Uber

Kubernatize Big Data and ML Workloads @ Uber

Mayank Bansal, Data Infra, Uber

Min Cai, ML Platform, Uber

Igniting opportunity by setting the world in motion

Uber

15 billion trips

15M trips per day

6 continents, 65 countries and 700 cities

100M active monthly users

3.9M active drivers

26,000+ employees worldwide

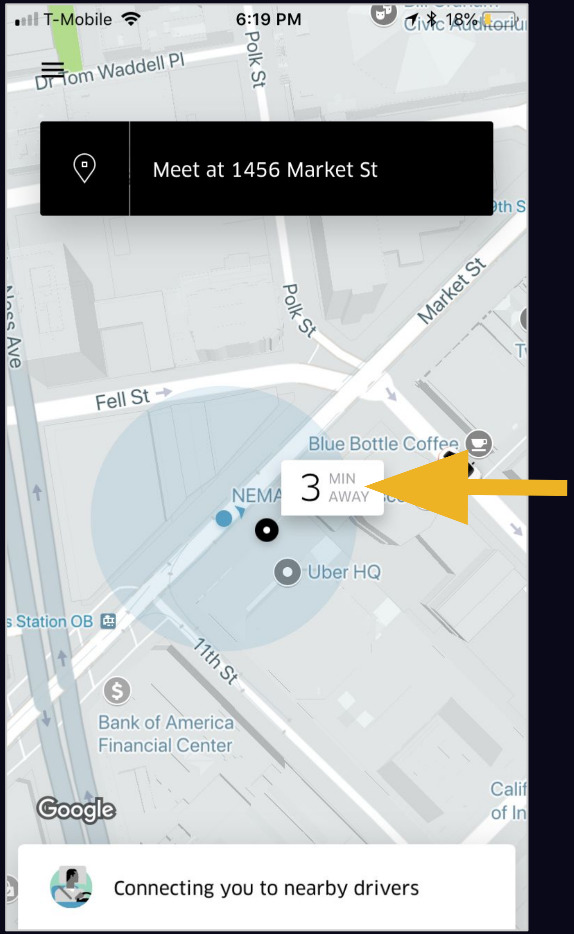
3700+ developers worldwide

Big Data and ML Use Cases at Uber

- Uber Eats
- ETAs
- Autonomous Cars
- Customer Support
- Dispatch
- Personalization
- Demand Modeling
- Dynamic Pricing
- Forecasting
- Maps
- Fraud
- Anomaly Detection
- Capacity Planning
- And many more...

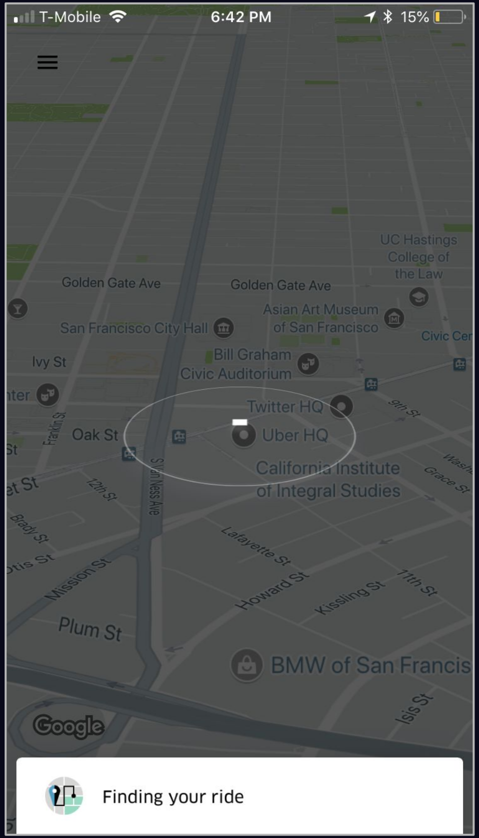
Big Data and ML at Uber - ETAs

- ETAs are core to customer experience
- ETAs used by myriad internal systems
- ETA are generated by route-based algorithm
- ML model predicts the route-based ETA error
- Use the predicted error to correct the ETA
- ETAs now dramatically more accurate



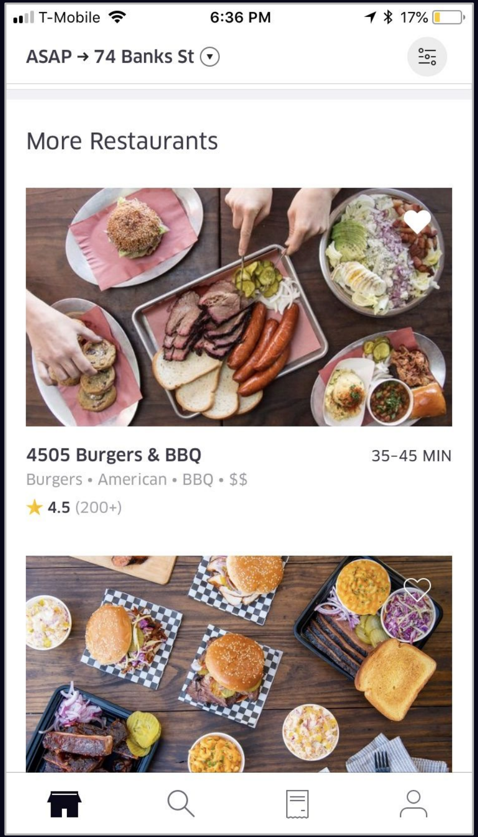
Big Data and ML at Uber - Dispatch

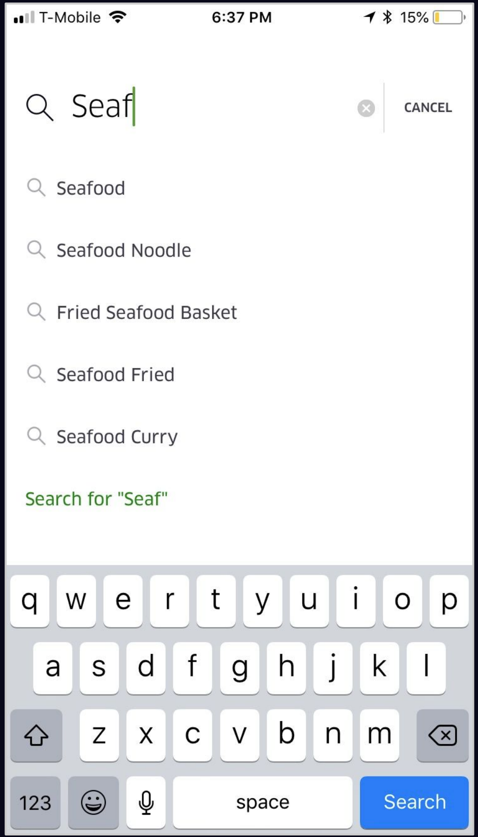
- Optimize matching of rider and driver
- Predict if open rider app will make trip request



