Uber System Architecture

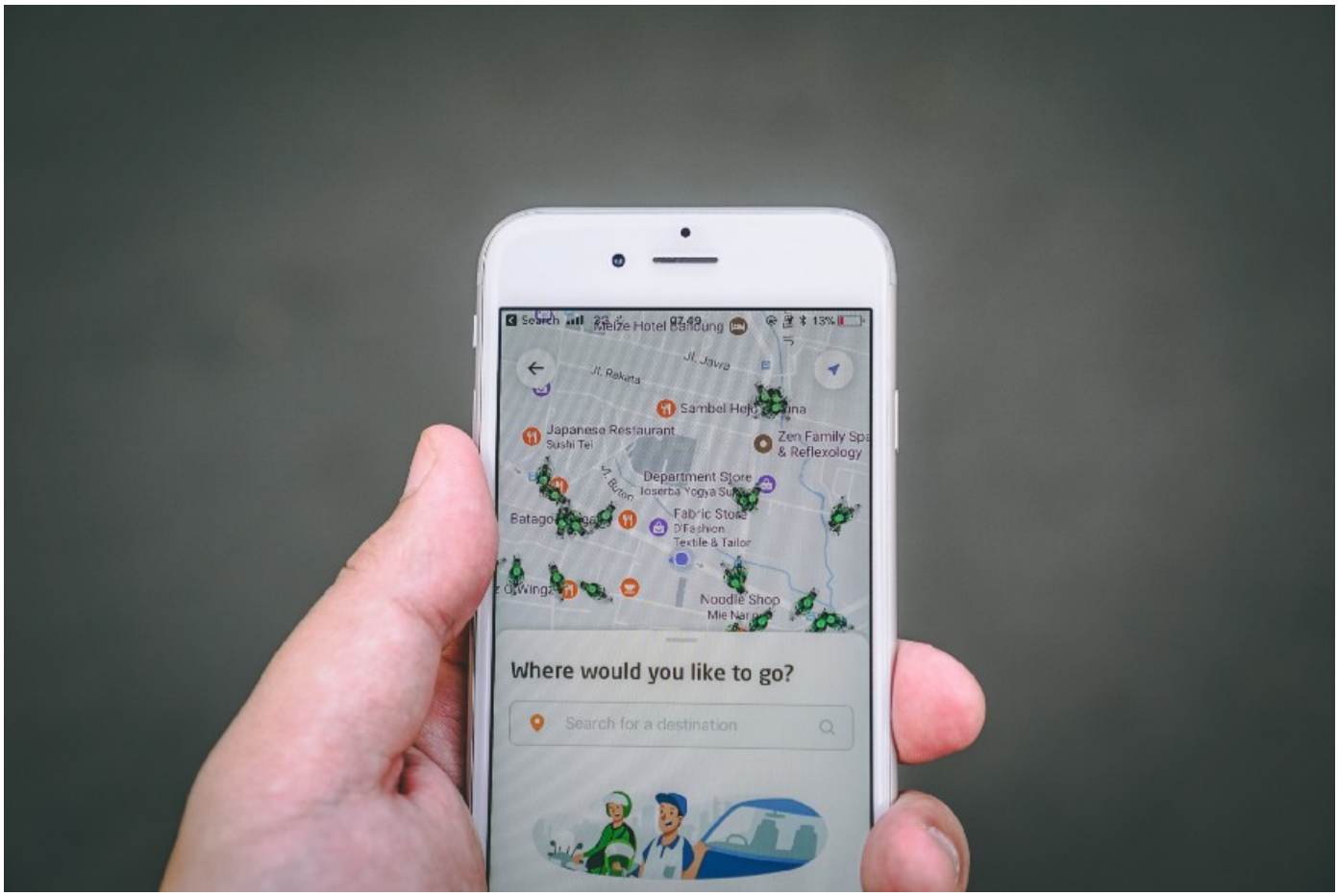


Photo by Fikri Rasyid on Unsplash

Uber began from a monolithic architecture to service-oriented Architecture. From the beginning, Uber is built for San Francisco, which they called UberBlack. As core domain models grew and new features were added, the components become tightly coupled. Continuous integration turned into a liability because deploying the codebase meant deploying everything at once. Adding new features, fixing bugs, and resolving technical debt all in a single repo became very difficult. It is the reason that Uber adopt a service-oriented architecture. It caused the Uber engineering team to restructure Uber's new Rider App.

Is Microservice Architecture the best choice for Machine Learning Deployment?

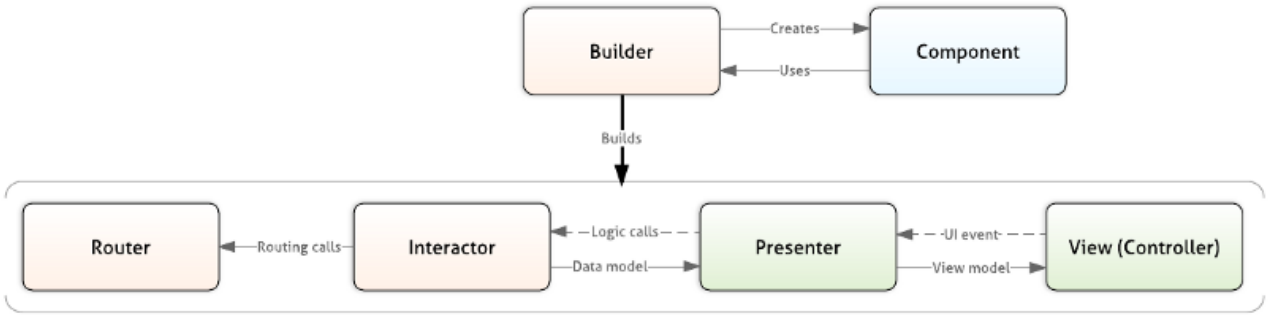
Understanding the difference between Monolithic Architecture and Microservice Architecture

towardsdatascience.com

Engineering the Architecture Behind Uber's New Rider App

In November 2016 Uber unveiled a sleek new rider app. The app implements a new mobile architecture across both iOS and…

eng.uber.com

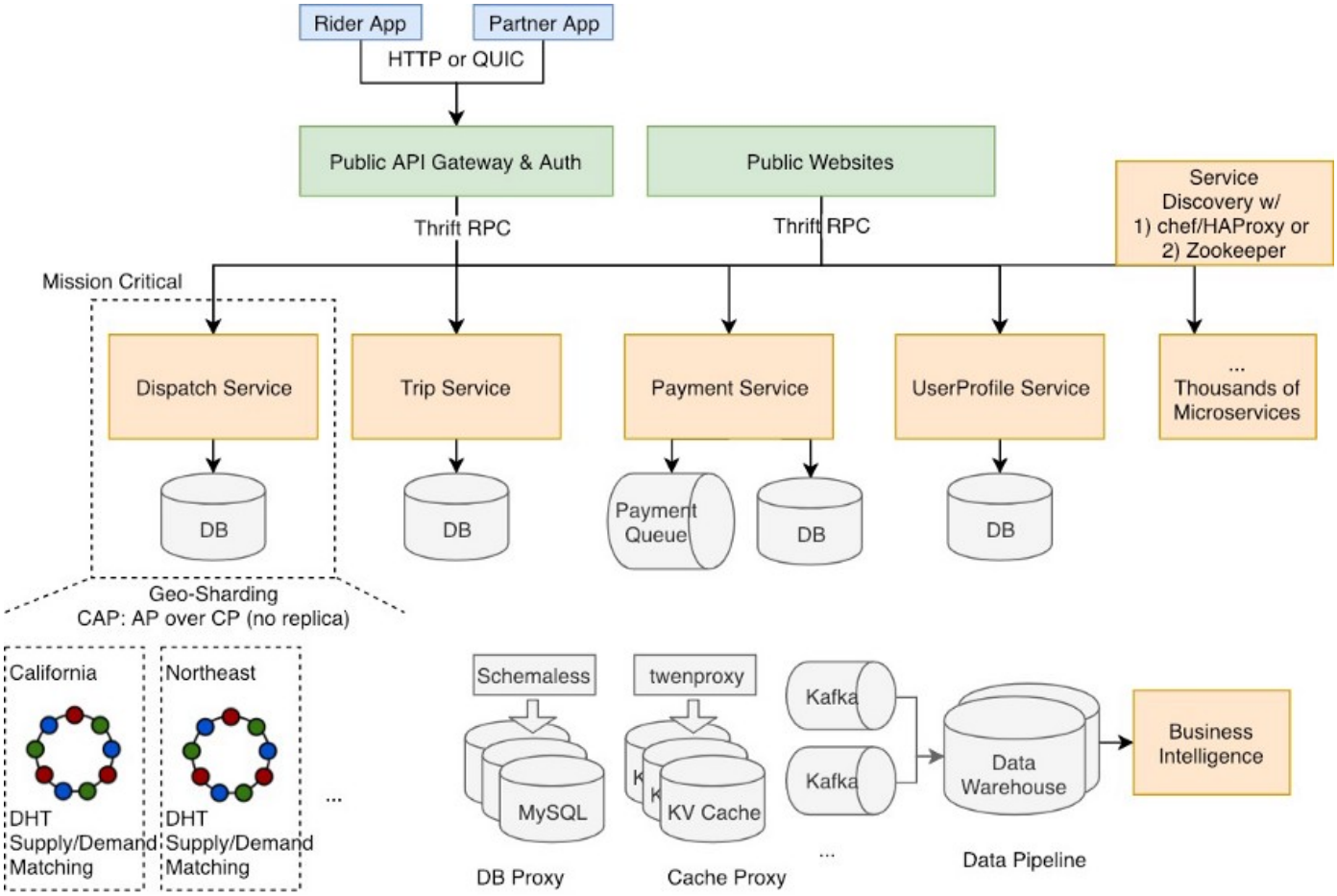


The new rider's app added UberPool, scheduled rides, and promotional vehicle views.

Objective

1. To achieve 99.99% reliability of the core travel experience on Uber (only have a total of one hour of downtime per year, and a maximum of one minute per week, in other words, every 10,000 operations can only fail 1 time at a time
2. Codebase divided by 2: Core code, and optional code. Core code is compulsory when the rider register, calls and completes or cancels travel requirements. Any modification to the core code must go through a rigorous review process. Optional code is less reviewed and can be dynamically closed at any time. This encourages mutual independence at the code level, allowing us to try new features and stop them at any time.
3. Core architecture: class names, inheritance relationships between business logic units (inheritance relationship between business logic units), main business logic, plugin point (name, dependency, structure, etc), reactive programming chains (relationship between reactive programming), unified platform components (unified platform-level modules)

