Uber HLD Notes - Location Sharing & System Design

# 1. Requirements & Considerations

- 500M total customers, 1M daily active users (DAU)  
- 1M drivers, 500k DAU  
- Drivers share location every 3 seconds; when a ride is booked, location sharing becomes real-time.  
- Non-Functional Requirements: Availability, Scalability, Low Latency

# 2. Location Tracking System

Drivers update their location frequently, and customers need real-time tracking during a trip.  
A Data Structure (DS) is required to manage frequent driver location updates geographically.

# 3. Using QuadTree for Spatial Indexing

- QuadTree divides space into 4 child regions per node (recursive partitioning)  
- Each grid contains: gridId, latitude, longitude, and possibly driver info  
- Querying is done using a bounding box (e.g., Lat between (X±y), Long between (Y±d))  
- Dynamic grids can help with high/low density areas but are harder to maintain due to location churn

# 4. Caching Strategy

- Driver locations are cached every 3 seconds  
- QuadTree data structure is updated every 15 seconds  
- Driver Object = 36 bytes: DriverId(Long), OldLat, OldLong, NewLat, NewLong  
- Total = 1M drivers \* 36B = 36 MB cache (segmented by region/country)

# 5. Bandwidth Calculation

- Location payload: DriverId (4 bytes) + NewLat/Long (16 bytes) = 20 bytes  
- 500k active drivers -> 500k \* 20B = 10 MB every 3 seconds

# 6. Communication Models

1. API-Based Pull (polling)  
2. Pub/Sub Model (Kafka):  
 - DriverId acts as a publisher  
 - CustomerId subscribes to updates from a specific DriverId

# 7. Partitioning Options

- Partition based on DriverId: evenly distributes load  
- Partition based on Ranking (preferred driver selection): supports dynamic routing and filtering

# 8. Database Schema (SQL & NoSQL)

## SQL Tables (Structured, Relational)

### Table: users

CREATE TABLE users (  
 user\_id UUID PRIMARY KEY, -- Unique ID for each customer  
 name VARCHAR(255),  
 phone VARCHAR(20),  
 email VARCHAR(255),  
 created\_at TIMESTAMP  
);

### Table: drivers

CREATE TABLE drivers (  
 driver\_id UUID PRIMARY KEY, -- Unique ID for each driver  
 name VARCHAR(255),  
 phone VARCHAR(20),  
 email VARCHAR(255),  
 status VARCHAR(50), -- Available, Busy, Offline  
 created\_at TIMESTAMP  
);

### Table: rides

CREATE TABLE rides (  
 ride\_id UUID PRIMARY KEY,  
 user\_id UUID REFERENCES users(user\_id),  
 driver\_id UUID REFERENCES drivers(driver\_id),  
 pickup\_location TEXT,  
 dropoff\_location TEXT,  
 status VARCHAR(50), -- Pending, Active, Completed, Cancelled  
 start\_time TIMESTAMP,  
 end\_time TIMESTAMP  
);

## NoSQL Collections (Flexible, High-Volume)

### Collection: driver\_location\_updates

{  
 "driver\_id": "UUID",  
 "timestamp": "2024-04-25T12:00:00Z",  
 "lat": 28.6139,  
 "long": 77.2090,  
 "status": "available"  
}

Updated every 3 seconds. Used for real-time tracking and matching nearby drivers.

### Collection: quad\_tree\_grid

{  
 "grid\_id": "IN-DEL-001",  
 "lat\_range": [28.60, 28.65],  
 "long\_range": [77.20, 77.25],  
 "drivers": ["driver\_1", "driver\_2", "..."]  
}

Used to find drivers in a specific geo area. Updated every 15 seconds from cache.

### Collection: ride\_events

{  
 "event\_id": "UUID",  
 "ride\_id": "UUID",  
 "timestamp": "2024-04-25T12:01:00Z",  
 "event\_type": "location\_update",  
 "location": {"lat": 28.62, "long": 77.21}  
}

Captures ride telemetry and live updates, useful for analytics and playback.

# 9. Cloud Services (Optional Components)

- Kafka / PubSub: Real-time communication between drivers and users.  
- Redis / Memcached: Store recent location updates before persisting.  
- AWS DynamoDB / MongoDB: Store location updates and geospatial grid data.  
- AWS RDS / Cloud SQL: Store structured data like users, drivers, and rides.  
- Amazon S3 / GCS: (Optional) Store logs, events, archived ride data.