UBER | DATA

“Big Data Processing the UBER Way”

Praveen Murugesan

Uber’s Mission

“Transportation as reliable as running water, everywhere, for everyone”

75+ Countries

500+ Cities

And growing...

Agenda

- UBER's Data Audience
- Data Infra - A Brief History
- What we Solved
- What we are Currently Solving

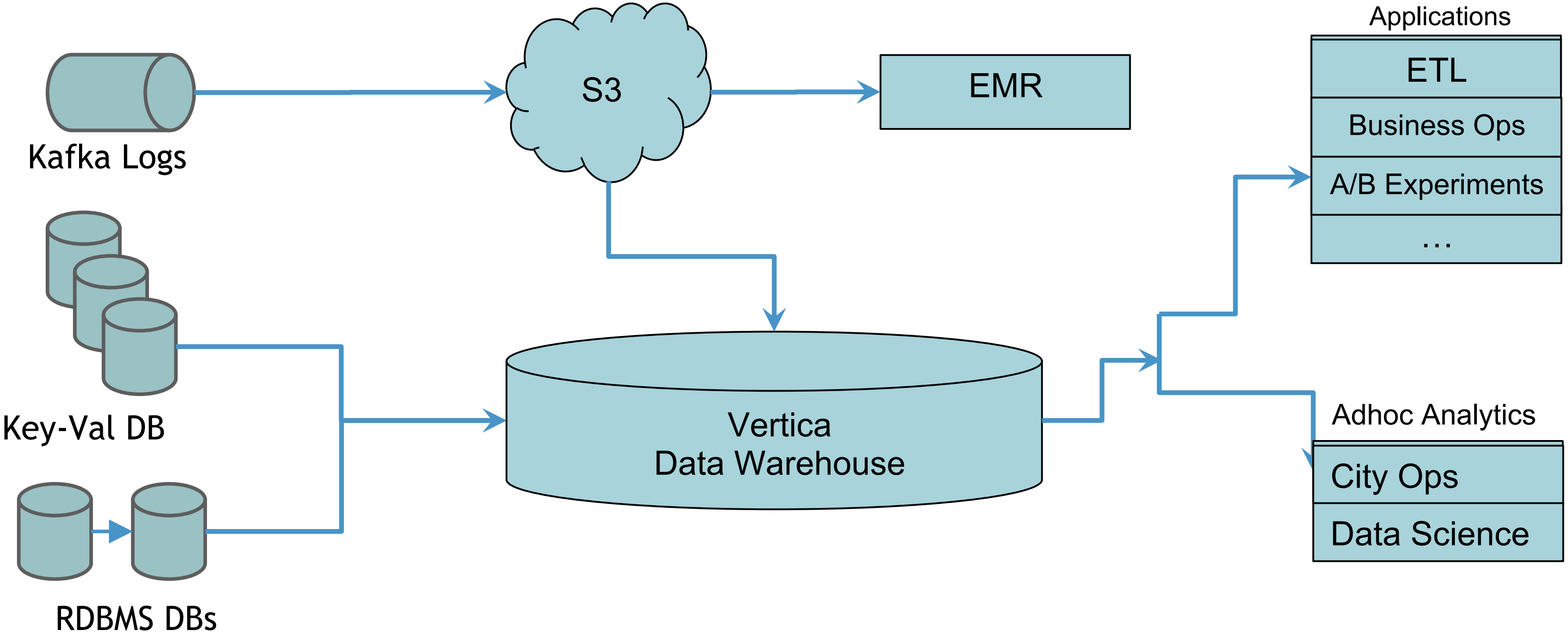
Uber’s Data Audience

- 1000s of City Operators (Uber Ops!)
  - On the ground team who run and scale uber’s transportation network
- 100s of Data Scientists and Analysts
  - Spread across various functional groups including Engineering, Marketing, BizDev etc
- 10s of Engineering Teams
  - Focussed on building automated Data Applications

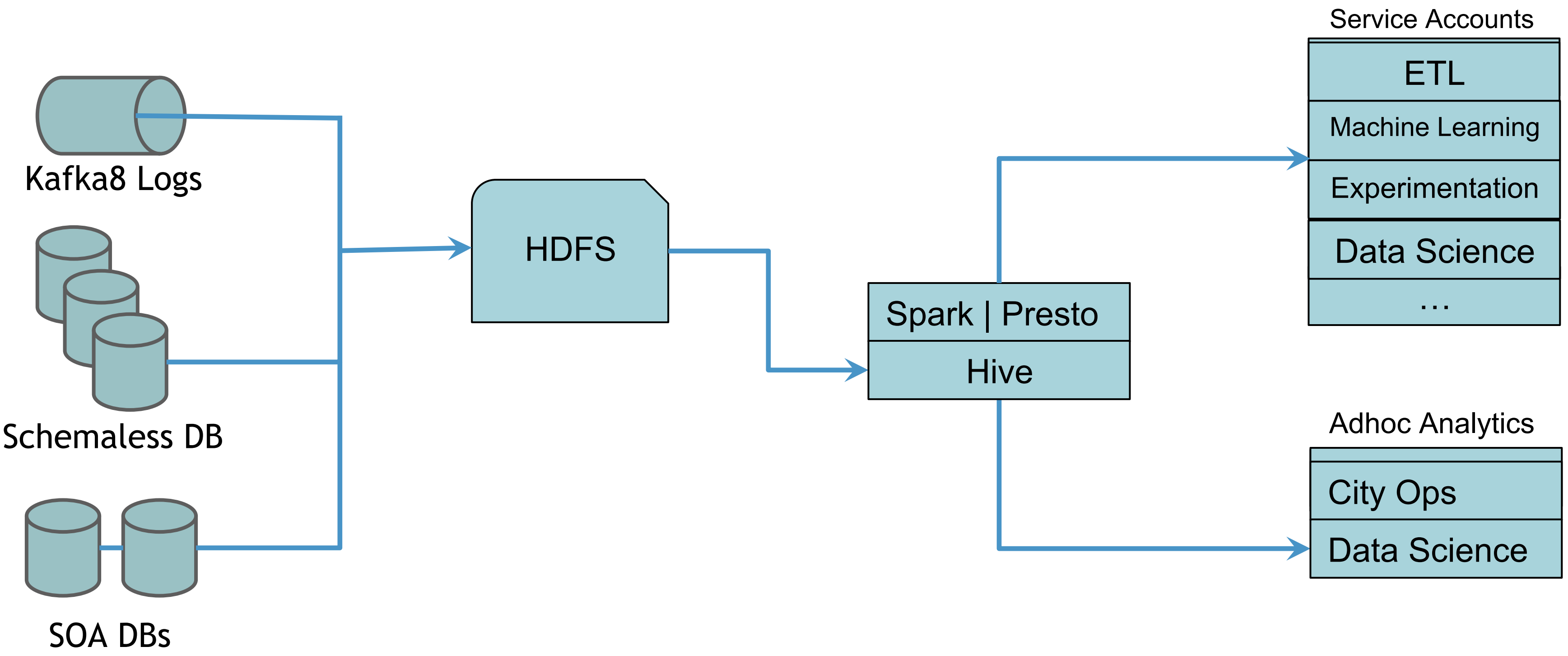
Uber’s Data Audience

- 1000s of City Operators (Uber Ops!)
  - driver funnel, rider retention, business performance and other daily/weekly reports
- 100s of Data Scientists and Analysts
  - A/B experimentations, Spend analysis etc
- 10s of Engineering Teams
  - real-time fraud detection, map search, location prediction etc

Once Upon a time.. (2014)



Data Infrastructure Today



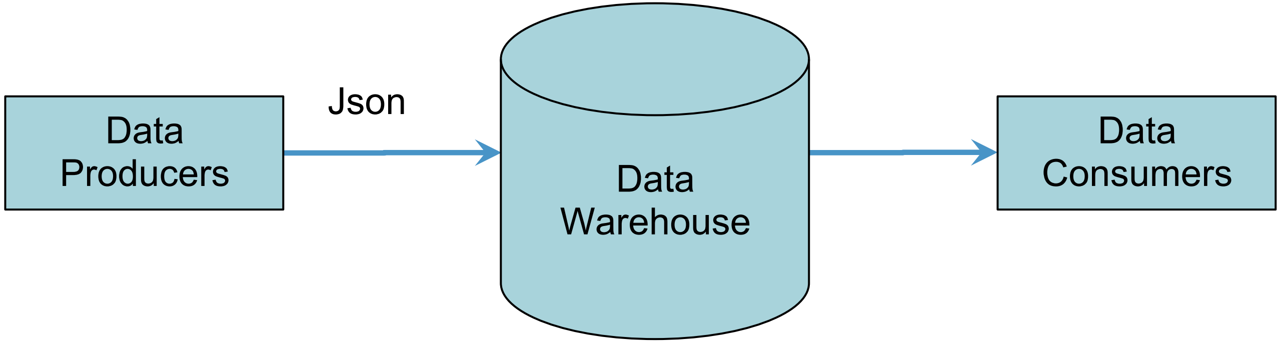
A Things we solved along the way..