Solutions



1. Apply IOS architecture (From MVC to VIPER, and create Riblets)

iOS Architecture Patterns

Demystifying MVC, MVP, MVVM and VIPER

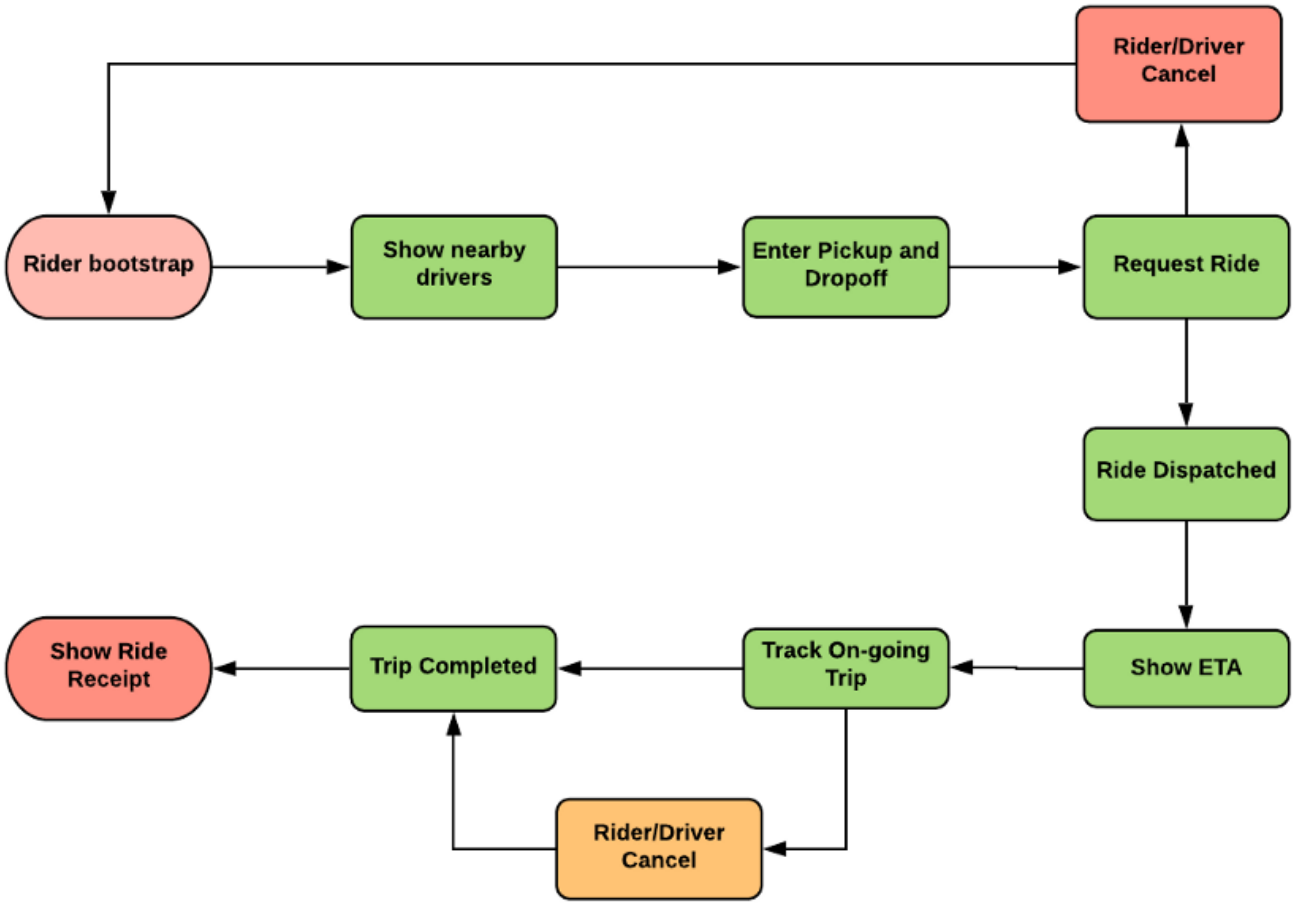
medium.com

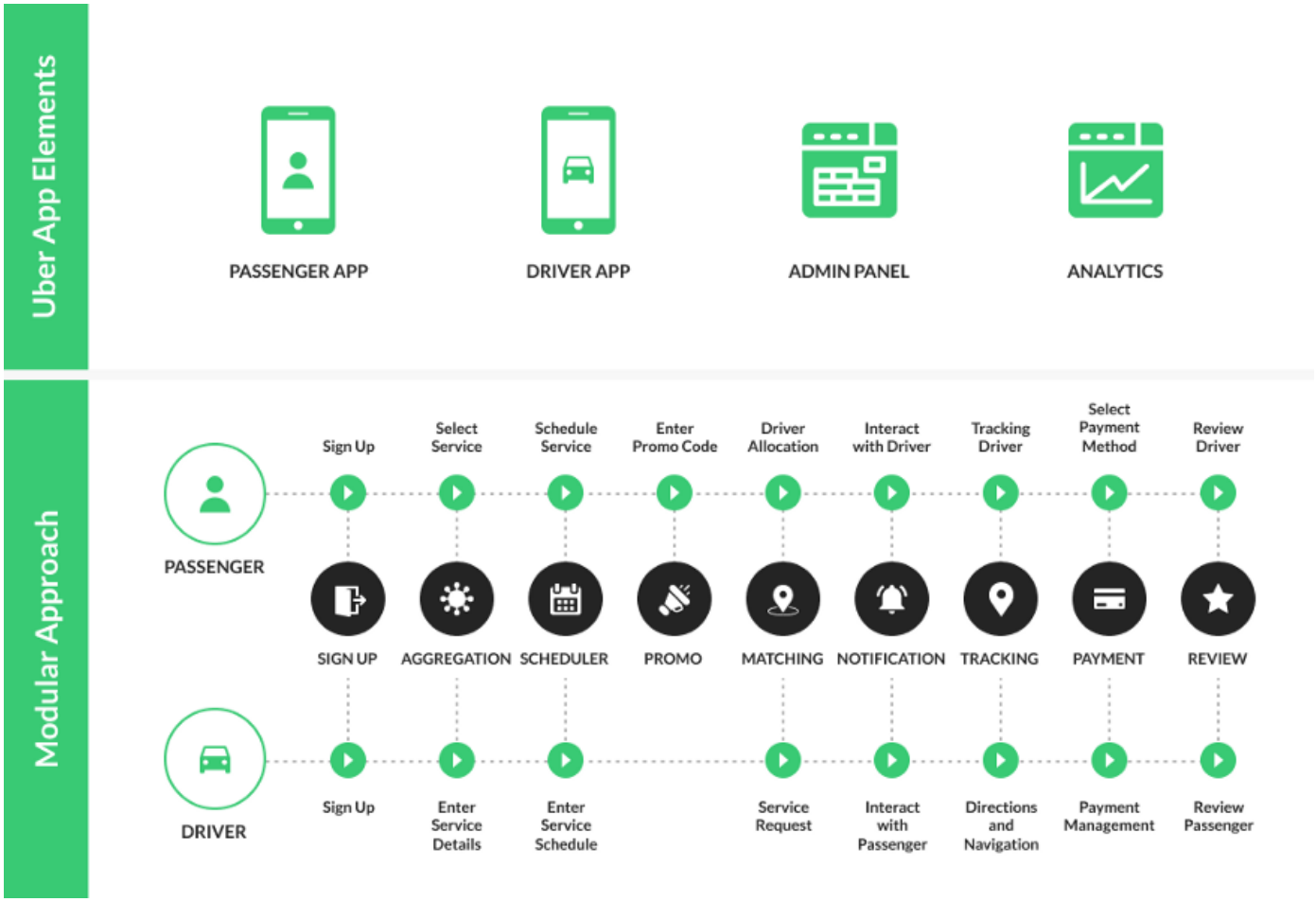
Ensuring Product Success With Transparency | Mutual Mobile

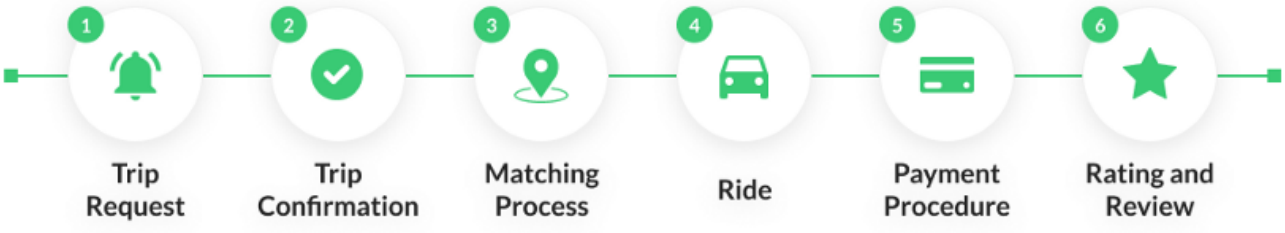
Our collaborative design and engineering model allows us to do more - attack complex design and development dilemmas…