Big Data and ML at Uber - Eats

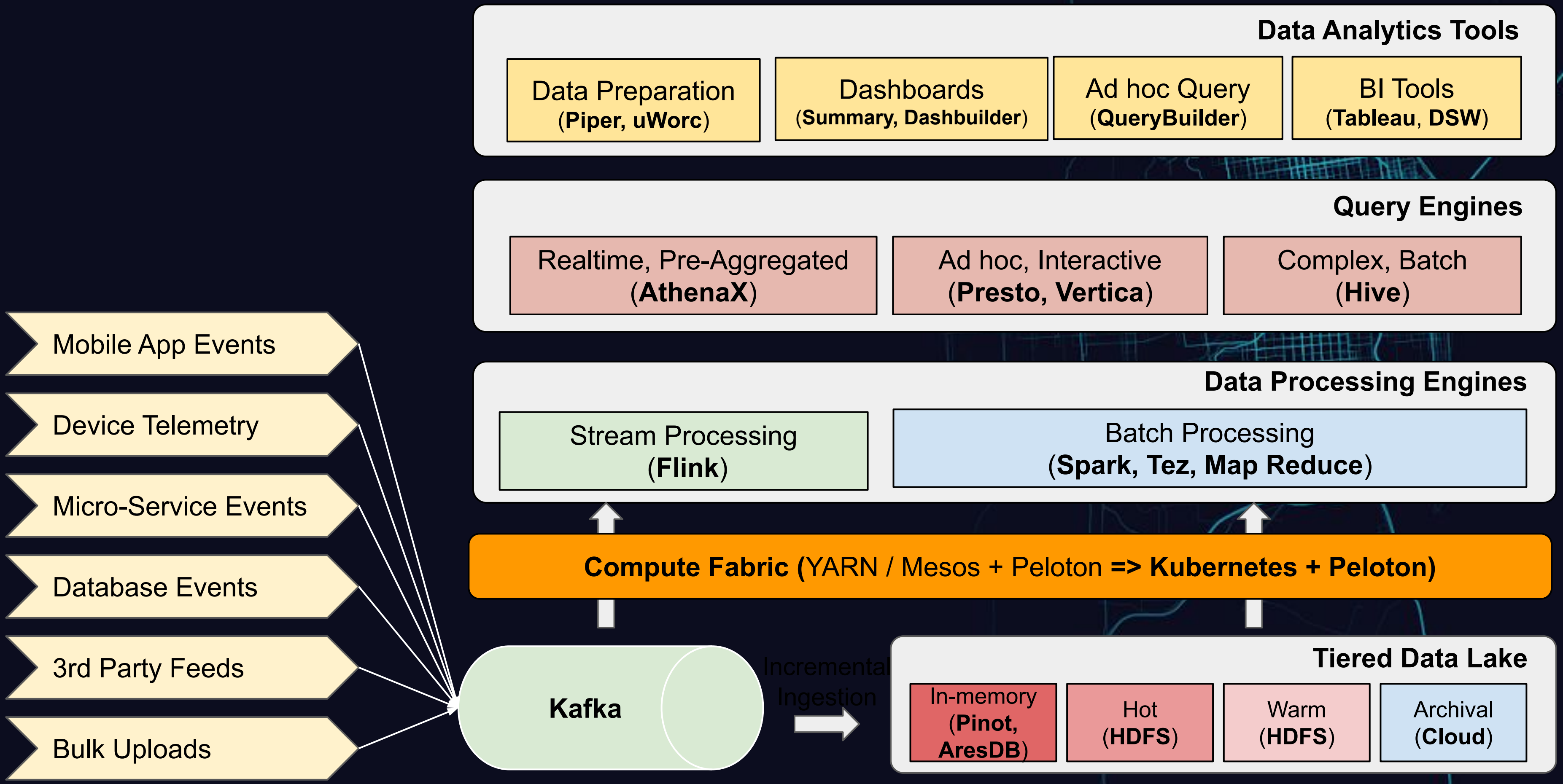
- Models used for
  - Ranking of restaurants and dishes
  - Delivery times
  - Search ranking
- 100s of ML models called to render Eats homepage



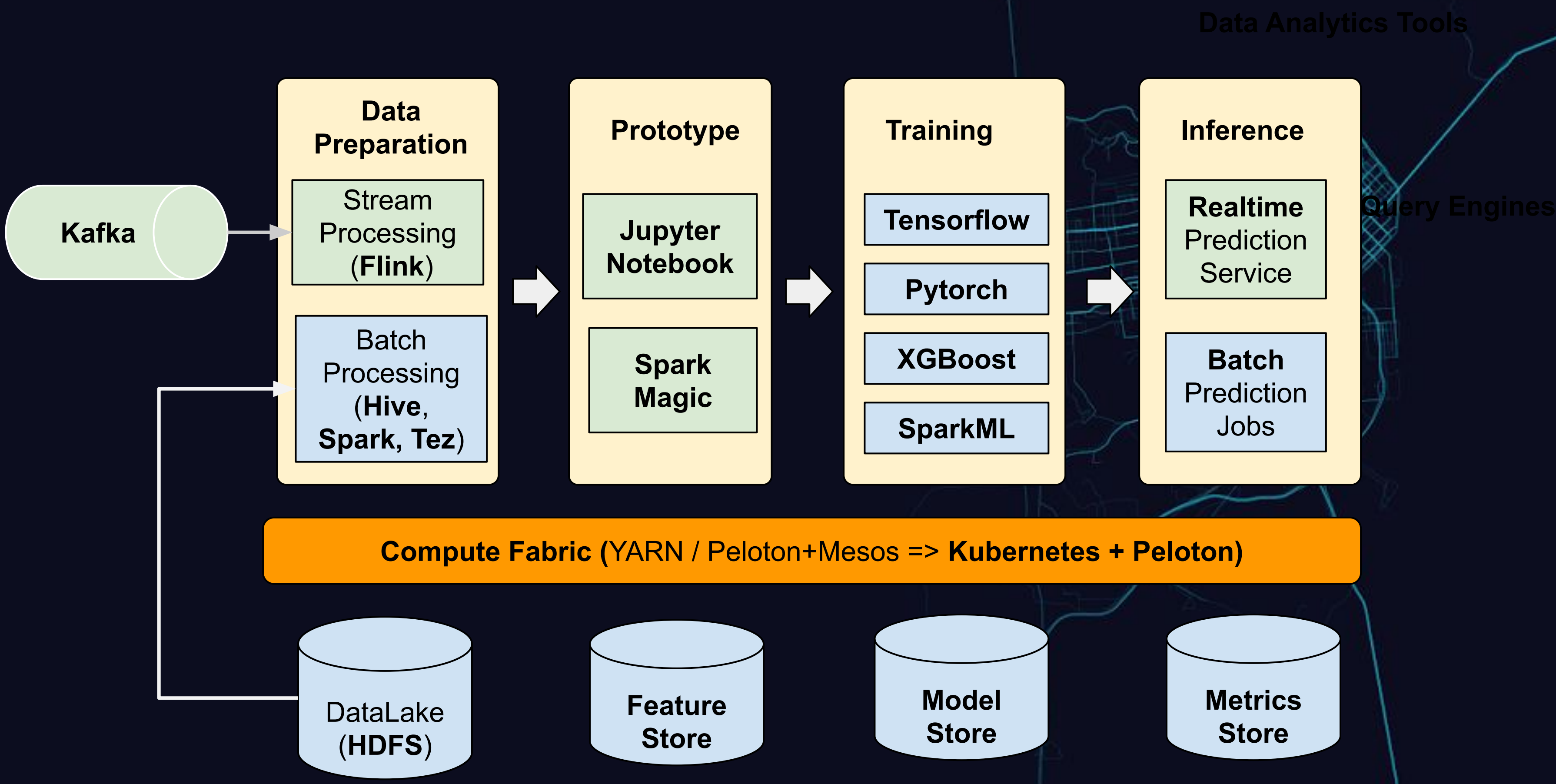


Big Data and ML at Uber - Self Driving Vehicles

Uber's Big Data Stack



Uber’s ML Stack - Michelangelo



Why Kubernetes ?

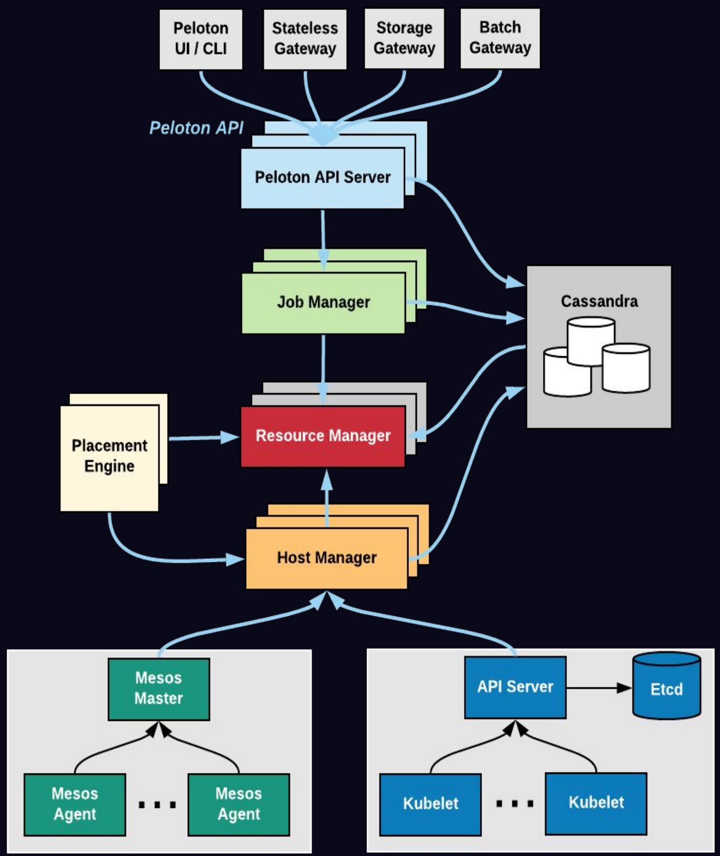
- Lots of features and extensions for mixed workloads
    - Pod, Deployment, StatefulSet, Job, DaemonSet, etc
- Growing community and ecosystem support
- Wide adoption and native integration from open source Big Data and ML projects
    - E.g. Spark, Flink, Kafka, Tensorflow etc
- Cloud native support in AWS, GCP, and Azure as managed clusters
- Feasible extension model that allows other Uber teams such as SWN, Storage, Data, and Security teams to build extensions.