- Scalable Ingestion Model
  - home-grown streaming ingestion solution
  - https://eng.uber.com/streamific/
- Built a Hadoop Data Lake
  - No more limited to storage, (EL from Data Sources instead of ETL)
  - JSON -> Avro -> Parquet

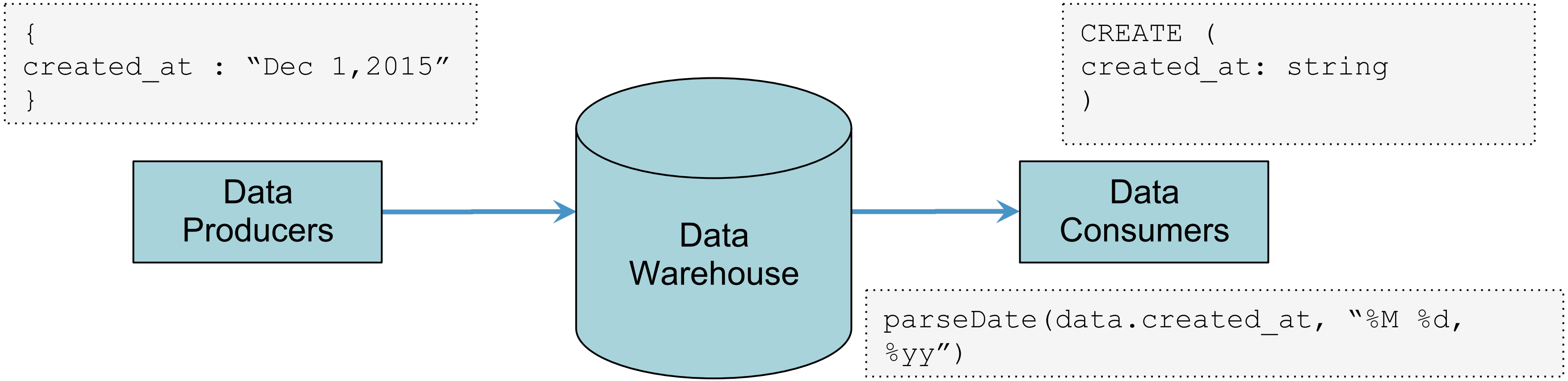
A Things we solved along the way..

- Strict Schema Management
  - Because our largest data audience are SQL Savvy! (1000s of Uber Ops!)
  - SQL = Strict Schema
- BigData Processing Tools Unlocked - Hive, Presto and Spark
  - Migrate SQL savvy users from Vertica to Hive & Presto (1000s of Ops & 100s of data scientists & analysts)
  - Spark for more advanced users - 100s of data scientists
- GeoSpatial Computation Platform
  - Because everyone runs geo based Queries
- Data Tools
  - Spark UDK - To reduce barrier to entry for writing Spark Jobs
  - Attis - A tool to analyze query costs and status

Pre-Strict Schema Management



Pre-Strict Schema Management

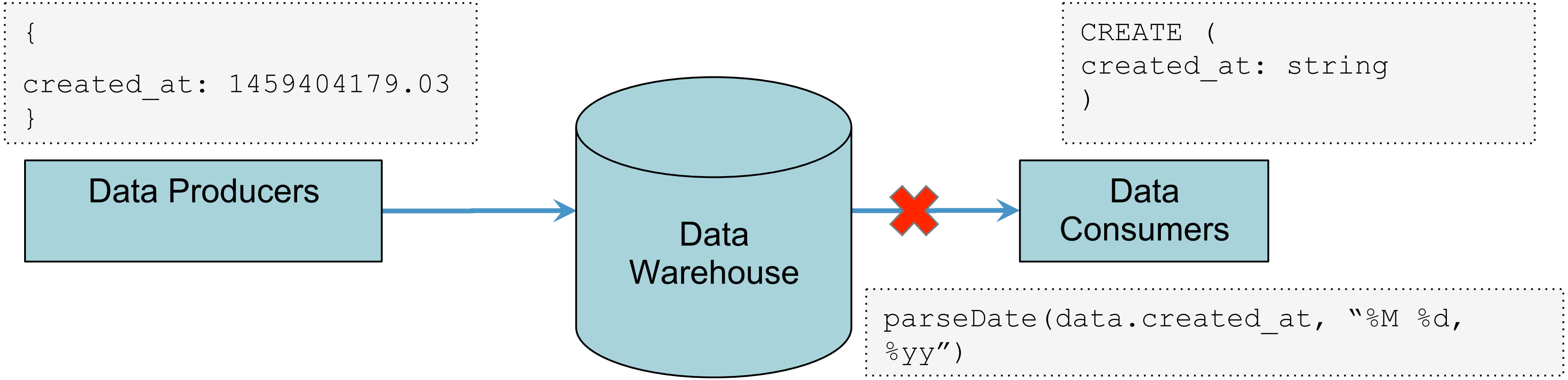


```
{
created_at : “Dec 1,2015”
}
```

```
CREATE (
created_at: string
)
```

```
parseDate(data.created_at, “%M %d,
%yy”)
```

