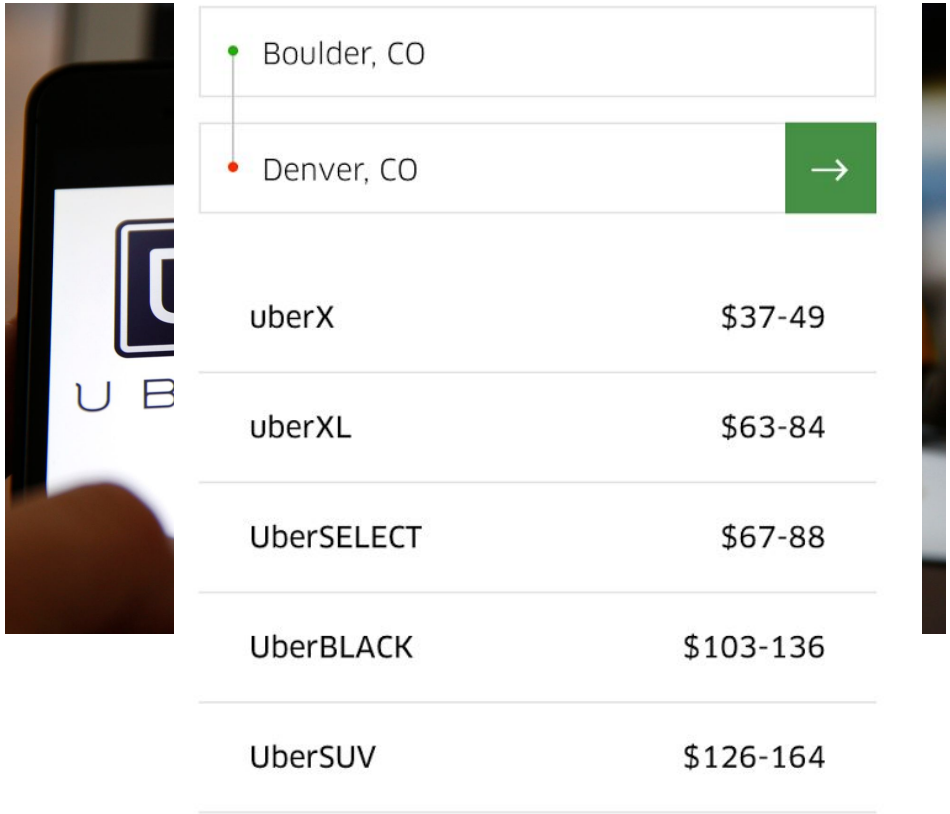
# Cab Sharing System

Mohammad Hashemi
Praveen Kumar Devaraj
Nachiket Bhagwat

CSCI 5448
University of Colorado Boulder
Spring 2016

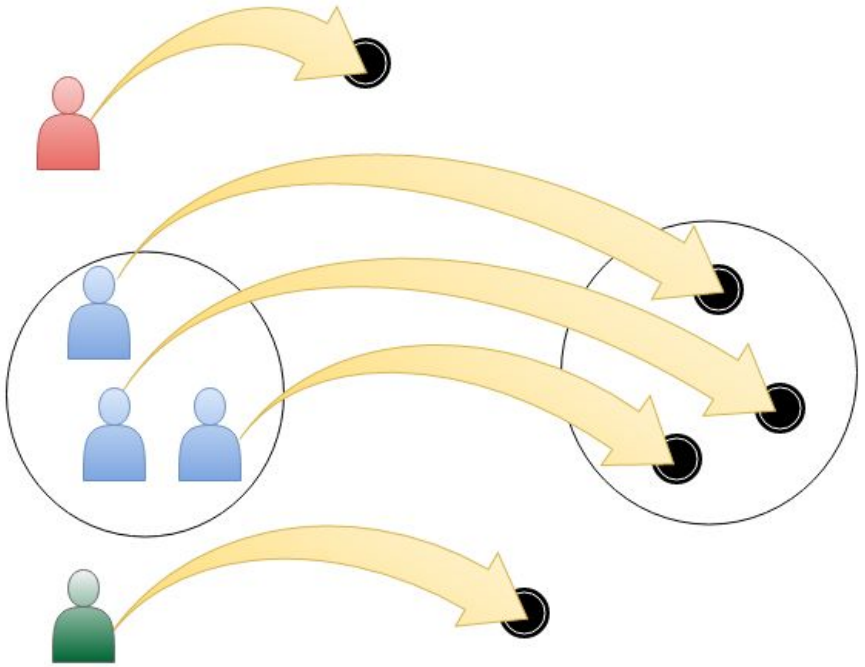
# Introduction

- Related Work
  - Uber
- Problem With Uber
  - Expensive



# Solution

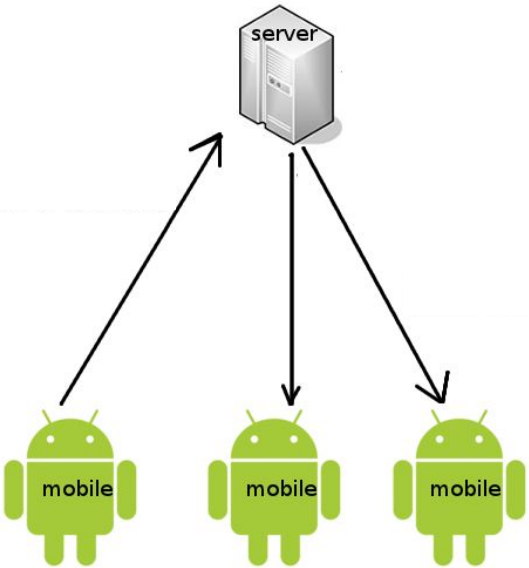
- Share a cab among passengers
- Group passengers
  - Close pickup locations
  - Close dropoff locations