Why Not Kubernetes As-Is?

- Elastic resource sharing with hierarchical resource pools
- Gang scheduling for ML workloads
- Support batch and stateless workload co-location
- High-throughput for Big Data workloads (> 1K pod / sec)
- Lack of resource oversubscription other than CPU quota / shares.
- Lack of dynamic port allocation
- Lack of cluster federation for multi-region and multi-zone

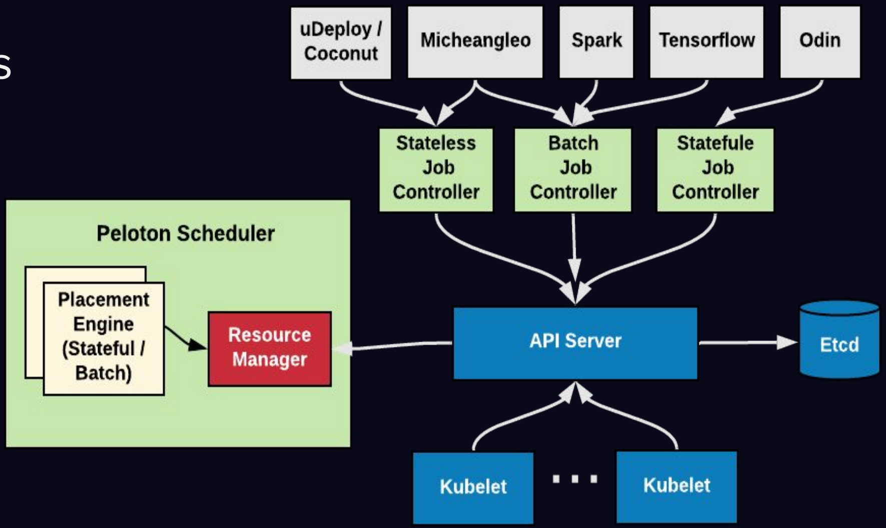
Peloton Overview

- Unified Resource Scheduler for co-locating mixed workload on compute clusters
- Integrates with Apache Spark, TensorFlow, YARN, uDeploy (Uber internal)
- Can be run on-premise or in Cloud

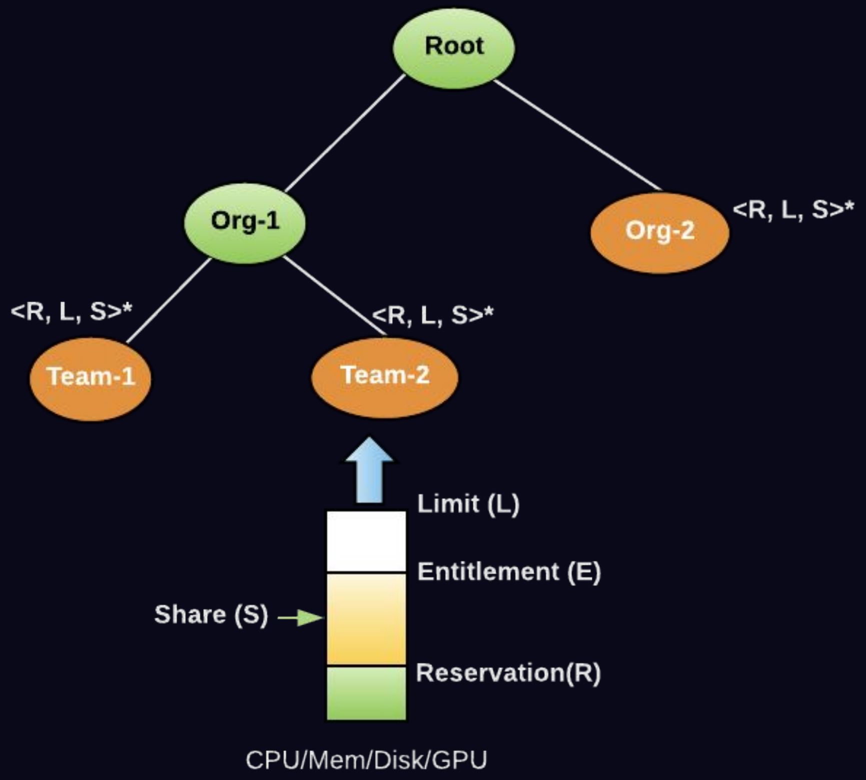


Peloton as Kubernetes Plugins

- Idiomatic for Kubernetes ecosystem
- Reuse Kubernetes API and components like api-server, etcd, kubelet.
- Support all Kubernetes drivers for Big Data / ML applications
- Optimized for Big Data / ML workloads
  - Elastic resource sharing
  - Gang scheduling
  - High-throughput for Big Data workloads