mutualmobile.com

Requirements





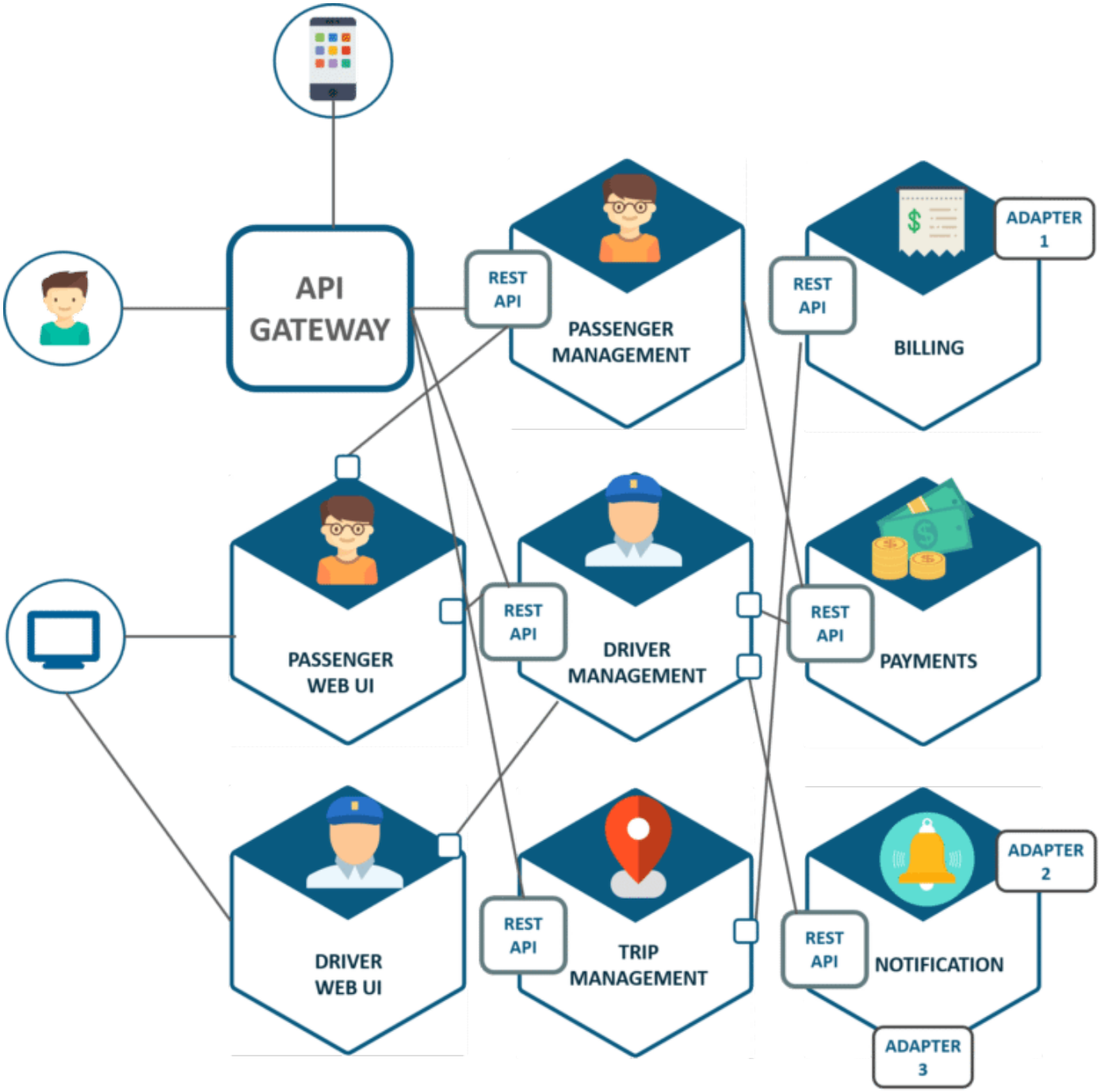


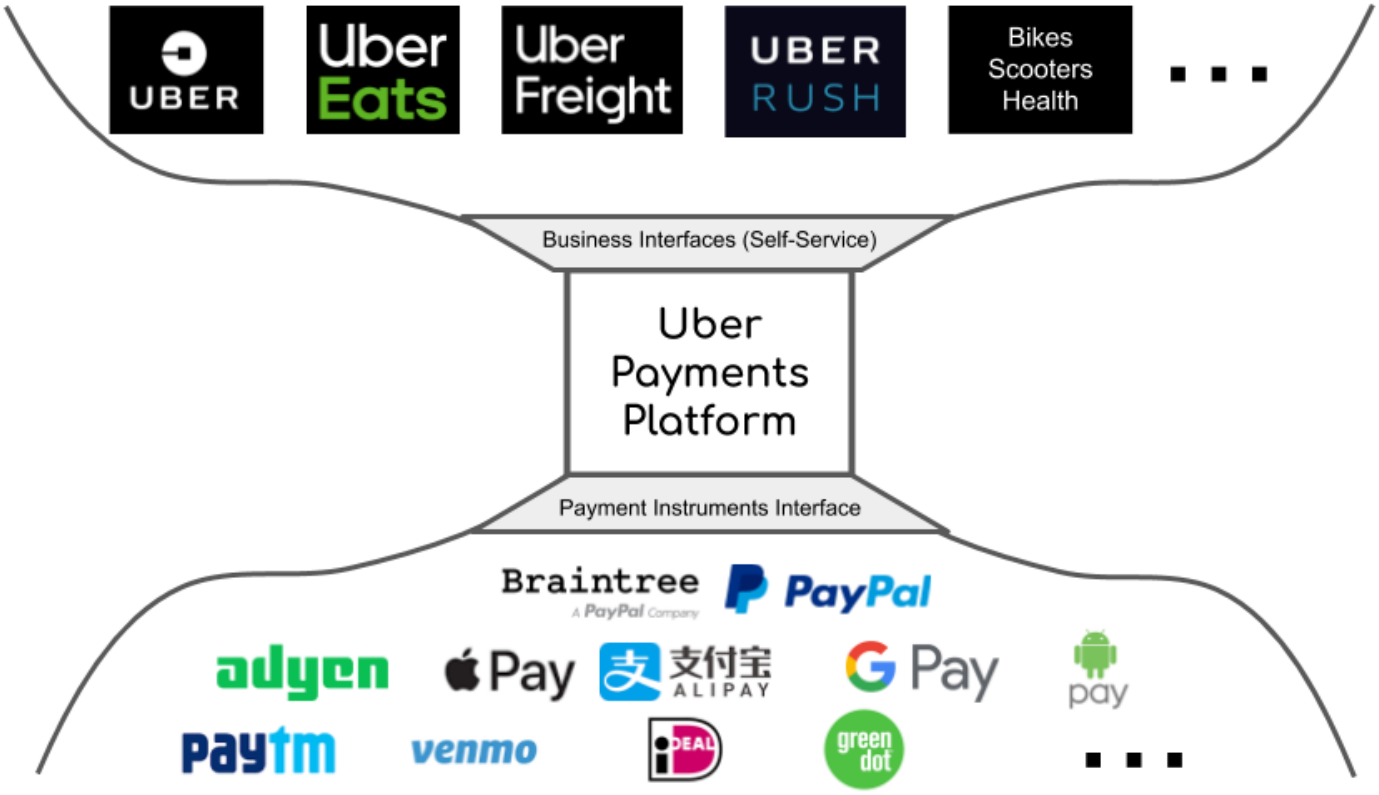
1. Riders can view nearby drivers
2. Riders can request a ride
3. Riders can view the driver's ETA and approximate price
4. Driver accepts the trip, the rider can view the driver's location and communicate until trip completion.
5. Book a cab

6. Match riders with drivers
7. See cabs in your vicinity
8. Location tracking
9. Post-tri actions: Collect ratings, send emails, update databases, schedule payments
10. Price and Surge: the price is increased when there are more demand and less supply with the help of prediction algorithms. According to Uber surge helps to meet supply demands. by increasing the price, more cabs will be on the road when the demand is more.

Non-Functional Requirements

1. Global
2. Low latency
3. High availability
4. High consistency
5. Scalable
6. Total Datacenter Failure: to handle an unexpected cascading failure or an upstream network provider could fail. Uber maintains a backup data center and the switches are in place to route everything over to the backup data center. The problem is the data for in-process trips may not be in the backup data center. Rather than replicate data they use driver phones as a source of trip data.


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
ADAPTER
1
BILLING
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
ADAPTER
2
NOTIFICATION
ADAPTER
3



DISCO — the foundation of this system

1. supply service (operates at the driver's end)
2. demand service (operates at the rider's end)

Dispatch optimization or DISCO is the area of Uber's system that works on location data to match demands to supply. In mapping drivers and riders, DISCO maintain minimum ETA and minimum driving. Instead of simply using latitude and longitude to locate riders and drivers, DISCO uses a more accurate Google S2 Library which divides the map of the location into tiny cells. Depending on the requirements, you can have 1km square cells across the map. Each of these cells is allocated a unique ID, so it is simpler to store cell data in the distributed system and access the ID and can use consistent hashing to store cell data.

The dispatch system is written in NodeJS since it is event-based and asynchronous and allows the exchange of messages from the application through web sockets whenever you want. Uber uses ring pop to scale its DISCO servers. Ring pop maintains consistent hashing

to distribute load efficiently across servers. It will automatically detect if nodes are added or removed from a cluster and will reallocate work accordingly through SWIM/Gossip Protocol that Ringpop uses. It uses RPC (Remote Procedure Call) protocol to manage calls between servers.

Demand Service

- rider requests for a taxi
- know the location where rider request
- the microservice receives the rider's request through web sockets
- Track the rider's GPS location
- also, receive the certain requirement set by the rider
- hands over the request, along with the additional requirements, to the dispatch system to connect it to the supply service.

Supply service

- service works at the driver's end
- tracks the cabs using latitude and longitude data (geolocation)
- All the active cabs continue sending their location every 5 seconds through a web application firewall to the load balancer
- The load balancer directs the GPS location of the cab to Kafka Rest API.
- The cab's location is updated into Kafka, a copy sent to the database and DISCO so that each service may use the latest cab location

DISCO — Dispatch optimization

1. Reduce extra driving

2. Reduce waiting time
3. lowest overall ETA

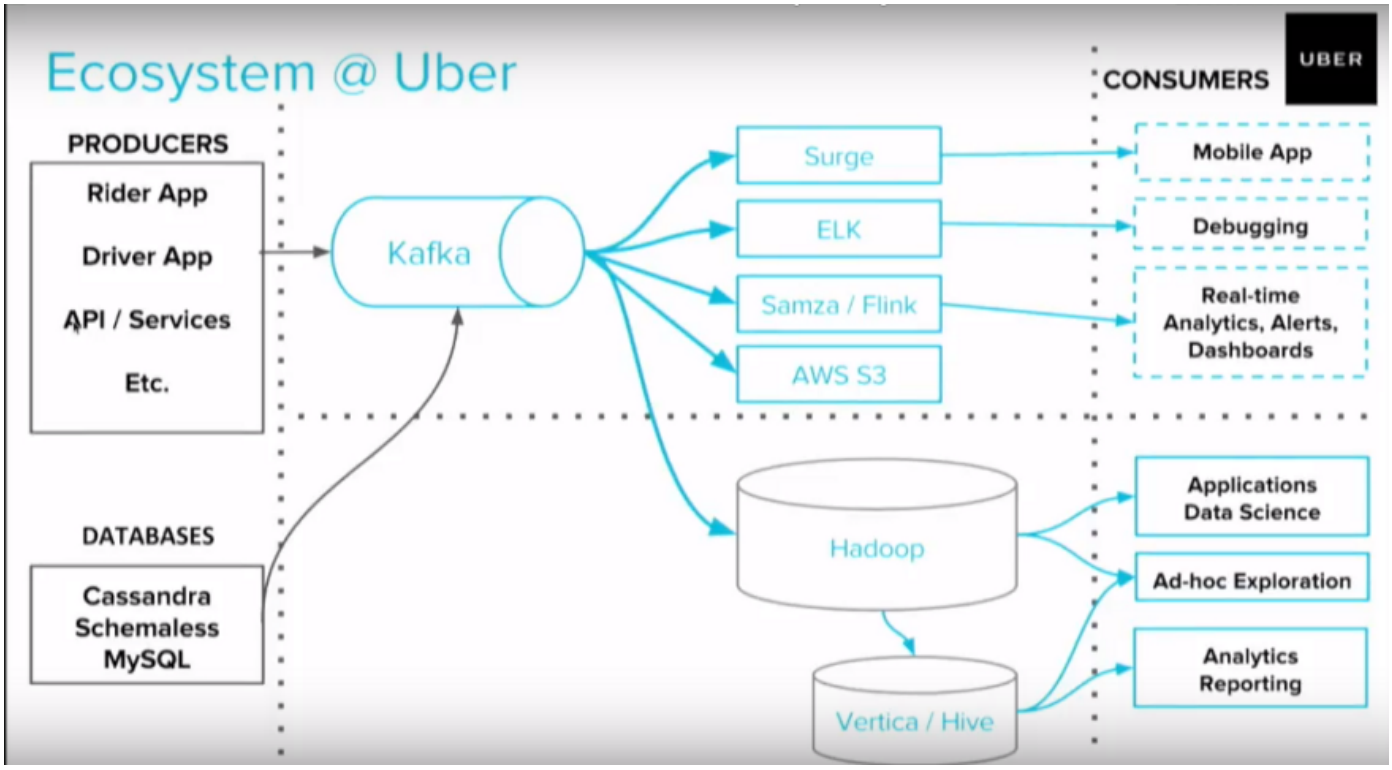
Trip data

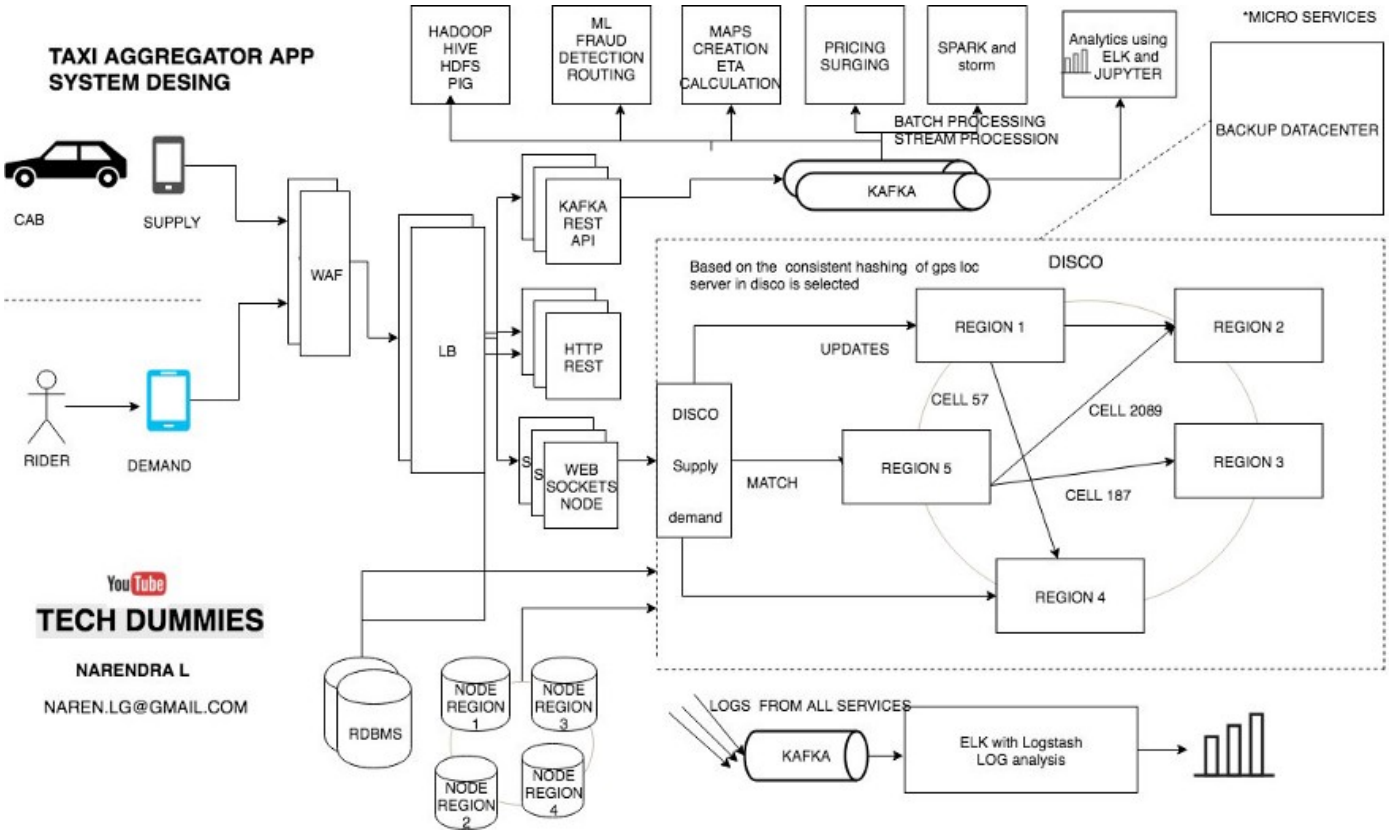
- Cab's location data
- Billing data once the ride trip is marked complete. The timestamp is provided to know the start and the end. So that, Uber can calculate the fare and bill the rider