# Architectural Pattern

- Client/Server

- Centralized view of users
  - Matching preferences
  - Grouping passengers



# Technology

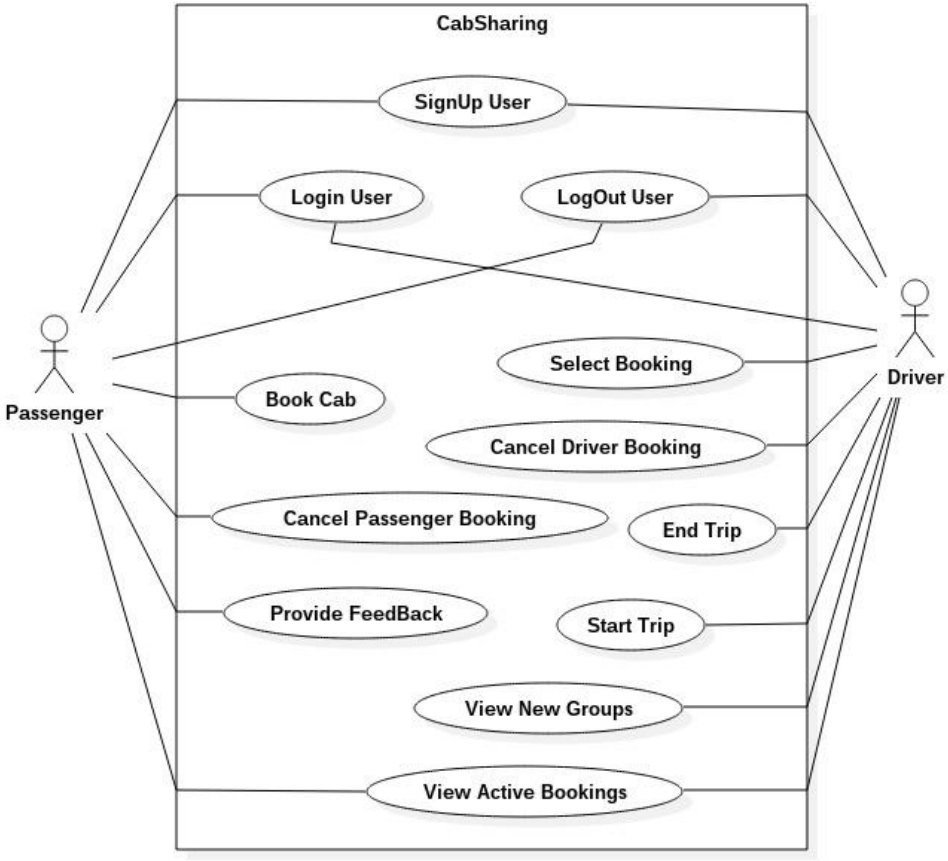
- Server Side
  - Python
  - MongoDB


- Client Side
  - Android Application

# System Design

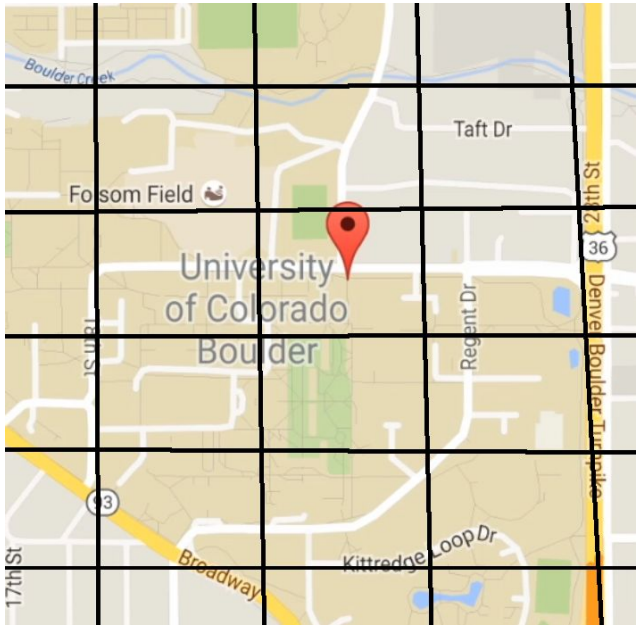
- 12 High-Level Use Cases

- 2 Different Users
  - Passenger
  - Driver



# Booking Use Case

- ❏ Find User Grid
- ❏ Add Booking to User Grid
- ❏ Add User to matching Group
- ❏ Update Group
- ❏ Broadcast Full Group to Driver in same Grid
