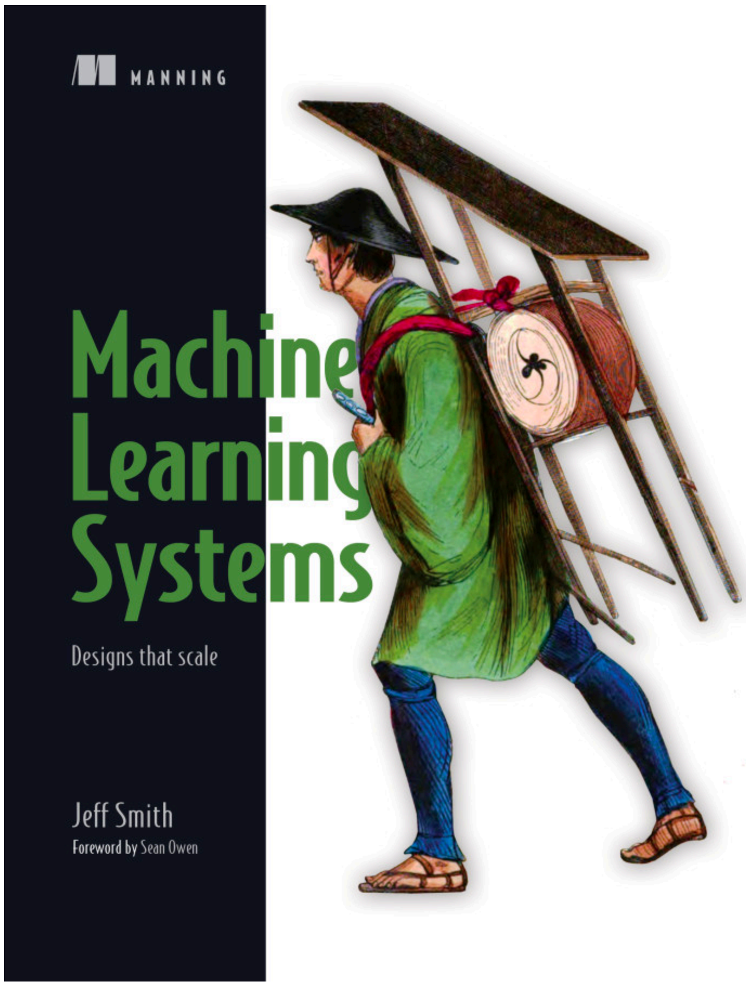
Machine Learning Platforms

Pooyan Jamshidi
USC

Learning goals

- Review principal components of an ML platform.
- Identify key challenges of scaling an ML pipeline to a large number of heterogeneous models.
- Define solutions that scale model training and serving.
- Characterize system properties and how to build a production-ready ML pipeline that guarantee reliability, resiliency, responsiveness, and elasticity.

Main sources



UBER Engineering

THE NETFLIX TECH BLOG

Spotify Labs

Uber Data

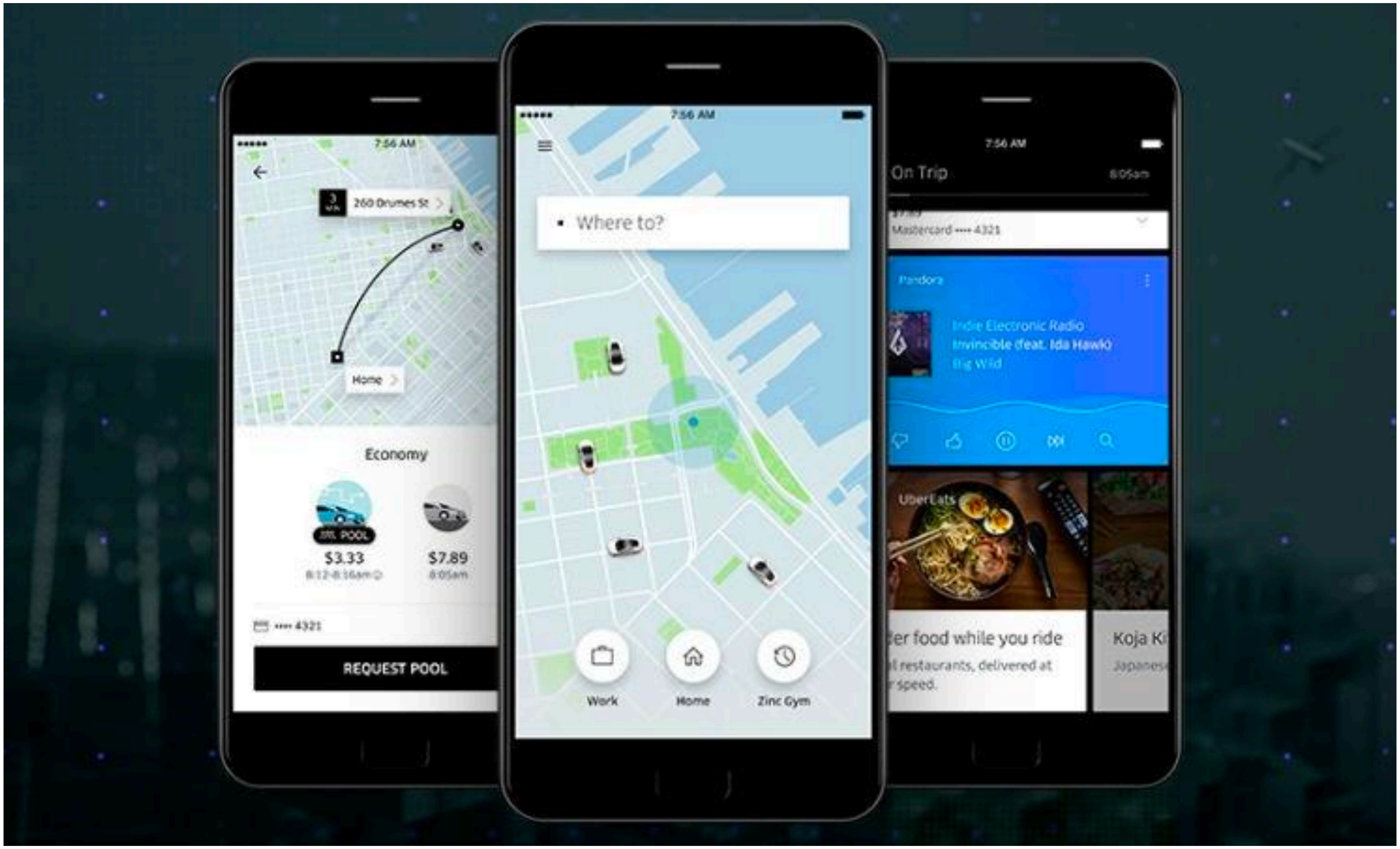
Meet Michelangelo: Uber’s Machine Learning Platform

September 5, 2017

Search

64 6K 6K 4 194 G+ 6K SHARES

Categories



Mission

Enable engineers and data scientists across the company to easily build and deploy machine learning solutions at scale.

How Uber uses ML?

In what business activities?

ML at Uber

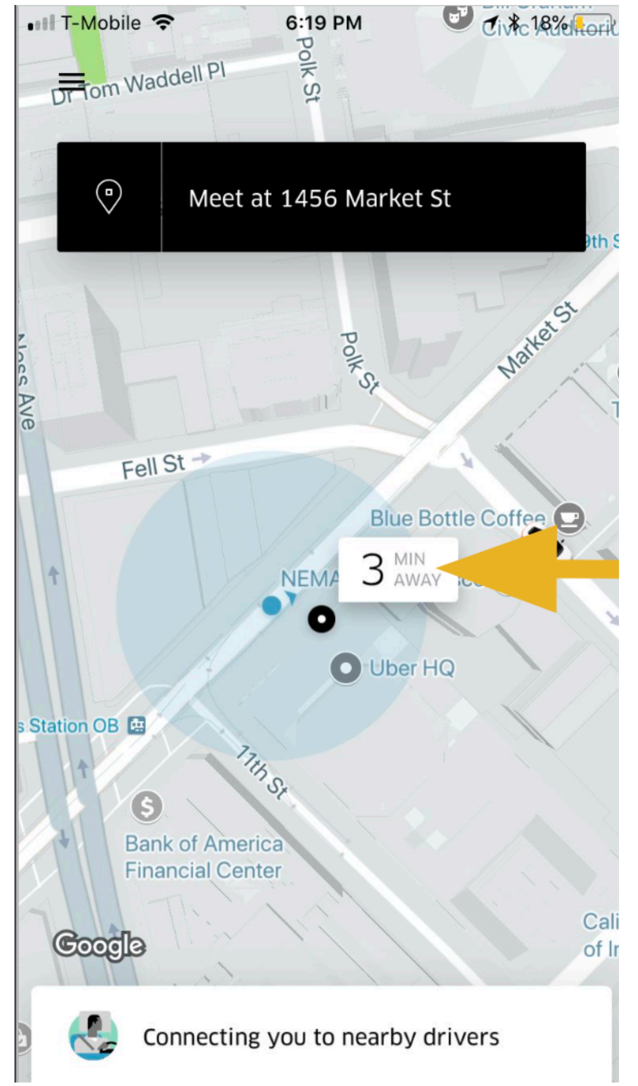
- Uber Eats
- ETAs
- Autonomous Cars
- Customer Support
- Dispatch
- Personalization
- Demand Modeling
- Dynamic Pricing

ML at Uber

- Forecasting
- Maps
- Fraud
- Destination Predictions
- Anomaly Detection
- Capacity Planning
- And many more...

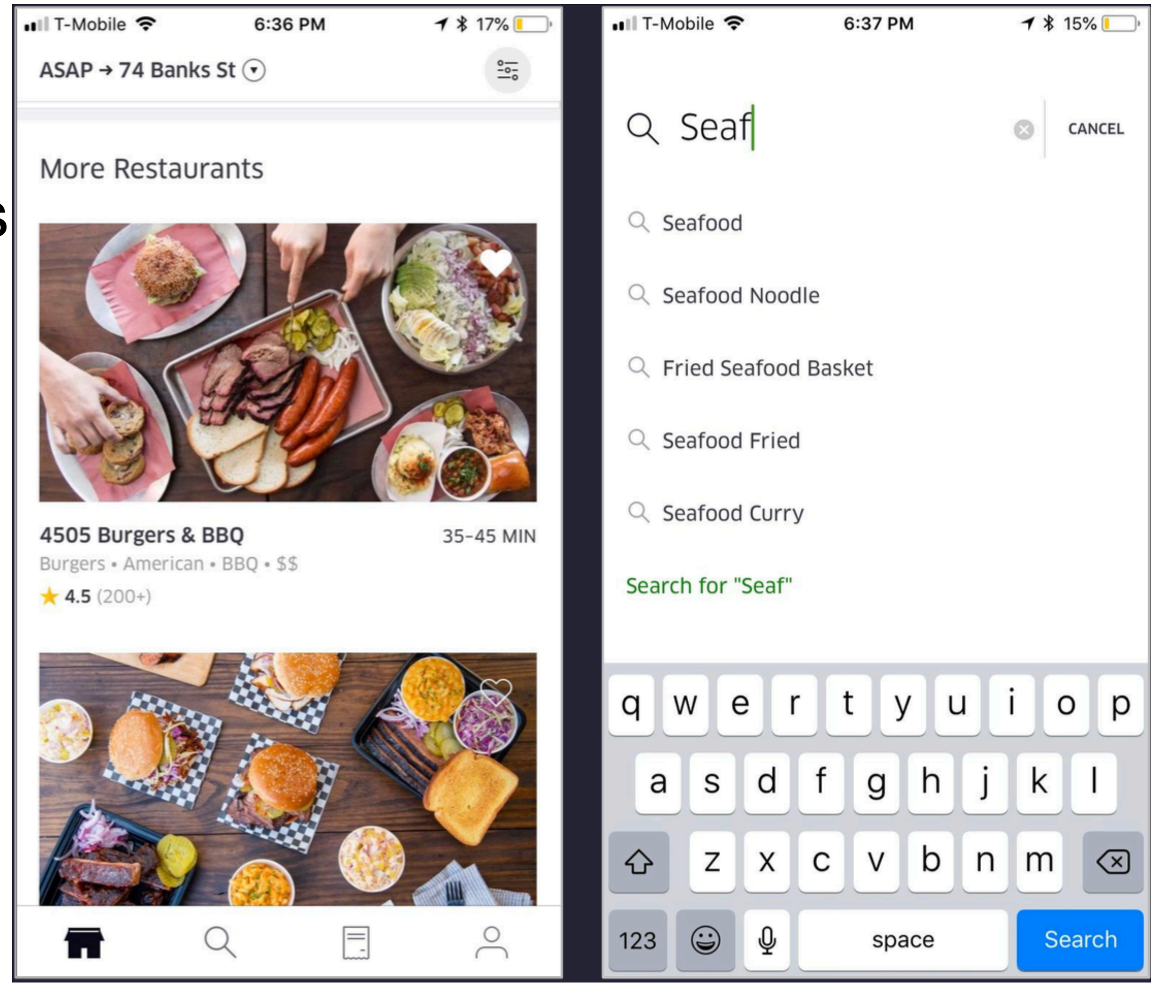
ML at Uber - ETAs

- ETAs are core to customer experience and used by many internal systems
- ETA are generated by route-based algorithms
- It is often incorrect - but it's incorrect in predictable ways
- ML model predicts the error
- Use the predicted error to correct the ETA
- ETAs now dramatically more accurate

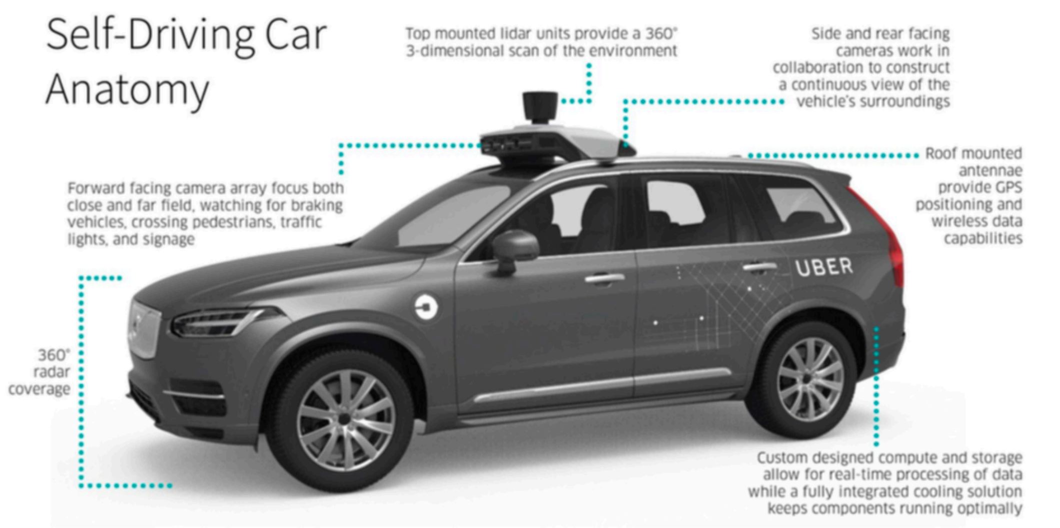


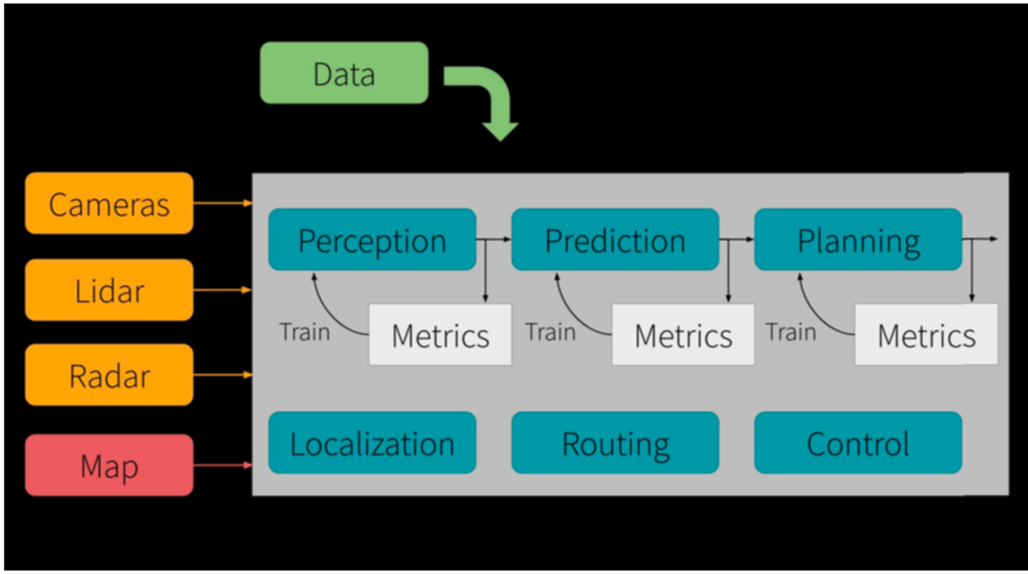
ML at Uber - Eats

- Models used for
  - Ranking of restaurants and dishes
  - Delivery times
  - Search ranking
- 100s of ML models called to render Eats homepage



