIONS

# Transaction

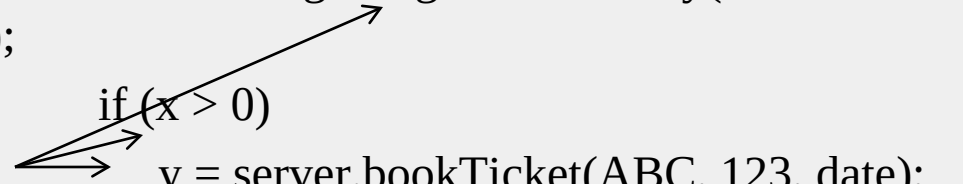
- Series of operations executed by client
- Each operation is an RPC to a server
- Transaction either
  - Completes and *commits* all its operations at server
    - Commit = reflect updates on server-side objects
  - Or *aborts* and has no effect on server

# Example: Transaction

Client code:

```
int transaction_id = openTransaction();
x = server.getFlightAvailability(ABC, 123, date);
if (x > 0)
    y = server.bookTicket(ABC, 123, date);
server.putSeat(y, “aisle”);
// commit entire transaction or abort
closeTransaction(transaction_id);
```

RPCs

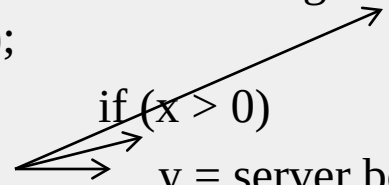


# Example: Transaction

Client code:

```
int transaction_id = openTransaction();
x = server.getFlightAvailability(ABC, 123, date);
if (x > 0)
    y = server.bookTicket(ABC, 123, date);
server.putSeat(y, “aisle”);
// commit entire transaction or abort
closeTransaction(transaction_id);
```

RPCs



```
// read(ABC, 123, date)

// write(ABC, 123, date)
// write(y)
```

# Atomicity and Isolation

- Atomicity: All or nothing principle: a transaction should either i) complete successfully, so its effects are recorded in the server objects; or ii) the transaction has no effect at all.
- Isolation: Need a transaction to be indivisible (atomic) from the point of view of other transactions
  - No access to intermediate results/states of other transactions
  - Free from interference by operations of other transactions
- But…
- Clients and/or servers might crash
- Transactions could run concurrently, i.e., with multiple clients
- Transactions may be distributed, i.e., across multiple servers

# ACID Properties for Transactions

- **A**tomicity: All or nothing
- **C**onsistency: If the server starts in a consistent state, the transaction ends the server in a consistent state.
- **I**solation: Each transaction must be performed without interference from other transactions, i.e., non-final effects of a transaction must not be visible to other transactions.
- **D**urability: After a transaction has completed successfully, all its effects are saved in permanent storage.

# Multiple Clients, One Server

- What could go wrong?

# 1. Lost Update Problem

**Transaction T1**

```
x = getSeats(ABC123);
   // x = 10
if(x > 1)
  x = x – 1;
write(x, ABC123);


commit
```

**Transaction T2**

```
x = getSeats(ABC123);
if(x > 1)      // x = 10


   x = x – 1;
write(x, ABC123);


commit
```

At Server: seats = 10

seats = 9

seats = 9

**T1's or T2's update was lost!**

# 2. Inconsistent Retrieval Problem

**Transaction T1**

x = getSeats(ABC123);

y = getSeats(ABC789);

write(x-5, ABC123);

**// ABC123 = 5 now**

write(y+5, ABC789);

commit

**Transaction T2**

x = getSeats(ABC123);

y = getSeats(ABC789);

**// x = 5, y = 15**

print("Total:" x+y);

**// Prints "Total: 20"**

commit

At Server:
ABC123 = 10
ABC789 = 15

**T2's sum is the wrong value!**
**Should have been "Total: 25"**

# Next

- How to prevent transactions from affecting each other