Pre-Strict Schema Management



```
{
created_at: 1459404179.03
}
```

Data Producers

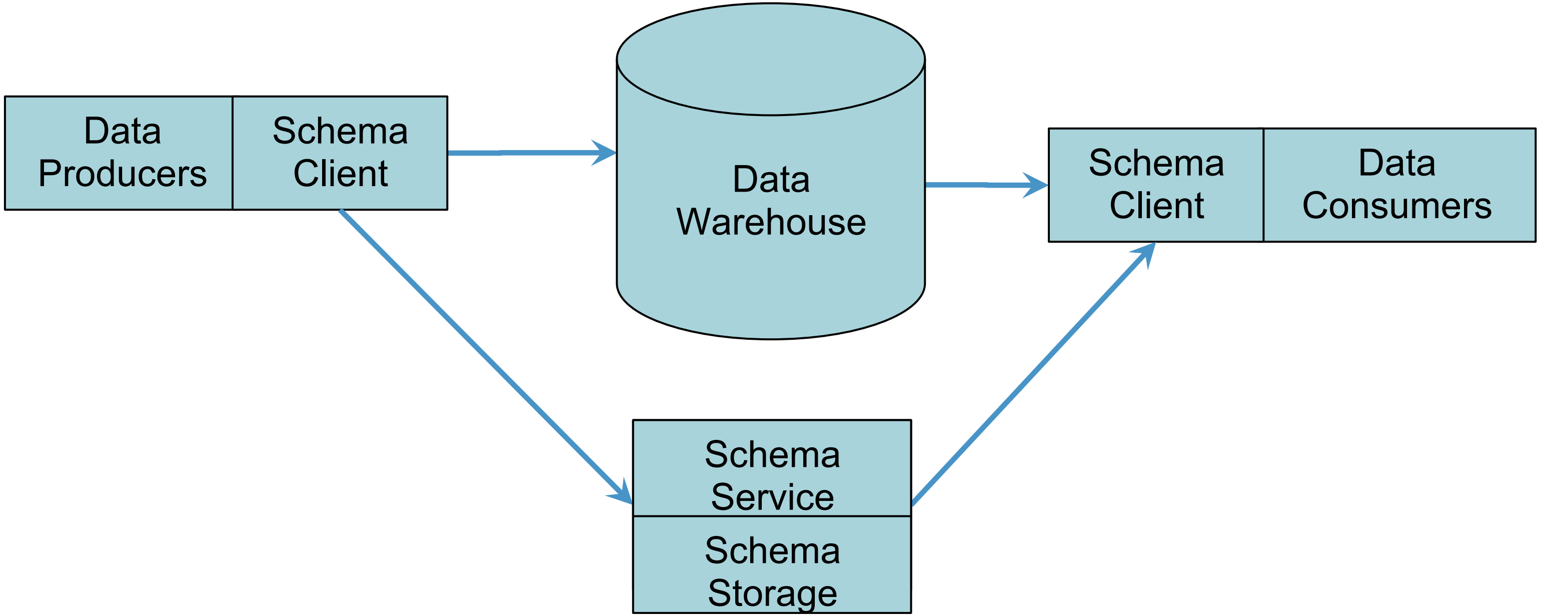
Data Warehouse

Data Consumers

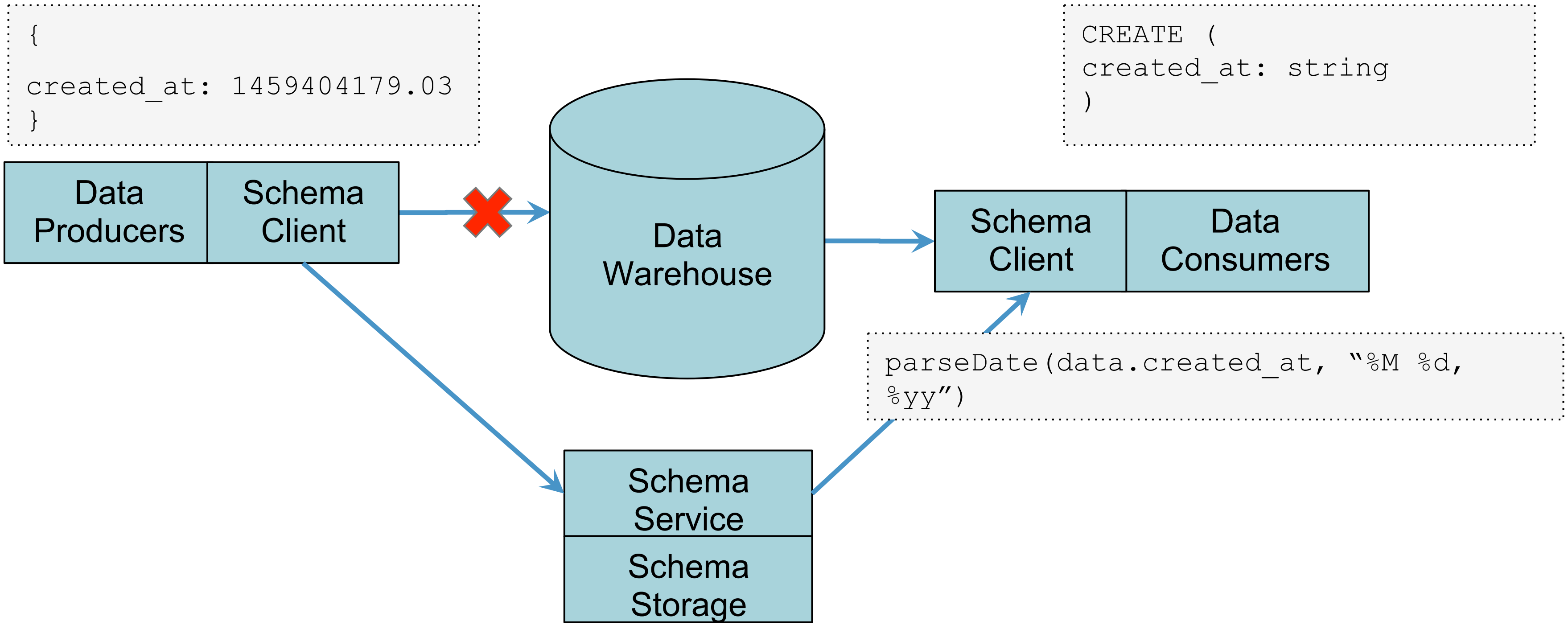
```
CREATE (
created_at: string
)
```

```
parseDate(data.created_at, “%M %d,
%yy”)
```

Solution: Centralized Schema Management



Solution: Centralized Schema Management



```
{
created_at: 1459404179.03
}
```

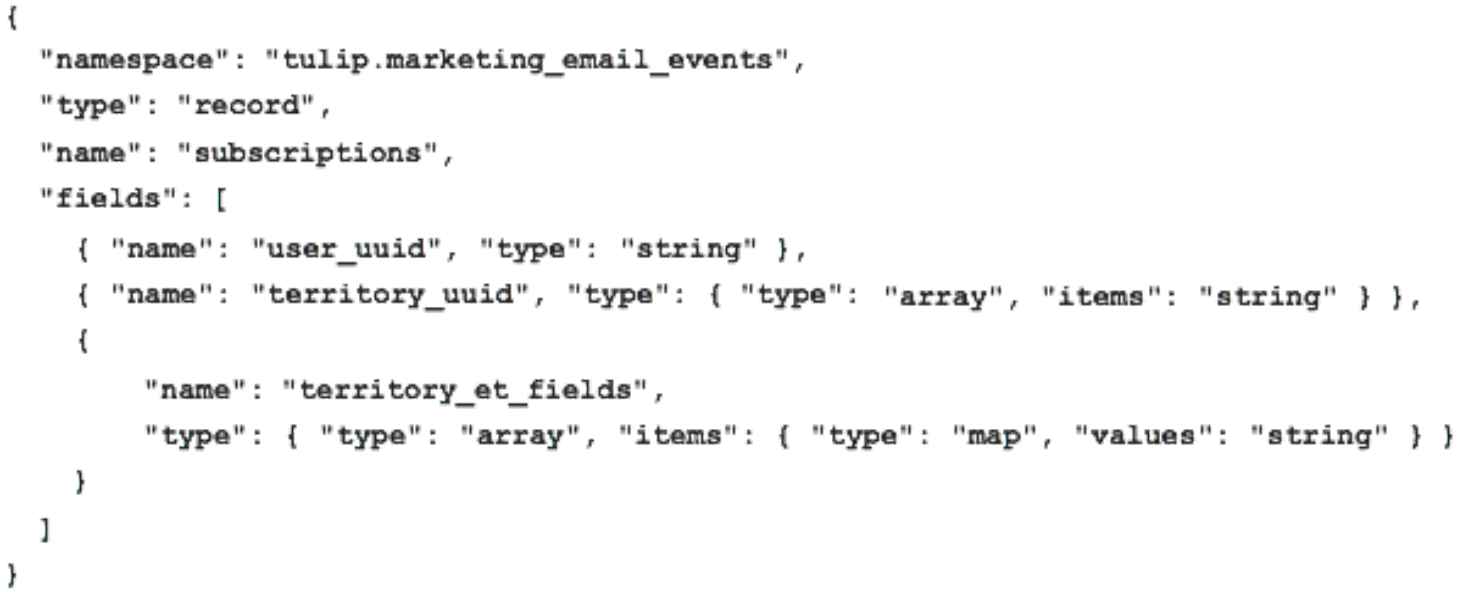
```
CREATE (
created_at: string
)
```

```
parseDate(data.created_at, “%M %d,
%yy”)
```

Solution: Strict Centralized Schema Management

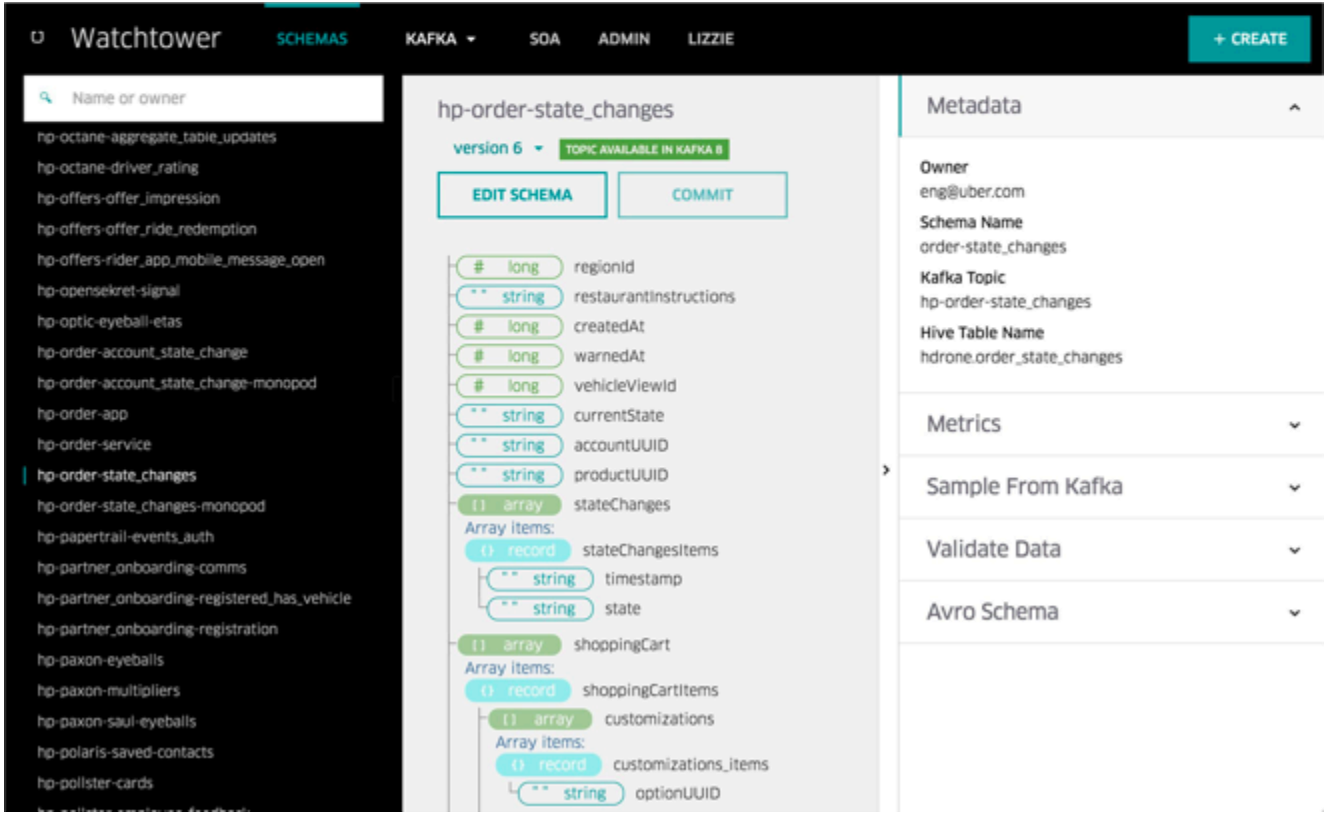
- A central versioned schema Contract for every dataset
  - used by teams, producers and consumers to negotiate data contracts
  - we use Heatpipe - an uber library which is a wrapper over Apache Avro as the serialization format
- A schema evolution system
  - Which ensures schemas evolution is compatible with previous data
  - Strictly typed
- A web UI schema manager
  - To easily create, edit, consume avro schemas.
  - Serves as documentation for data

Avro Schema Example



```
{
  "namespace": "tulip.marketing_email_events",
  "type": "record",
  "name": "subscriptions",
  "fields": [
    { "name": "user_uuid", "type": "string" },
    { "name": "territory_uuid", "type": { "type": "array", "items": "string" } },
    {
        "name": "territory_et_fields",
        "type": { "type": "array", "items": { "type": "map", "values": "string" } }
    }
  ]
}
```