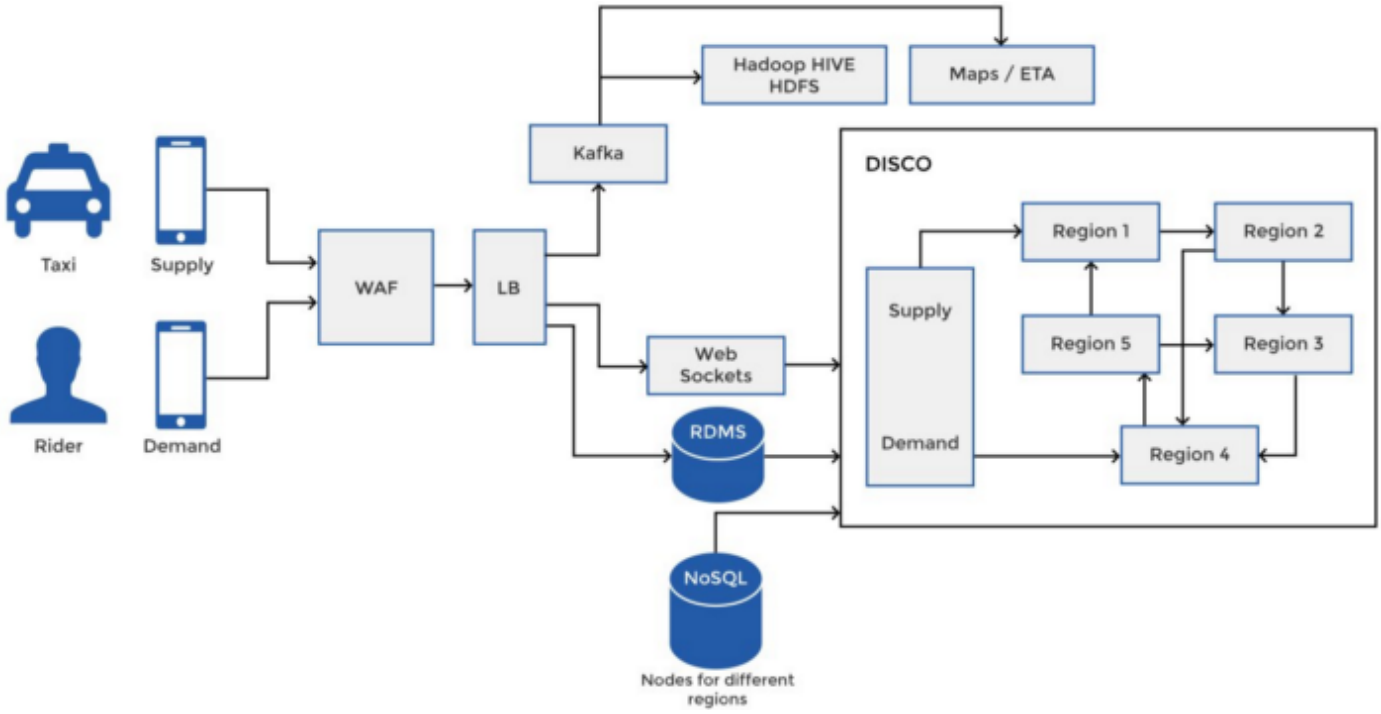
Database Architecture

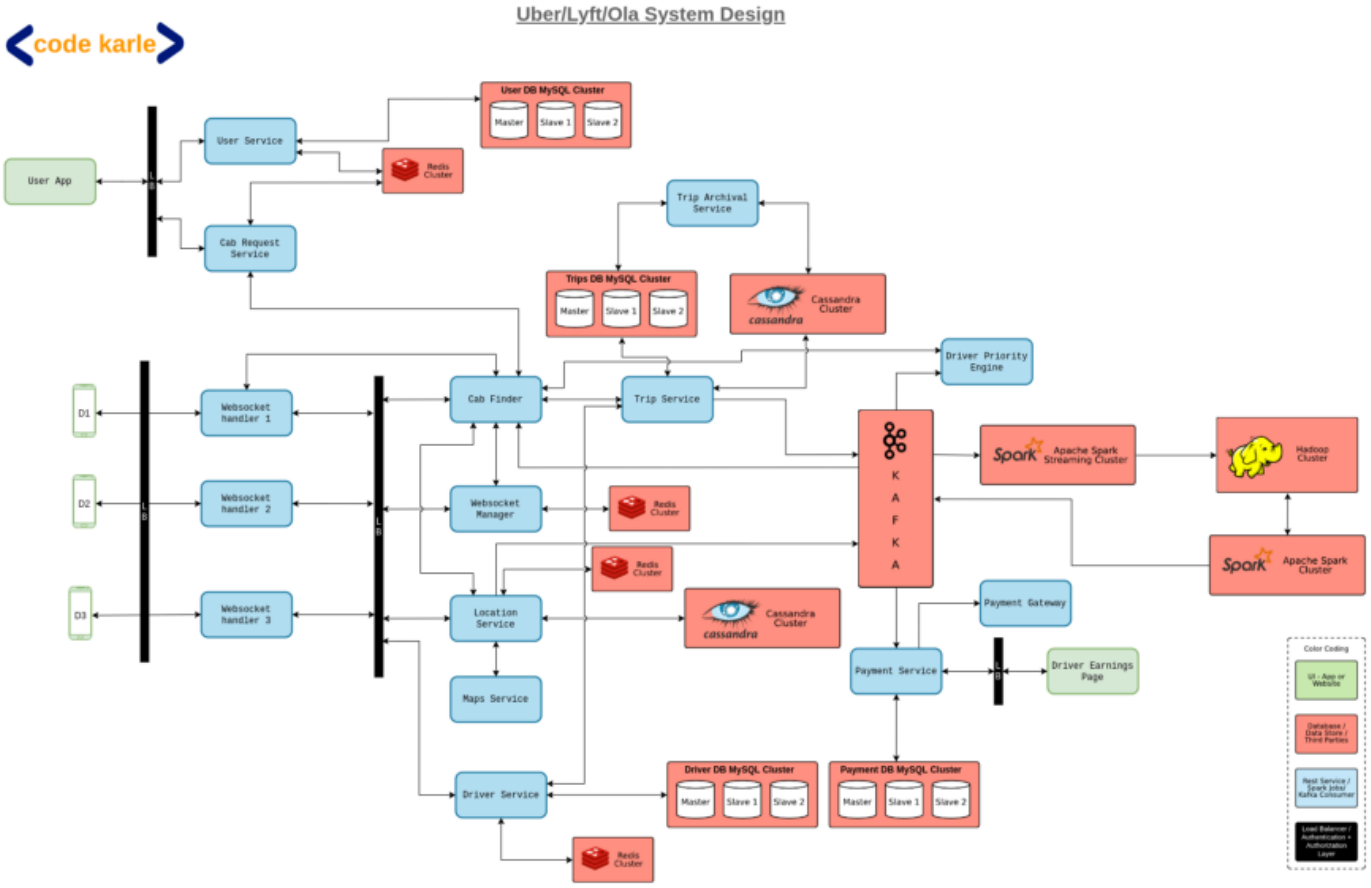
- The application can both read and write-heavy
- Since we are accepting cab's location updates every 5 seconds, there will be a lot of writes. Cab requests are heavy, which mean there are also many reads
- Started off from RDBMS PostgreSQL to Schema-less, NoSQL database built over MySQL

System Architecture

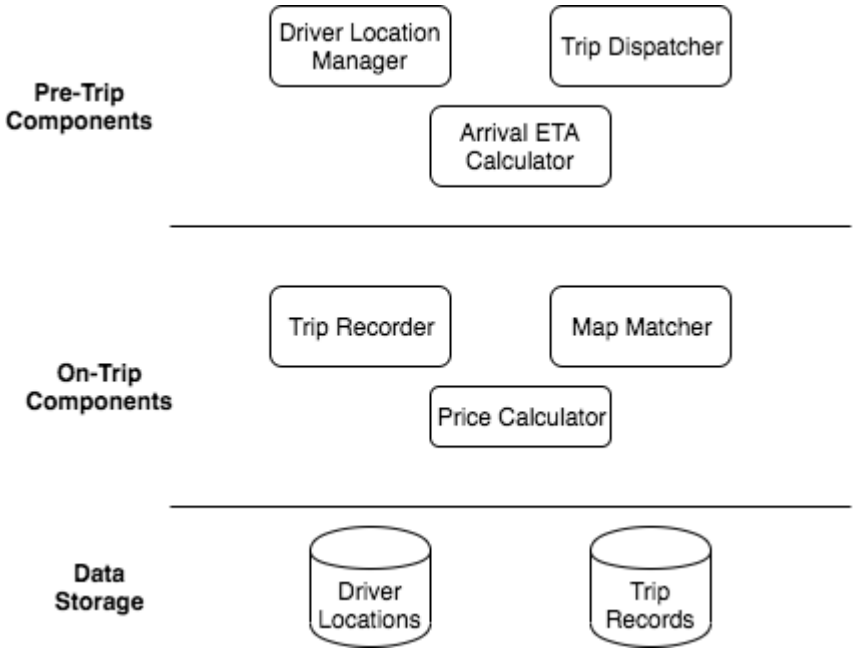

Ecosystem @ Uber
UBER
PRODUCERS
Rider App
Driver App
API / Services
Etc.
DATABASES
Cassandra
Schemaless
MySQL
Kafka
Surge
ELK
Samza / Flink
AWS S3
Hadoop
Vertica / Hive
CONSUMERS
Mobile App
Debugging
Real-time Analytics, Alerts, Dashboards
Applications Data Science
Ad-hoc Exploration
Analytics Reporting


TAXI AGGREGATOR APP
SYSTEM DESING
HADOOP HIVE HDFS PIG
ML FRAUD DETECTION ROUTING
MAPS CREATION ETA CALCULATION
PRICING SURGING
SPARK and storm
Analytics using ELK and JUPYTER
*MICRO SERVICES
BACKUP DATACENTER
BATCH PROCESSING
STREAM PROCESSION
KAFKA
CAB
SUPPLY
RIDER
DEMAND
WAF
LB
KAFKA REST API
HTTP REST
WEB SOCKETS NODE
Based on the consistent hashing of gps loc server in disco is selected
DISCO
UPDATES
REGION 1
REGION 2
REGION 3
REGION 4
REGION 5
CELL 57
CELL 2089
CELL 187
DISCO
Supply
demand
MATCH
YouTube
TECH DUMMIES
NARENDRA L
NAREN.LG@GMAIL.COM
RDBMS
NODE REGION 1
NODE REGION 2
NODE REGION 3
NODE REGION 4
LOGS FROM ALL SERVICES
KAFKA
ELK with Logstash LOG analysis


Hadoop HIVE HDFS
Maps / ETA
Kafka
DISCO
Region 1
Region 2
Region 5
Region 3
Region 4
Supply
Demand
Taxi
Supply
Rider
Demand
WAF
LB
Web Sockets
RDMS
NoSQL
Nodes for different regions


Uber/Lyft/Ola System Design
code karle
User DB MySQL Cluster
Master
Slave 1
Slave 2
User App
User Service
Redis Cluster
Cab Request Service
Trip Archival Service
Trips DB MySQL Cluster
Cassandra Cluster
Driver Priority Engine
D1
D2
D3
Websocket handler 1
Websocket handler 2
Websocket handler 3
Cab Finder
Trip Service
Websocket Manager
Location Service
Maps Service
Driver Service
K A F K A
Spark Apache Spark Streaming Cluster
Hadoop Cluster
Spark Apache Spark Cluster
Payment Gateway
Payment Service
Driver Earnings Page
Driver DB MySQL Cluster
Payment DB MySQL Cluster

System Components



Map — Cab location is sent to the rider

- Rider makes a ride request, the application displays the location of all the nearby drivers on a map Uber application will push the current location of all cabs to the riders. When the customer opens the map, a query is sent to the server to find all the nearest drivers.
- The real-time location data provided by Kafka will be used to calculate the ETA to allow riders to know how long the driver can reach them. It also notifies them of the estimated time for their destination.
- Dijkstra's algorithm can be used to find the shortest path on the map with road networks. Since the shortest path(distance-wise) isn't always the quickest path(heavy traffic may impact the arrival time), more complex AI algorithms may also be used to generate the best time estimates.