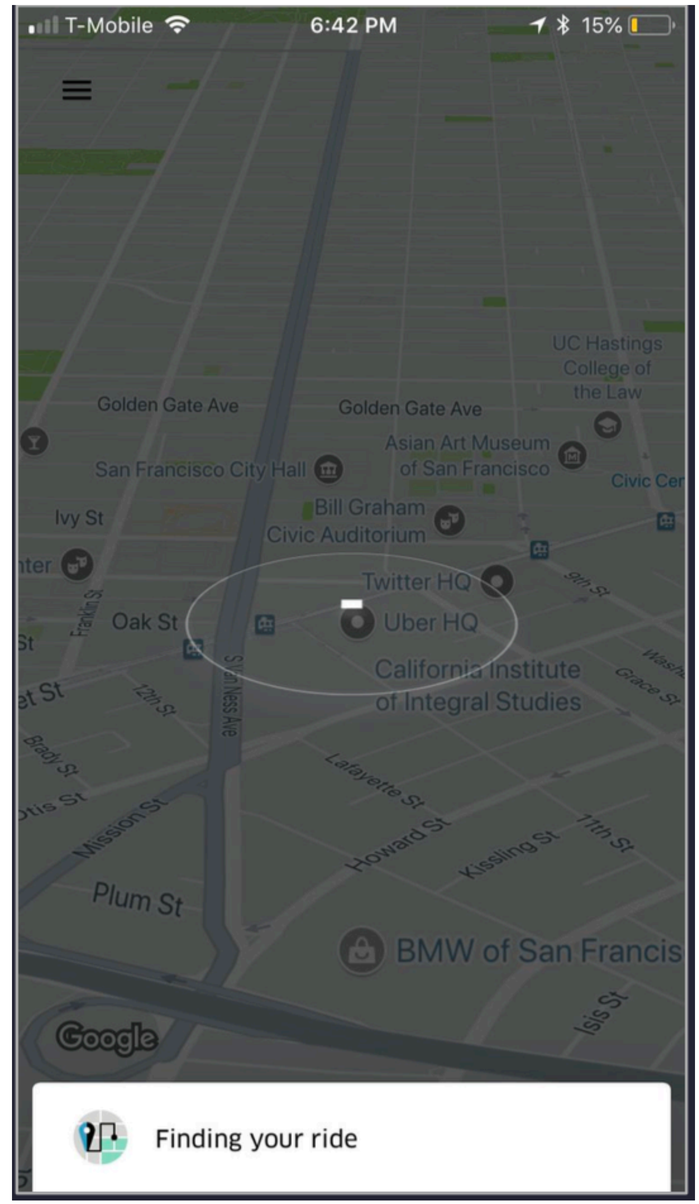
ML at Uber - Autonomous Cars



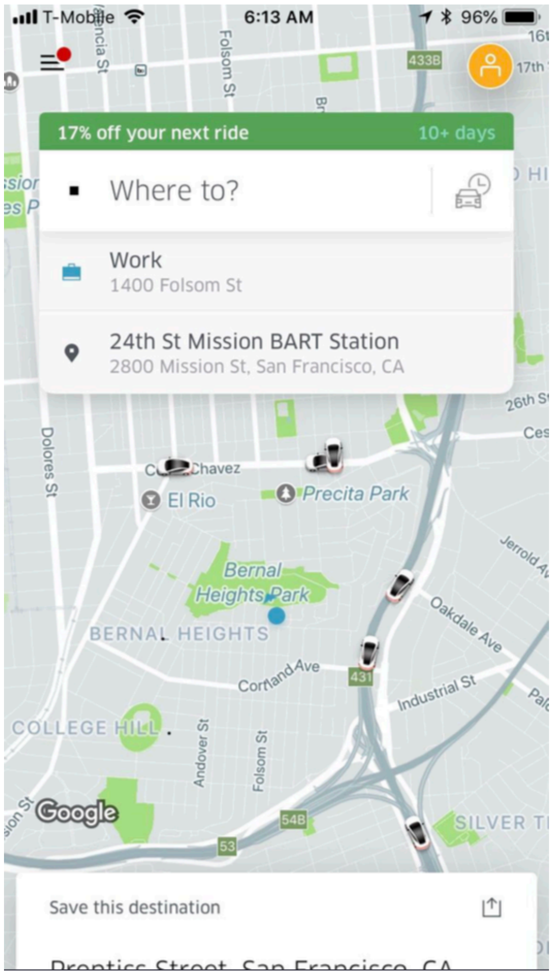


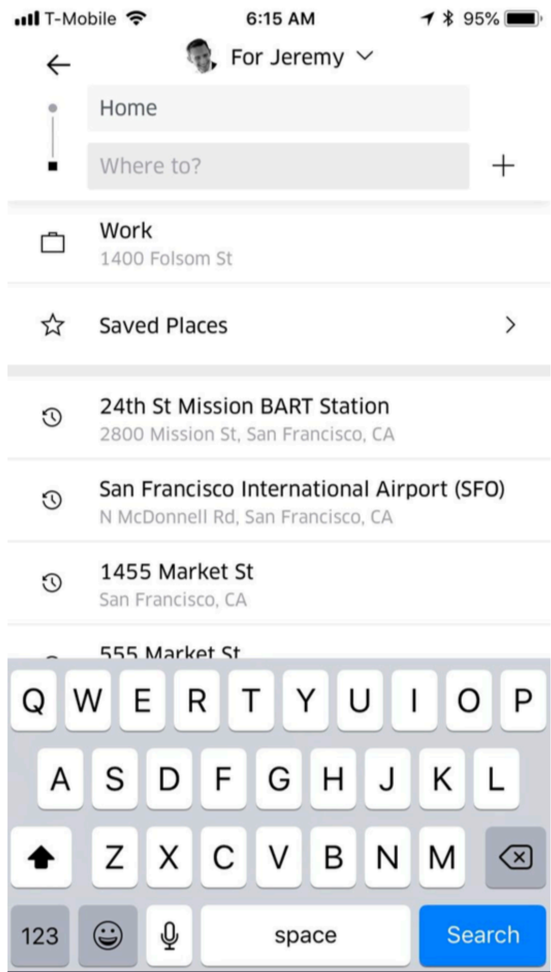
ML at Uber - Dispatch

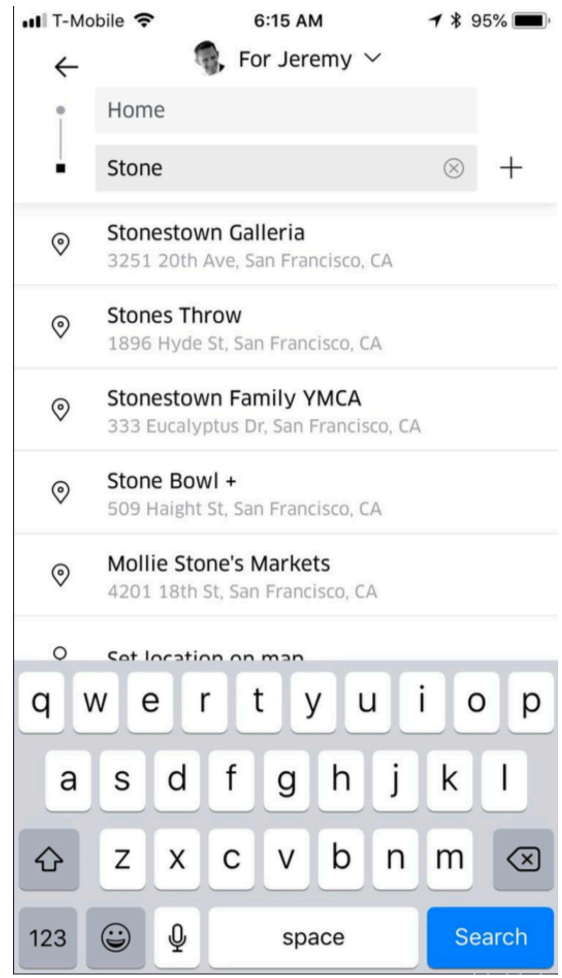
- Optimize matching of rider and driver
- Predict if open rider app will make trip request



ML at Uber - Destination Prediction

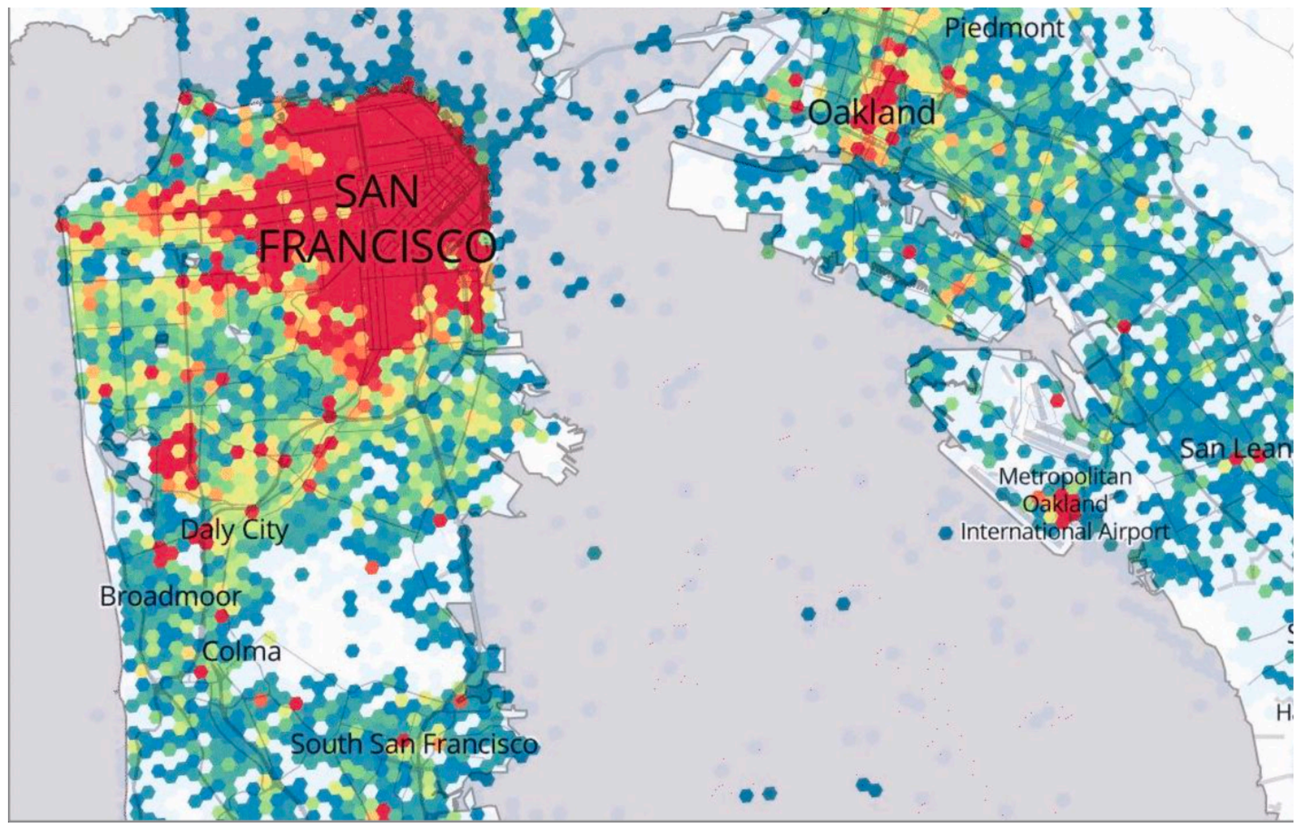






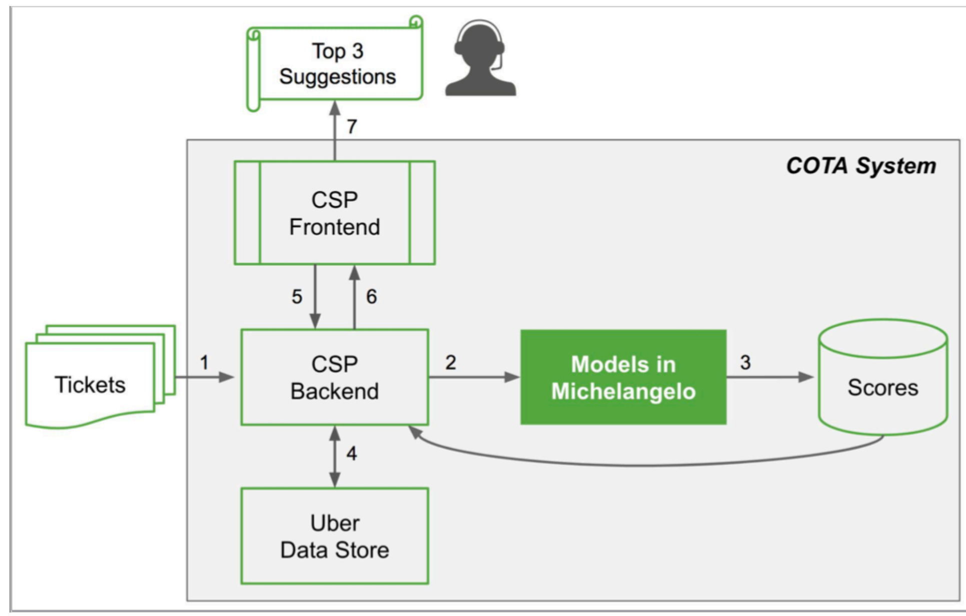
ML at Uber - Spatiotemporal Forecasting

- Supply
  - Available Drivers
- Demand
  - Open Apps
- Other
  - Request Times
  - Arrival Times
  - Airport Demand



ML at Uber - Customer Support

- 5 customer-agent communication channels
- Hundreds of thousands of tickets surfacing daily on the platform across 400+ cities
- NLP models classify tickets and suggest response templates
- Reduce ticket resolution time by 10%+ with same or higher CSAT

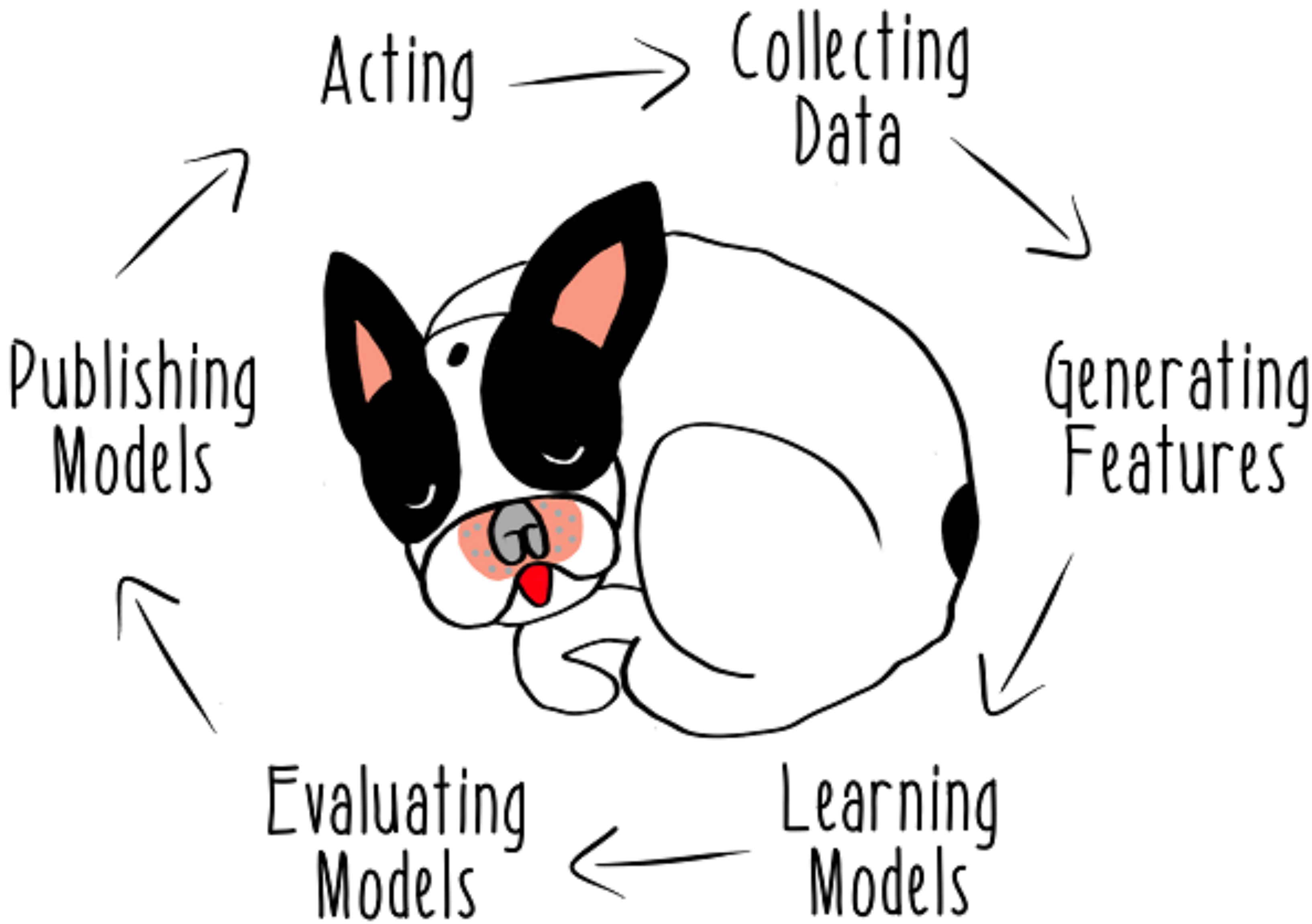


Why build an ML platform?