- ❏ Send notification if a Driver accepts a Group



# Grid System

- At 38 degrees North latitude, one minute equals 1.15 miles
- GPS allows to track location accurately
- Divide groups of users in Grids of 1 minute difference
- Distance remains manageable for non polar countries

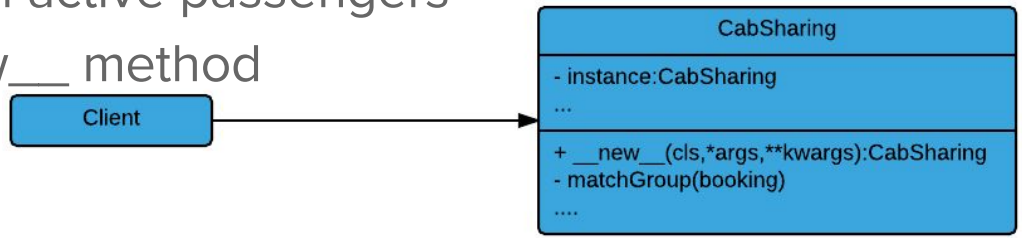
# End Trip

- End Trip ⇨ Generate Travel Summary ⇨ Payment Calculation
- Travel Summary is pushed to Server.

$$\text{Individual Fare} = \frac{\text{Total Fare} + (\text{Sharing fee} * \text{Number of Sharing passengers})}{\text{Total Number of Passengers}}$$