Web application Firewall

- A firewall is applied because of security reasons. It allows Uber to block requests from suspicious sources or from regions where the service is not supported.

Load Balancer

- Uber uses 3 layers of load balancers, layer 3, layer 4, and layer 7. Layer 3 works for IP-based load balancing. Layer 4 works for DNS-based load balancing while layer 7 handles application load balancing.

Kafka

- Kafka provides a logging layer for Uber. It can instantly log updates to the driver location so it can be used by the different microservices of the system. Kafka ingests all these updates in real-time with the guarantee that no information will be lost. In reality, there's a cluster of Kafka servers for this purpose.

Web Sockets

- For simpler communication between the client (both rider and driver) and the server in such types of applications. Web sockets keep a connection open between the driver's application or user's application and the server.

Hadoop

- Analytics data which Uber uses to improve their service. Kafka will periodically store and archive data in Hadoop. This information can be helpful when analyzing different trends in the usage of the app. For example, what time and where have more Uber traffic or more requests.

Payment MySQL DB

- Payment service will sit on top of Kafka once the trip completion event. As soon as the trip is completed, based on distance, time, etc it will come up with the amount of the money that needs to be paid and it inserts all this information in a Payment MySQL DB. The payment service will also be connected with a payment gateway in case the payment method requires so. It will also expose APIs to give all the previous payment-related information for the customer's or driver's account.

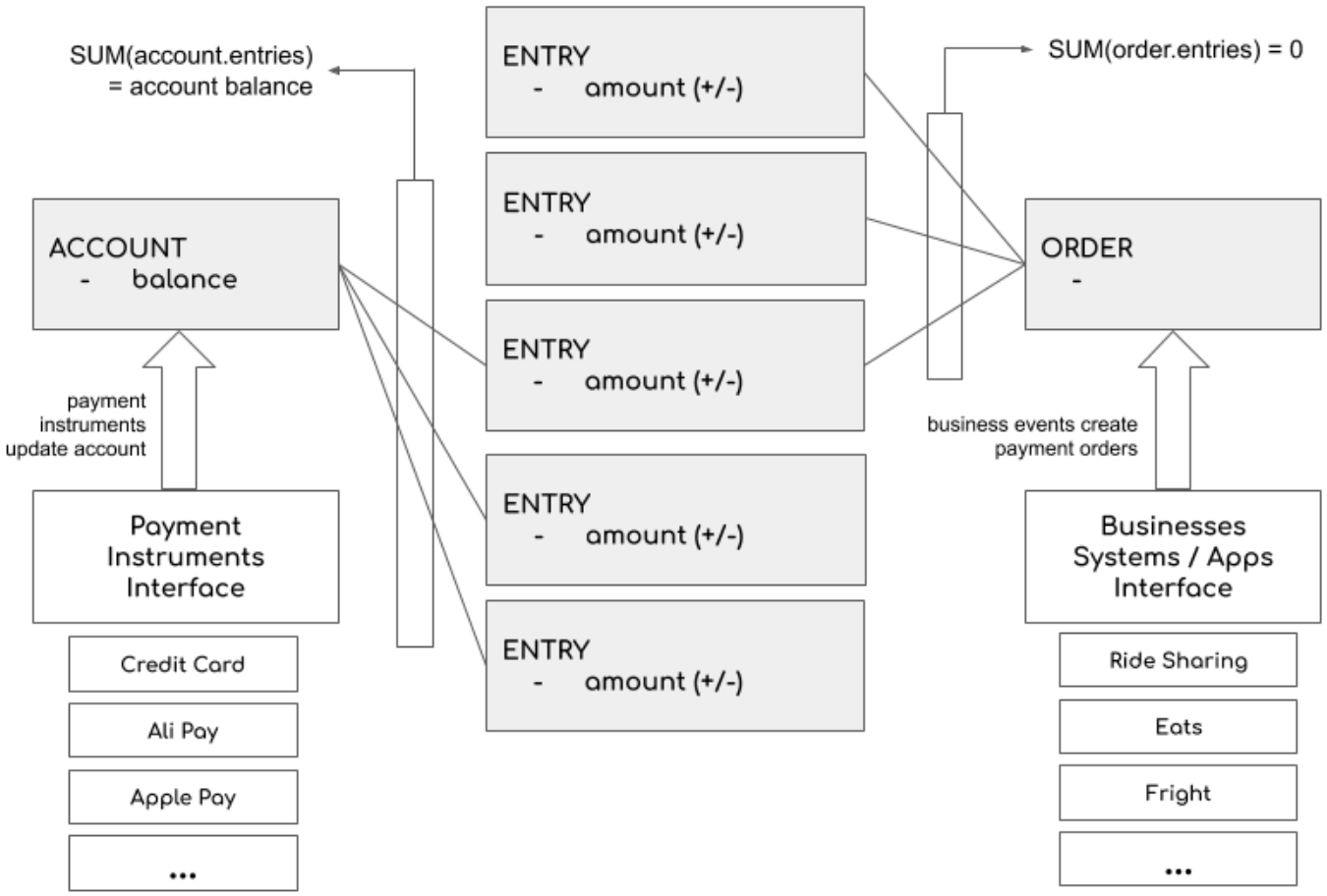
- Onboarding of payment options, facilitating users in adding new payment profiles,

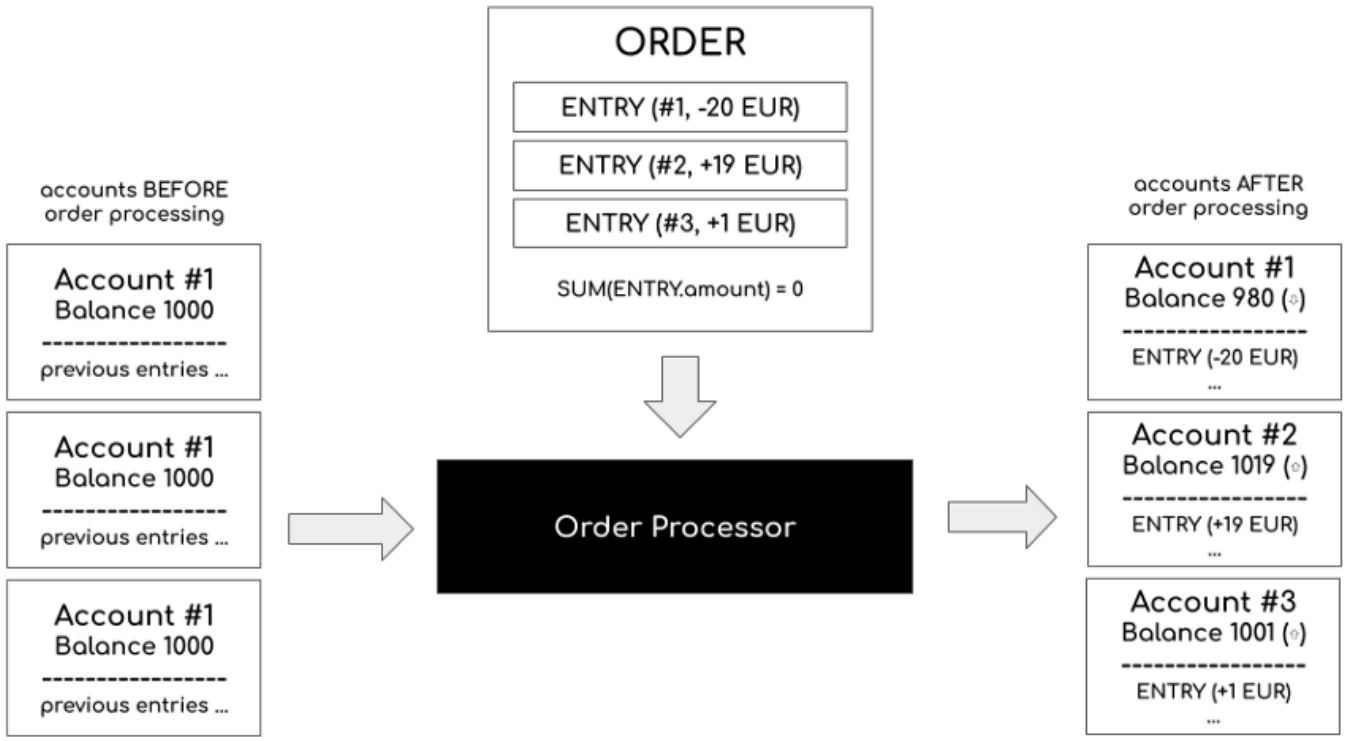
- authorization of payment for transactions, to guarantee that a specific amount of money is available for charging,
- voiding of payments, refunding
- charging and capturing the customer account, moving money from user to Uber
- Deleting payment options and profiles
- Tipping,
- scheduled service
- Promotions,
- scheduled dunning of unsettled payments,
- Switching payment methods during a service
- Default payment method fallback/selection
- Duplicate payments,
- incorrect currency conversion,
- incorrect payment
- lack of payment
- Dangling authorization