Hierarchical Resource Pools

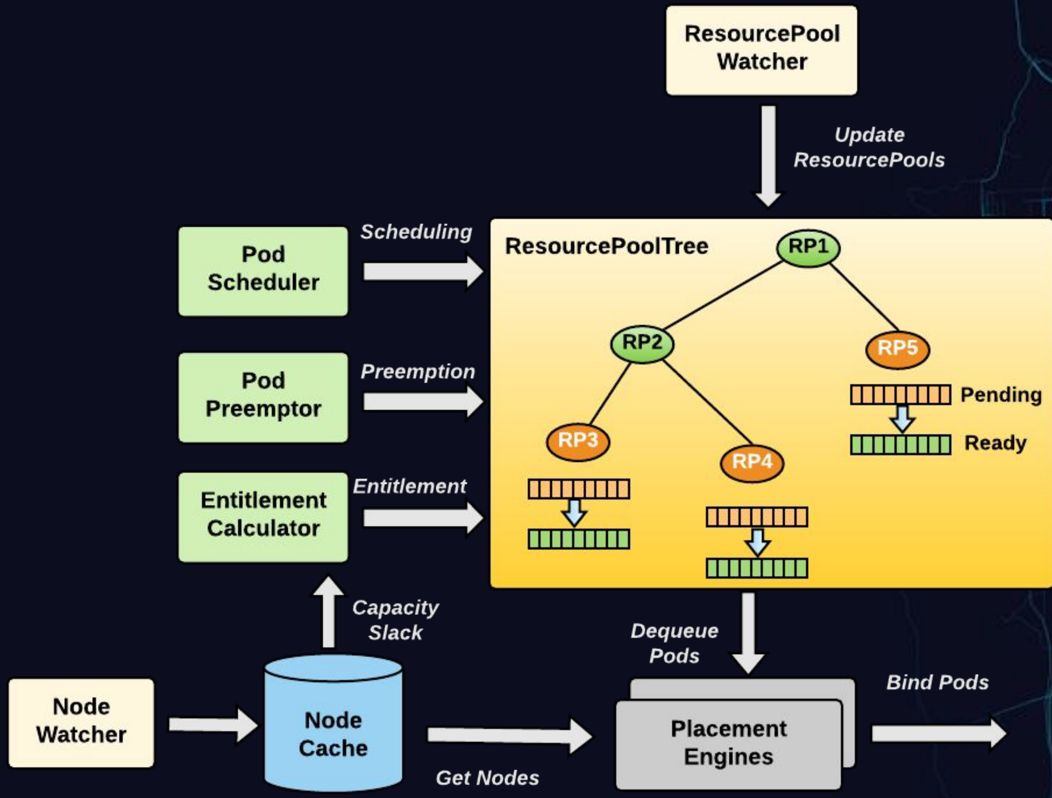


Resource Pool as Kubernetes CRD

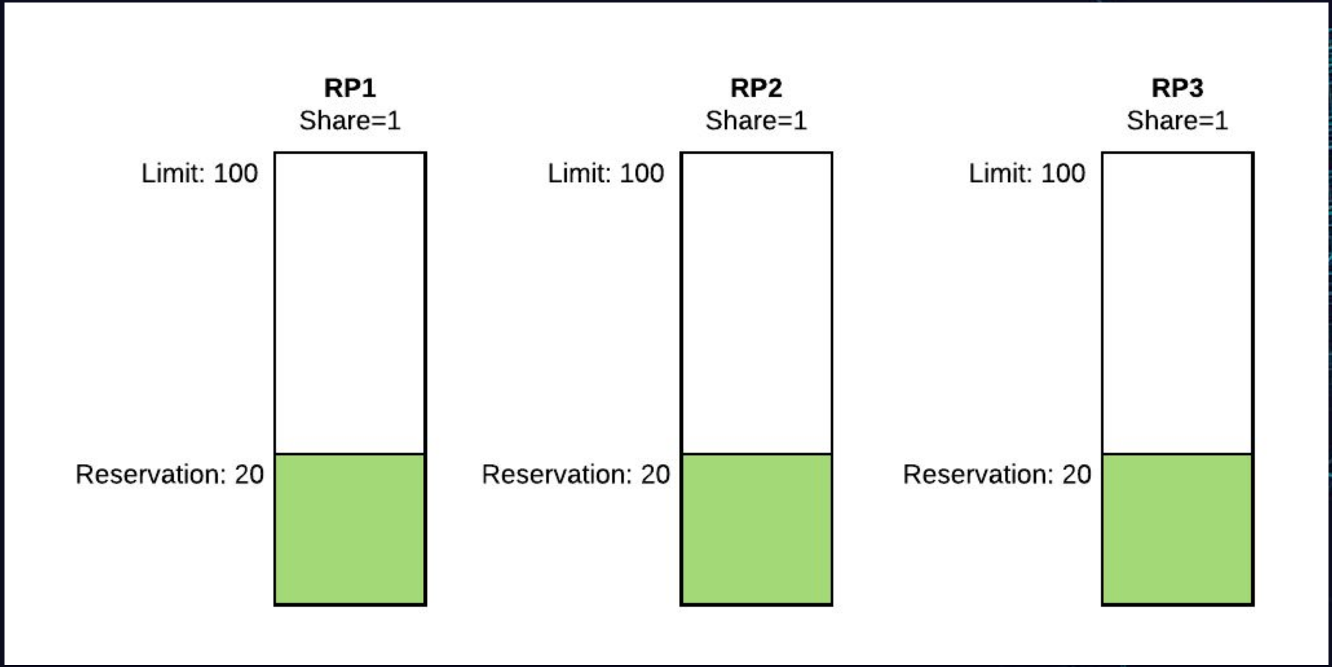
```yaml
apiVersion: "peloton.uber.com/v1alpha1"
kind: "ResourcePool"
metadata:
  name: "marketplace.uber.com"
spec:
  resources:
    reservation:
      cpu: 512
      memory: "256G"
    limit:
      cpu: 1024
      memory: "1024G"
    share:
      cpu: 1
      memory: 4
```

```
$ kubectl get CustomResourceDefinition
NAME                              AGE
resourcepools.peloton.uber.com    2h

$ kubectl apply -f respool-marketplace.yaml
resourcepool.peloton.uber.com "marketplace.uber.com" created

$ kubectl get ResourcePool
NAME                   AGE
marketplace.uber.com   5s
```

