Machine Learning and Big Data @ Uber: A Tale of Two Systems

Zhenxiao Luo

Engineering Manager @ Uber

Agenda

Mission

Uber Business Highlights

Machine Learning @ Uber

Pain Points

Big Data @ Uber

Machine Learning support in Big Data

Ongoing Work

Uber Mission

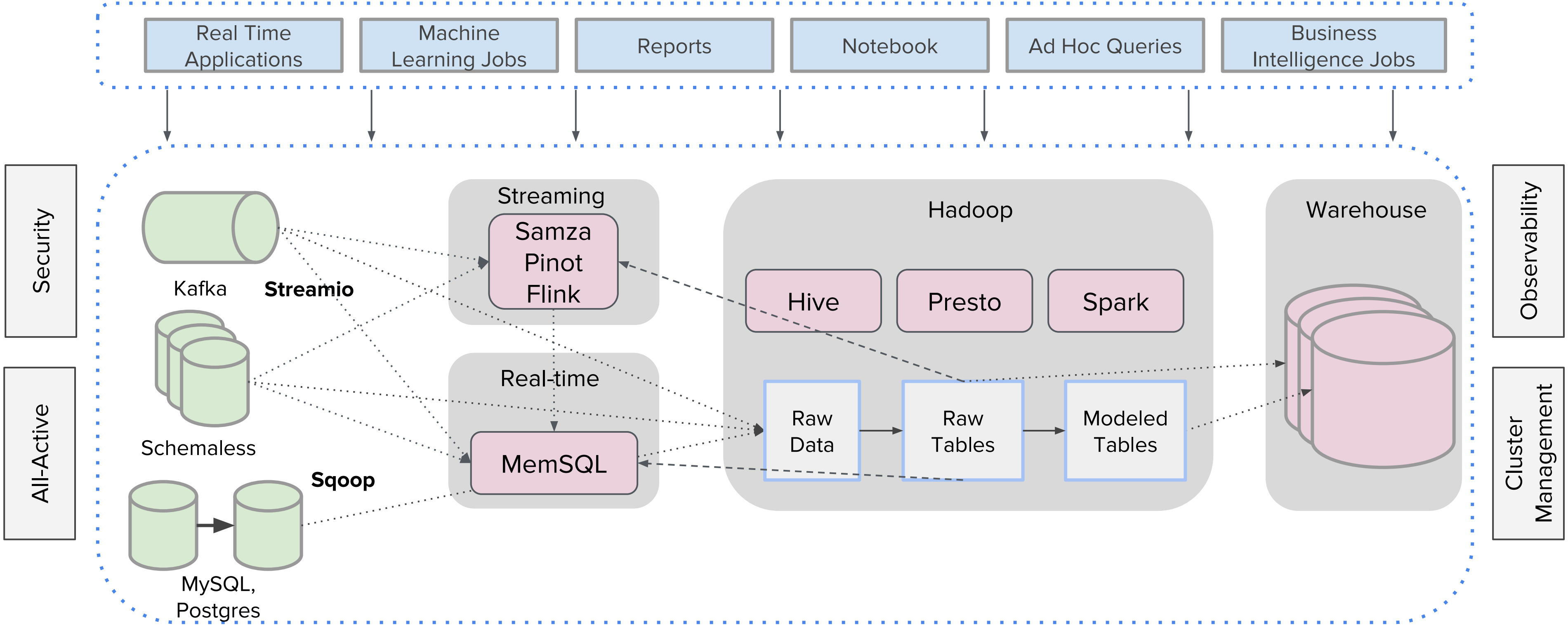
Transportation as reliable as running water, everywhere, for everyone