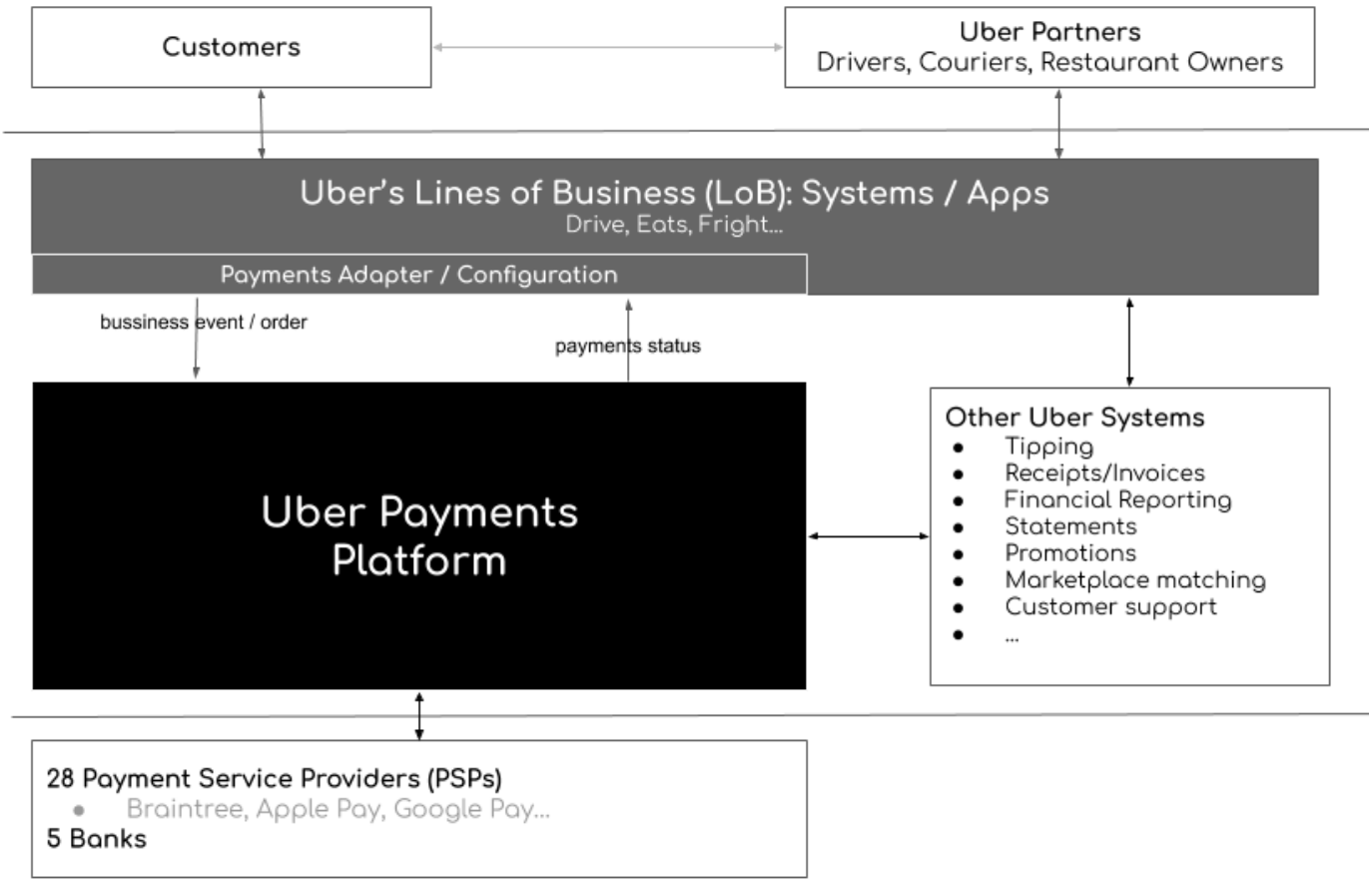
Uber's Payments Platform

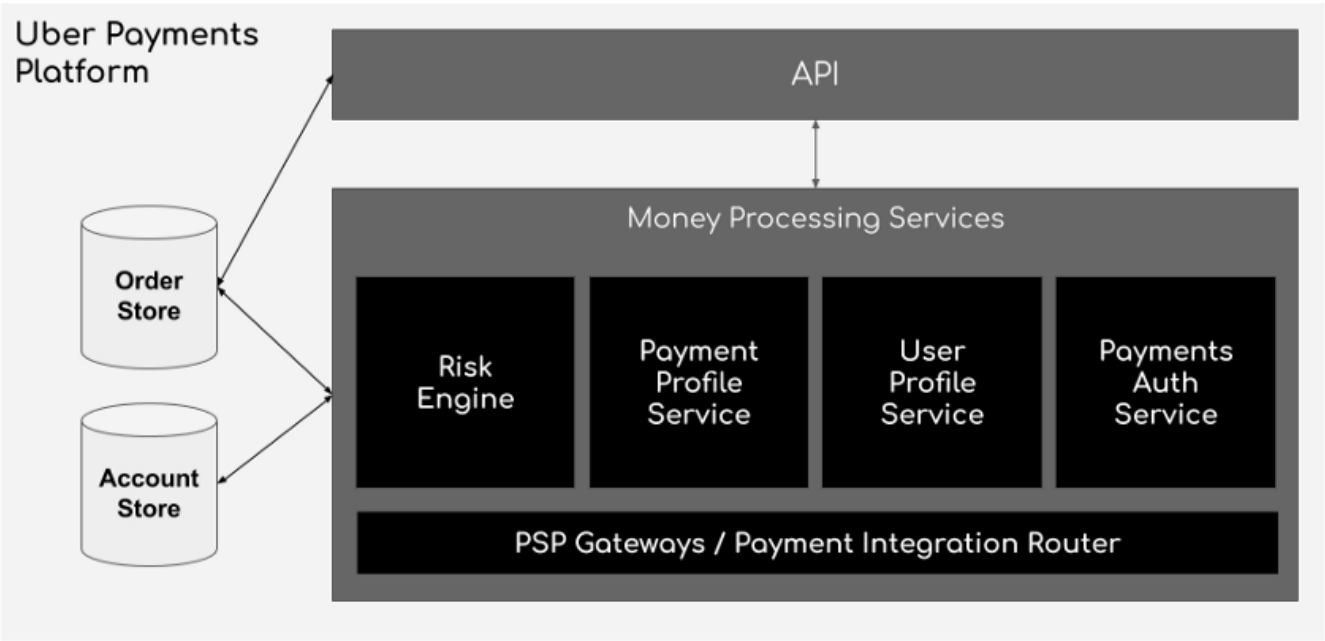
Recently, Uber has announced deeper push into financial services with Uber Money, a new division, including products…

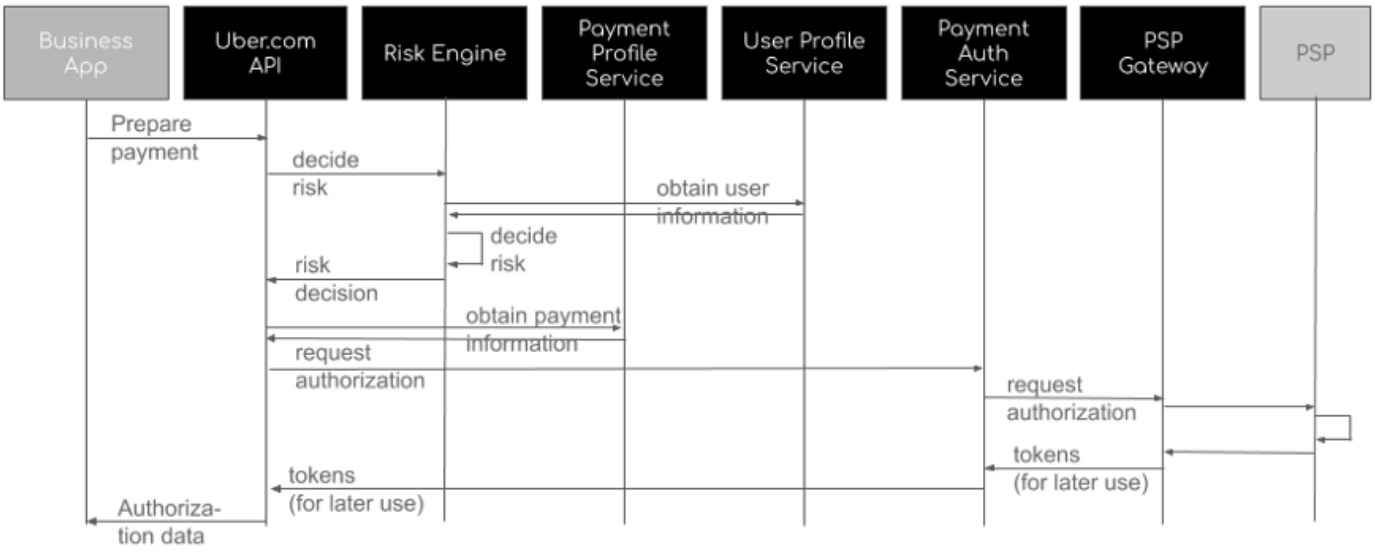
underhood.blog

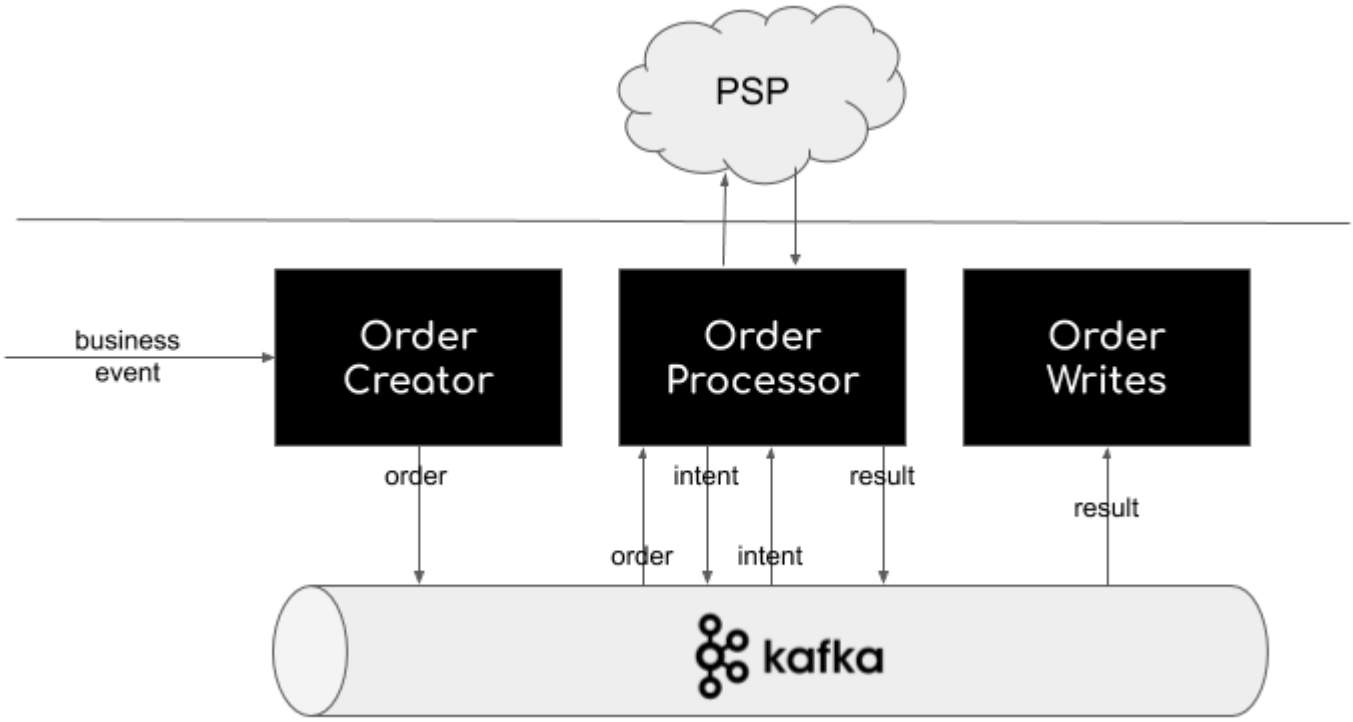

SUM(account.entries)
= account balance
ENTRY
- amount (+/-)
ENTRY
- amount (+/-)
ENTRY
- amount (+/-)
ENTRY
- amount (+/-)
ENTRY
- amount (+/-)
SUM(order.entries) = 0
ACCOUNT
- balance
ORDER
-
payment
instruments
update account
business events create
payment orders
Payment
Instruments
Interface
Credit Card
Ali Pay
Apple Pay
...
Businesses
Systems / Apps
Interface
Ride Sharing
Eats
Fright
...

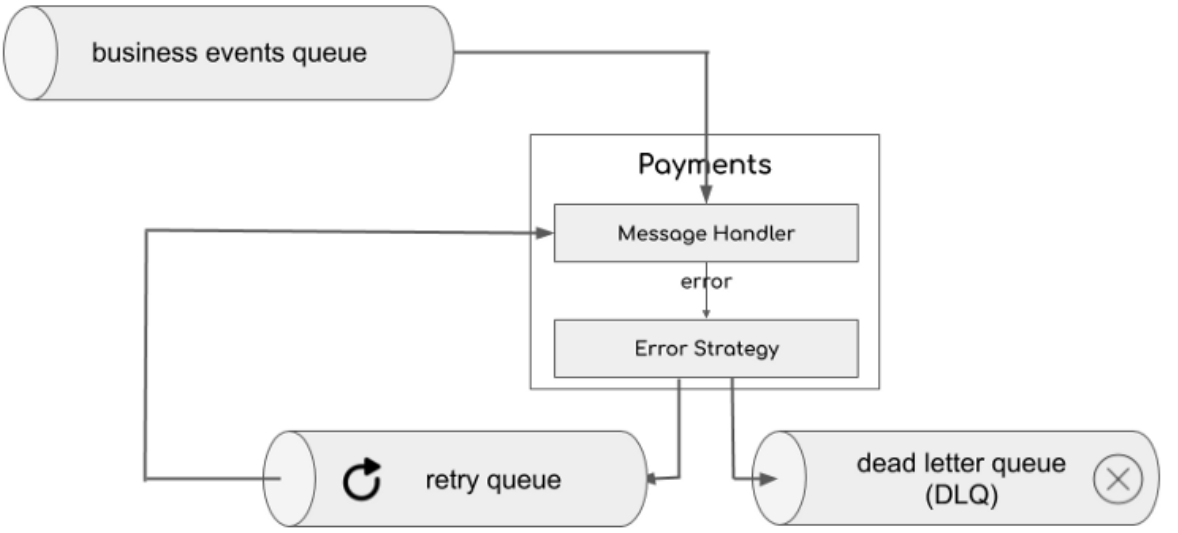

ORDER
ENTRY (#1, -20 EUR)
ENTRY (#2, +19 EUR)
ENTRY (#3, +1 EUR)
SUM(ENTRY.amount) = 0
accounts BEFORE
order processing
Account #1
Balance 1000
previous entries ...
Account #1
Balance 1000
previous entries ...
Account #1
Balance 1000
previous entries ...
Order Processor
accounts AFTER
order processing
Account #1
Balance 980 (◊)
ENTRY (-20 EUR)
...
Account #2
Balance 1019 (◊)
ENTRY (+19 EUR)
...
Account #3
Balance 1001 (◊)
ENTRY (+1 EUR)
...