Schema Manager Web Ui aka Watchtower



hp-octane-aggregate_table_updates
hp-octane-driver_rating
hp-offers-offer_impression
hp-offers-offer_ride_redemption
hp-offers-rider_app_mobile_message_open
hp-opensekret-signal
hp-optic-eyeball-etas
hp-order-account_state_change
hp-order-account_state_change-monopod
hp-order-app
hp-order-service
hp-order-state_changes
hp-order-state_changes-monopod
hp-papertrail-events_auth
hp-partner_onboarding-comms
hp-partner_onboarding-registered_has_vehicle
hp-partner_onboarding-registration
hp-paxon-eyeballs
hp-paxon-multipliers
hp-paxon-saul-eyeballs
hp-polaris-saved-contacts
hp-pollster-cards

hp-order-state_changes

version 6 TOPIC AVAILABLE IN KAFKA 8

```
# long regionId
" " string restaurantInstructions
# long createdAt
# long warnedAt
# long vehicleViewId
" " string currentState
" " string accountUUID
" " string productUUID
[ ] array stateChanges
Array items:
 { } record stateChangesItems
   " " string timestamp
   " " string state
[ ] array shoppingCart
Array items:
 { } record shoppingCartItems
   [ ] array customizations
   Array items:
    { } record customizations_items
      " " string optionUUID
```

Metadata

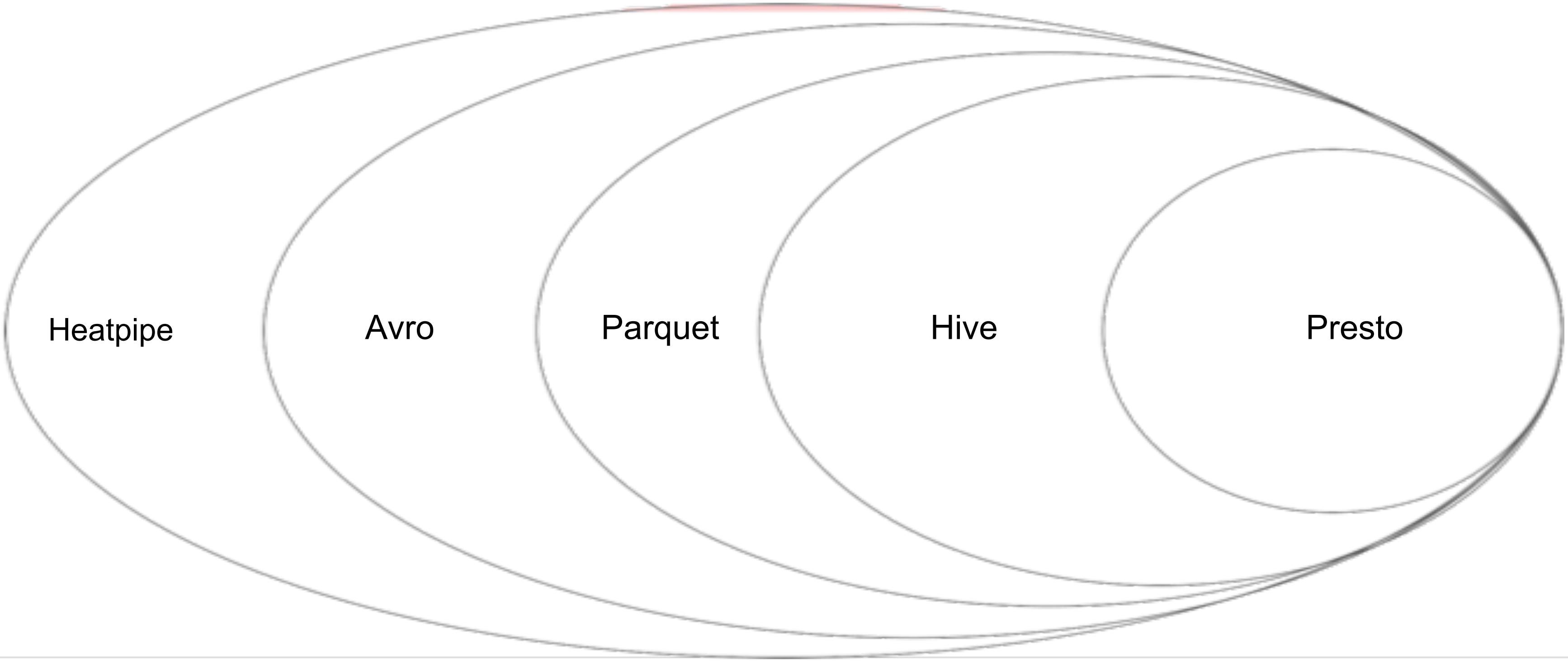
Owner
eng@uber.com

Schema Name
order-state_changes

Kafka Topic
hp-order-state_changes

Hive Table Name
hdrone.order_state_changes

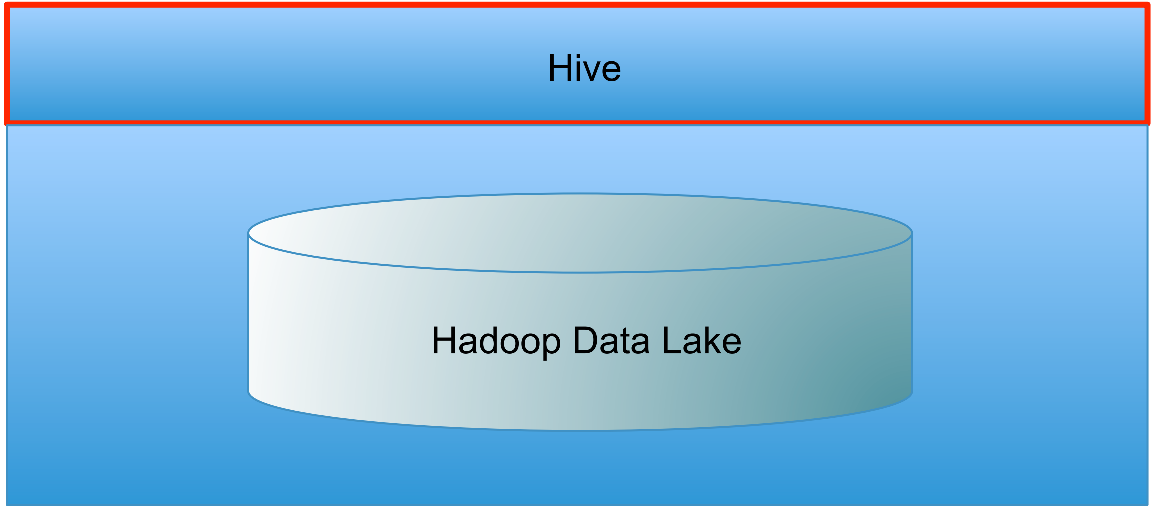
Schema Evolution Venn



Parquet for Hadoop Data Lake Storage

- Supports Schema
- 2 to 4 times faster than json/gzip
  - column pruning
    - wider nested table support (at uber)
  - filter predicate push-down
  - columnar compression
- Strong Open Source Support
  - Hive, Presto, Spark

Queryable Big Data Warehouse (2016)



But Hive is Slow..