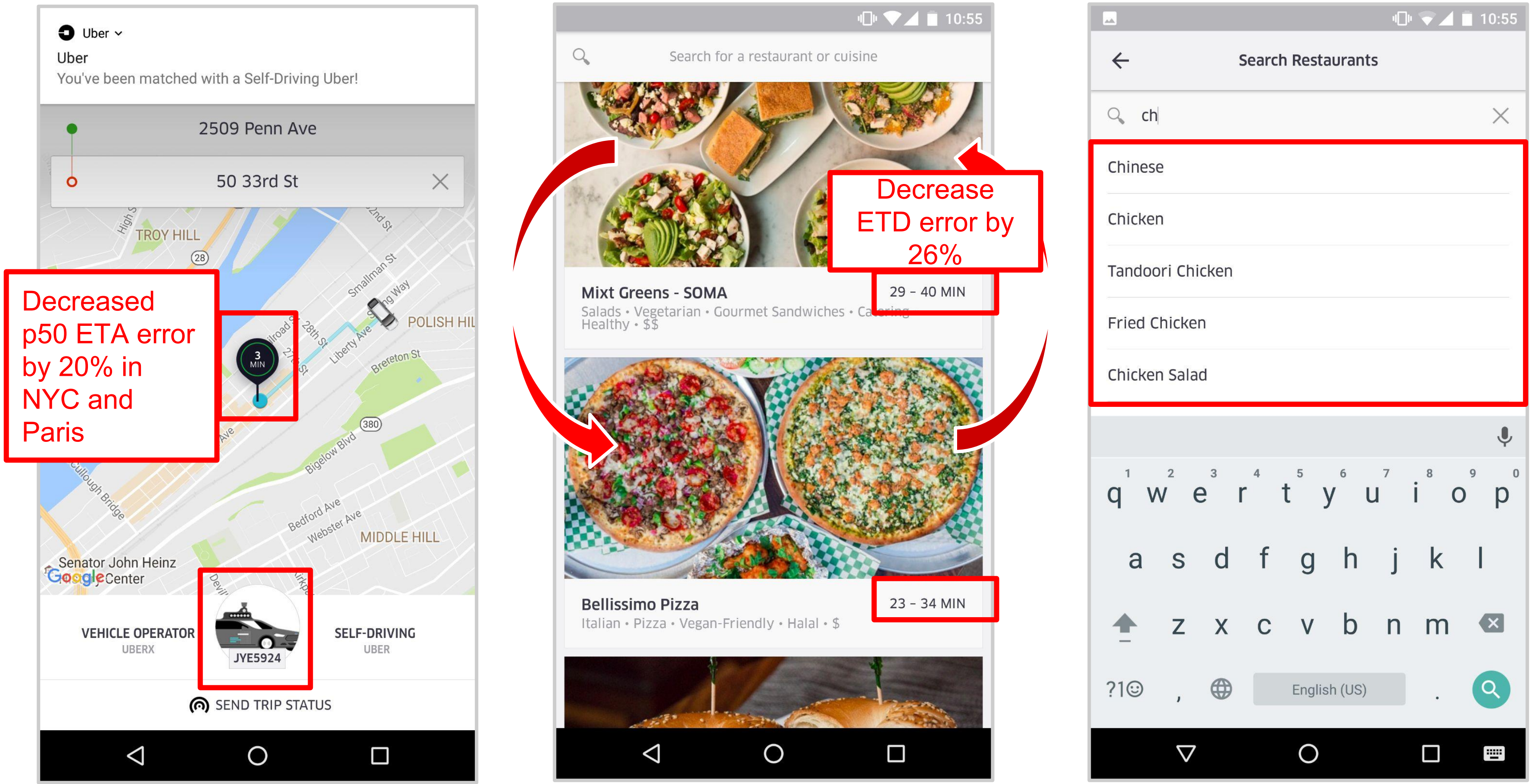
Uber Stats

6 Continents	73 Countries	450 Cities	12,000 Employees

10+ Million Avg. Trips/Day	40+ Million MAU Riders	1.5+ Million MAU Drivers

Data Infrastructure @ Uber



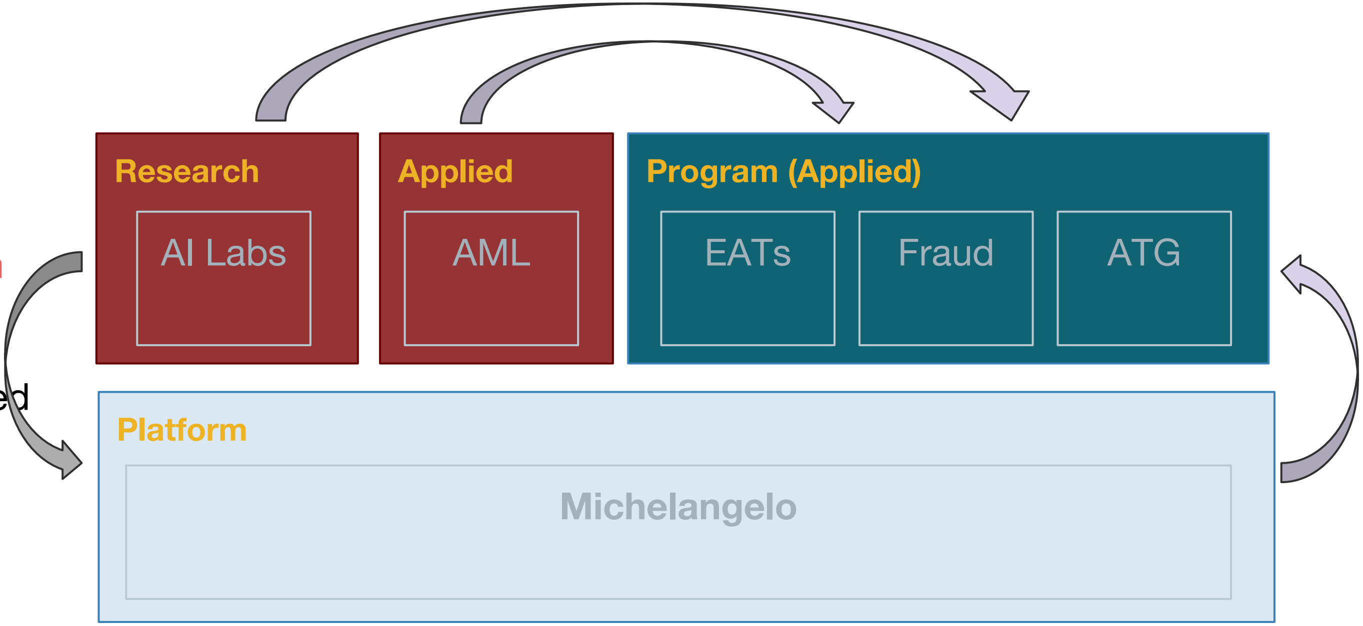

Uber
Uber
You've been matched with a Self-Driving Uber!
2509 Penn Ave
50 33rd St
Decreased p50 ETA error by 20% in NYC and Paris
3 MIN
VEHICLE OPERATOR
UBERX
JYE5924
SELF-DRIVING
UBER
SEND TRIP STATUS
10:55
Search for a restaurant or cuisine
Decrease ETD error by 26%
Mixt Greens - SOMA