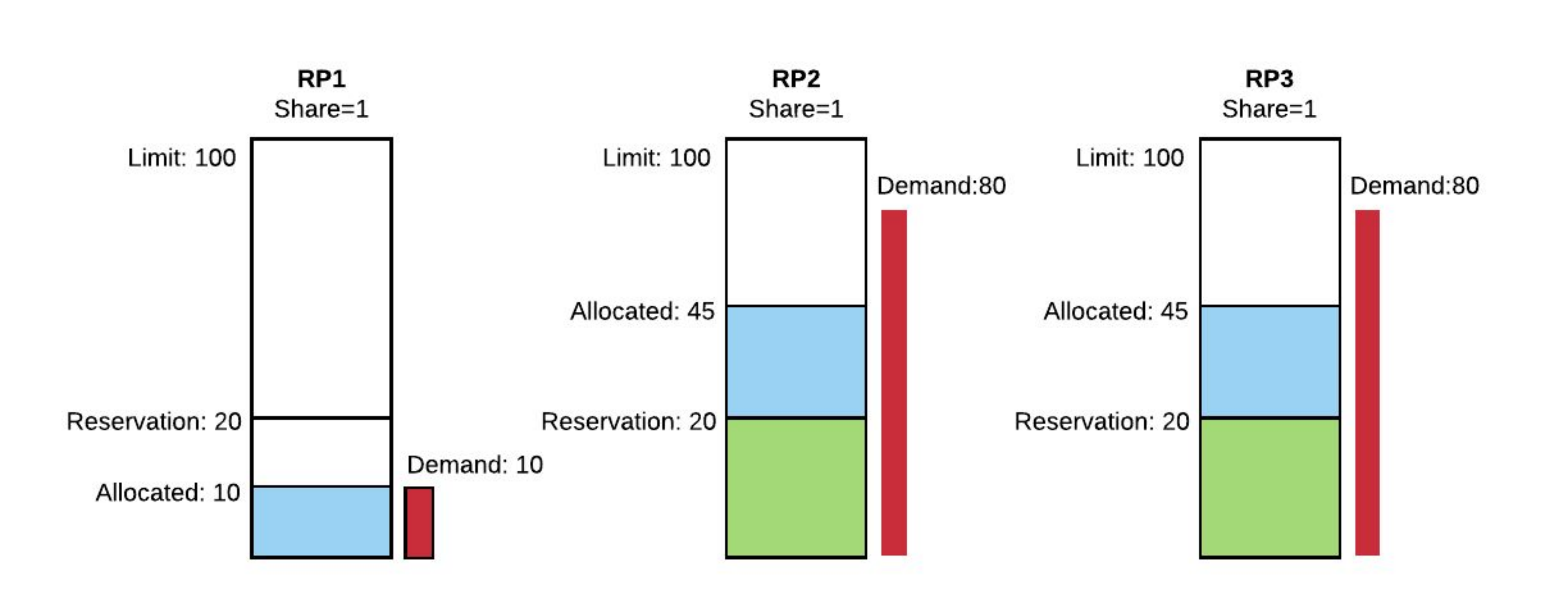
Peloton Scheduler Architecture



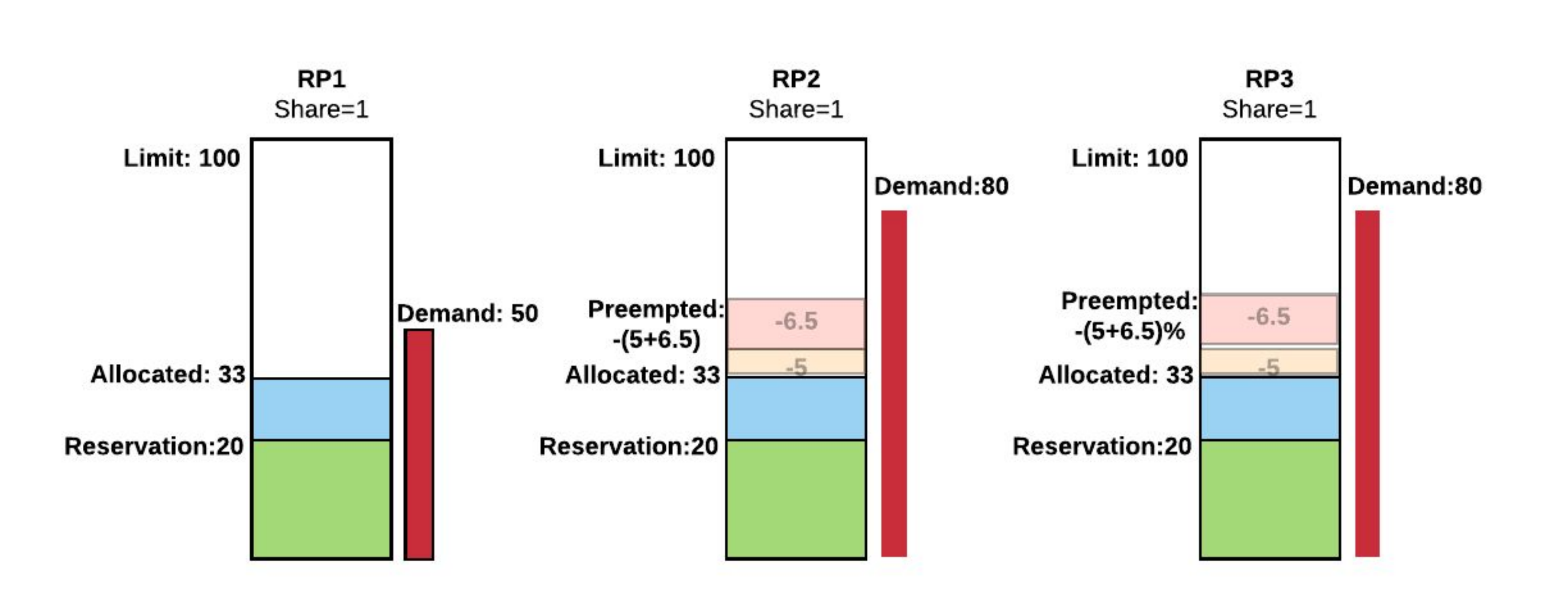
Elastic Resource Pool Example



Elastic Resource Pool Example (cont.)



Elastic Resource Pool Example (cont.)



Spark on Peloton + Kubernetes

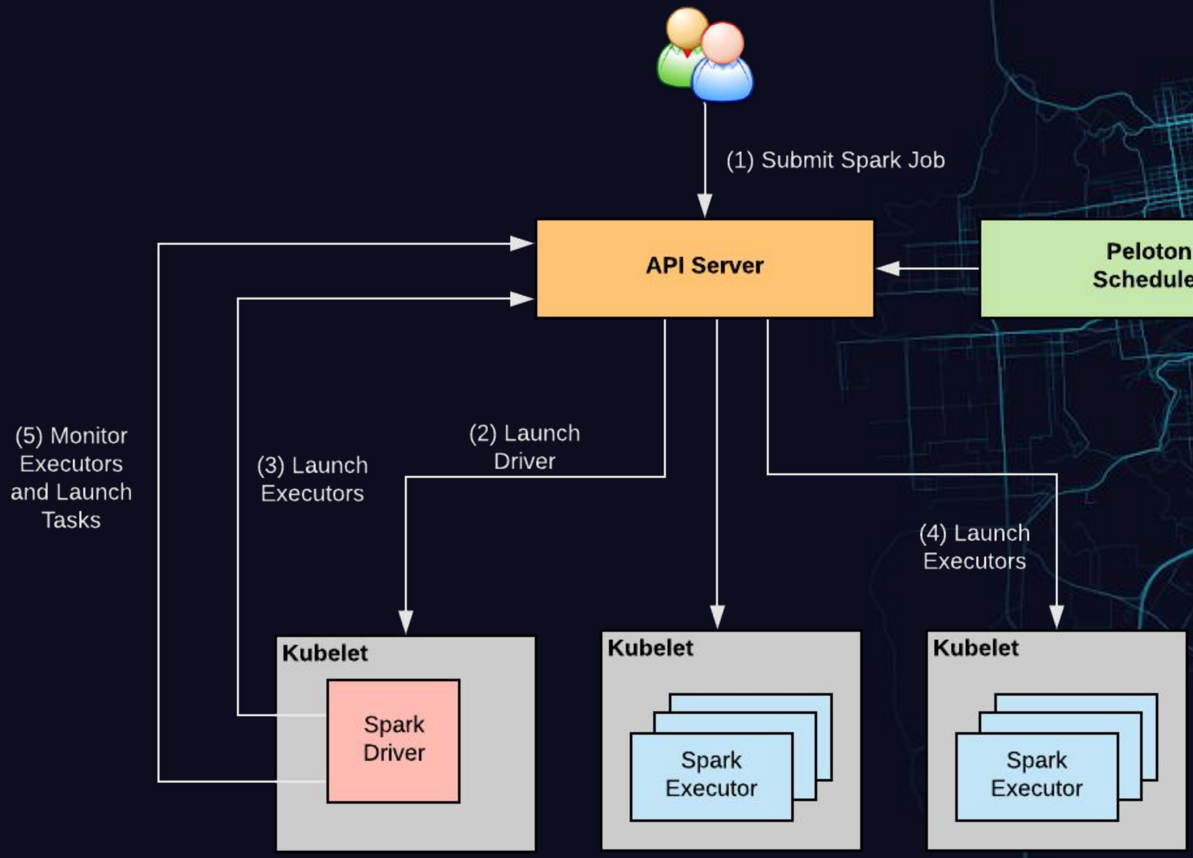
Apache Spark @ Uber

- Challenges for Spark on YARN
  - Lack of Docker support
  - Lack of big containers support
- Challenges for Spark on Mesos
  - Lack of elastic resource sharing
  - Spark job registers as a new framework
- Spark on Peloton
  - Custom spark drivers for Peloton (v2.1, v2.3, v2.4)
  - In production for 2 years
  - 6+ production clusters

Why Spark on Kubernetes ?

- Kubernetes is becoming de facto for ML/AI workloads
- Expensive to maintain custom Peloton drivers for Spark
- Unify Big data and ML workloads in one resource scheduler
    - Remove cluster fragmentation
    - Prioritize workloads
- Leverage Kubernetes growing community and ecosystem
    - Out of the shelf Spark driver support

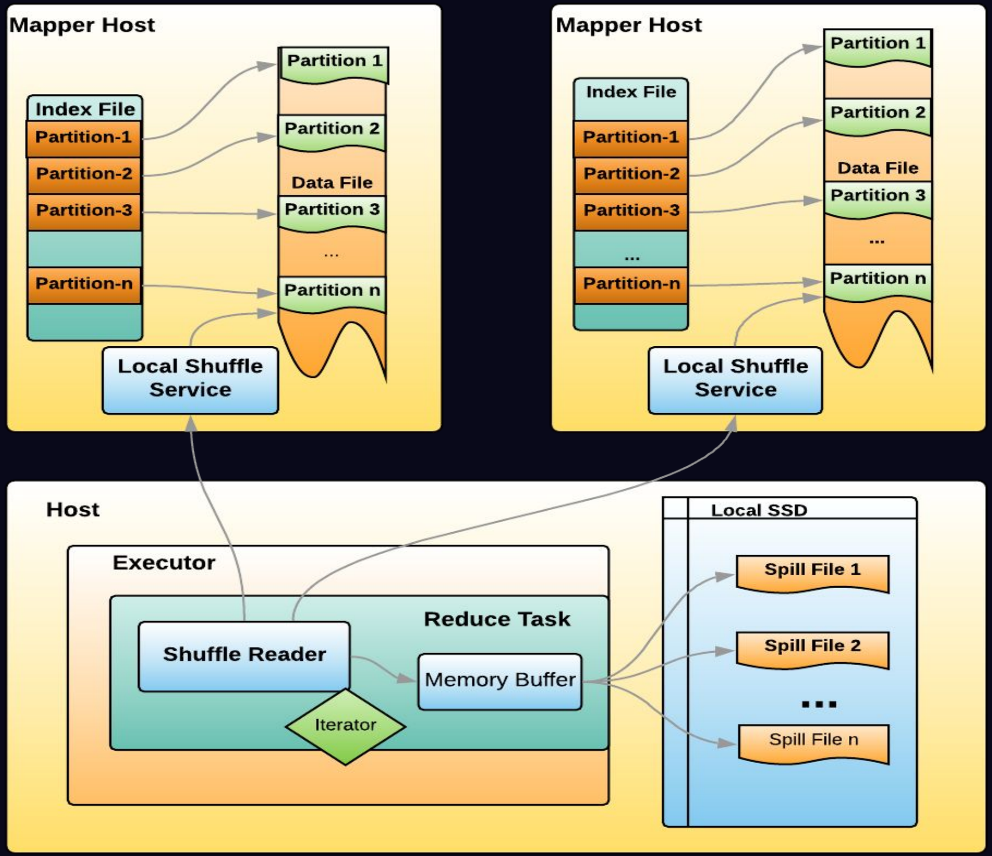
How Does Spark on Kubernetes Work ?