Motivation behind Michelangelo

- Early **challenges** with machine learning
  - Limited scale with Python and R
  - Pipelines not reliable or reproducible
  - Many one-off production systems for serving
- **Goals** of platform
  - Standardize workflows and tools
  - Provide scalable support for end-to-end ML workflow
  - Democratize and accelerate machine learning through ease of use

ML Pipeline


Acting
Collecting Data
Generating Features
Learning Models
Evaluating Models
Publishing Models

Key Platform Components

Key Components: Feature Store & Feature Engineering

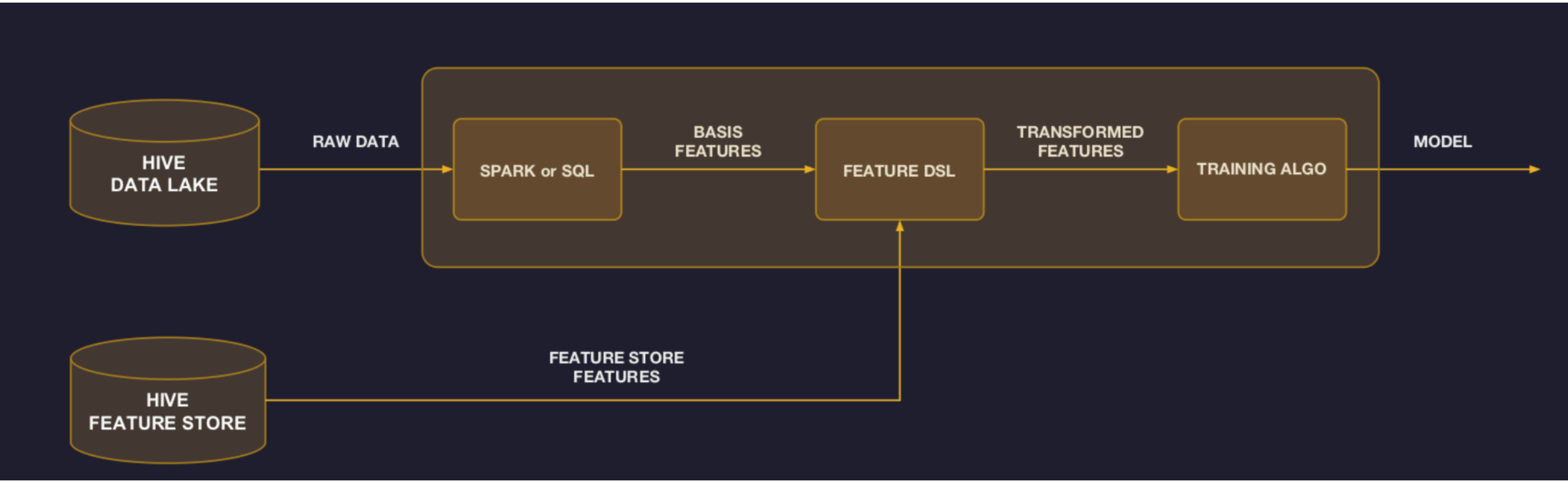
Feature Store (aka Palette)

- Problem
  - Hardest part of ML is finding good features
  - Same features are often used by different models built by different teams
- Solution
  - Centralized feature store for collecting and sharing features
  - Platform team curates core set of widely applicable features
  - Modelers contribute more features as part of ongoing model building
  - Meta-data for each feature to track ownership, how computed, where used, etc
  - Modelers select features by name & join key. Offline & online pipelines auto-configured

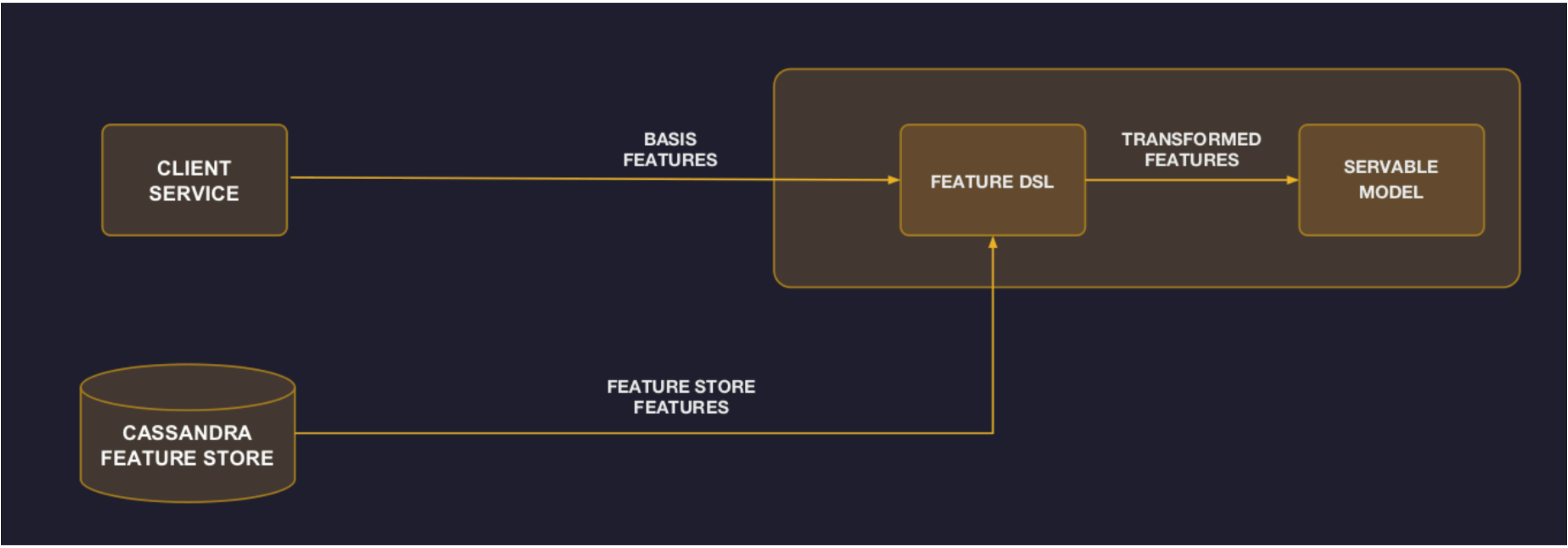
Functionality of feature store

- It allows users to easily add features they have built into a shared feature store.
- They are very easy to consume, both online and offline, by referencing a feature's simple canonical name in the model configuration.

Pipeline for offline training with Feature Store



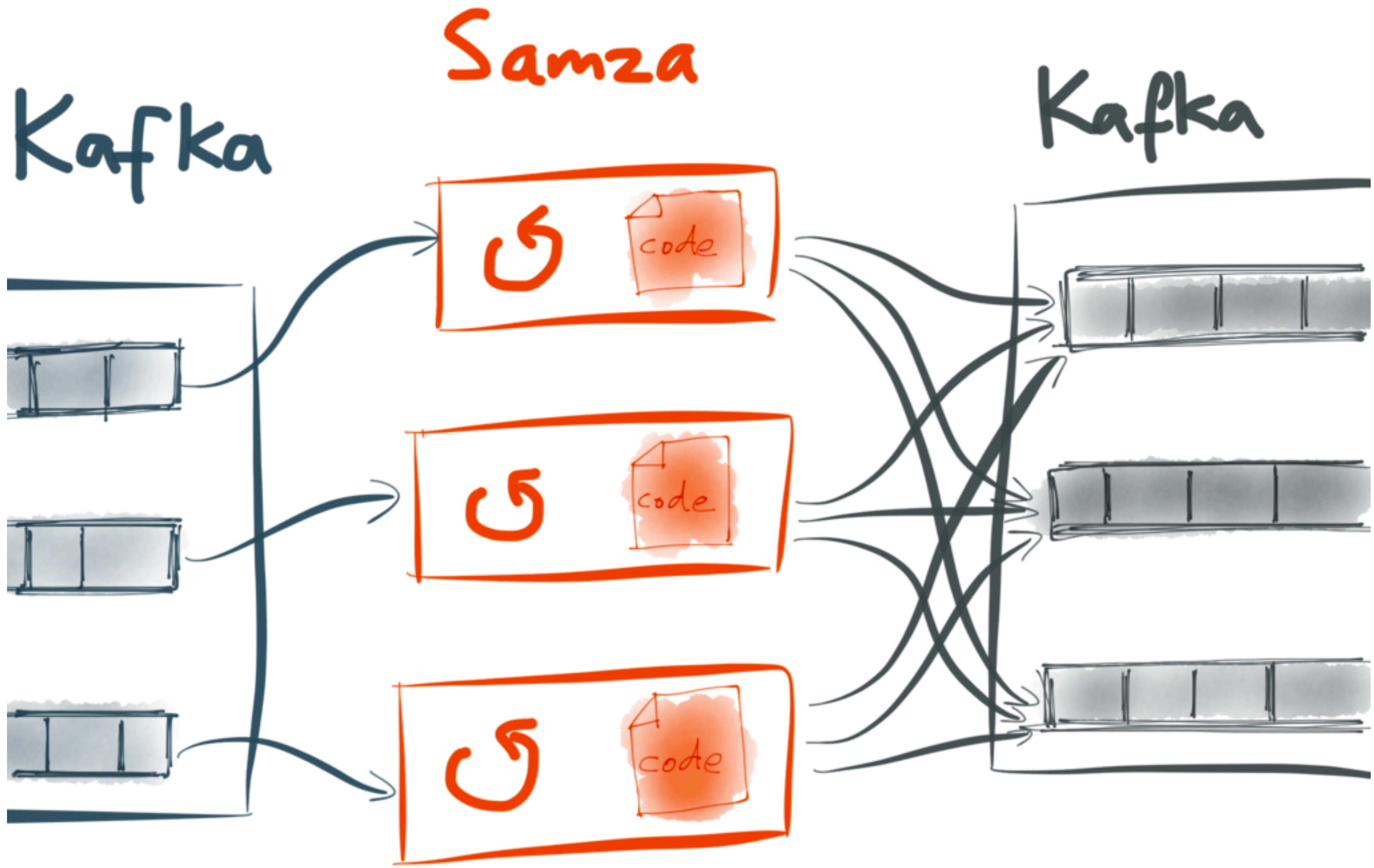
Pipeline for online serving with Feature Store



Options for computing online-served features

- Batch precompute:
  - To conduct bulk precomputing and loading historical features from HDFS into Cassandra on a regular basis.
  - ‘restaurant’s average meal preparation time over the last seven days.’
- Near-real-time compute:
  - Publish relevant metrics to Kafka and then run Samza-based streaming compute jobs to generate aggregate features at low latency. These features are then written directly to Cassandra for serving and logged back to HDFS for future training jobs.
  - ‘restaurant’s average meal preparation time over the last one hour.’

Apache Kafka? Apache Samza?



Domain specific language for feature selection and transformation

- Often the features generated by data pipelines or sent from a client service are not in the **proper format** for the model, and they may be missing values that need to be filled.

- Moreover, the model may only need a **subset** of features provided.

- In some cases, it may be more useful for the model to transform a timestamp into an **hour-of-day or day-of-week** to better capture seasonal patterns.

- In other cases, feature values may need to be **normalized** (e.g., subtract the mean and divide by standard deviation).

Domain specific language for feature selection and transformation

- To select, transform, and combine the features that are sent to the model at training and prediction times.

- The DSL is implemented as sub-set of Scala.

- It is a pure functional language with a complete set of commonly used functions.

- It has the ability for customer teams to add their own user-defined functions.

- @palette:store:orders:prep_time_avg_1week:rs_uuid

Key Components:
Scalable Model Training

Distributed training of non-DL models

- Large-scale distributed training (billions of samples)
  - Decision trees
  - Linear and logistic models
  - Unsupervised learning
  - Time series forecasting
  - Hyperparameter search for all model types
- Smart pipeline management to balance speed and reliability
  - Fuse operators into single job for speed
  - Break operators into separate jobs to reliability

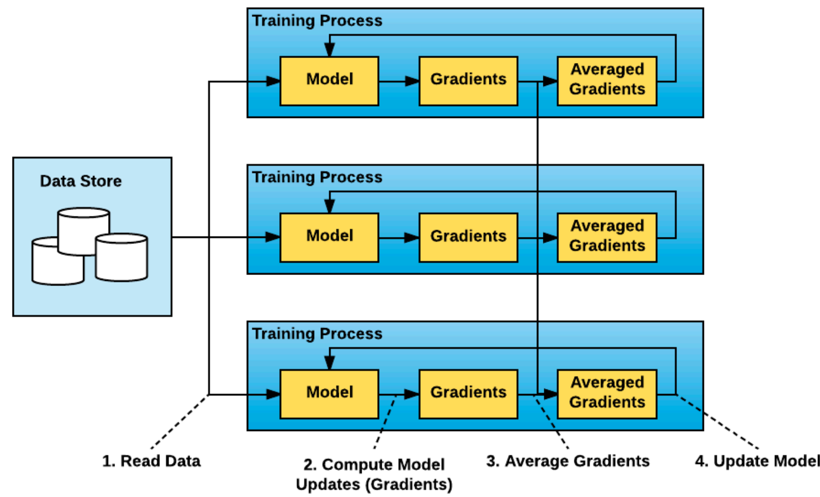
Distributed training of deep learning models with Horovod

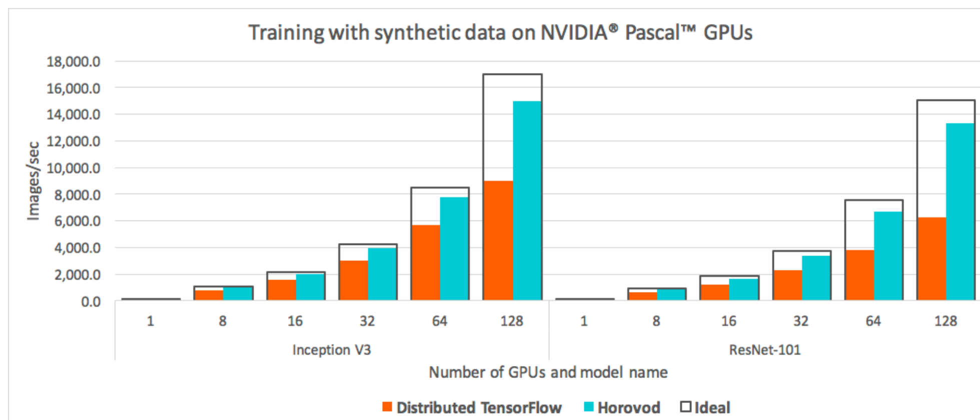
- Data-parallelism works best when model is small enough to fit on each GPU