29 – 40 MIN
Salads • Vegetarian • Gourmet Sandwiches • Catering
Healthy • $$
Bellissimo Pizza
23 – 34 MIN
Italian • Pizza • Vegan-Friendly • Halal • $
10:55
Search Restaurants
ch
Chinese
Chicken
Tandoori Chicken
Fried Chicken
Chicken Salad
English (US)

Michelangelo: Uber Machine Learning Platform

Research teams focus their research based on business priorities from program teams

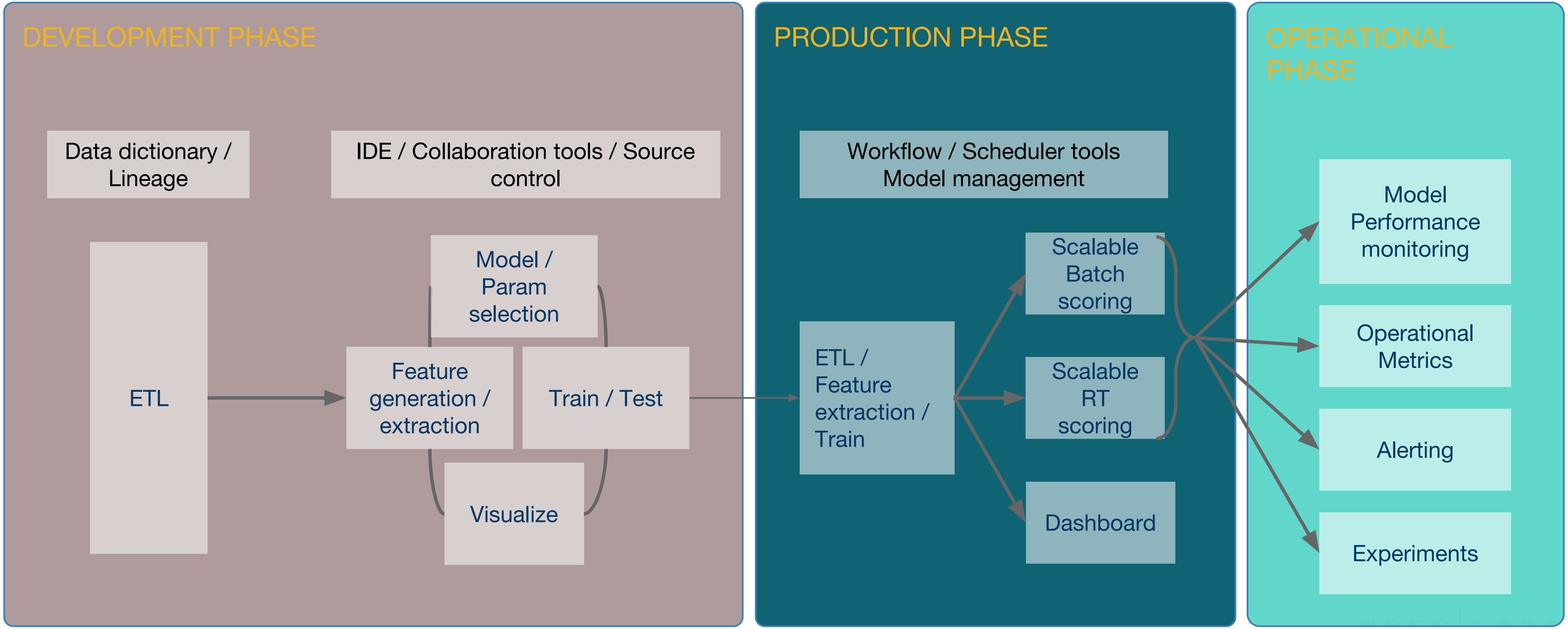
Applied ML teams helps with problems when Program team doesn't have right expertise; platform-izes the solutions