Spark on Kubernetes Challenges

- Lack of elastic resource sharing
  - Solution: Peloton resource pools
- Only support global priority
  - Solution: Priorities at org / resource pool level
- Lack of dynamic resource allocation support
  - Solution: remote spark shuffle service
- Lack of support for secure HDFS
  - Solution: Pass Kerberos token as Kubernetes secret

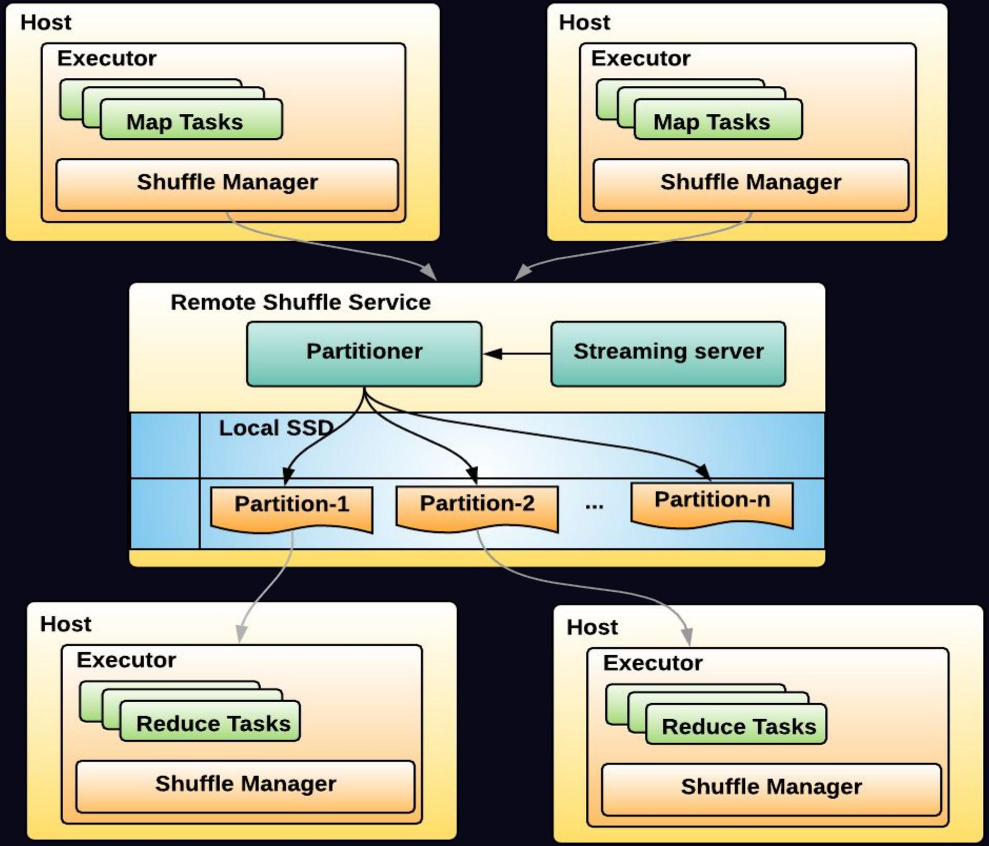
How Does Spark Shuffle Service Work?



Limitations of Spark Shuffle Service

- SSD wearing out Issues
- Reliability
- Kubernetes dynamic allocation
- Collocation

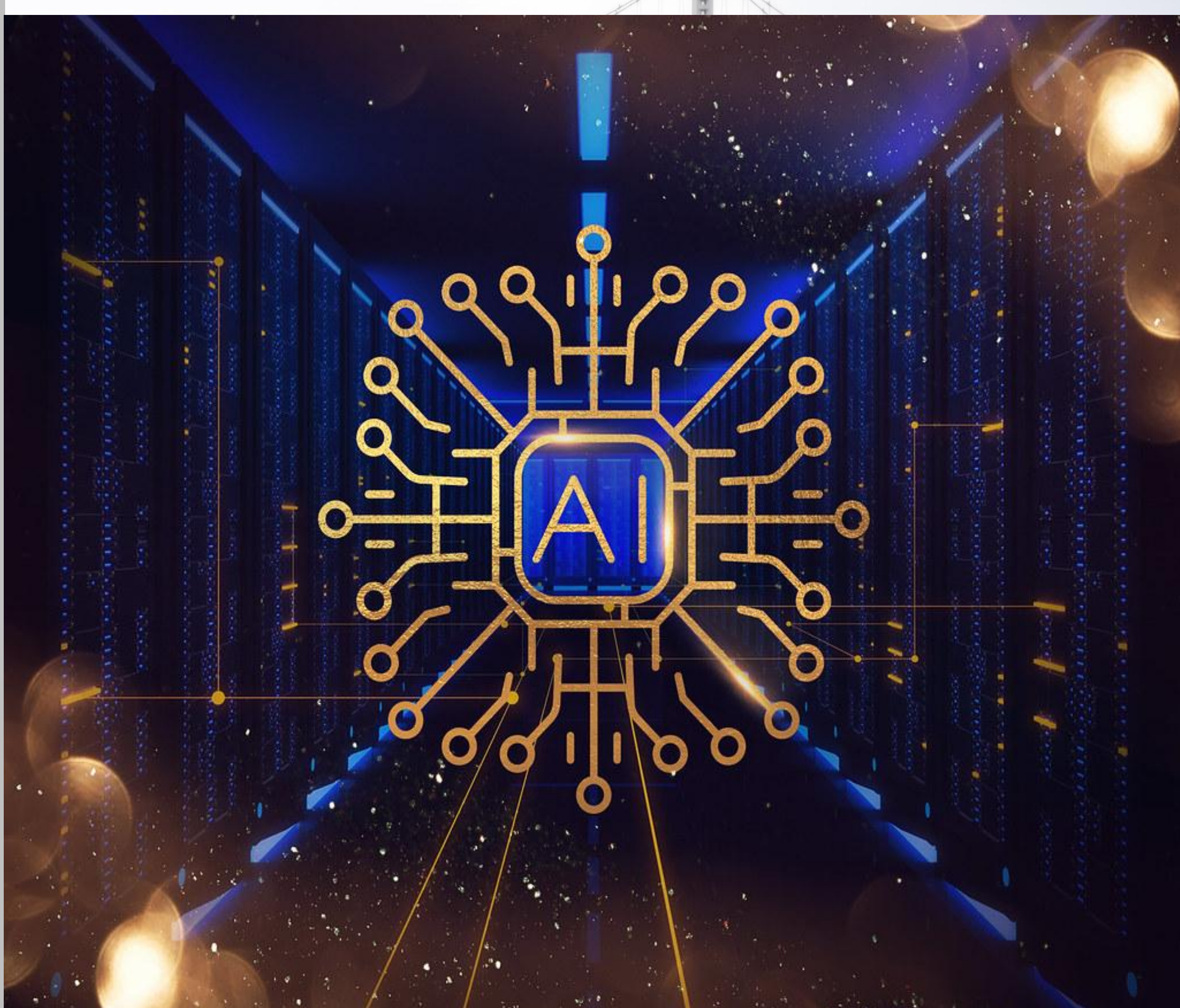
Remote Spark Shuffle Service



Remote Spark Shuffle Service - Production Status

- In Production from last 3 months for YARN and Peloton on Mesos
- Thousand’s of application running every day
- Job latencies are on par with external shuffle
- Working towards on boarding all Spark workloads
- Open Source soon