- Ring-allreduce is more efficient than parameter servers for averaging weights

- Faster training and better GPU utilization

- Much simpler training scripts



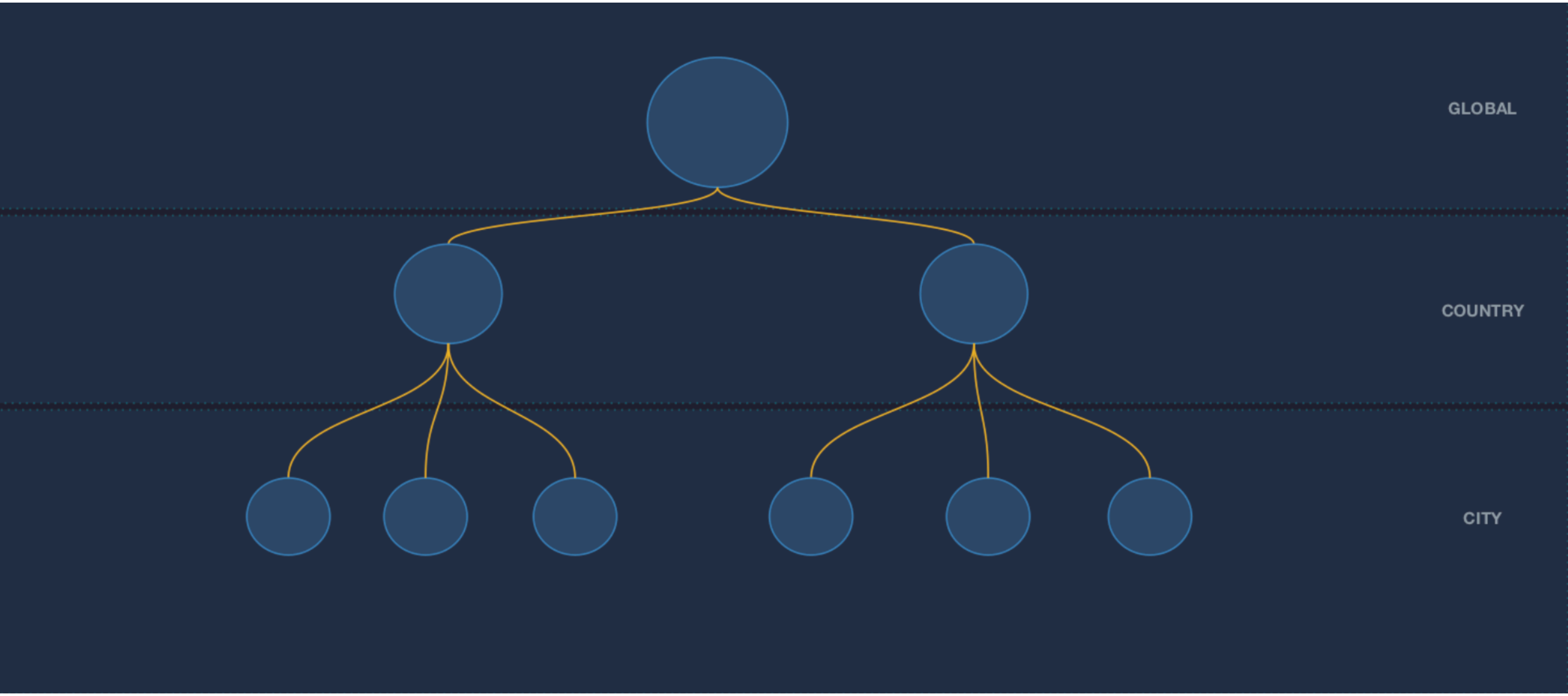


Key Components: Partitioned Models

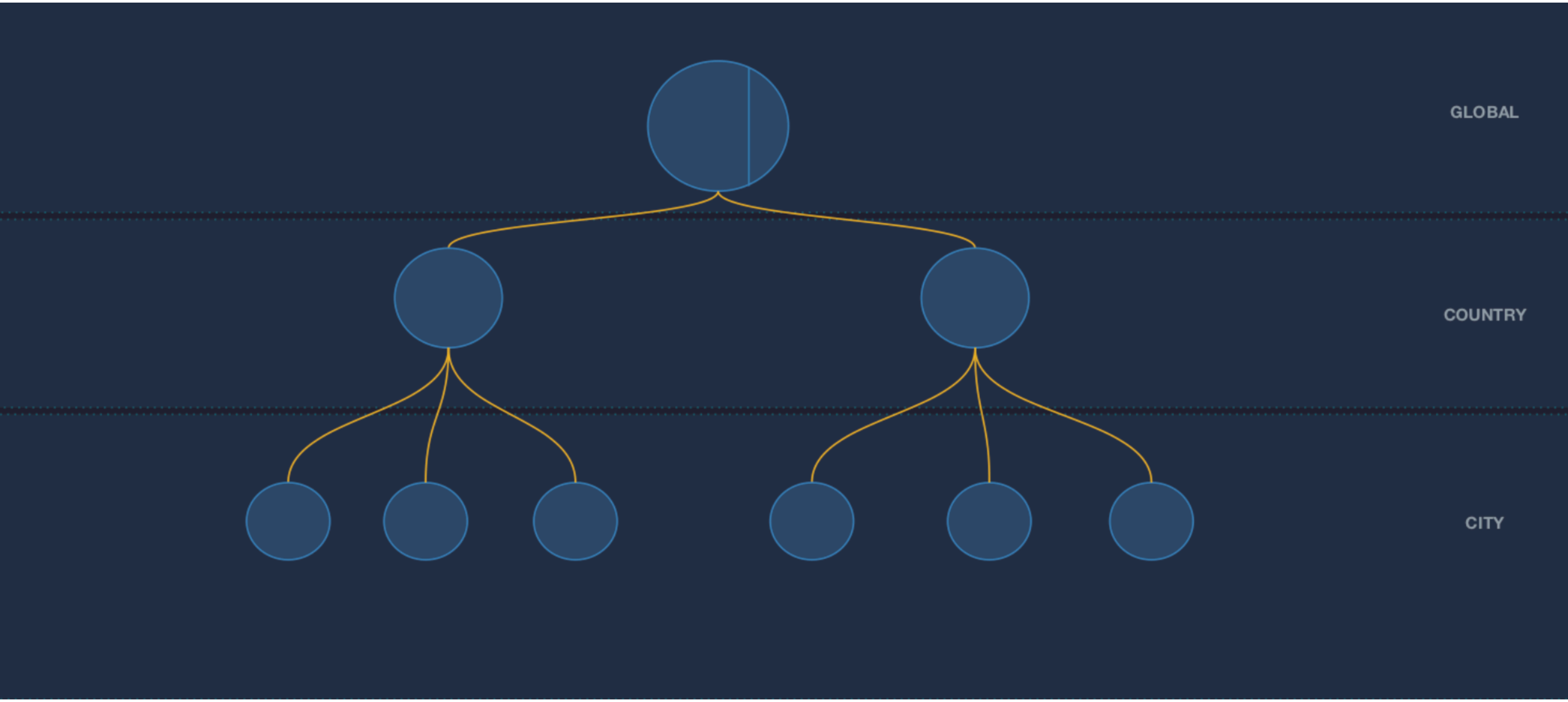
Partitioned models

- Problem
  - Often want to train a model per city or per product
  - Hard to train and deploy 100s or 1000s of individual models
- Solution
  - Let users define hierarchical partitioning scheme
  - Automatically train model per partition
  - Manage and deploy as single logical model

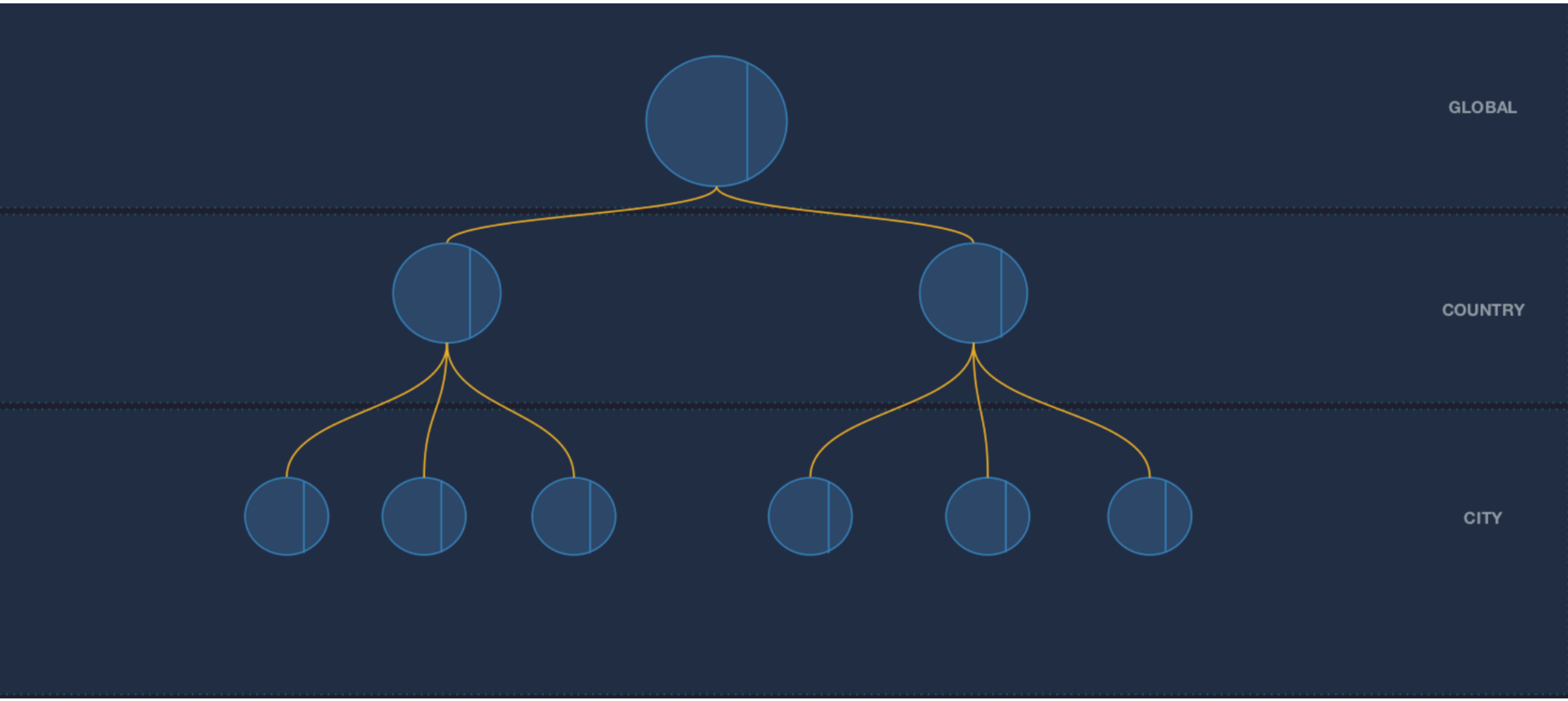
Define partition scheme



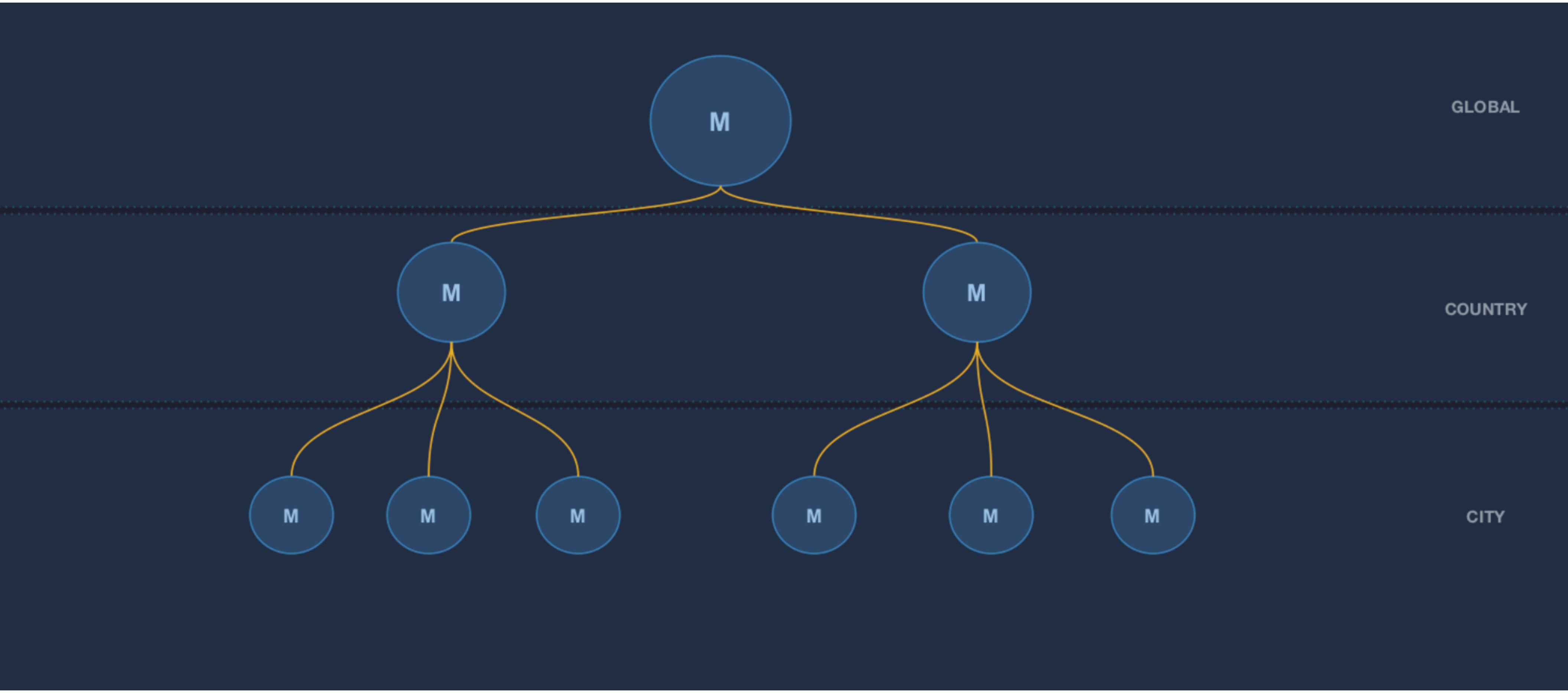
Make train / test split



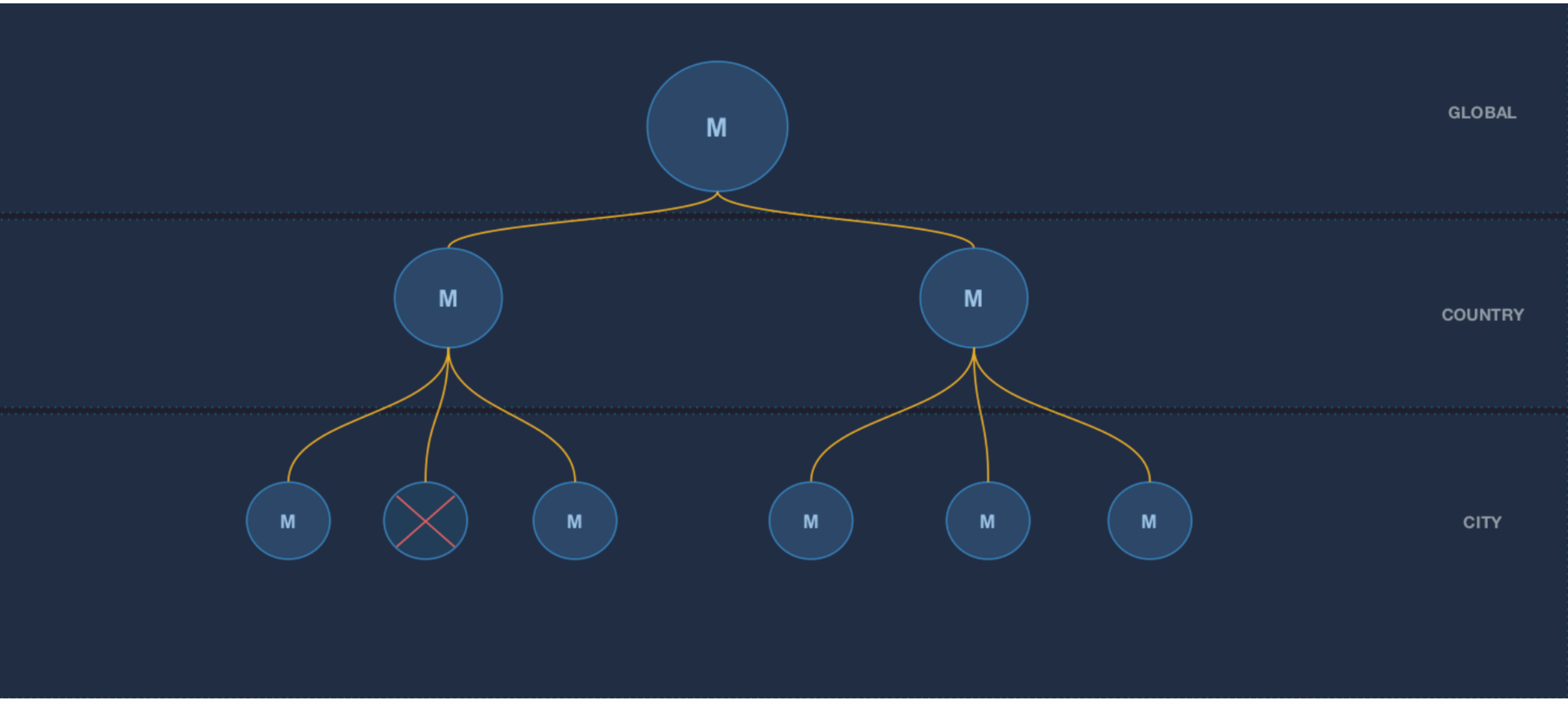
Keep same split and partition for each level