# Singleton Design Pattern

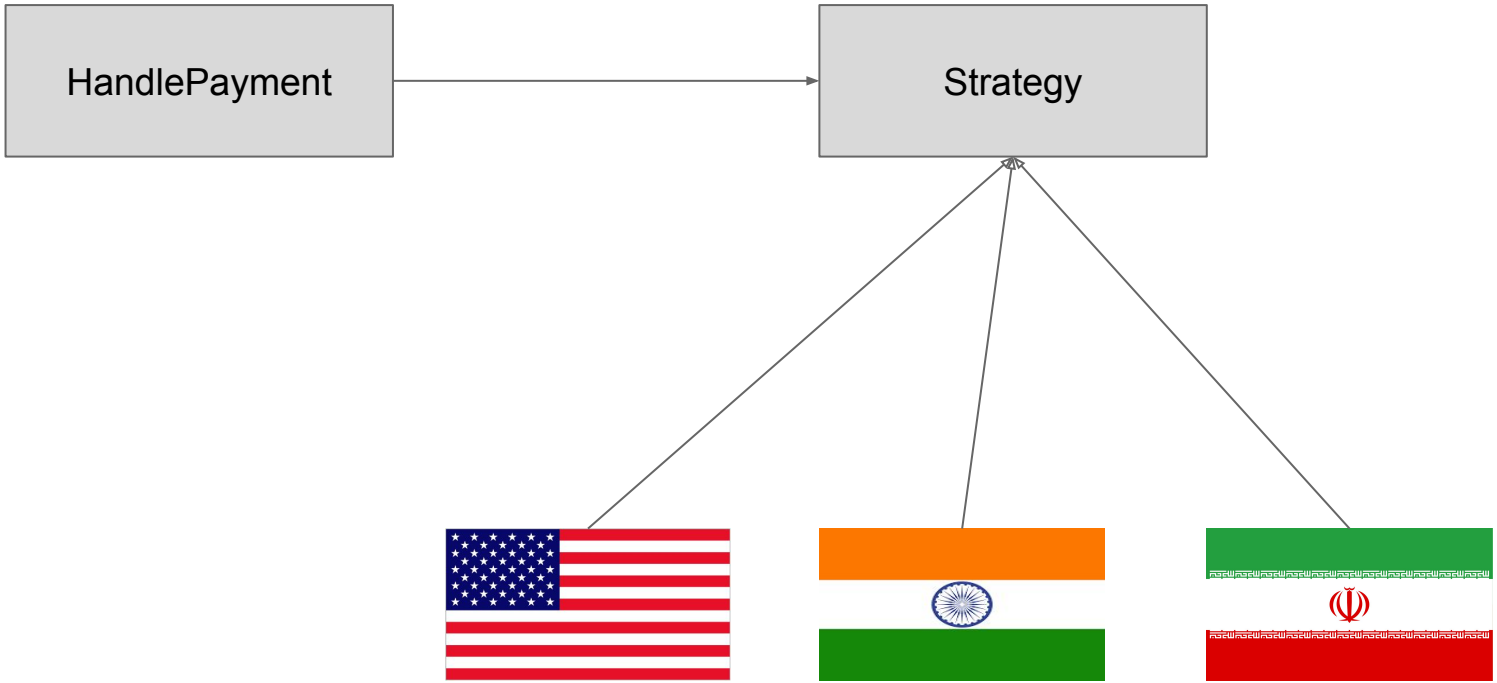
- CabSharing as a Singleton class
  - Centralized view of all active passengers
  - Overriding the __new__ method