Research team invents new techniques which are added to the platform



Program teams build ML apps on the platform

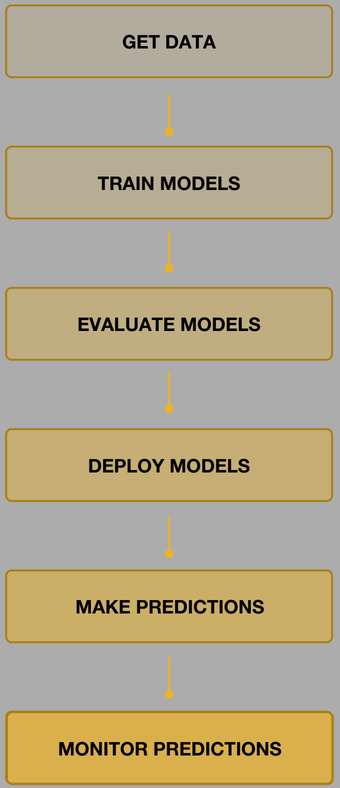
Workflow of a Machine Learning Project

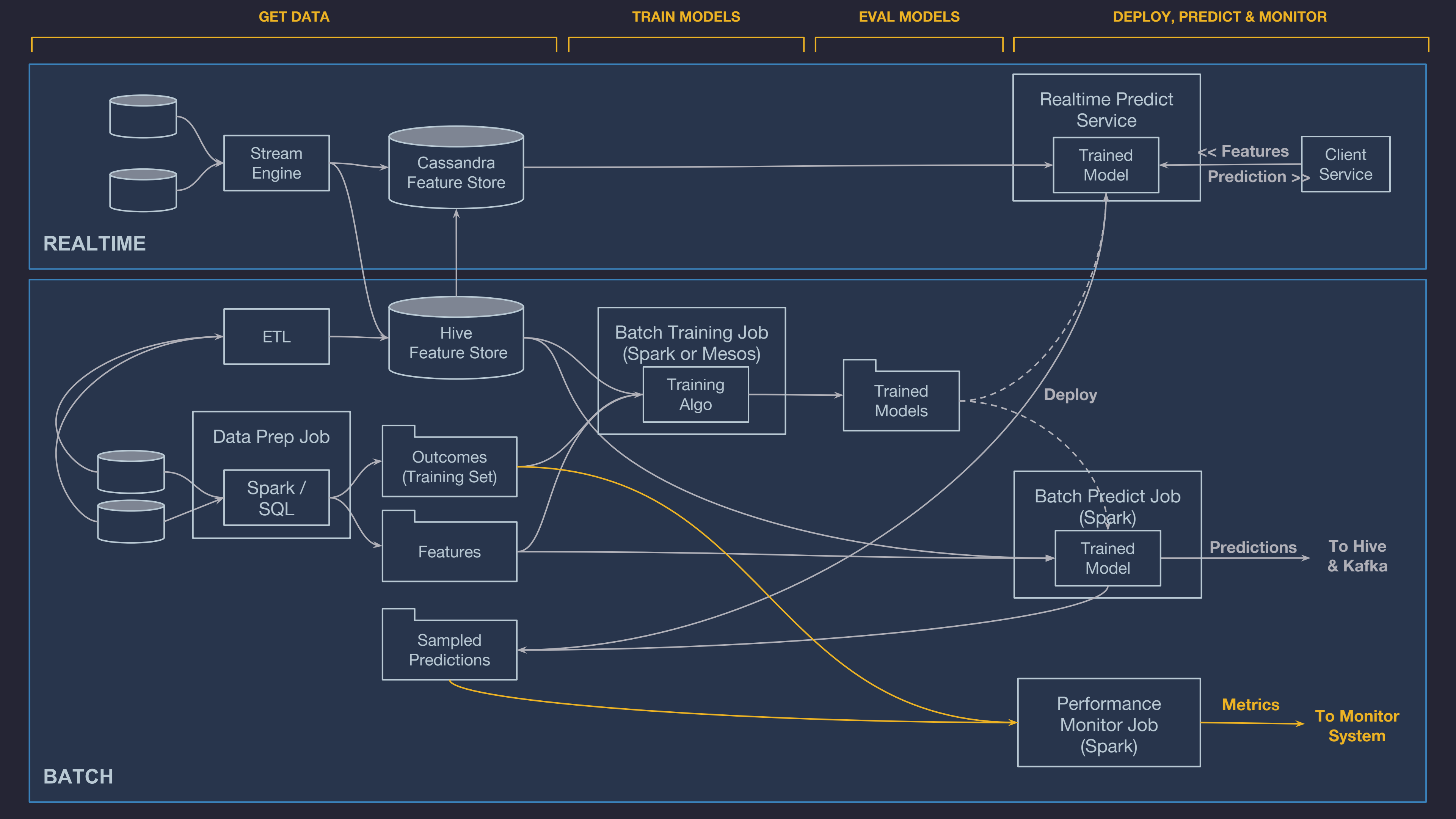


Workflow

Same basic ML workflow & system requirements for

- Traditional ML & Deep Learning
- Supervised, Unsupervised & Semi-supervised learning
- Online learning
- Batch, Realtime and Mobile deployments
- Timeseries Forecasting

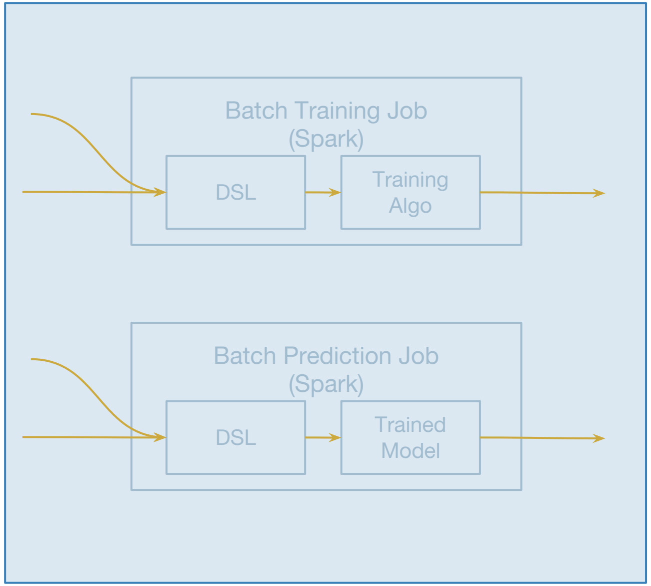