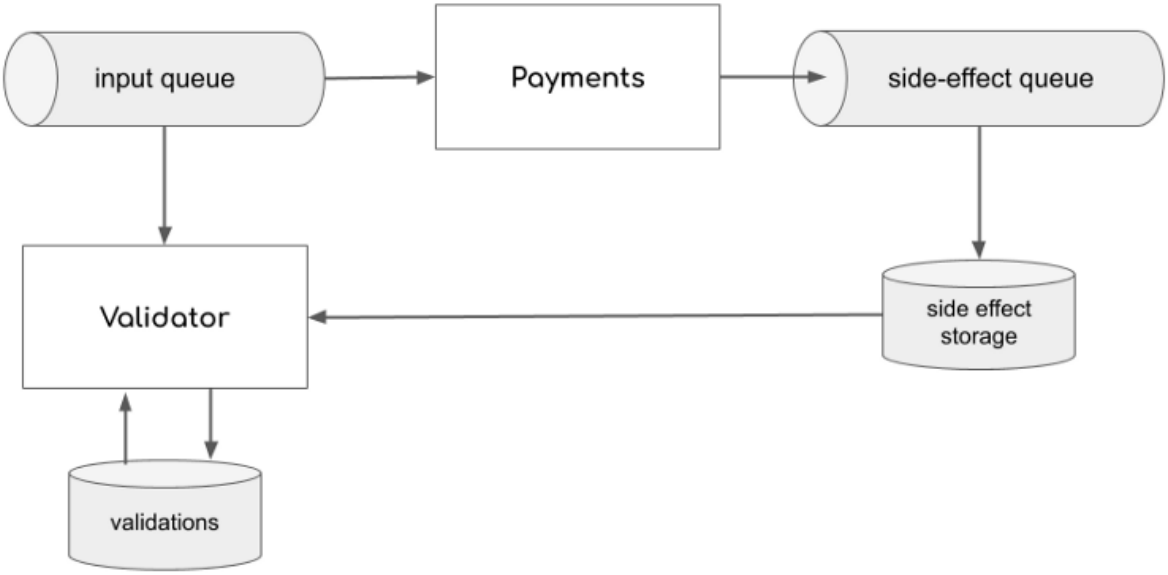

Customers
Uber Partners
Drivers, Couriers, Restaurant Owners
Uber's Lines of Business (LoB): Systems / Apps
Drive, Eats, Fright...
Payments Adapter / Configuration
bussiness event / order
payments status
Uber Payments Platform
Other Uber Systems
Tipping
Receipts/Invoices
Financial Reporting
Statements
Promotions
Marketplace matching
Customer support
...
28 Payment Service Providers (PSPs)
Braintree, Apple Pay, Google Pay...
5 Banks

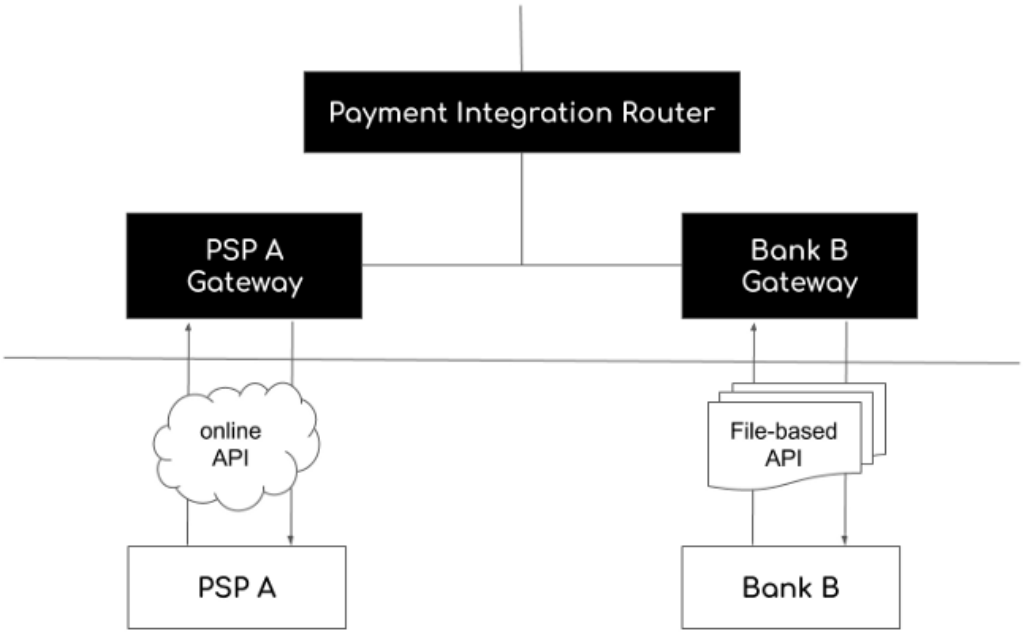

Uber Payments Platform
API
Money Processing Services
Order Store
Account Store
Risk Engine
Payment Profile Service
User Profile Service
Payments Auth Service
PSP Gateways / Payment Integration Router

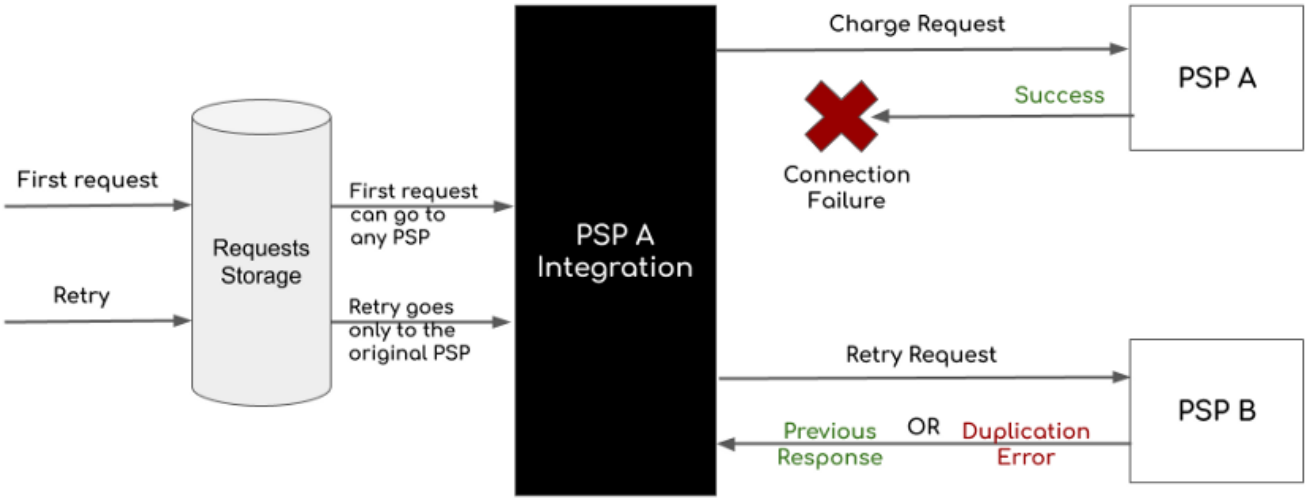

Business App
Uber.com API
Risk Engine
Payment Profile Service
User Profile Service
Payment Auth Service
PSP Gateway
PSP
Prepare payment
decide risk
obtain user information
decide risk
risk decision
obtain payment information
request authorization
request authorization
tokens (for later use)
tokens (for later use)
Authoriza-tion data


PSP
business event
Order Creator
Order Processor
Order Writes
order
intent
result
order
intent
result
kafka


business events queue
Payments
Message Handler
error
Error Strategy
retry queue
dead letter queue
(DLQ)


input queue
Payments
side-effect queue
Validator
side effect
storage
validations





Spark Streaming Cluster

- keep track of basic events like drivers not found, identifying areas with scarcity of drivers, and other basic analytics. This spark streaming cluster will also dump all the events in the Hadoop cluster for further analytics like user profiling, driver profiling, etc to classify customers or drivers as premium or regular.

Driver Profile engine

- Will classify the drivers based on ratings, ETA accuracy, etc to score the drivers will be powered by the same data.

Fraud Engine

- If the same driver is always taking trips from the same user, there is a high chance that they are friends or the same person trying to exploit any incentive programs offered by our application. We could also use this same data to fetch the traffic or road condition information that can further power the map service.

Kibana/Graphana- Elastic search

- do some log analysis
- Track HTTP APIs, manage profile, collect feedback and ratings, promotion and coupons, fraud detection, payment fraud, incentive abuse, compromised accounts