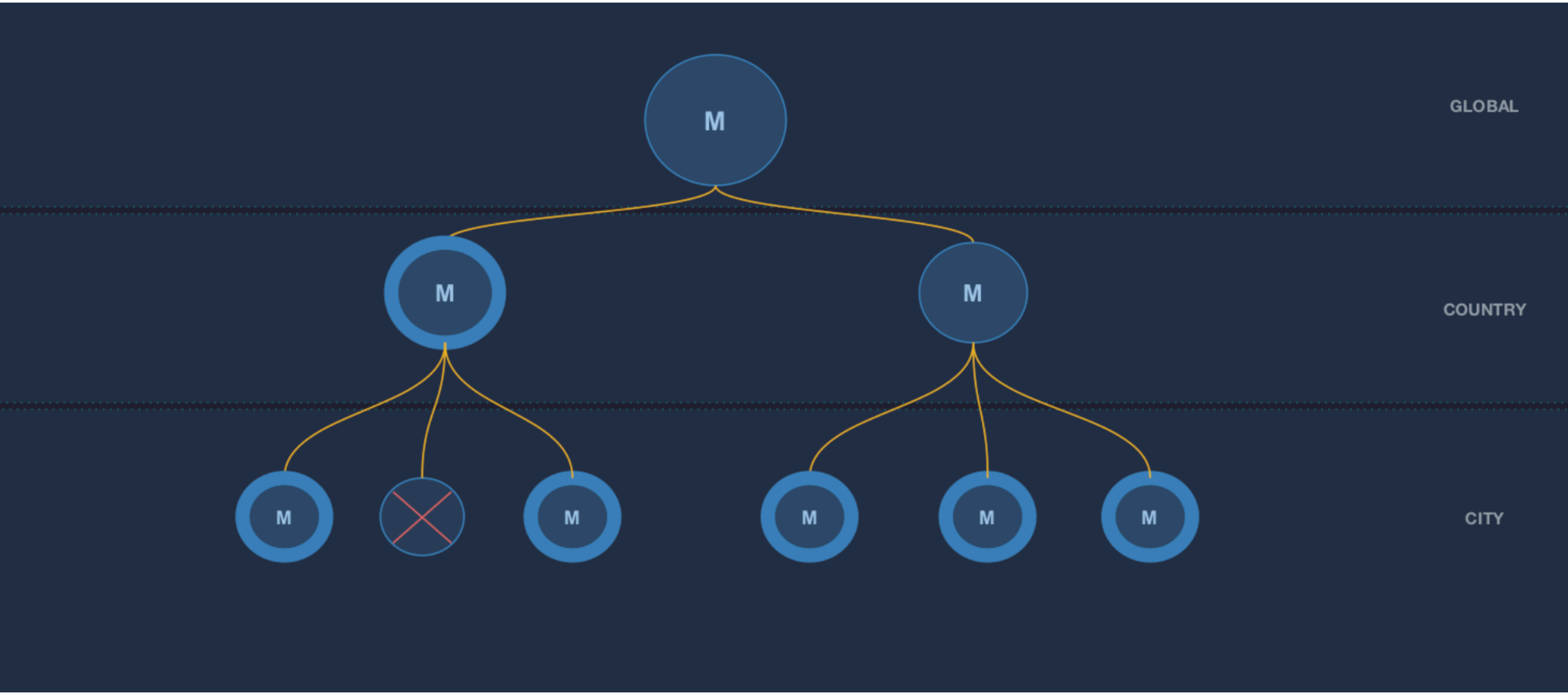
Train model for every node



Prune bad models



At serving time, route to best model for each node

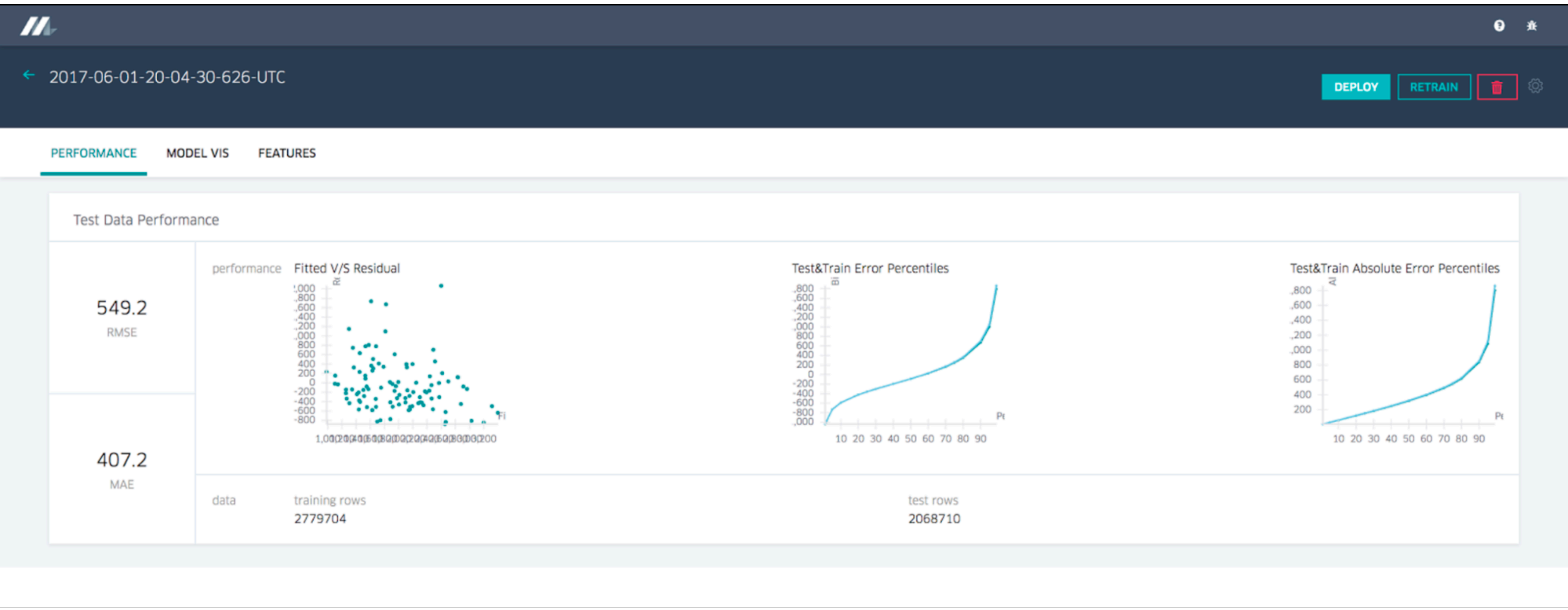


Key Components: Model Visualization

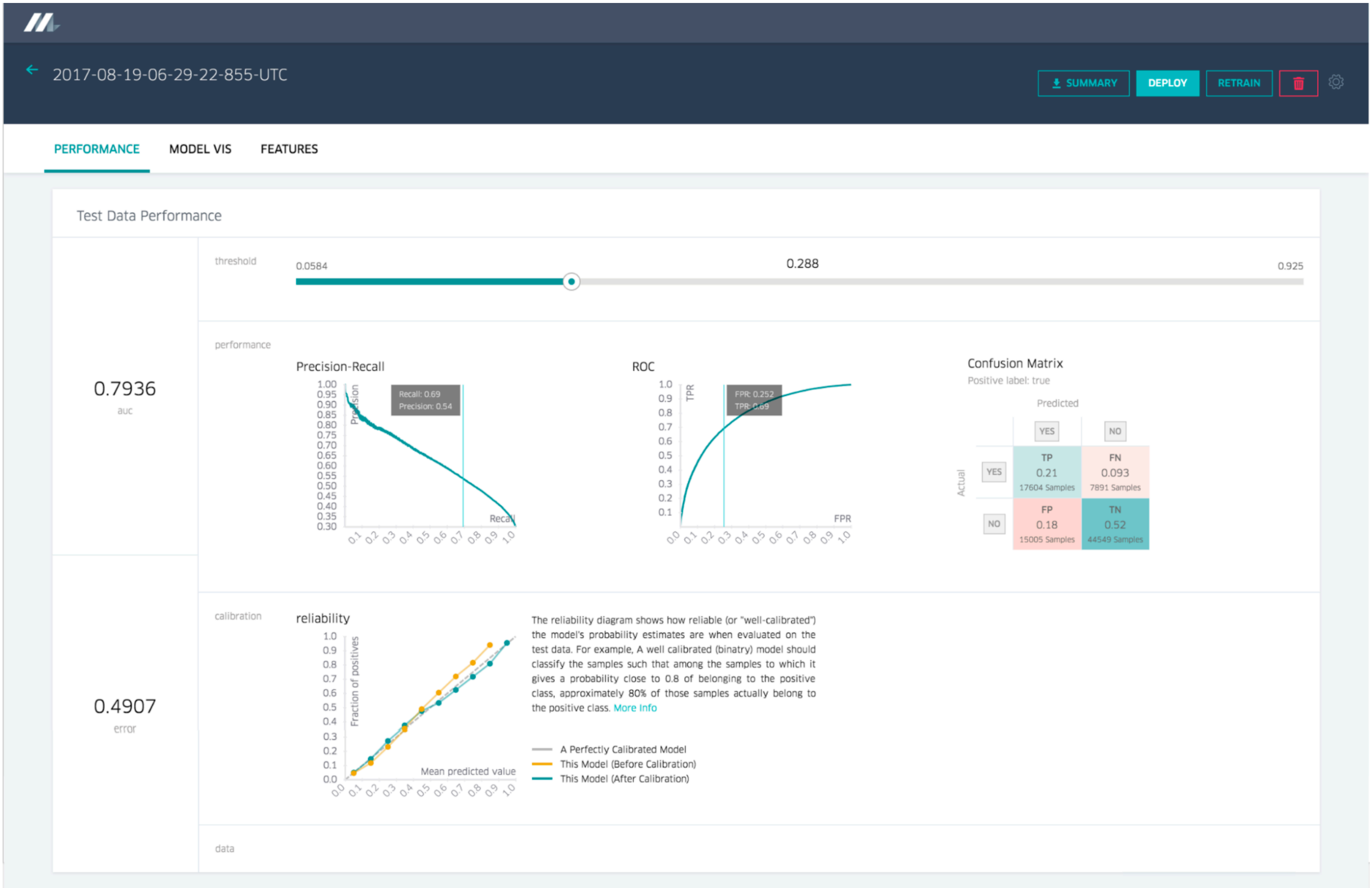
Evaluate models

- Problem
  - It takes many iterations to produce a good model
  - Keeping track of how a model was built is important
  - Evaluating and comparing models is hard
- With every trained model, we capture standard metadata and reports
  - Full model configuration, including train and test datasets
  - Training job metrics
  - Model accuracy metrics
  - Performance of model after deployment

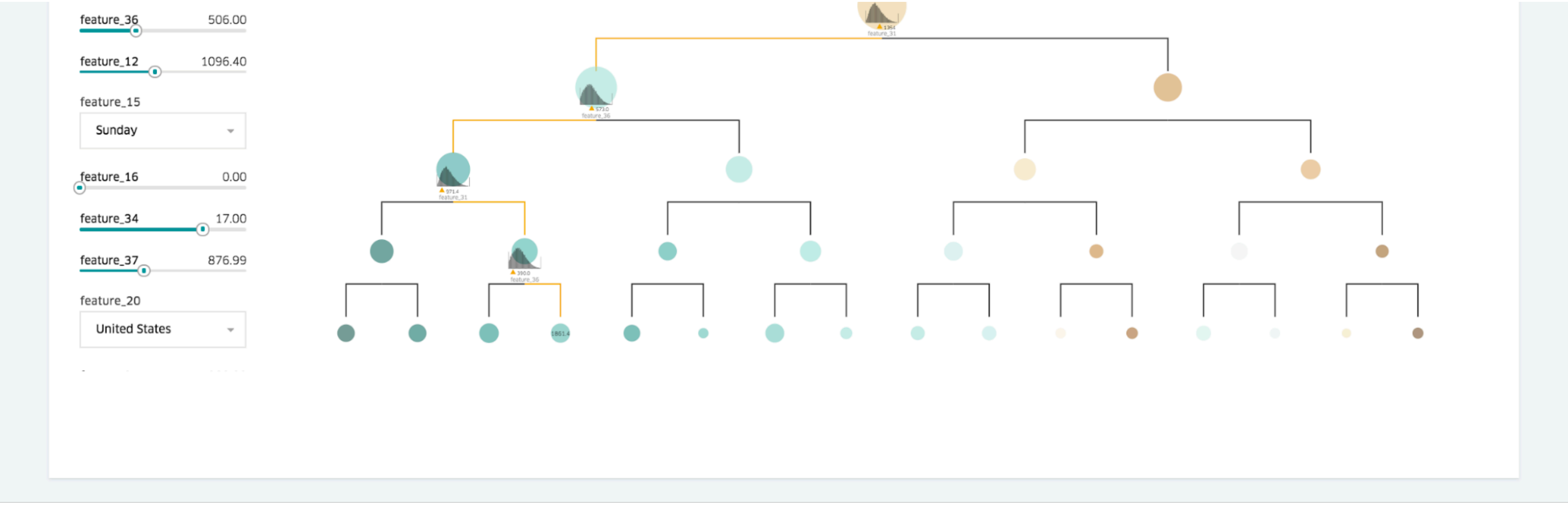
Model visualization - regression model



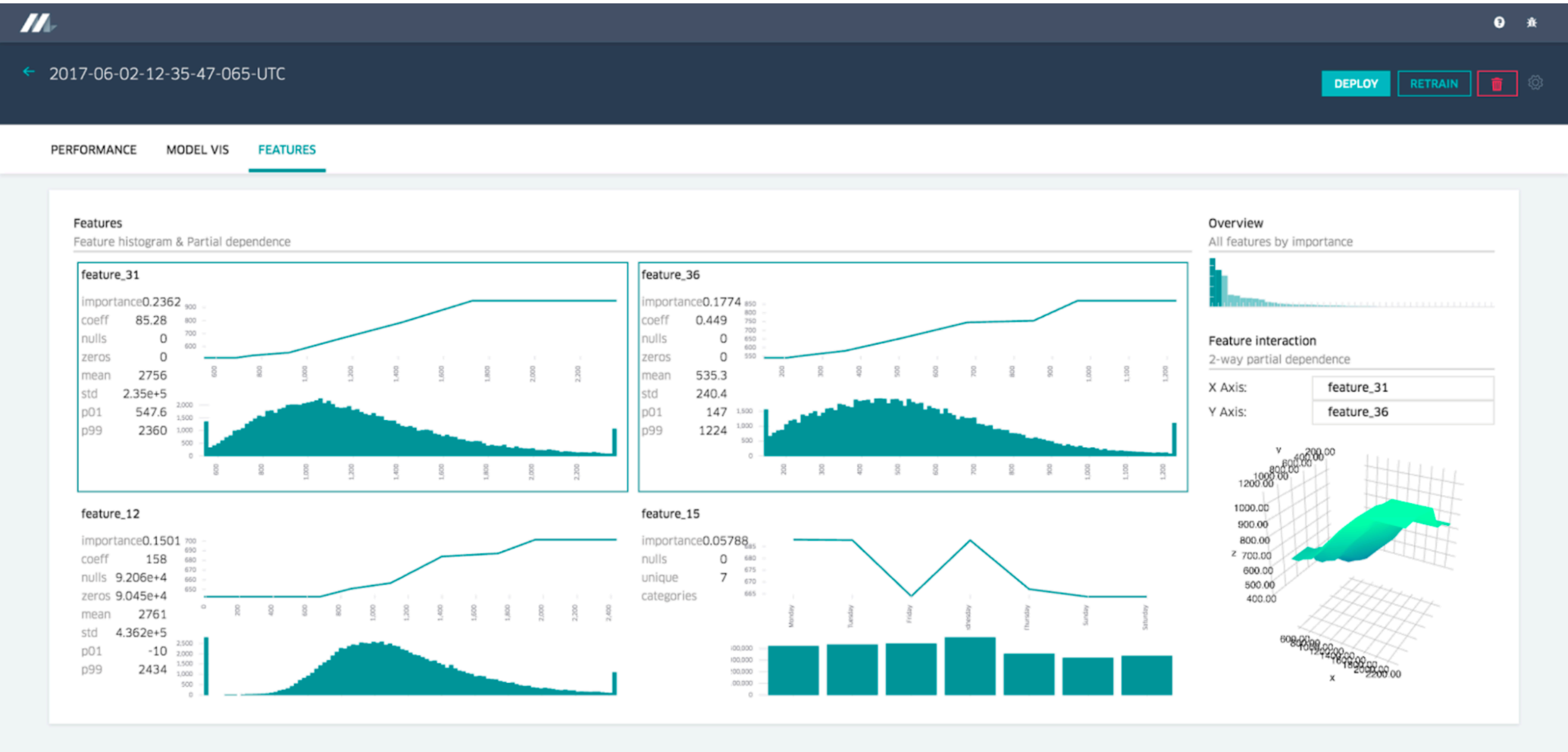
Model visualization - classification model