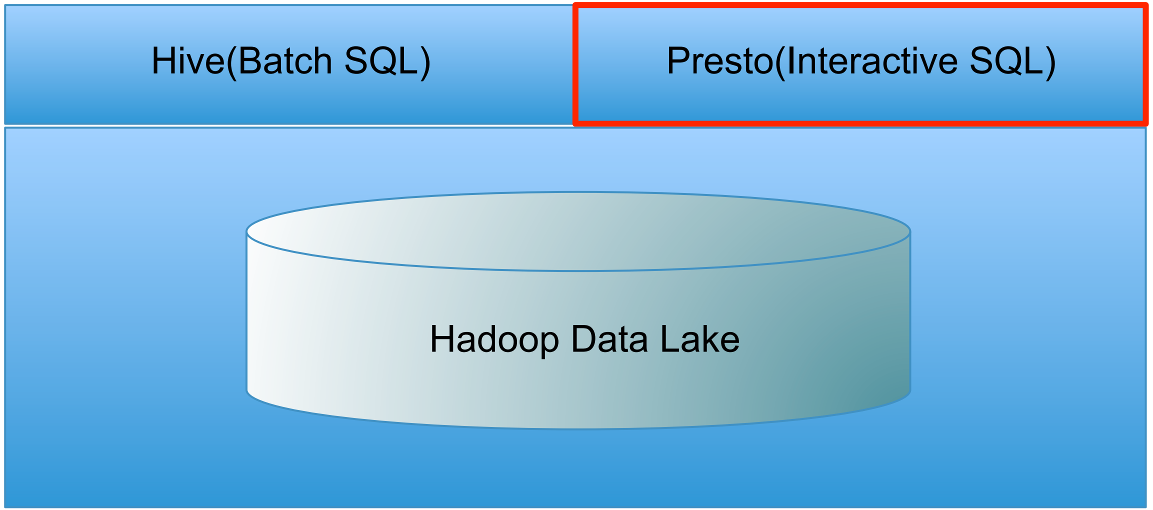
Vertica

Fast…
but **cannot scale** cheaply

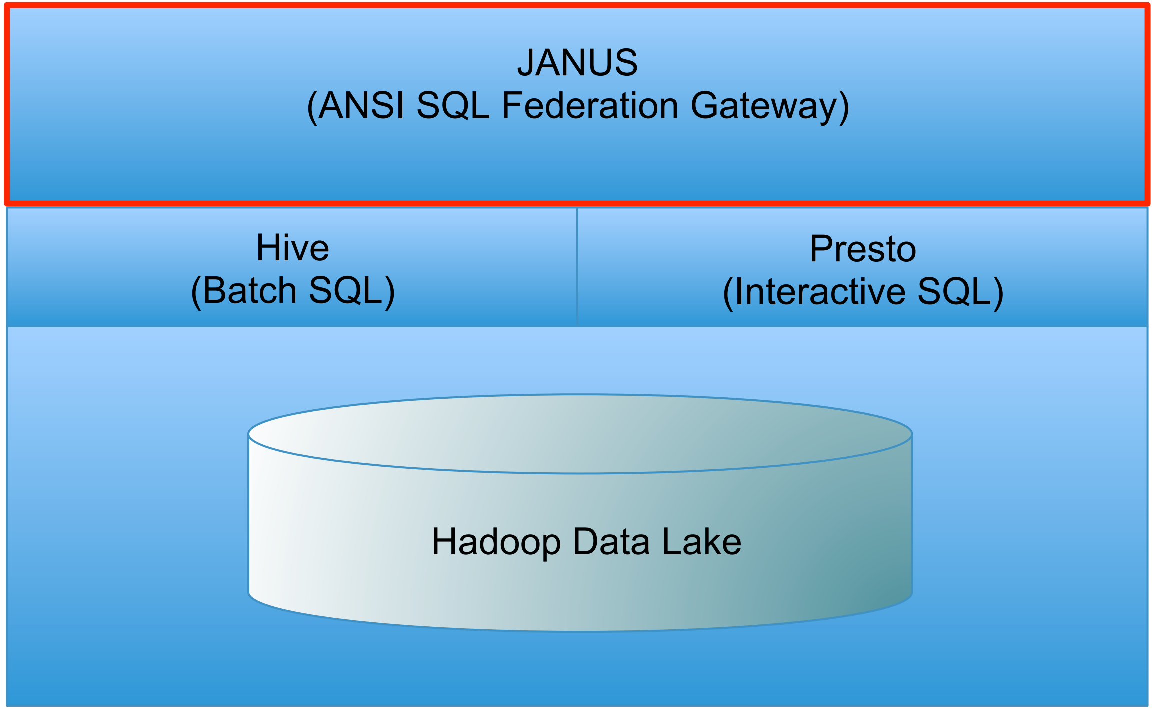
Hive

Scales cheaply and reliably...
but is not **fast**

Queryable Big Data Warehouse (2016)



Queryable Big Data Warehouse (2016)



JANUS
(ANSI SQL Federation Gateway)

Hive
(Batch SQL)

Presto
(Interactive SQL)

Hadoop Data Lake

Query Federation

- Adhoc/Scheduled SQL
  - Query via Janus - Gateway Service
  - Uses ANSI SQL standard (Presto, Hive underneath today..)
  - Dynamically picks YARN Queues
  - Keep bad queries out!

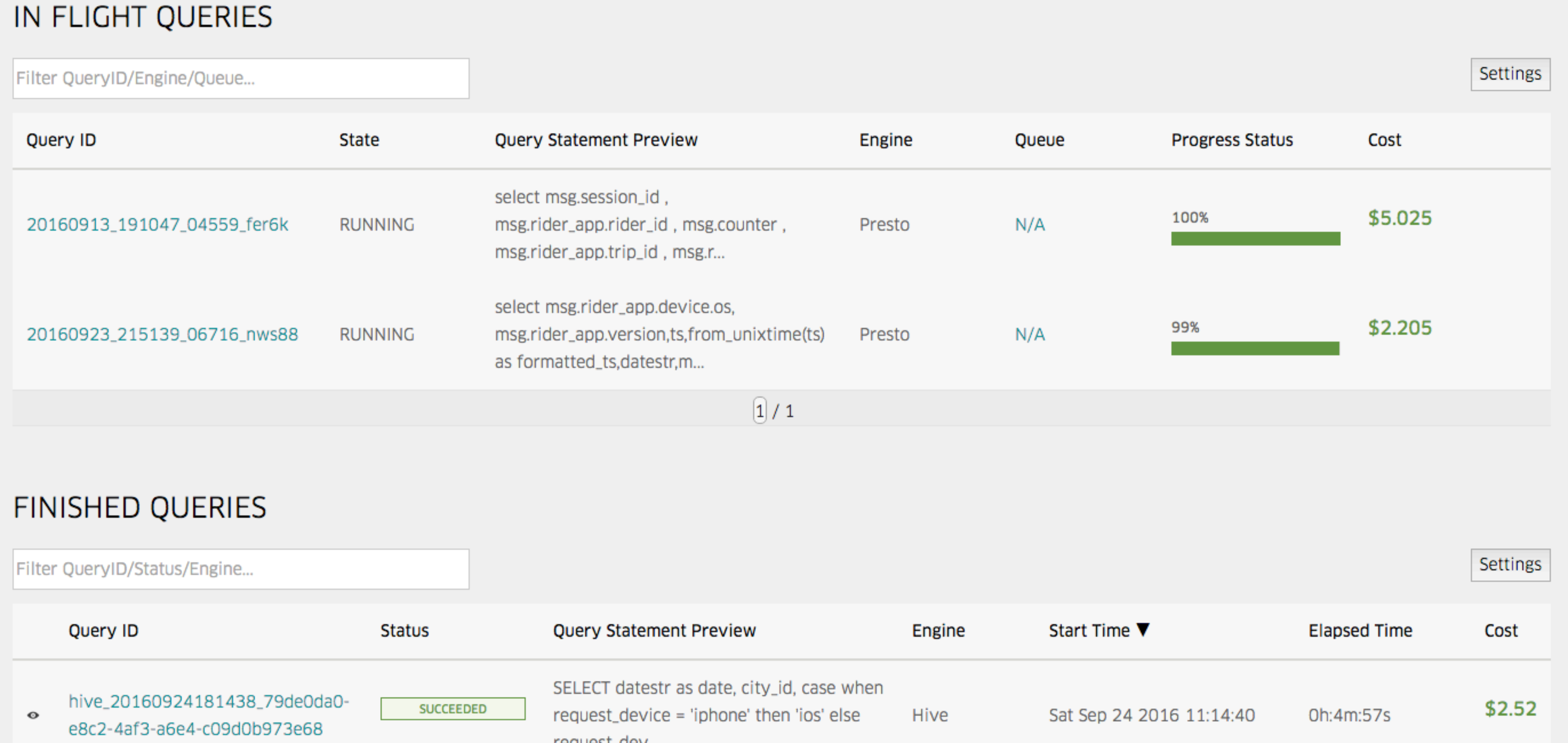
Query Engine Enhancements

- Presto
    - Nested Column Pruning for Parquet Columns
        - Making Presto fasterr!
    - Geospatial support
        - Filling in the UBER Gap!
- Hive
    - Hive on Parquet schema evolution fixes

Attis - Our Query Monitoring Tool

- Oracle AWR like reports
  - top queries by CPU
  - top queries by runtime
- Cost Analysis
  - Approx Cost to run query on AWS (by CPU and Memory)
- Realtime Query Tracker

Query Engine Monitor



IN FLIGHT QUERIES

Query ID	State	Query Statement Preview	Engine	Queue	Progress Status	Cost
20160913_191047_04559_fer6k	RUNNING	select msg.session_id , msg.rider_app.rider_id , msg.counter , msg.rider_app.trip_id , msg.r...	Presto	N/A	100%	$5.025
20160923_215139_06716_nws88	RUNNING	select msg.rider_app.device.os, msg.rider_app.version,ts,from_unixtime(ts) as formatted_ts,datestr,m...	Presto	N/A	99%	$2.205

FINISHED QUERIES

	Query ID	Status	Query Statement Preview	Engine	Start Time ▼	Elapsed Time	Cost
	hive_20160924181438_79de0da0-e8c2-4af3-a6e4-c09d0b973e68	SUCCEEDED	SELECT datestr as date, city_id, case when request_device = 'iphone' then 'ios' else request_dev	Hive	Sat Sep 24 2016 11:14:40	0h:4m:57s	$2.52

What next to solve for Data Warehousing?

- True Query Federation
  - Predict if a Query should be run on Hive or Presto?
- Query Translation
  - Can we convert expensive Presto Queries to Hive
- UDF Management across Hive/Presto

SQL solves for the most part....

But, What about Complex Data Applications?

- Machine Learning algos
- Low Latent batch processing
- Stitching HDFS files
- etc

SQL solves for the most part....

But, What about Complex Data Applications?

- Machine Learning algos
- Low Latent batch processing
- Stitching HDFS files
- etc

Use Spark!