GPUs & Deep Learning

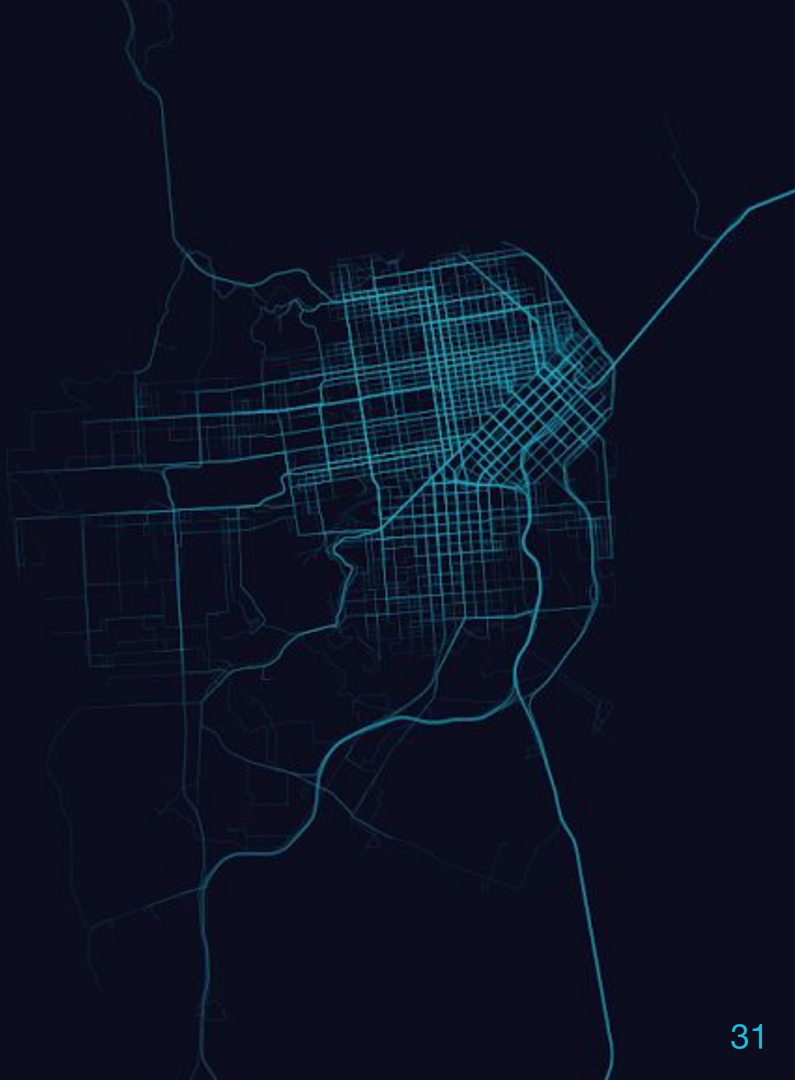


Deep Learning Use Cases

- Self-Driving Vehicles
- Trip Forecasting
- Fraud Detection
- More ...

Distributed TensorFlow Challenges

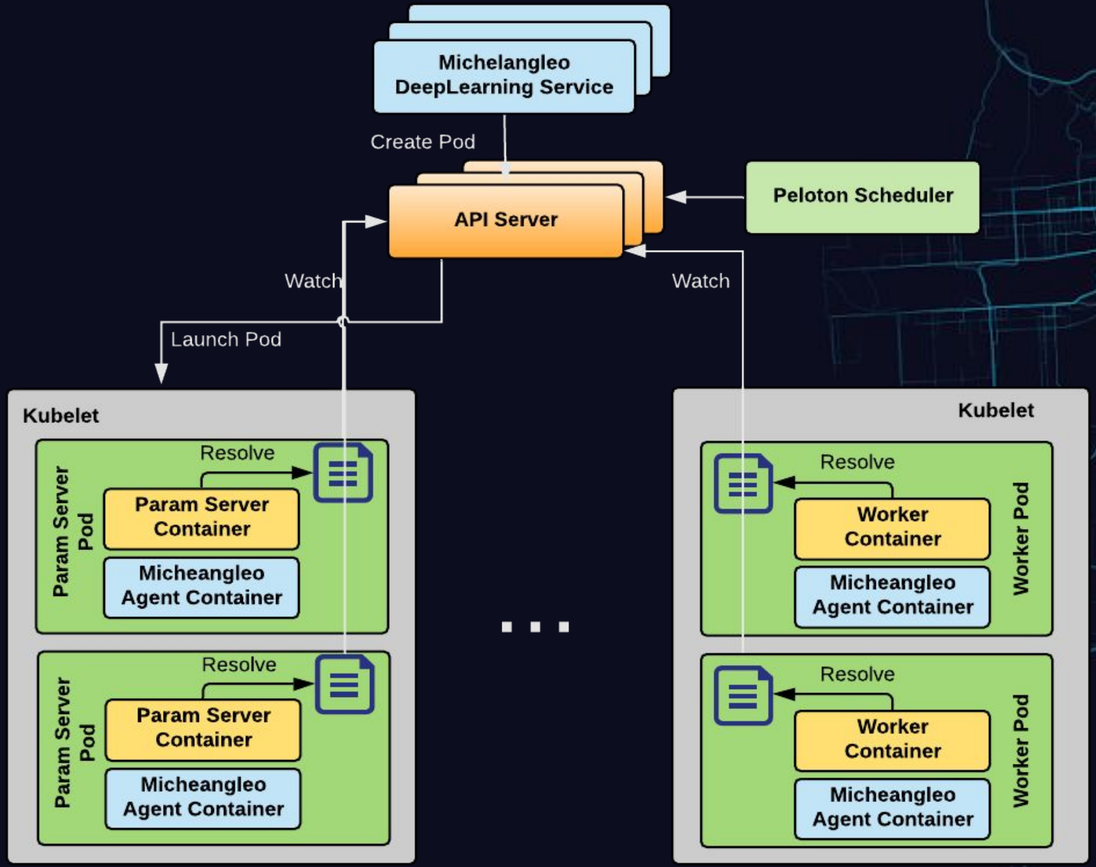
- Elastic GPU Resource Management
- Locality and Network-aware Placement
- Task Discovery
- Gang Scheduling
- Failure Handling



Gang Scheduling

- A subset of tasks in a job can be specified for gang scheduling
- Gang tasks are a single scheduling unit
- Admitted, placed, preempted, and killed as a group
- Gang tasks are independent execution units
- Run in separate containers and may fail independently
- Gang execution is terminated if a gang task fails and cannot be restarted