Model visualization - decision tree



Model visualization - feature report

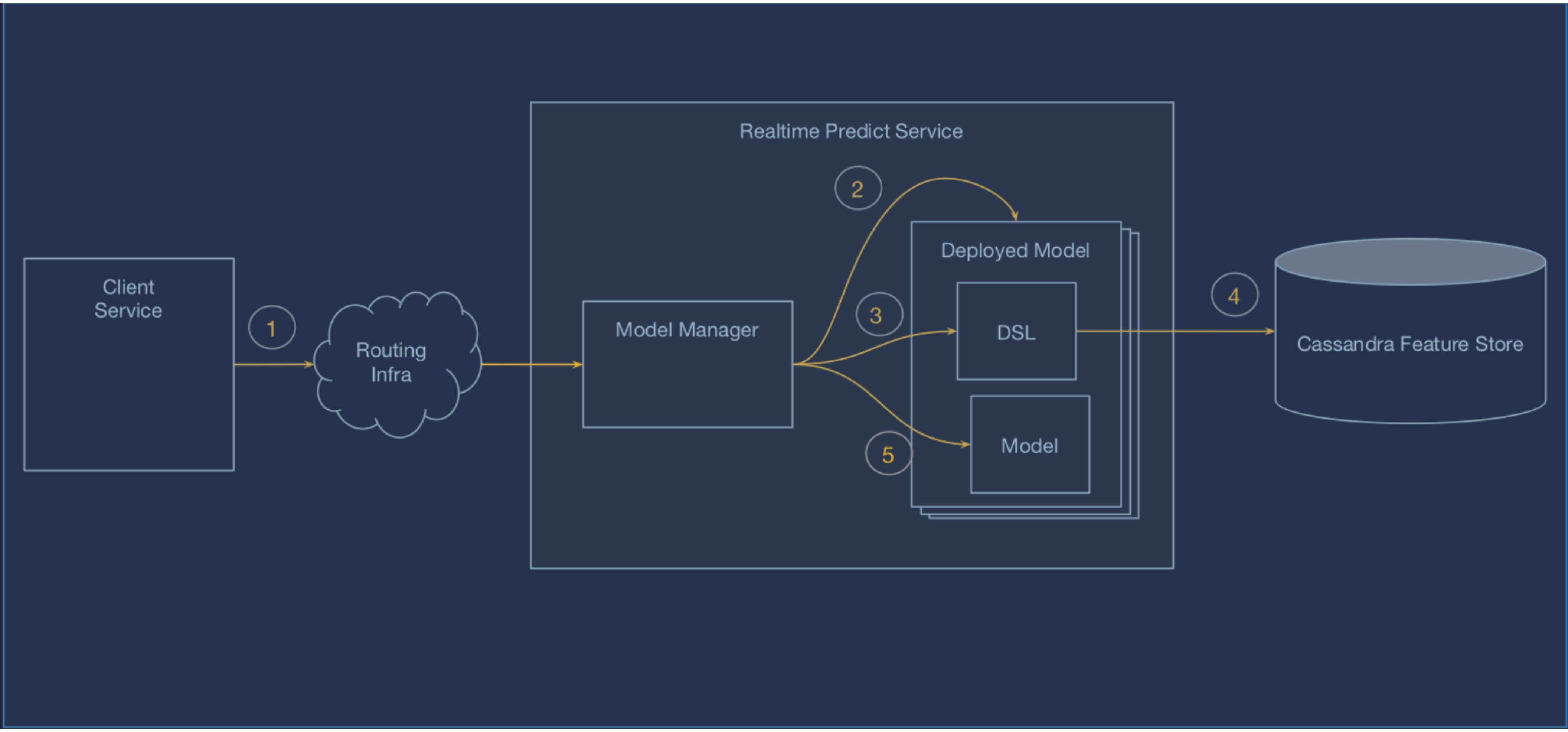


Key Components: Sharded Deployment and Serving

Online prediction service

- Prediction Service
  - Thrift service container for one or more models
  - Scale out in Docker on Mesos
  - Single or multi-tenant deployments
  - Connection management and batched / parallelized queries to Cassandra
  - Monitoring & alerting
- Deployment
  - Model & DSL packaged as JAR file
  - One click deploy across DCs via standard Uber deployment infrastructure
  - Health checks and rollback

Online prediction service



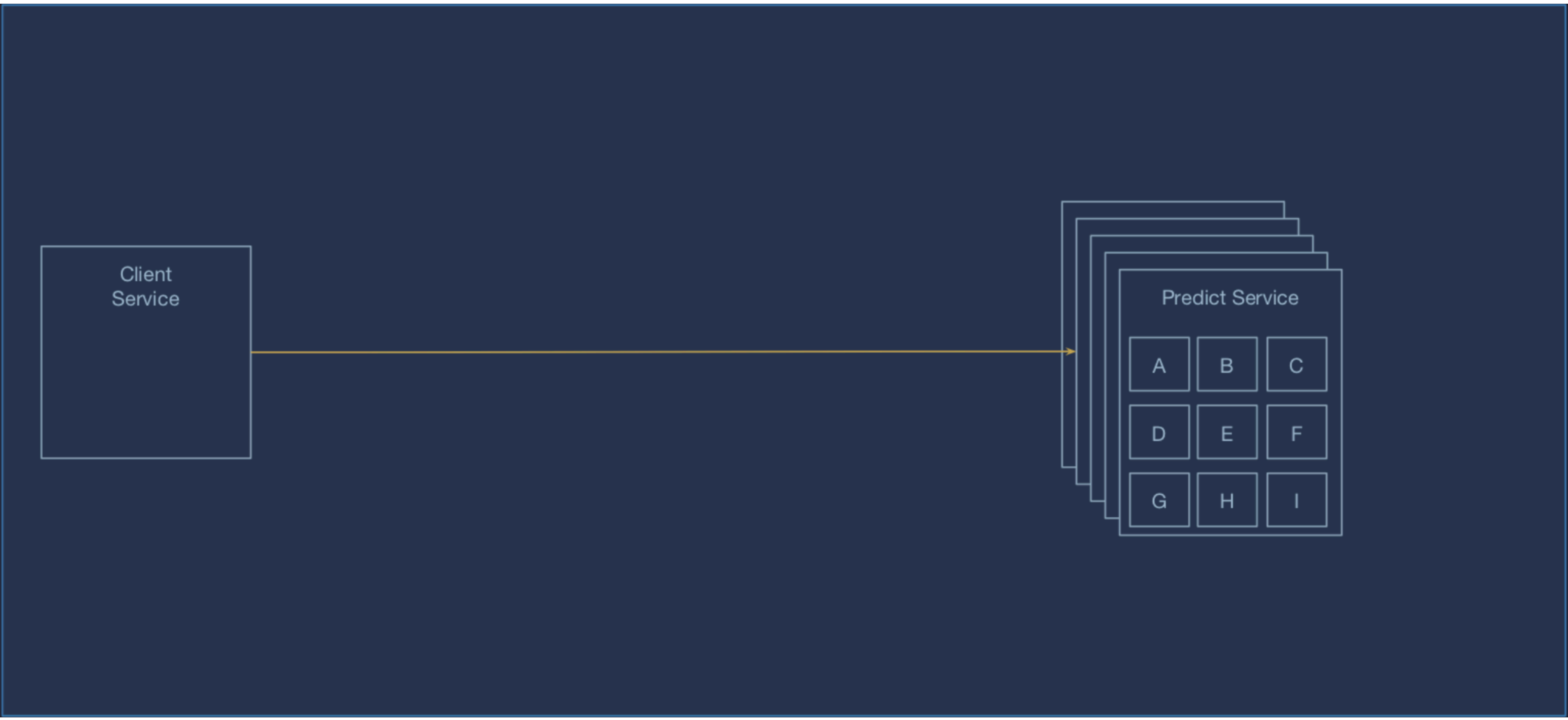
Online prediction service

- Typical p95 latency from client service
  - ~5ms when all features from client service
  - ~10ms when joining pre-computed features from Cassandra
- Peak prediction volume across current online deployments
  - 600k+ QPS

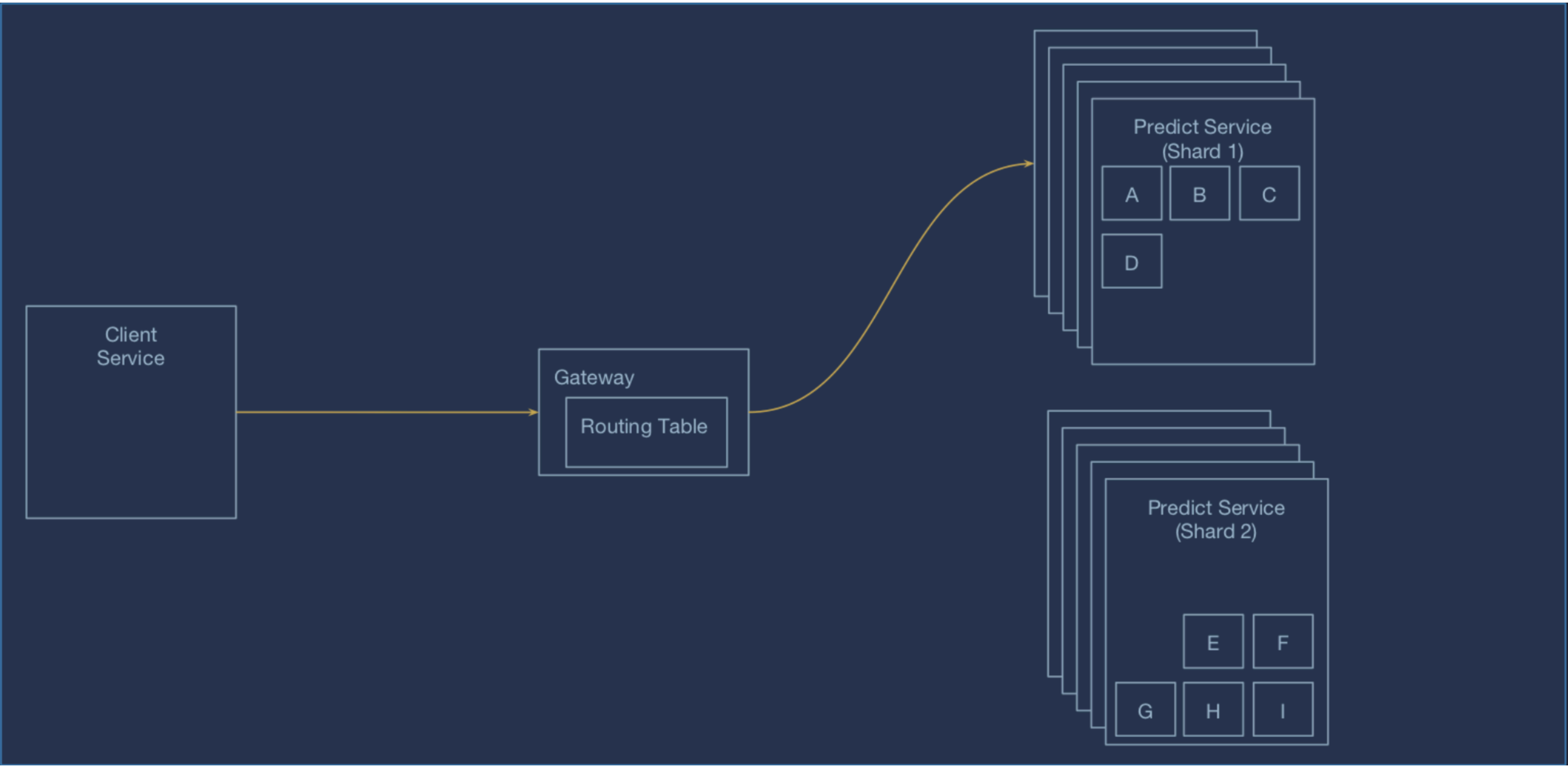
Sharded deployment

- Problem
  - Prediction service can serve as many models as will fit into memory
  - Easy to run out of memory with large deployments of complex models
- Solution
  - Organize serving cluster into number of physical shards
  - Introduce client facing concept of ‘virtual shard’ that is specified at deploy time
  - Virtual shards are mapped by system to physical shards
  - Models are loaded by service instances in the correct physical shard(s)
  - Gateway service routes to correct physical shard based on request header

Unsharded deployment



Sharded deployment



Key Components: Deployment Labels

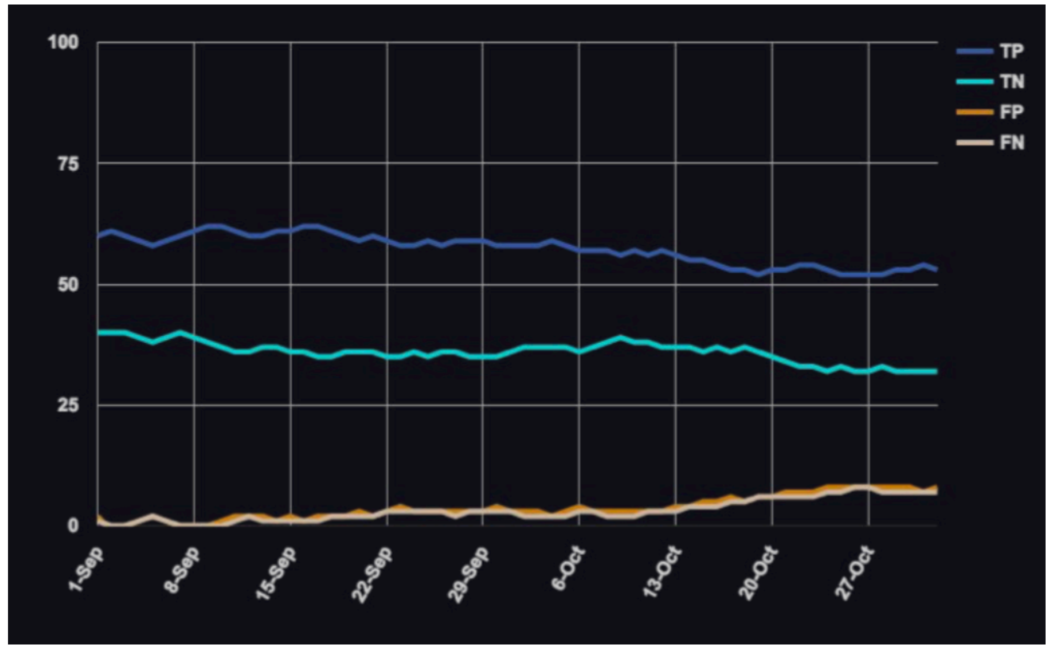
Deployment labels

- Problem
    - Multiple models per container (entirely different or multiple versions of same)
    - Support experimentation
    - Support automated retrain / redeploy
    - Cumbersome to have client service manage routing
- Solution
    - Models deployed to 'label'
    - Labels can be used for experimentation or different use cases
    - Predict service routes request to most recent model w/ specified label
    - Labels have schema so deploys won't break