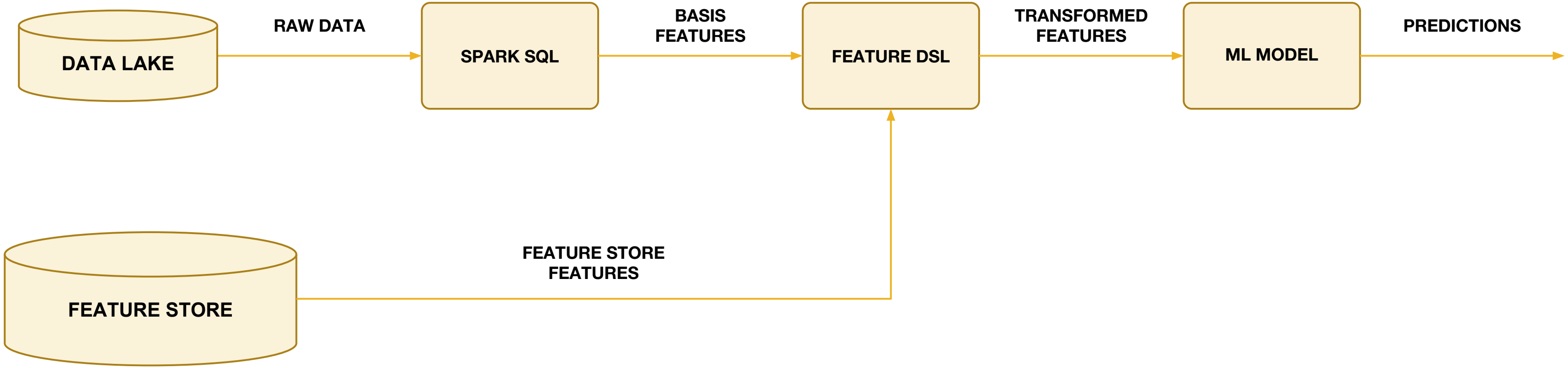
GET DATA
TRAIN MODELS
EVAL MODELS
DEPLOY, PREDICT & MONITOR
Stream Engine
Cassandra Feature Store
Realtime Predict Service
Trained Model
<< Features
Prediction >>
Client Service
REALTIME
ETL
Hive Feature Store
Batch Training Job (Spark or Mesos)
Training Algo
Trained Models
Deploy
Data Prep Job
Spark / SQL
Outcomes (Training Set)
Features
Batch Predict Job (Spark)
Trained Model
Predictions
To Hive & Kafka
Sampled Predictions
Performance Monitor Job (Spark)
Metrics
To Monitor System
BATCH