Spark UDK (Uber Developer Kit)

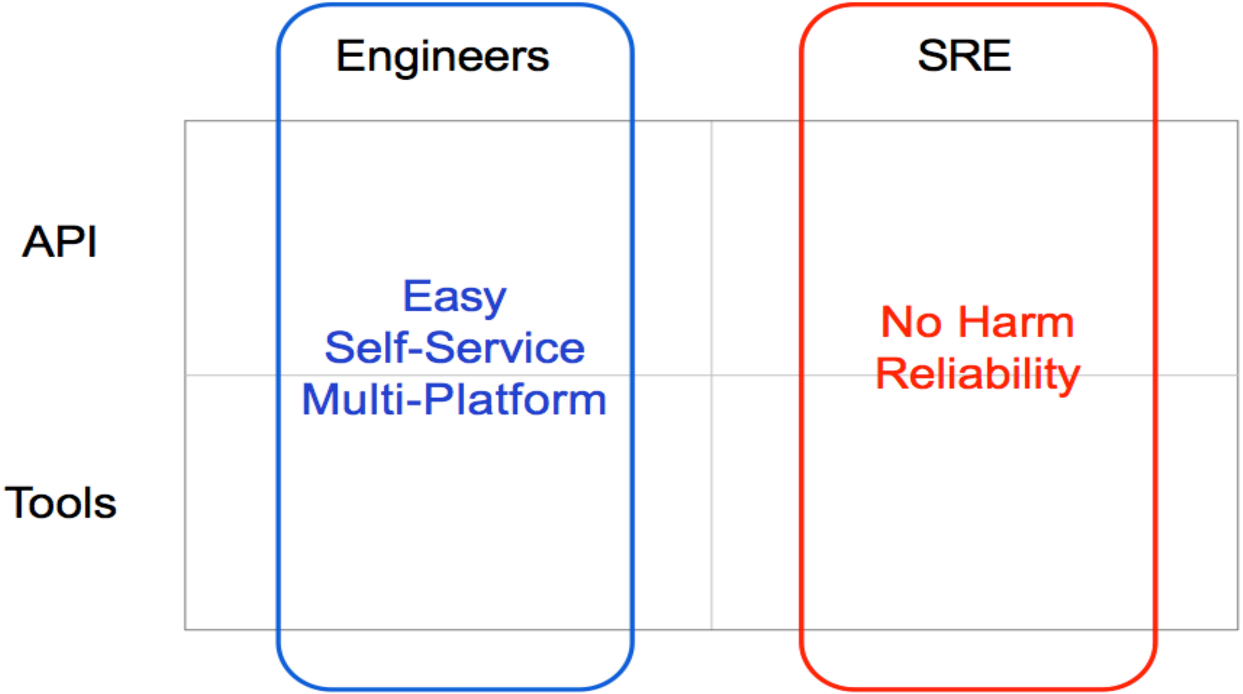
Goal:

- Self-Serve Development kit:
  - Reduce barrier to entry for new spark users
  - Application Lifecycle management
    - Scheduling, Monitoring etc
- Abstract our runtime environment
- Ensure a reliable multi-tenant infrastructure

Spark UDK

	Engineers	SRE
API		
Tools		

Spark UDK



Engineers

SRE

API

Tools

Easy
Self-Service
Multi-Platform

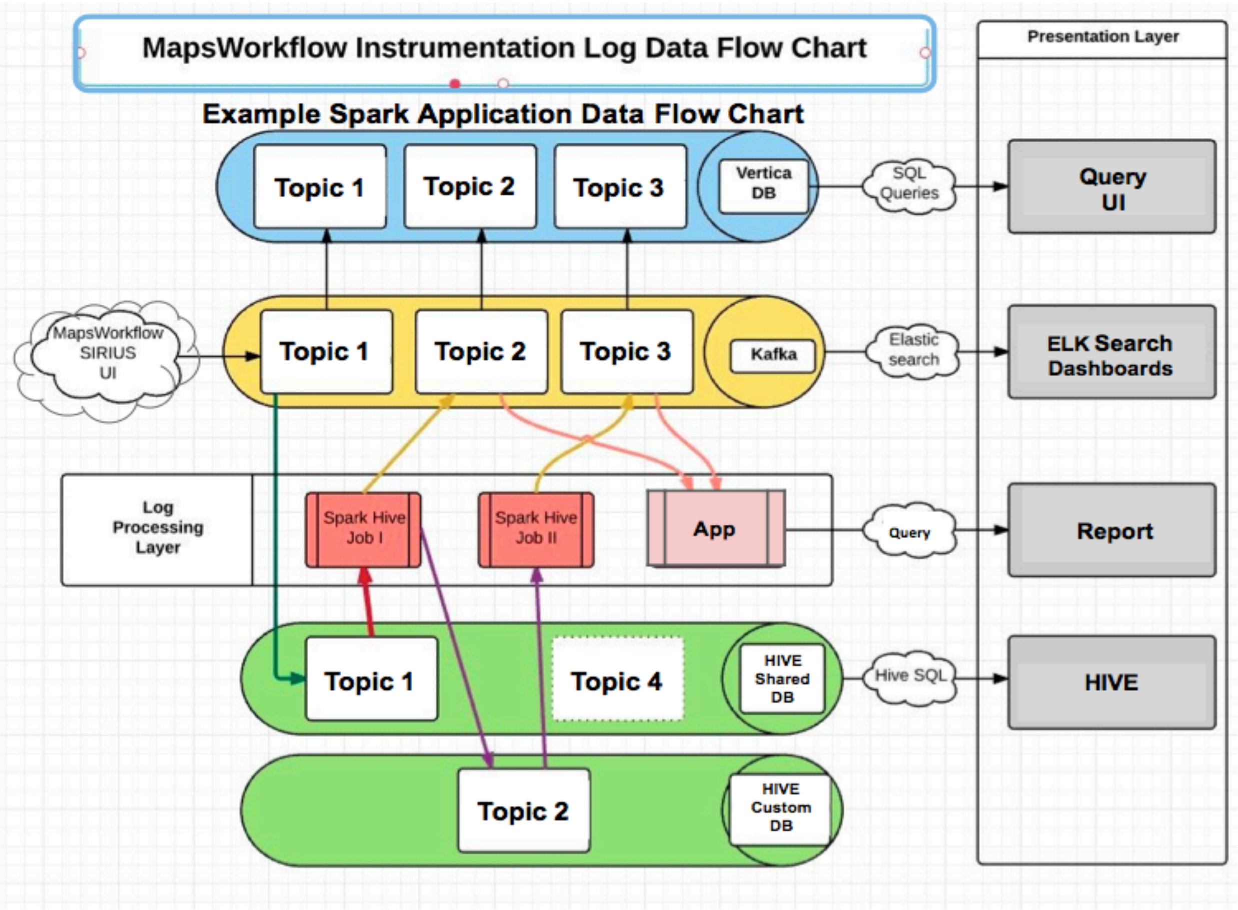
No Harm
Reliability

Spark UDK Engineering APIs

- SCBuilder
  - Encapsulates cluster environment details
  - Perf, debug optimized (history, event logs, YARN configs)
  - SRE approved CPU & Memory settings
- Data Dispersal
  - Kafka Dispersal
    - RDD - Parallelization
    - HA, Rate - limiting, schema enforcement
    - `publish(data: RDD, topic: String, schemaId: Int, appId: String)`
  - Also have connectors to Hive, Elastic Search

Spark UDK Tools

- Sparkplug
  - A collection of popular job templates
  - Two commands to run the first job in Dev
  - One use case per template
    - e.g. Ozzie + SparkSQL + Incremental processing
    - e.g. Incremental processing + Kafka dispersal
  - Best Practices
    - built-in unit tests, test coverage, Jenkins
    - built-in Kafka, HDFS mocks


MapsWorkflow Instrumentation Log Data Flow Chart
Example Spark Application Data Flow Chart
Topic 1
Topic 2
Topic 3
Vertica DB
SQL Queries
Query UI
Presentation Layer
MapsWorkflow SIRIUS UI
Topic 1
Topic 2
Topic 3
Kafka
Elastic search
ELK Search Dashboards
Log Processing Layer
Spark Hive Job I
Spark Hive Job II
App
Query
Report
Topic 1
Topic 4
HIVE Shared DB
Hive SQL
HIVE
Topic 2
HIVE Custom DB

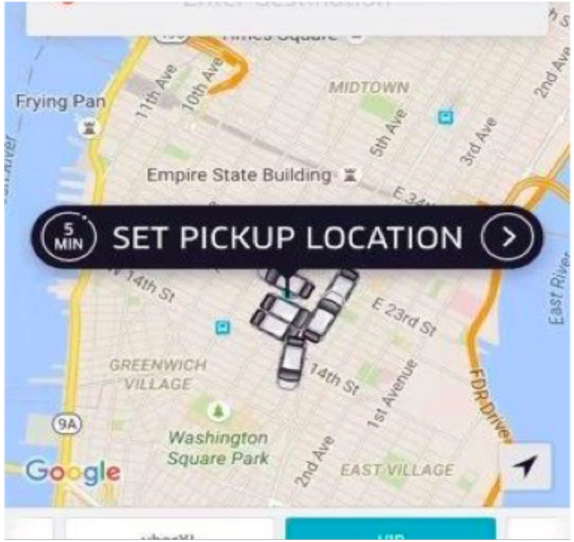
Future Work

	Engineers	SRE
API	• SCBuilder • Kafka dispersal • Hive table registration • Incremental processing • Geo-spatial processing • Debug logging • Metrics • Configurations • Data Freshness	• Resource usage
Tools	• Distributed Debugger • SparkPlug • Unit testing • Oozie integration)	• Resource usage auditing • Data access auditing • Machine learning on jobs