Finally,

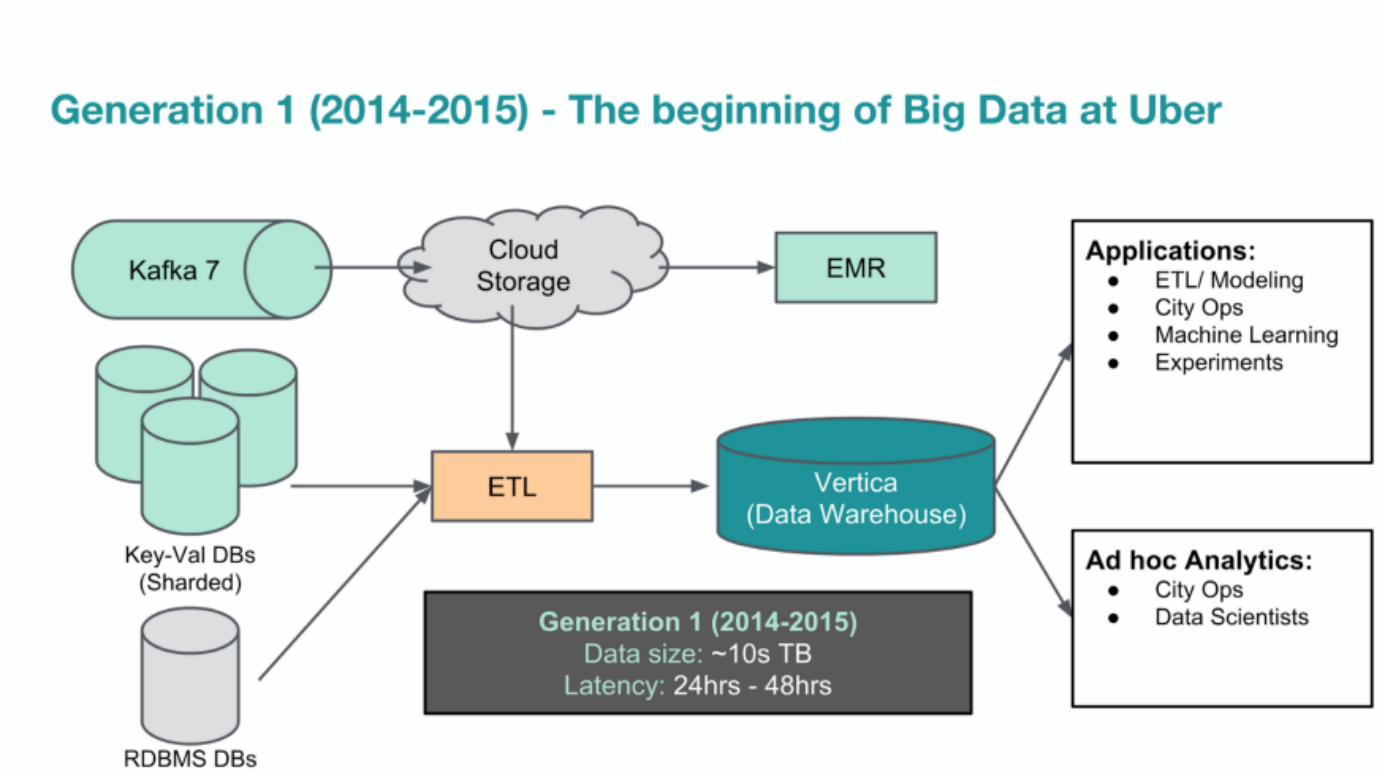
Big Data

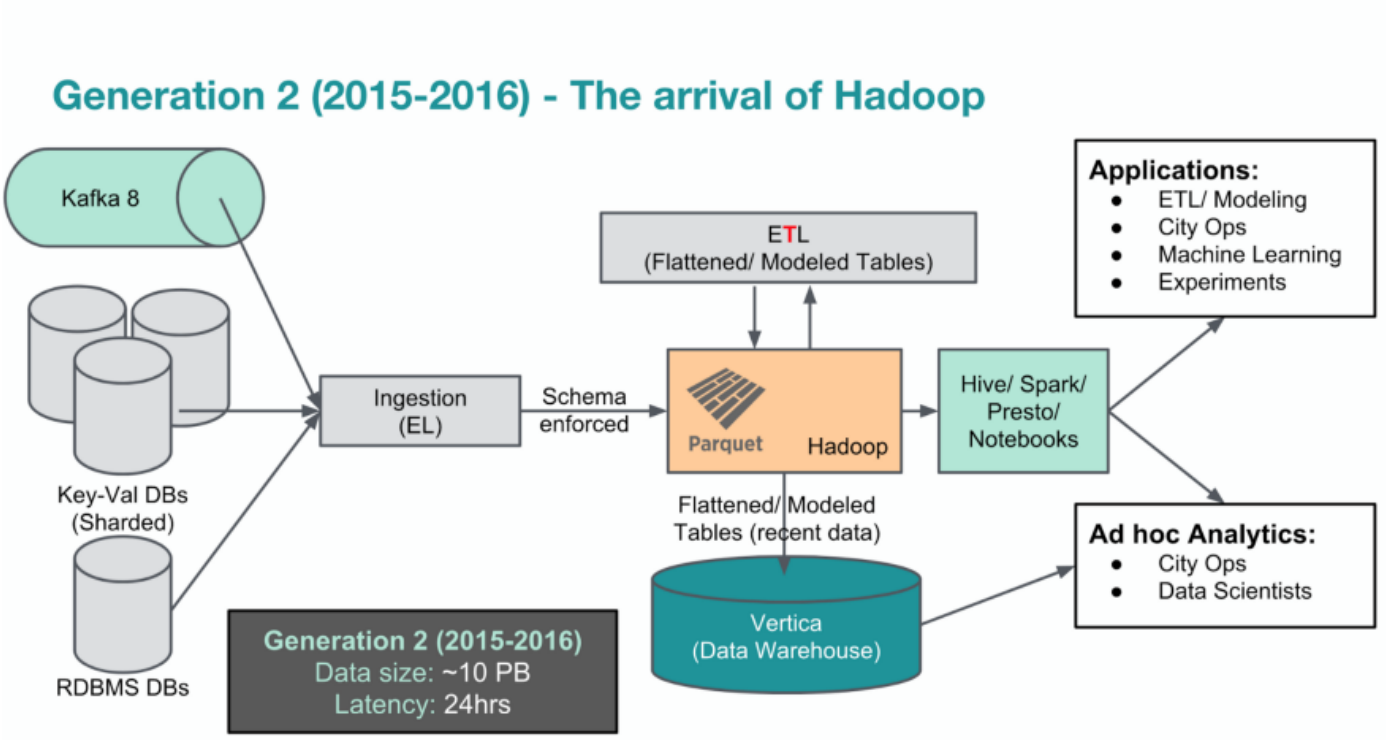
Uber's Big Data Platform: 100+ Petabytes with Minute Latency

Uber is committed to delivering safer and more reliable transportation across our global markets. To accomplish this…

eng.uber.com

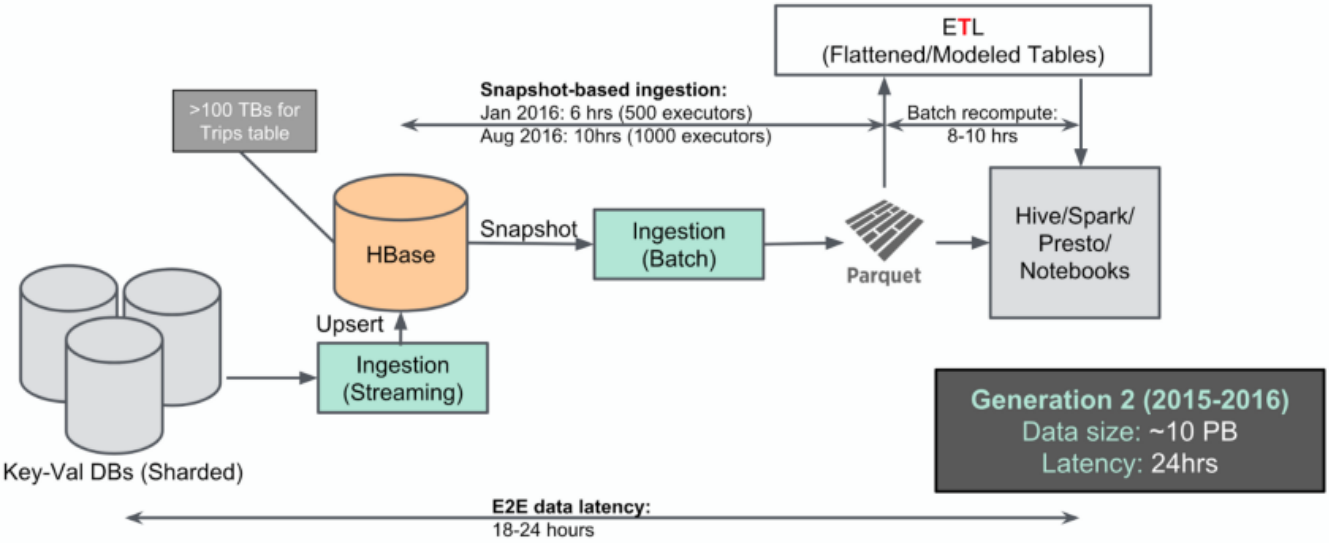
From the article, big data is a must to evolve to a better solution.


Generation 1 (2014-2015) - The beginning of Big Data at Uber
Kafka 7
Cloud Storage
EMR
Applications:
ETL/ Modeling
City Ops
Machine Learning
Experiments
Key-Val DBs (Sharded)
ETL
Vertica (Data Warehouse)
Ad hoc Analytics:
City Ops
Data Scientists
Generation 1 (2014-2015)
Data size: ~10s TB
Latency: 24hrs - 48hrs
RDBMS DBs


Generation 2 (2015-2016) - The arrival of Hadoop
Kafka 8
ETL
(Flattened/ Modeled Tables)
Applications:
ETL/ Modeling
City Ops
Machine Learning
Experiments
Key-Val DBs (Sharded)
Ingestion (EL)
Schema enforced
Parquet
Hadoop
Hive/ Spark/ Presto/ Notebooks
Flattened/ Modeled Tables (recent data)
Ad hoc Analytics:
City Ops
Data Scientists
Vertica (Data Warehouse)
RDBMS DBs
Generation 2 (2015-2016)
Data size: ~10 PB
Latency: 24hrs

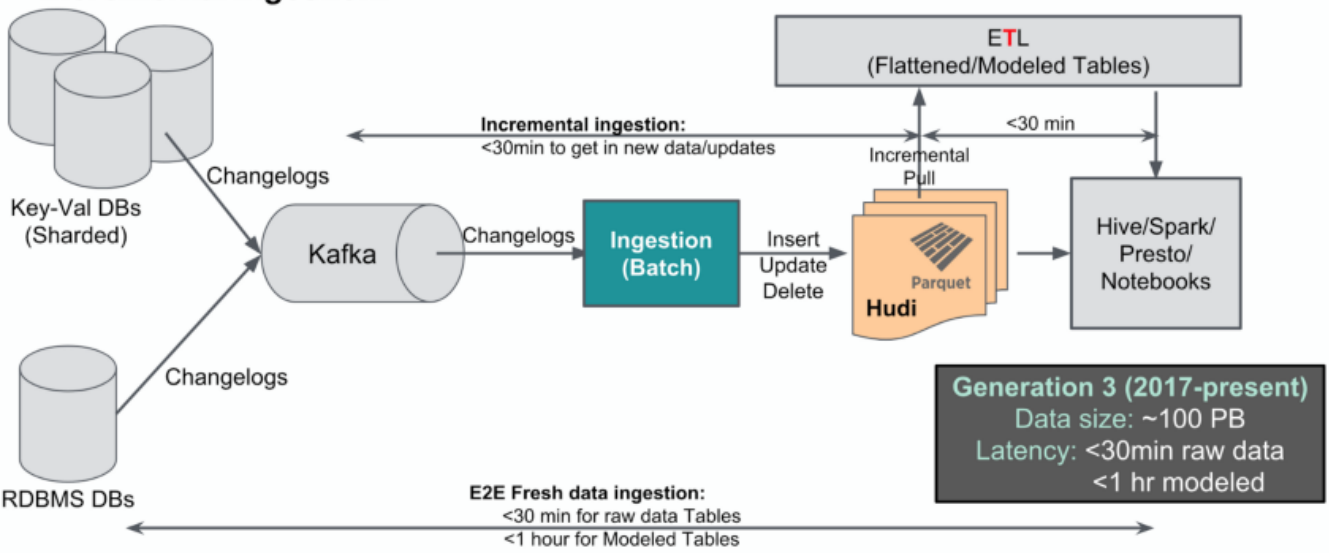
Generation 2 (2015-2016) - The arrival of Hadoop

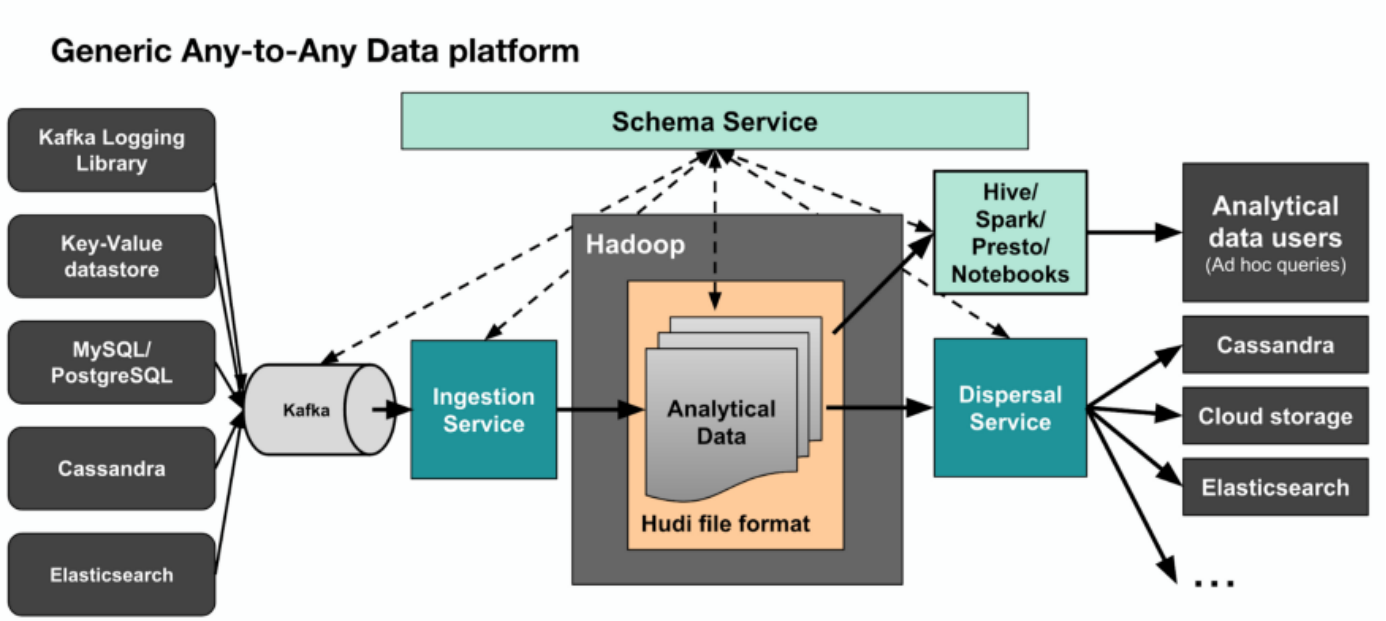
Why does data latency remain at 24 hours?



Generation 3 (2017-present) - Let's rebuild for long term

Incremental ingestion:





Further resources

1. https://techtakshila.com/system-design-interview/chapter-3/#high-level-design
2. https://dzone.com/
3. https://tianpan.co/hacking-the-software-engineer-interview#designing-uber-or-a-realtime-marketplace
4. https://weixia.info/ride-hailing-system-design.html
5. https://www.codekarle.com/system-design/Uber-system-design.html
6. https://underhood.blog/uber-payments-platform
7. https://github.com/raghav-chamarthy/Uber-replica
8. https://github.com/apache/hudi
9. https://eng.uber.com/engineering-routing-engine/
10. https://www.garyfox.co/uber-business-model/
11. https://eng.uber.com/michelangelo-machine-learning-platform/