DSL for Feature Engineering

- Pure function expressions for
  - Feature selection
  - Feature transformations (for derived and composite features)
- Standard set of accessor functions for
  - Feature store
  - Basis features
  - Column stats (min, max, mean, std-dev, etc)
- Standard transformation functions + UDFs
- Examples
  - @palette:store:orders:prep_time_avg_1week:rs_uuid
  - nFill(@basis:distance, mean(@basis:distance))



Data Pipeline for Predictions



Pain Points

- Users have to learn DSL for feature engineering
    - Waste of data scientist time
- ETL and Feature extraction could last for hours
    - Waste of data scientist time
- Features dump onto disk several times
    - Waste of data scientist time
- Feature store and data lake are separate
    - Waste of hardware resource
- Machine Learning and Data Analytics are separate
    - Waste of hardware resource and software resource

Ideally, Feature Engineering should ...

- Users do not need to learn anything new
- User do not need to worry about implementation details
- Machine Learning jobs are running as fast
- Features stored in Data Lake
- Features are cached in memory for repeated processing
- Machine Learning pipelines and Data Analytics share the same hardware, and maybe software?

Try Declarative Language

- SQL extensible language
- Users no need to worry about implementation details
- Feature store resides in data lake
- Data kept in memory for repeated processing
- Machine Learning and Data Analytics
  - share the same hardware
  - share the same software
  - Streaming, Pipelined, Parallel, Vectorized execution

The Data Model

- All datasets are tables
  - Training datasets
  - Validation datasets
- Labels and Features are columns
  - Label is integer
  - Feature is map<integer, float>
- Models are SQL extensible types
  - <size>:<model>
  - Model: Variable length serialized type
- Classify, Regress are SQL extensible functions

The Execution Model

- Data are read from Hadoop once
- Data are kept in memory for repeated processing
- Streaming, Pipelined, Parallel, Vectorized execution
- Machine Learning Functions are implemented as:
    - Map
        - Applies to each input row
    - Combine
        - Merge partial aggregations
    - Reduce
        - Final aggregation

In Summary

- Users use extensible SQLs to train and predict models
- All data are modeled as tables
- All data stored in data lake
- Data read from disk once, kept in memory for repeated processing
- A framework with execution primitives, to support:
    - SQL Analytics
    - Machine Learning

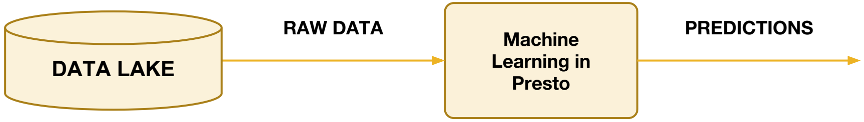
What does it look like?

```sql
SELECT evaluate_classifier_predictions(label, classify(features, model))
FROM (
    SELECT learn_classifier(label, features) AS model
    FROM training_data
)
CROSS JOIN validation_data
```

The Presto Machine Learning Plugin

- All data are modeled as tables
- Data read from disk just once
  - Data are kept in memory for repeated processing
  - Streaming, Pipelined, Parallel, Vectorized execution
- Presto Connectors
  - SQL or Machine Learning on any data, e.g. Hadoop, MySQL, Elasticsearch, Cassandra, MongoDB, etc.
- Presto provides execution primitives:
  - Input Function -- Map
  - Combine Function -- Combine
  - Output function -- Reduce

Significantly Reduce Complexity



- Data Scientists just use SQL for Machine Learning
- Model development time speed up > 5X
- Machine Learning pipeline speeds up > 10X
- Shared data lake
- Shared resource management
- Shared primitive executions