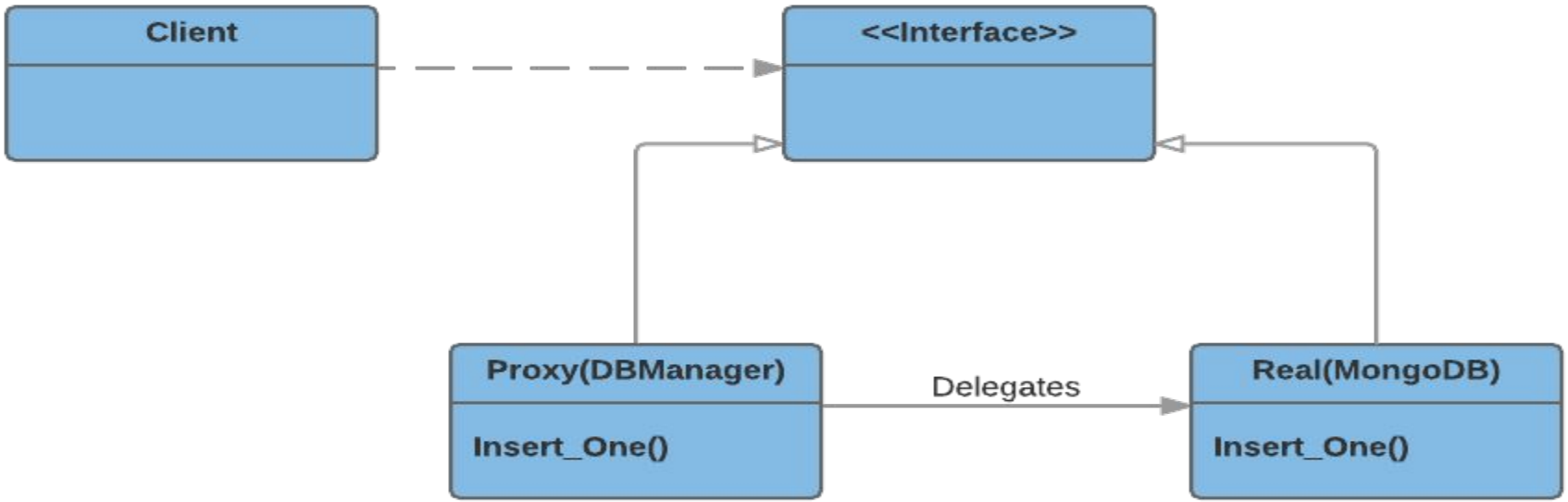

- Python Decorator
  - language feature
  - To adding synchr

```
@synchronized
def __new__(cls, *args, **kwargs):
        if not cls.__instance:
            cls.__instance = super(CabSharing, cls).__new__(cls, *args, **kwargs)
            print cls.__instance
        return cls.__instance
```

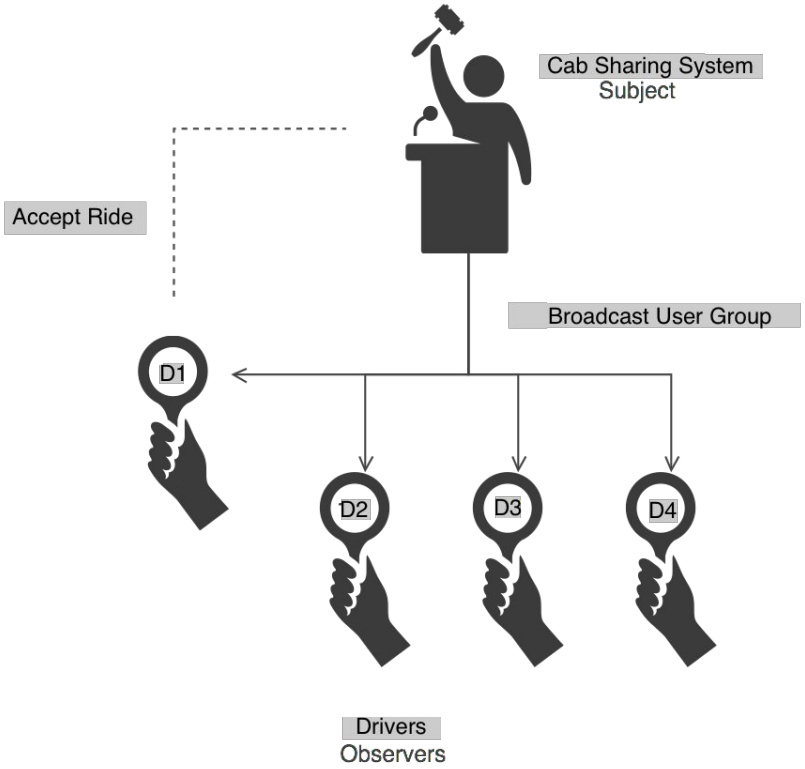
# Strategy Design Pattern



# Proxy Design Pattern



# Observer Design Pattern



# Demo