Key Components: Live Model Performance Monitoring

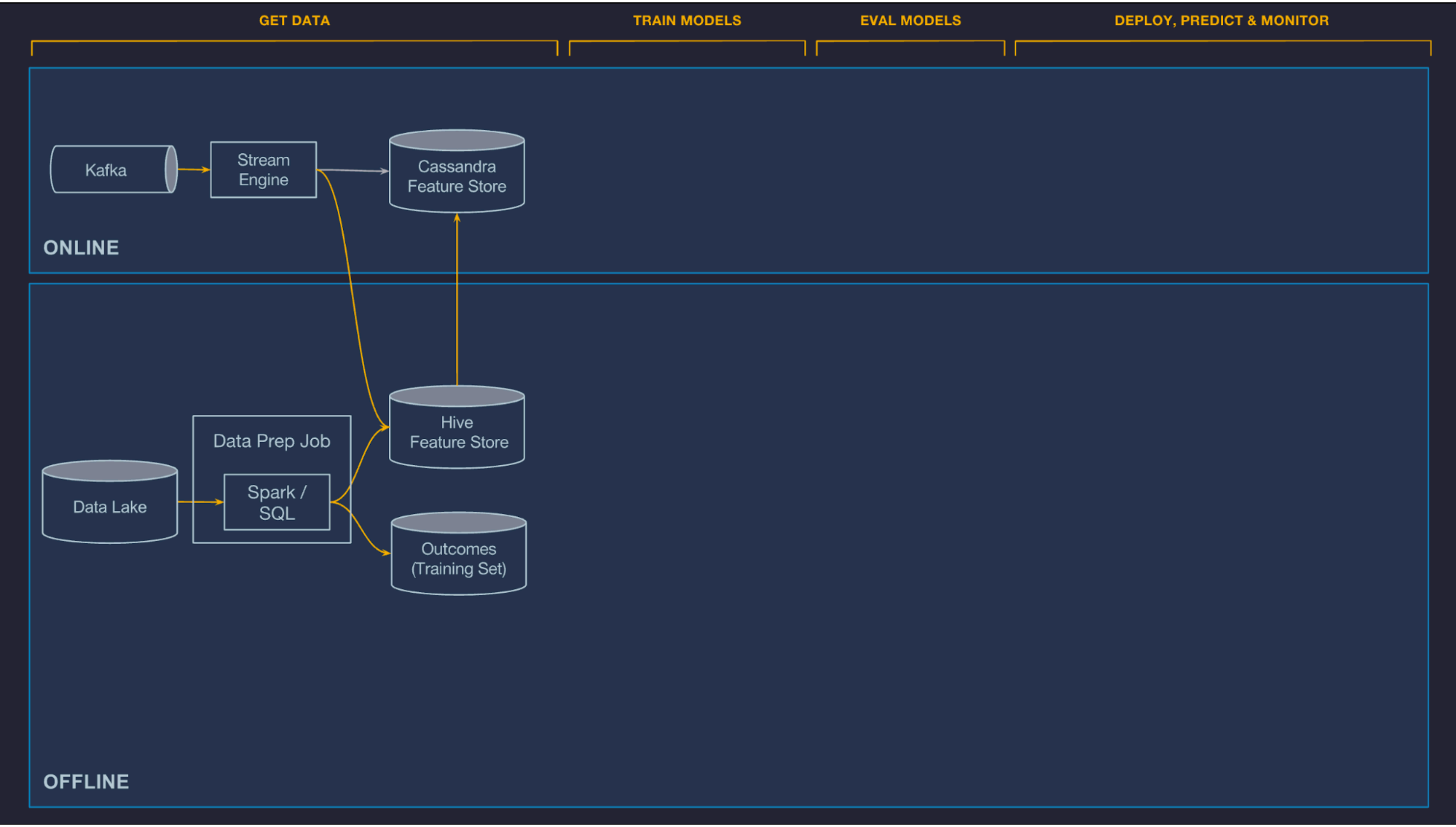
Monitor predictions

- Problem
  - Models trained and evaluated against historical data
  - Need to ensure deployed model is making good predictions going forward
- Solution
  - Log predictions & join to actual outcomes
  - Publish metrics feature and prediction distributions over time
  - Dashboards and alerts