What is Presto: Interactive SQL Engine for Big Data

Interactive query speeds

Horizontally scalable

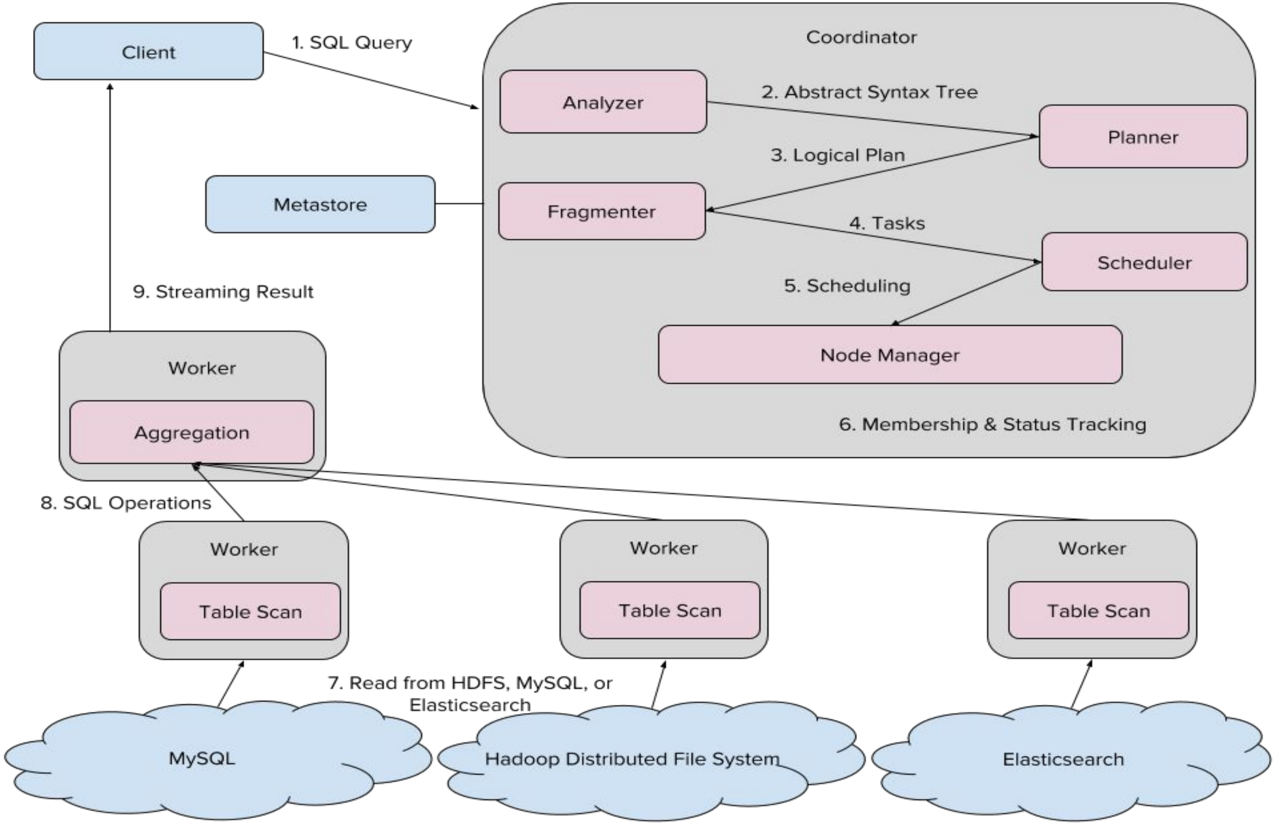
ANSI SQL, Extensible for Machine Learning

Battle-tested by Facebook, Uber, & Netflix

Completely open source

Access to petabytes of data in the Hadoop data lake

How Presto Works



Why Presto is Fast

- Data in memory during execution
- Pipelining and streaming
- Columnar storage & execution
- Bytecode generation
    - Inline virtual function calls
    - Inline constants
    - Rewrite inner loops
    - Rewrite type-specific branches