Distributed TensorFlow on Kubernetes

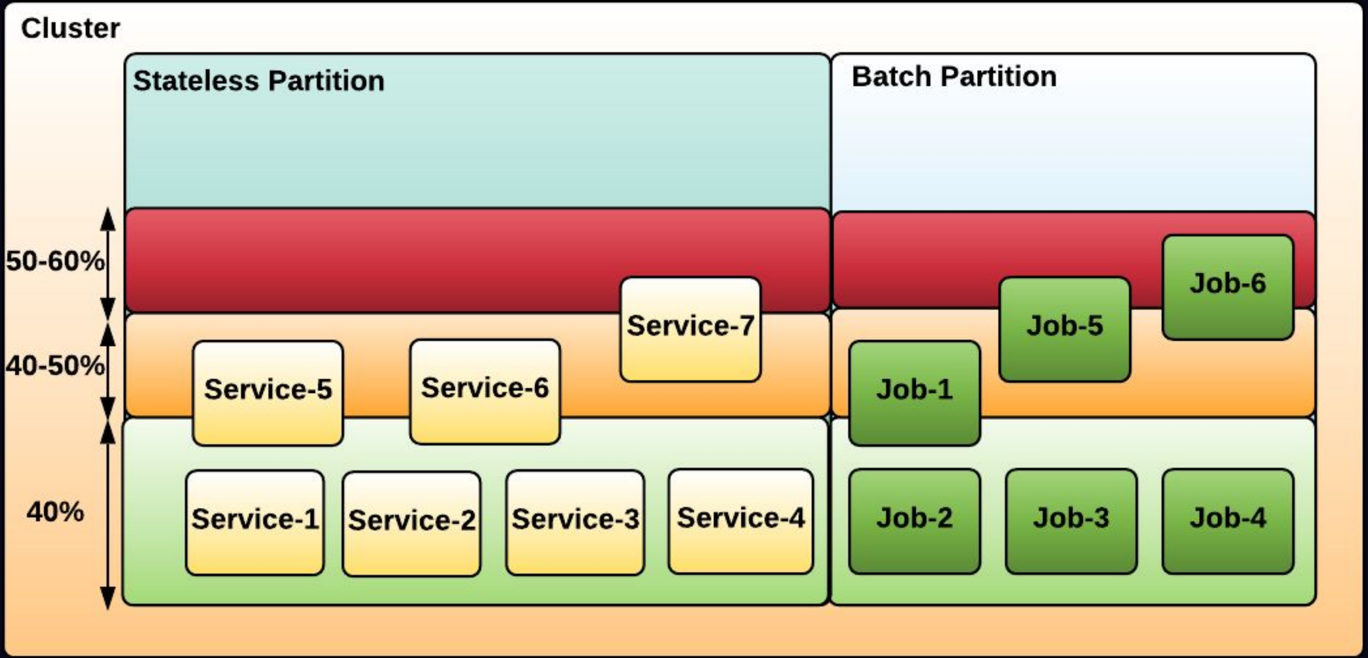


Workload Collocation

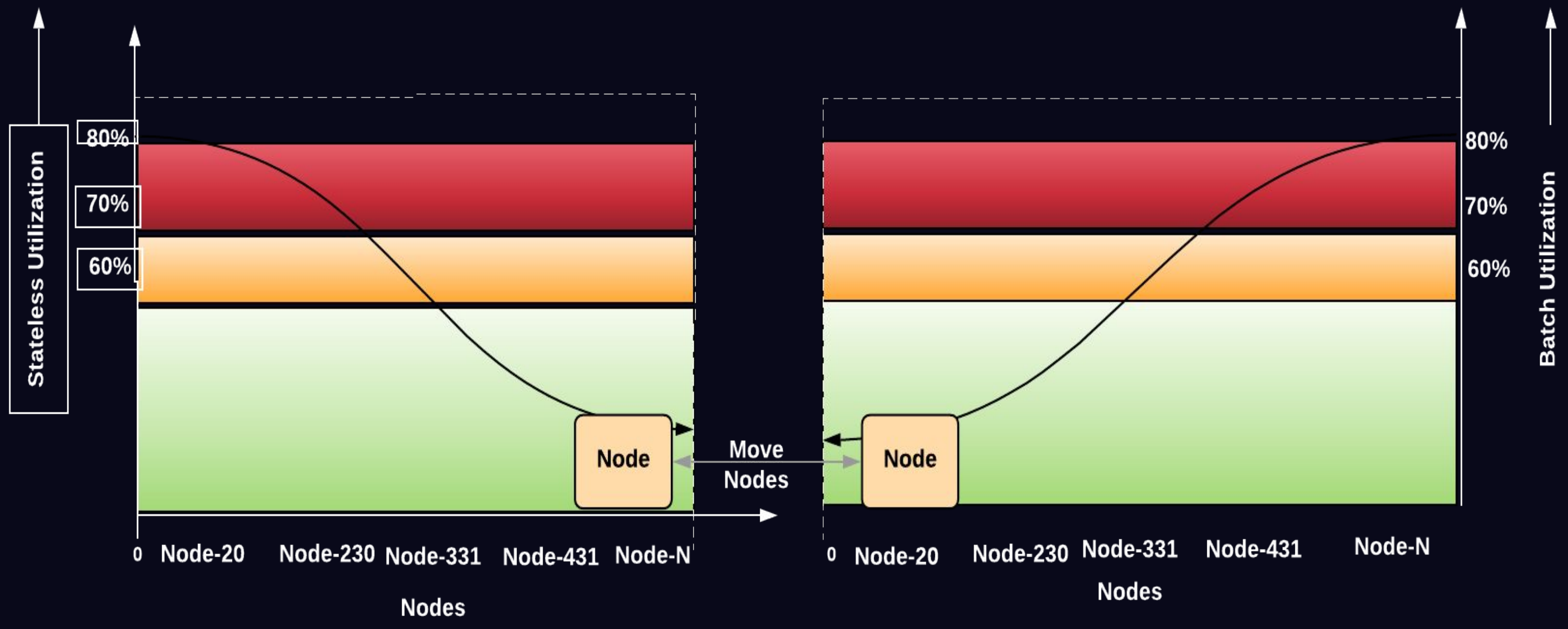
Colocating Batch and Stateless Workload

- Aim to save ~ 20-25% compute resources via collocation
- Challenges
  - Disk I/O
  - Network On Machine
  - CPU caches
  - Memory Oversubscription
- Dynamic Partitions
  - Create virtual partitions
  - Oversubscribe physical resources
  - Move machines if need to each partitions

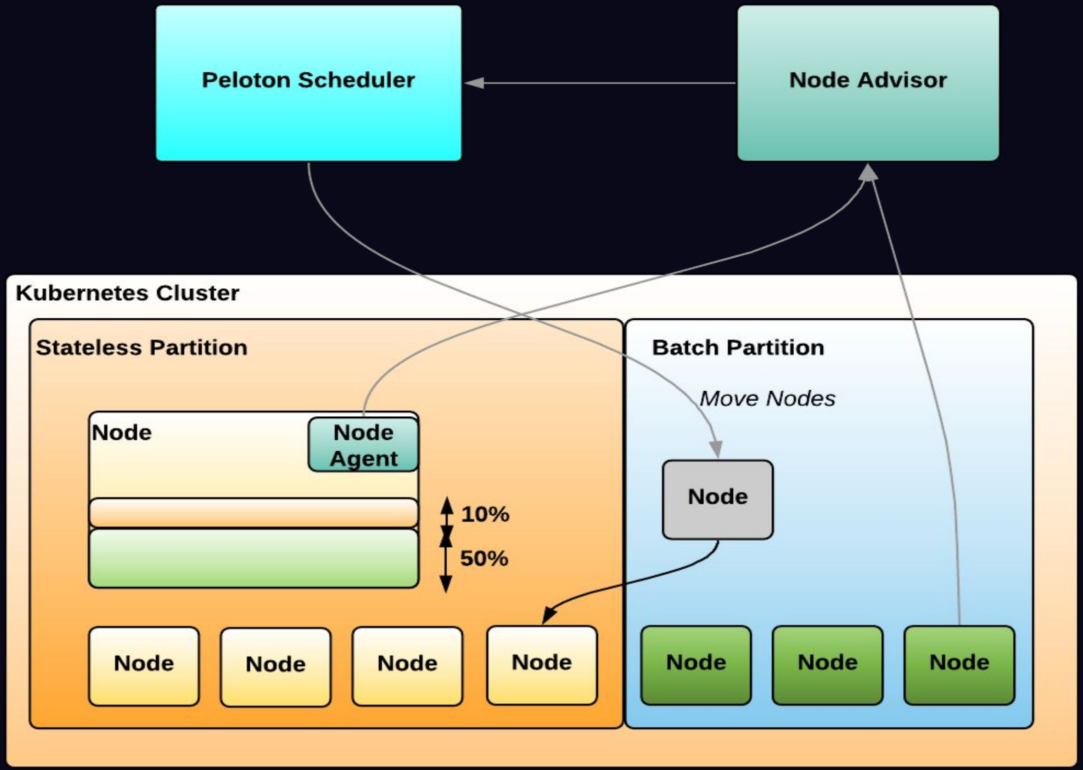
Dynamic Partition Collocation



Oversubscription



Dynamic Partition Collocation Architecture



Dynamic Partition Collocation

- Load Aware Placement
  - Not causing churn into system
- Batch and Stateless Scorers
  - Find best node to be evicted
- Virtual partition within batch partition
  - Contain imp batch jobs together
- Break Glass
  - Handle unusual spikes

Summary

- Kubernetes is the future for BigData and ML workloads
- Peloton as K8s scheduler POC is done
- Peloton on Mesos is in Production for stateless and batch
- Looking for collaboration to enable Kubernetes for BigData and ML workloads

We are hiring!

www.uber.com/careers/

Uber

eng.uber.com/peloton