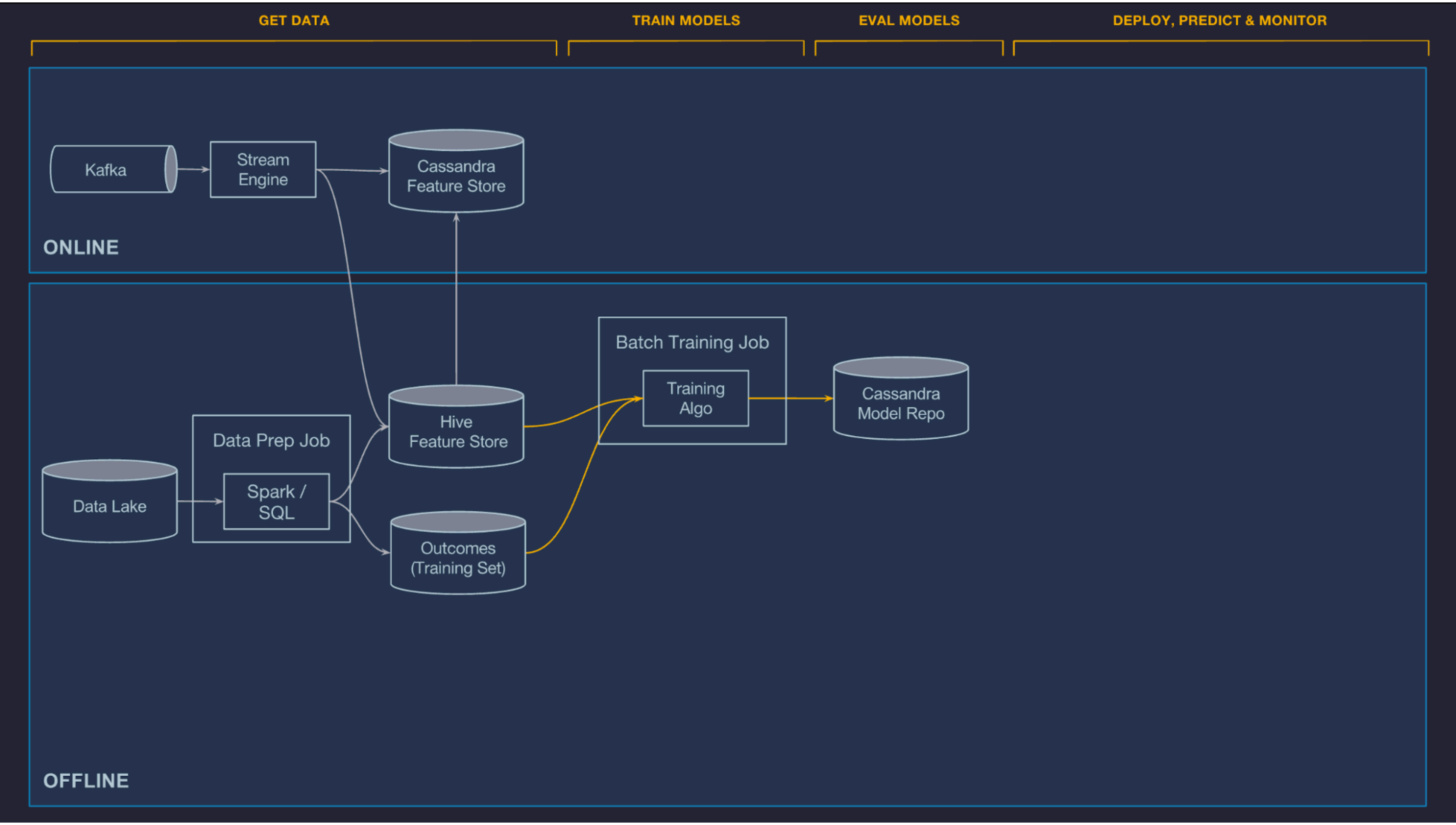


System Architecture

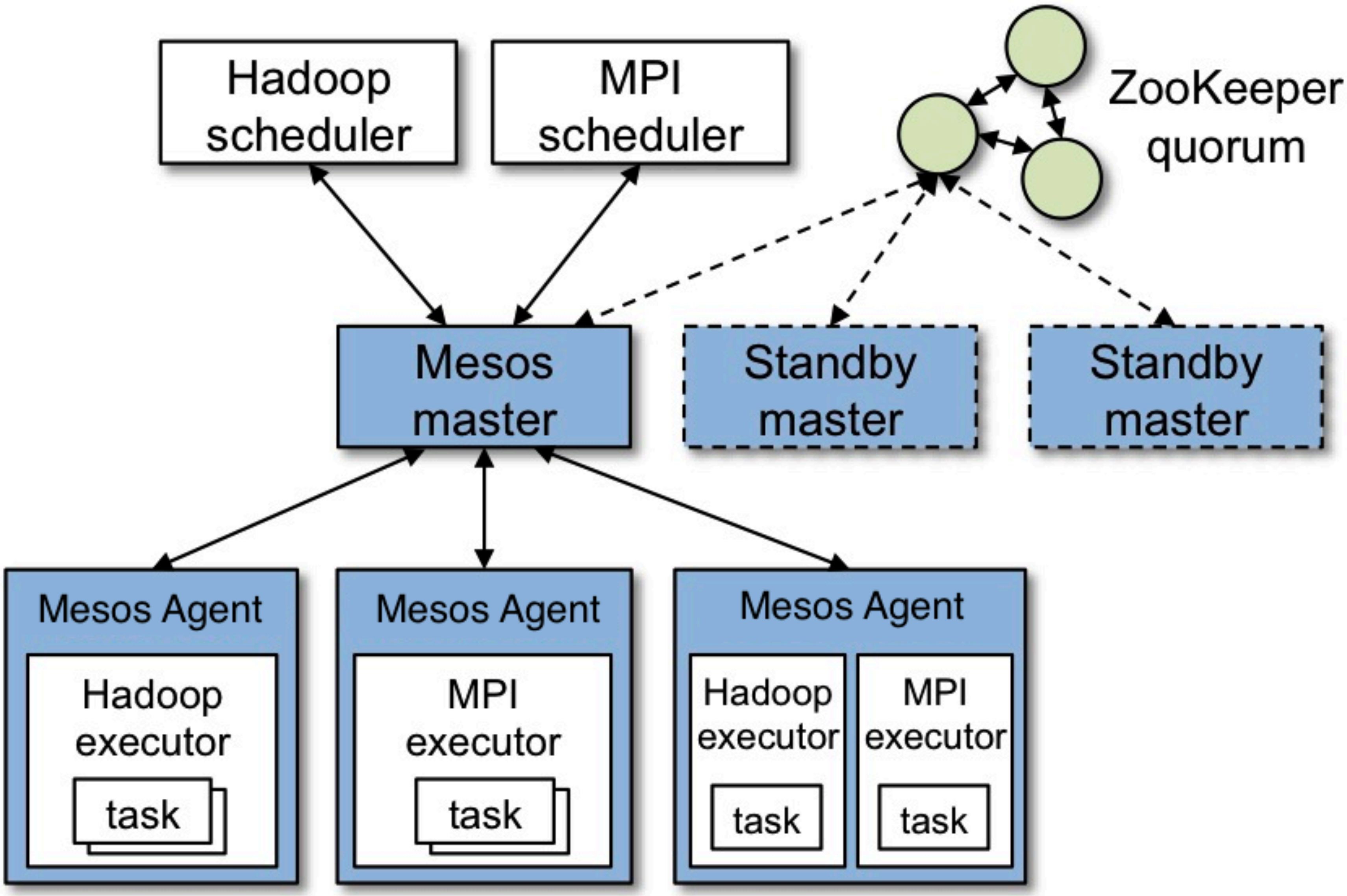
Data preparation



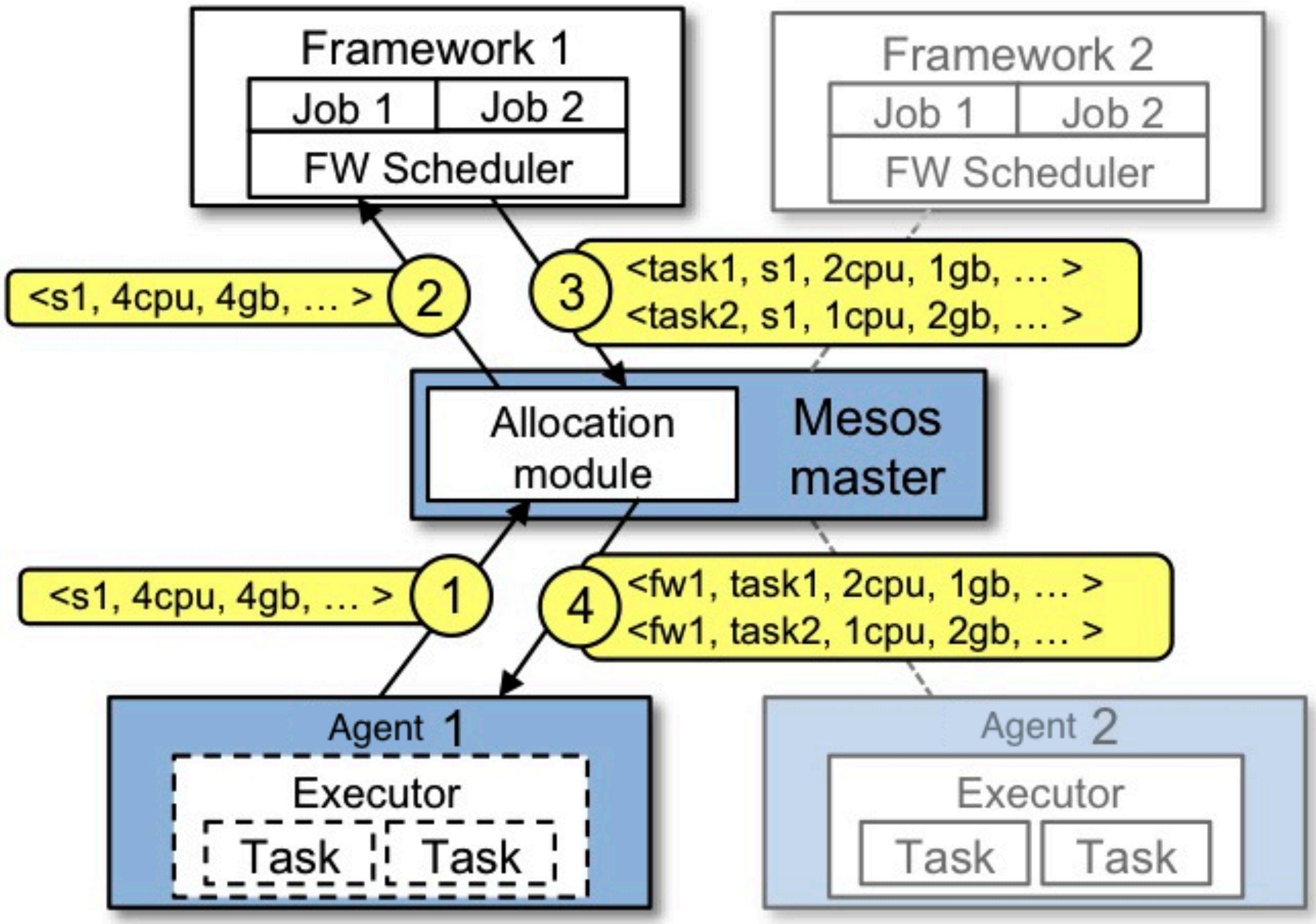
Model training & evaluation



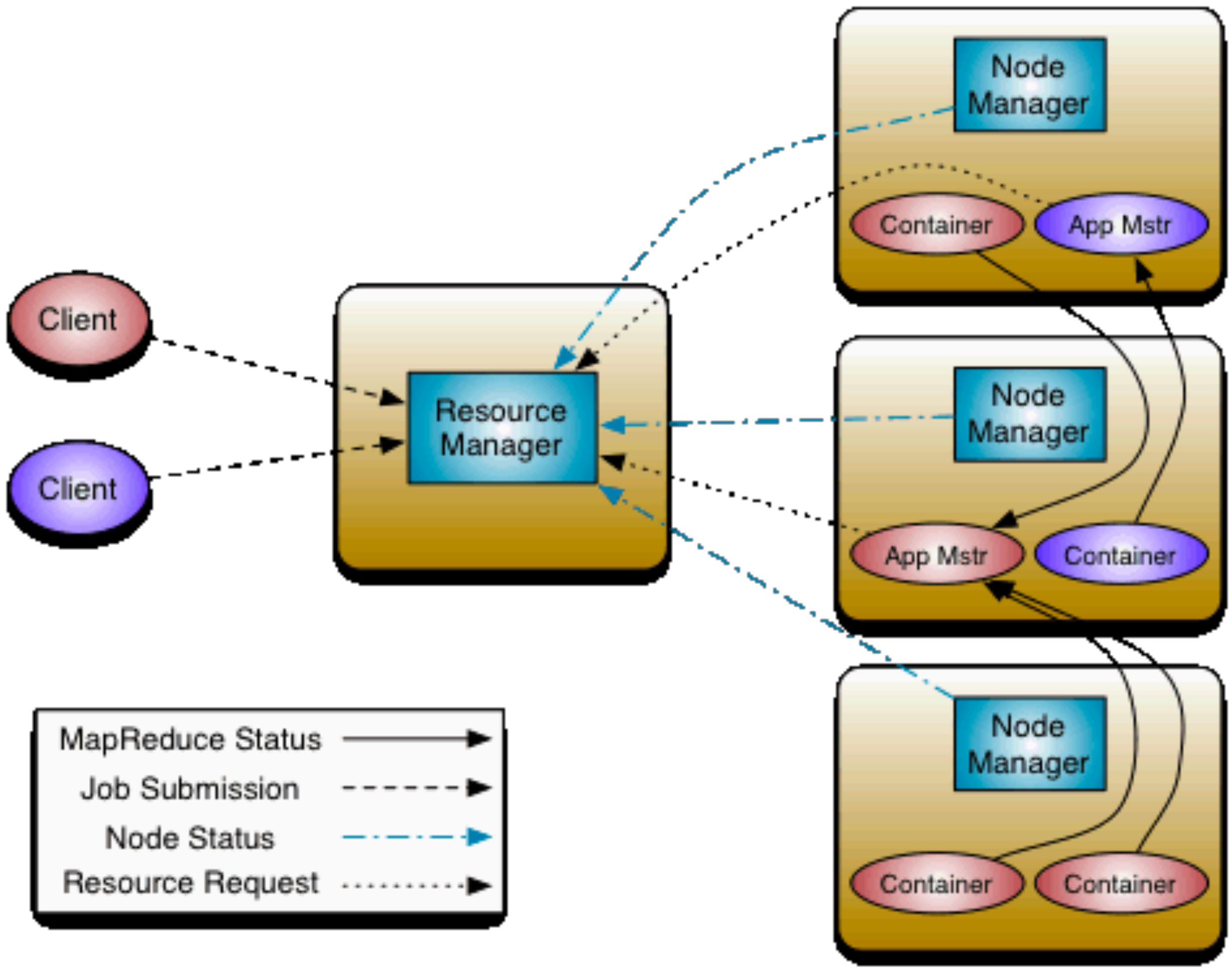
Mesos Architecture



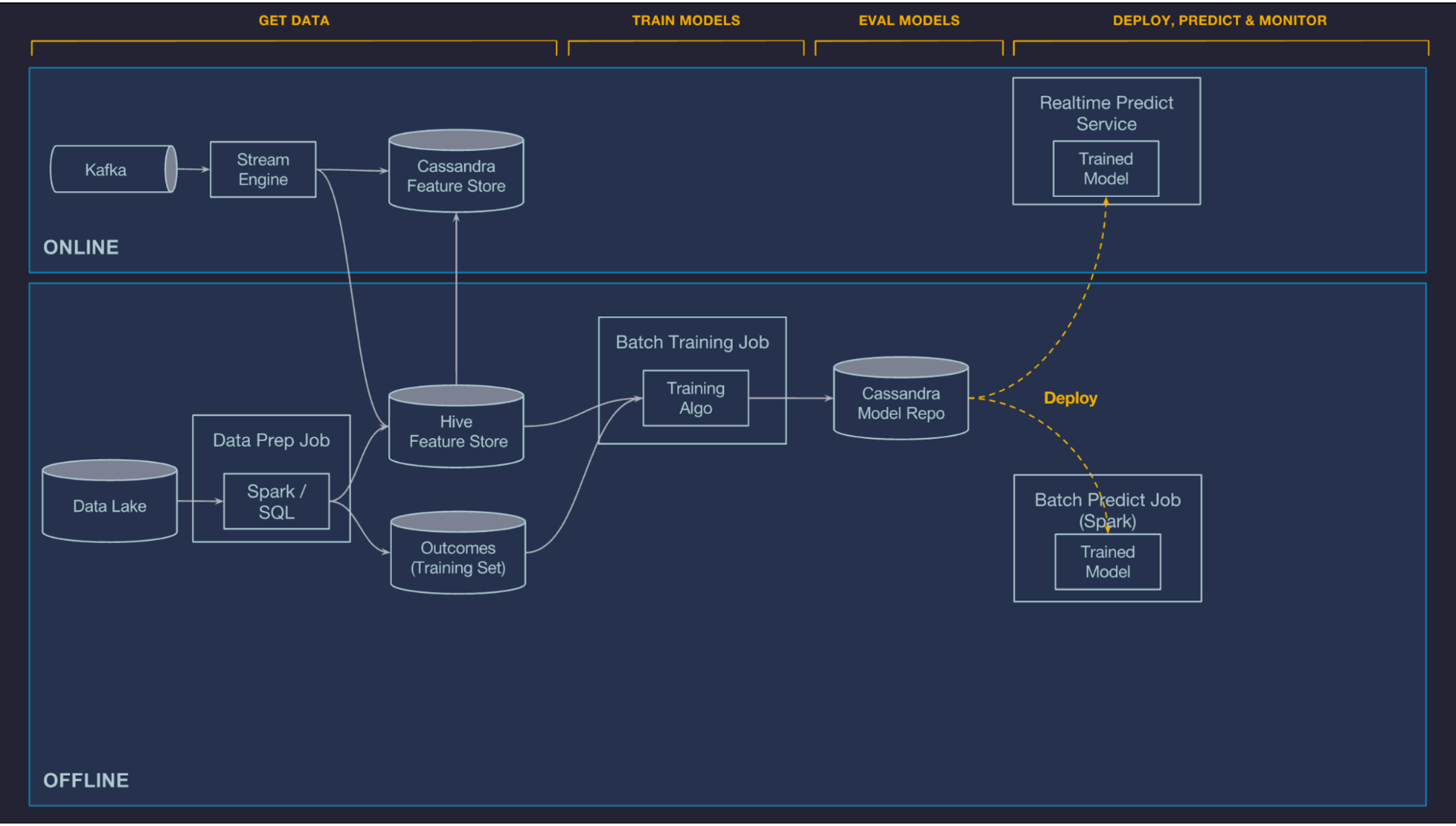
Example of resource offer



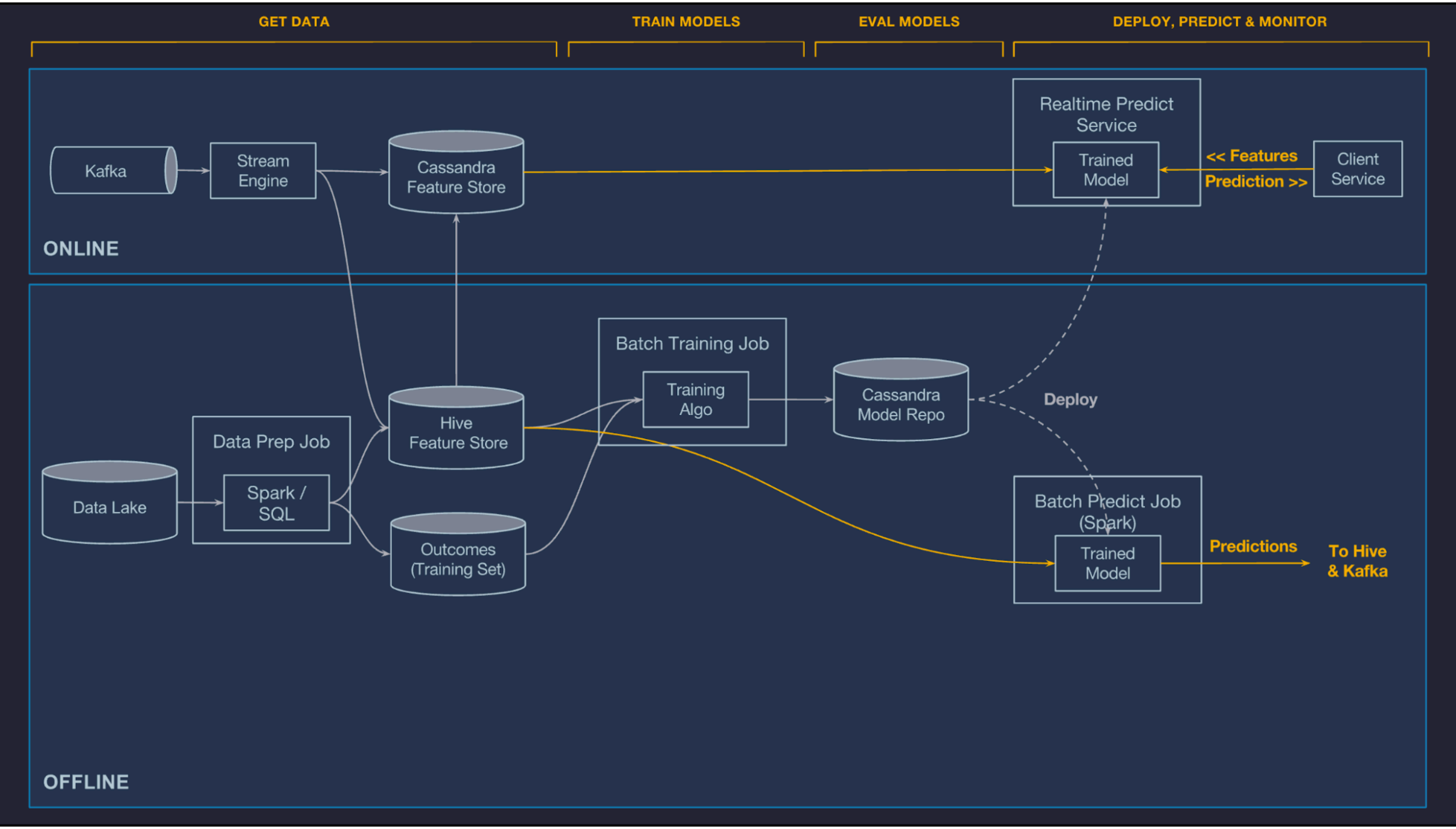
Apache Hadoop YARN



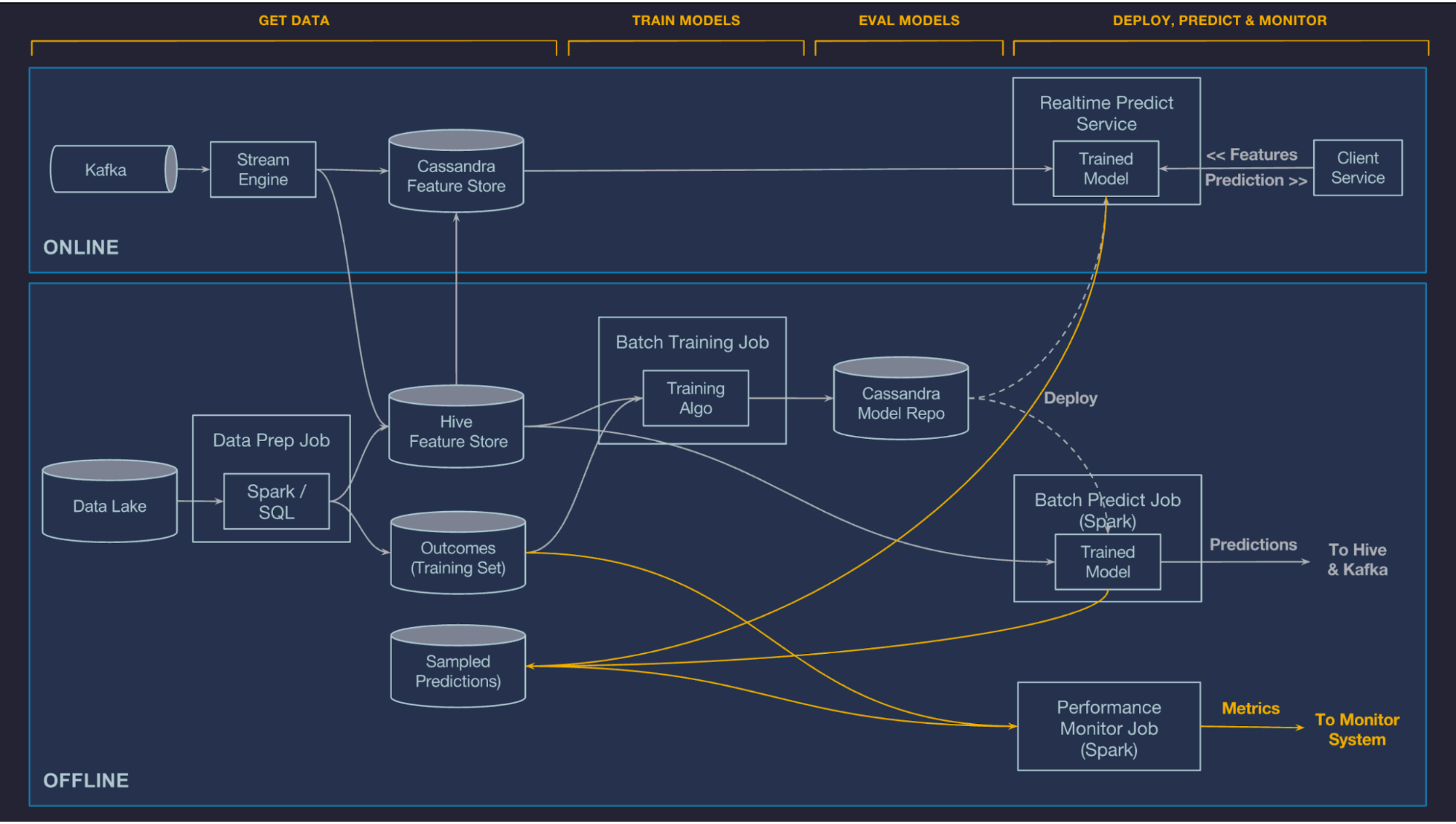
Model deployment



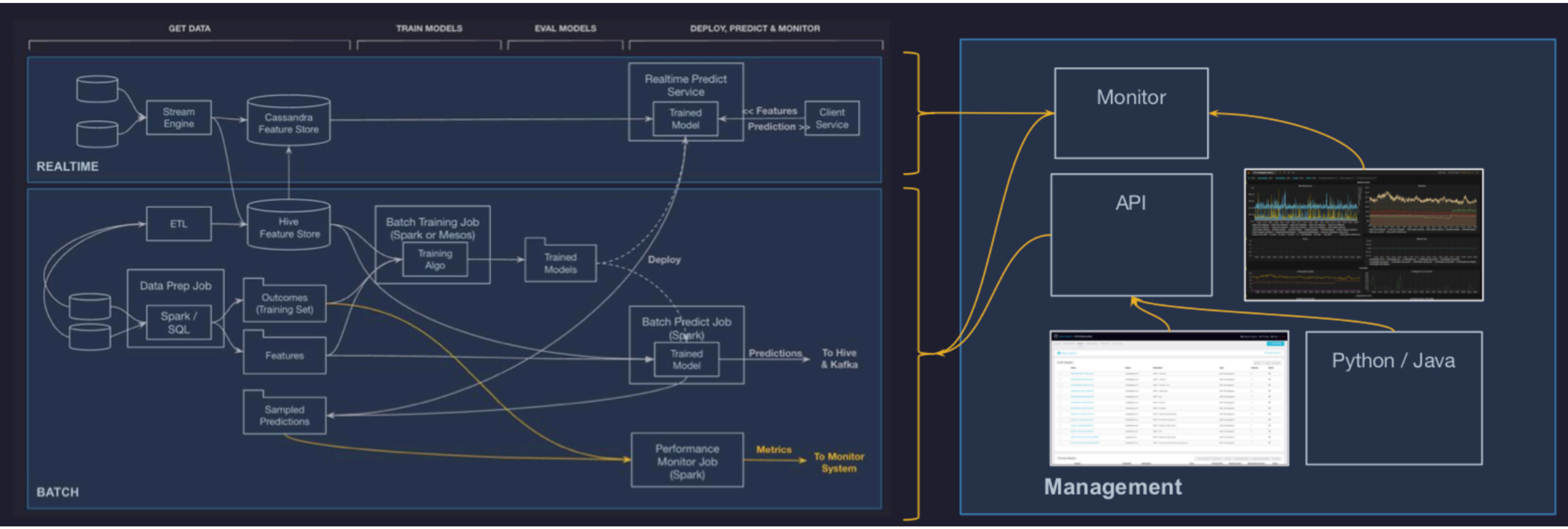
Model serving



Monitoring


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

Big picture of the ML platform



Summary

- We went through the key components of Michelangelo
- We reviewed challenges of building an ML pipeline
- We discussed solutions that scale
- We reviewed how an ML platform can facilitate building a pipeline.