Uber Geospatial Processing

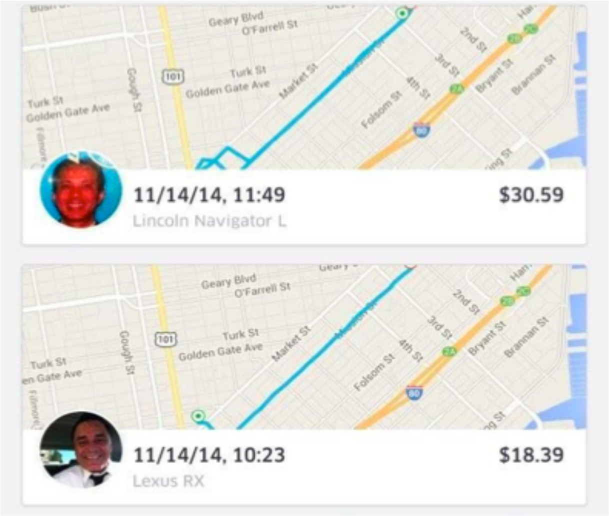
Denton
McK
ower Mound
PLANO
Grapevine
Sach
Garla
haw
Euless
DALLAS
RTH
ARLINGTON
Duncanville
Lancaster
Desoto
Mansfield
Midlothian

within(trip_location, city_shape)
Find if a car is within a city



contains(geofence, auto_location)
Find all cars in an area

Uber Geospatial Processing



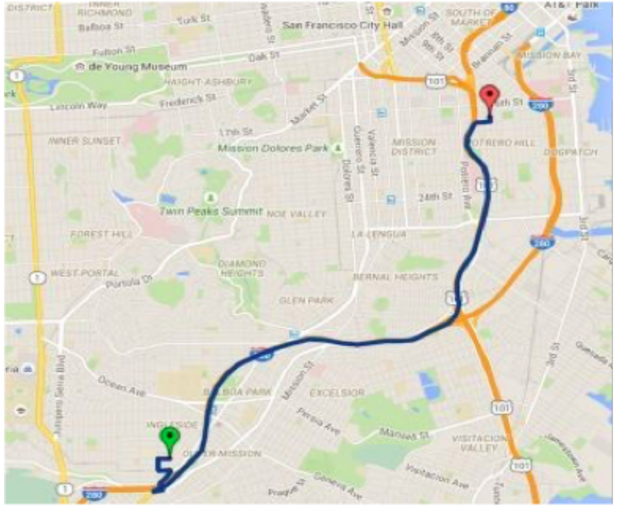
11/14/14, 11:49 $30.59

Lincoln Navigator L

11/14/14, 10:23 $18.39

Lexus RX

overlaps(trip1, trip2)
Find trips that have similar routes



intersects(trip_location, gas_locations)
Find all gas stations a trip has passed by

Spatial Join: The Problem

Objective: Associate all trips with city_id for a single day.

```
SELECT trip.trip_id, city.city_id
FROM trip JOIN city
WHERE contains(city.city_shape, trip.start_location)
AND trip.datestr = ‘2016-09-07’
```

Spatial Join: The Problem

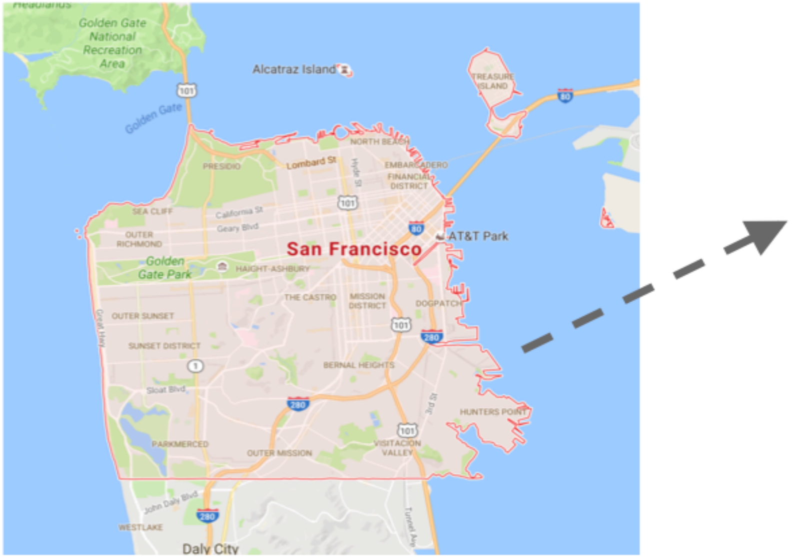
Objective: Associate all trips with city_id for a single day.

```
SELECT trip.trip_id, city.city_id
FROM trip JOIN city
WHERE contains(city.city_shape, trip.start_location)
AND trip.datestr = ‘2016-09-07’
```

Notice that a cross join is involved in the raw query which is prohibitively time consuming

City Boundary (field simplified_shape)

Geofence shape



OGC format

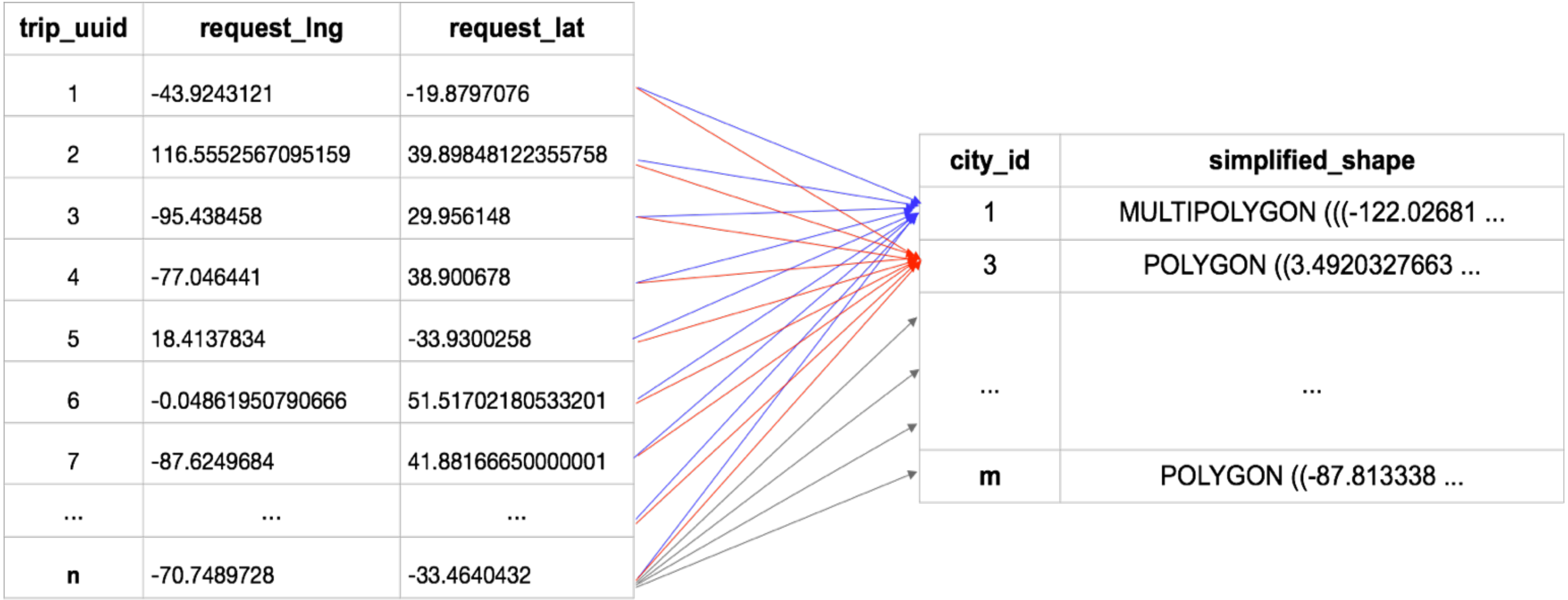
MULTIPOLYGON (((-122.02681 36.546405, -122.037459 36.560381, -122.041148 36.568211, -122.044402 36.580456, -122.044148 36.591354, -122.042748 36.596627, -122.034248 36.609673, -122.023636 36.620443, -122.020127 36.622798, -122.020406 36.624959, -122.019351 36.627761, -122.014173 36.636682, -122.003487 36.646399, -122.001124 36.647704, -122.068317 36.874953,

………..
………..
………..

-123.073063 37.682583, -123.074921 37.690066, -123.086057 37.70075, -123.09285 37.713688, -123.094801 37.714067, -123.104921 37.712648, -123.115575 37.714045, -123.126114 37.71673, -123.132597 37.719393, -123.156757 37.73784, -123.162751 37.743533, -123.169985 37.75593, -123.172926 37.7636, -123.173763 37.771831)))"

.

Trip City Association Spatial Cross Join



trip_uuid	request_lng	request_lat
1	-43.9243121	-19.8797076
2	116.5552567095159	39.89848122355758
3	-95.438458	29.956148
4	-77.046441	38.900678
5	18.4137834	-33.9300258
6	-0.04861950790666	51.51702180533201
7	-87.6249684	41.88166650000001
...	...	...
n	-70.7489728	-33.4640432

city_id	simplified_shape
1	MULTIPOLYGON (((-122.02681 ...
3	POLYGON ((3.4920327663 ...
...	...
m	POLYGON ((-87.813338 ...