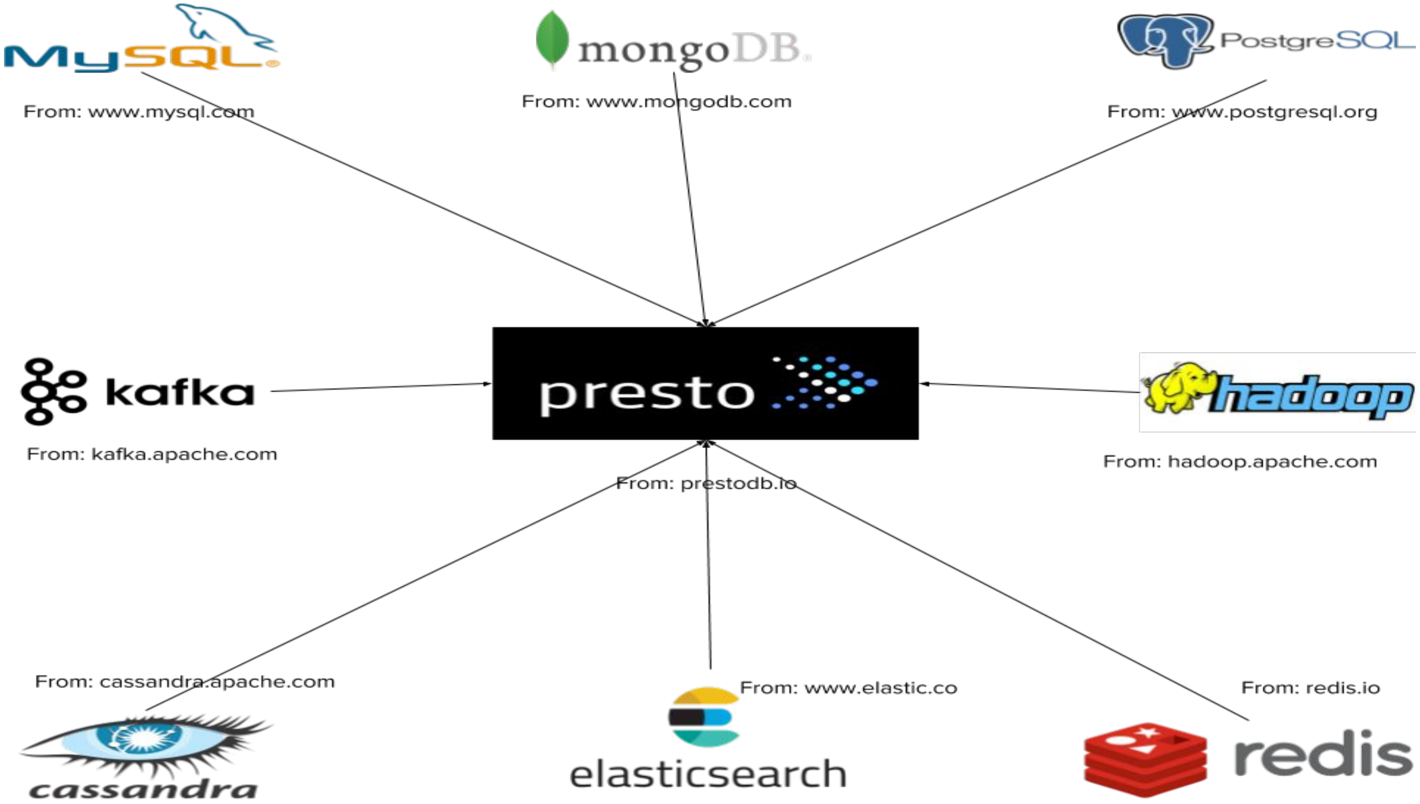
Scale of Presto @ Uber

- 2 clusters
    - Application cluster
        - Hundreds of machines
        - 100K queries per day
        - **P90: 30s**
    - Ad hoc cluster
        - Hundreds of machines
        - 100K queries per day
        - **P90: 60s**
- Access to both raw and model tables
    - 5 petabytes of data
- Total **200K+** queries per day

Resource Management

- Presto has its own resource manager
  - Not on YARN
  - Not on Mesos
- CPU Management
  - Priority queues
  - Short running queries higher priority
- Memory Management
  - Max memory per query per node
  - If query exceeds max memory limit, query fails
  - No OutOfMemory in Presto process

Presto Connectors: No Need to Copy Data



Presto Plugin

- Presto Connectors
  - ConnectorMetadata
  - ConnectorSplitManager
  - ConnectorSplit
  - ConnectorRecordCursor
- Extensible Types
  - Native Container Type
  - Type Signature
- Extensible Functions
  - Input Function, Combine Function, Output Function

Presto Machine Learning Plugin

- Types
  - Model Type : variable length type
  - Classifier Parametric Type
- Functions
  - Classify
  - Regress
  - Features
  - Learn_libsvm_regressor
  - Learn_libsvm_classifier
  - Evaluate_classifier_predictions

Presto Ongoing Work

- Machine Learning support in Presto
- Presto Elasticsearch Connector
- Cost based optimizer
- Spill to disk
- Multi-tenancy Support
- Authentication and Authorization

We are Hiring

https://www.uber.com/careers/list/27366/

Send resumes to:

luoz@uber.com

Thank you

Interested in learning more about Uber Eng?
Eng.uber.com
Follow us on Twitter:
@UberEng

UBER