Time complexity (n*m) = 10M trips x 1K cities
= 10B operations ~ 1 week computation time

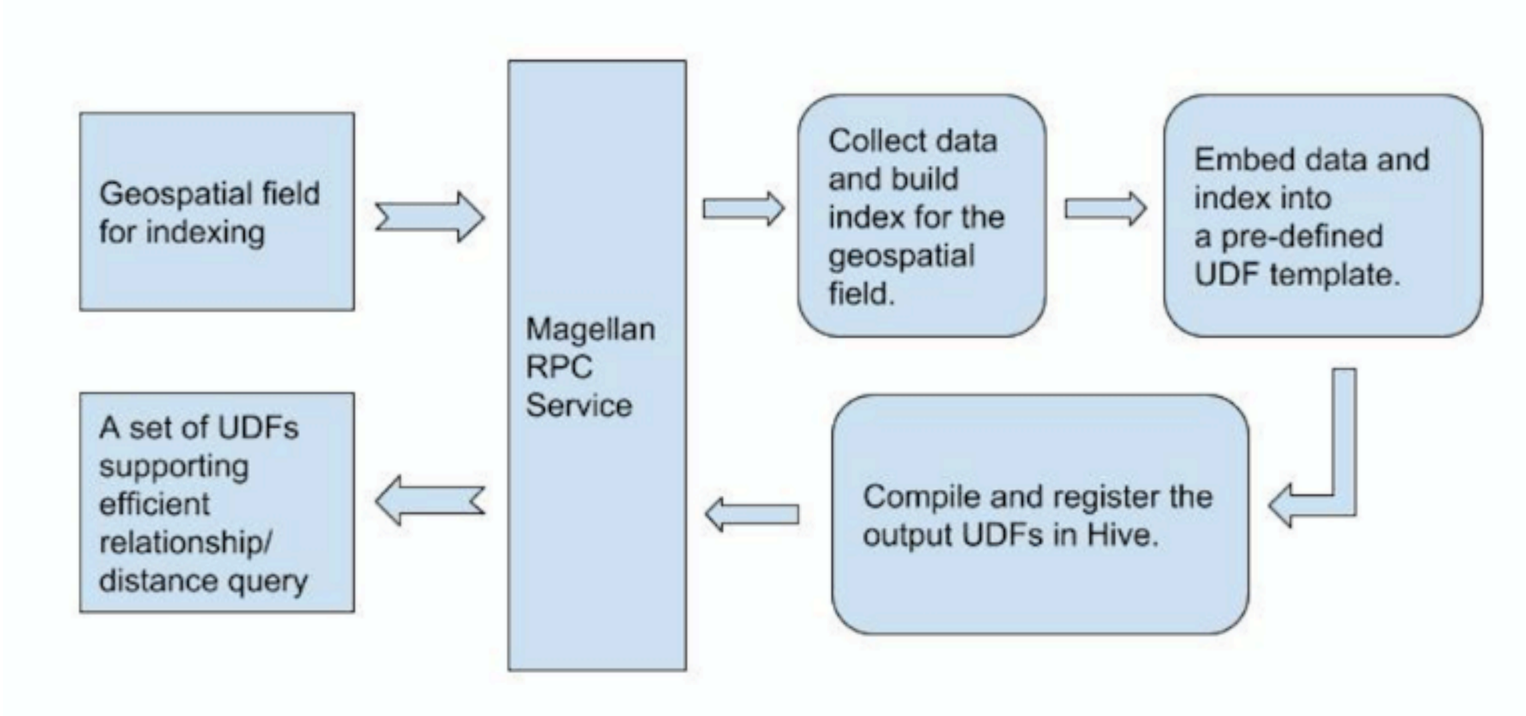
Spatial Join: Solution

- Use Generated UDFs which uses a geospatial index and avoid cross joins

```
SELECT trips.id, getCityId(trips.request_location)
FROM trips
```

We Build either quadtree or r-tree indexes dynamically

Magellan - A self serve Geospatial Service



Geospatial field for indexing

Magellan RPC Service

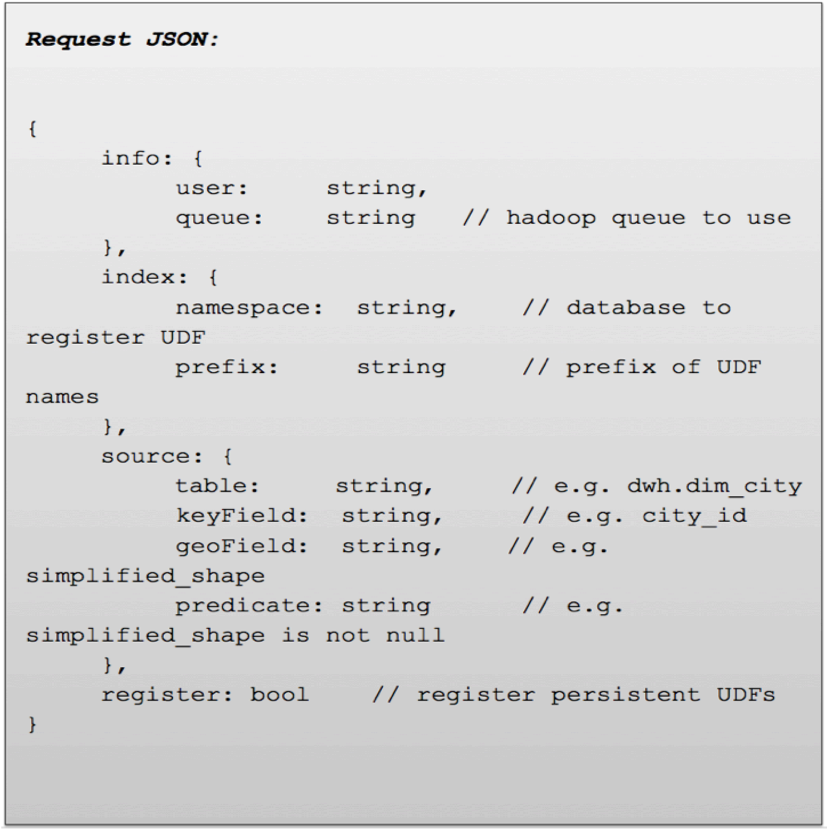
Collect data and build index for the geospatial field.

Embed data and index into a pre-defined UDF template.

Compile and register the output UDFs in Hive.

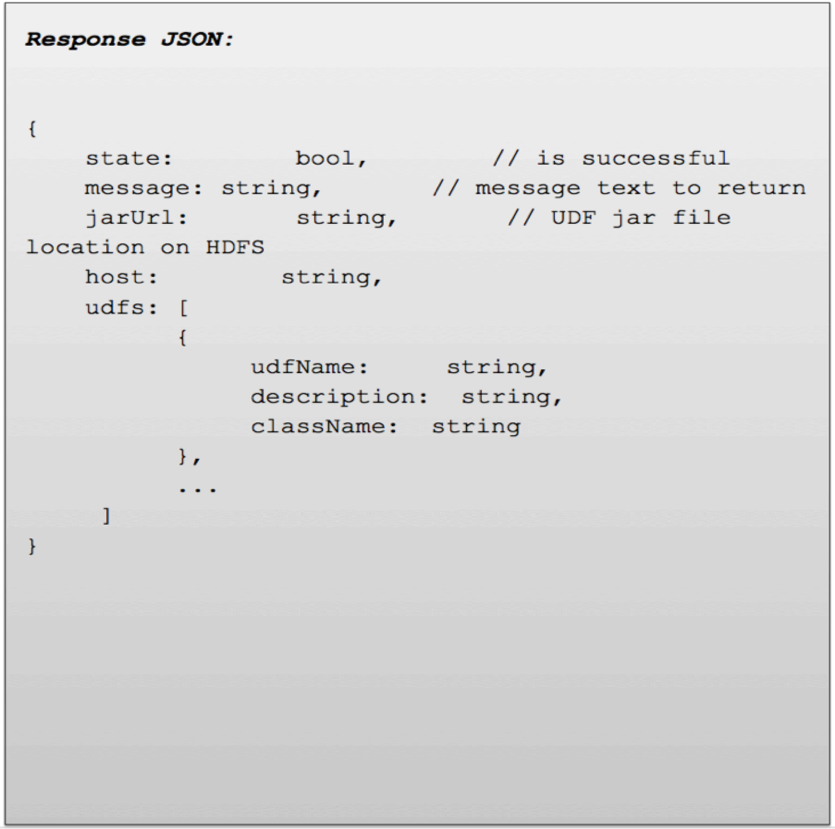
A set of UDFs supporting efficient relationship/ distance query

Magellan - A self serve Geospatial Service



Request JSON:

```
{
     info: {
          user:     string,
          queue:    string   // hadoop queue to use
     },
     index: {
          namespace:  string,    // database to
register UDF
          prefix:     string     // prefix of UDF
names
     },
     source: {
          table:     string,     // e.g. dwh.dim_city
          keyField:  string,     // e.g. city_id
          geoField:  string,    // e.g.
simplified_shape
          predicate: string      // e.g.
simplified_shape is not null
     },
     register: bool    // register persistent UDFs
}
```



Response JSON:

```
{
    state:         bool,         // is successful
    message: string,         // message text to return
    jarUrl:        string,         // UDF jar file
location on HDFS
    host:          string,
    udfs: [
          {
               udfName:      string,
               description:  string,
               className:  string
          },
          ...
     ]
}
```

What Next for Spatial Processing?

- Extend ingestion pipelines to support spatial-index fields
- Enhance query-engines (Hive, Presto, Spark) to auto optimize on supported index fields

Key Takeaways

- EL from Source to Data Lake
  - Going back to fetch from online sources over and over again is not a good idea especially at a large scale
- Always manage schemas if you have > 1 consumer
  - When an organization scales, you need automated ways to manage lineage & schema evolution to avoid pain
- Abstract Query Engines Access and Use Standards
  - ANSI SQL - Makes swapping query engines later easier
  - Use a gateway to audit, your SRE/Ops will like you for it
- Leverage Open Source Whenever Possible
  - While filling in the gaps, and contributing back!!

Thank you!