FUTUREWEI Technologies

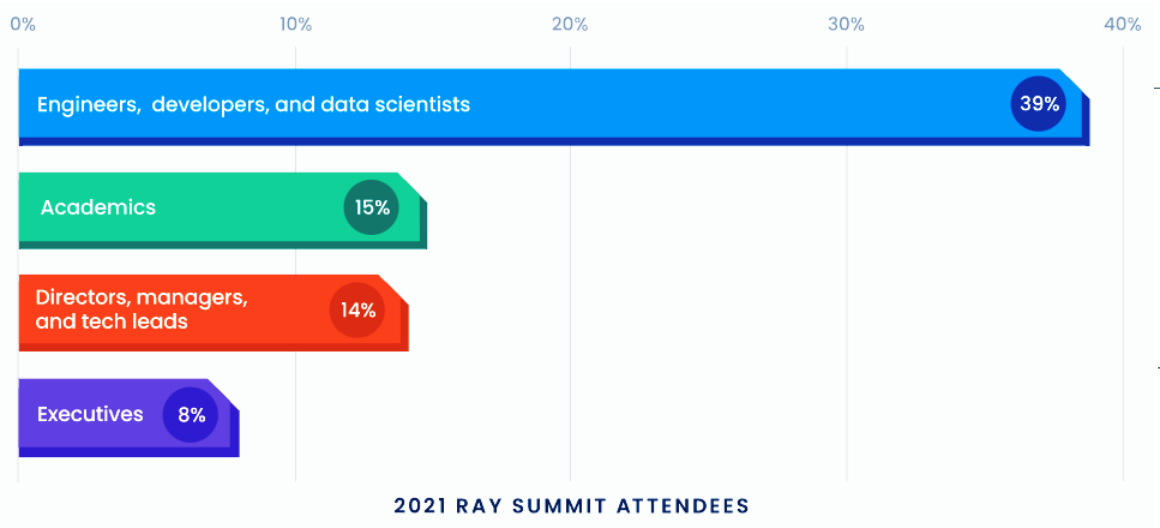
Ray Summit'22 Report

August 23-24, 2022

Norbert Egi
Infrastructure Software Lab

Ray Summit 2022

- August 23-24, 2022, San Francisco, CA
- Ray Summit is the conference for everything AI and more on Ray
- Ray Summit attendees are a mix of engineers and ML practitioners, developers, data scientists, and students and academics



PRESENTED BY

anyscale

PLATINUM SPONSORS

GOLD SPONSORS

ANT GROUP

IBM

intel

Weights & Biases

SILVER SPONSORS

MEDIA PARTNERS

aws

ByteDance

UNION

redis

AI

Technology

Selected Papers of Interest

- Introduction to Ray AI Runtime
- Distributed training with Ray on Kubernetes
- Serverless Resilient Graph Analysis with Hierarchical Persistent Storage on Ray
- KubeRay: A Kubernetes Ray clustering solution
- Highly Available Architectures for Online Serving in Ray
- State of Ray Serve in 2.0
- SkyPilot: Infra-less Machine Learning on Any Cloud
- Graph use at scale with Ray, for AI in Manufacturing
- BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

Introduction to Ray AI Runtime

Project Overview: AIR is an effort to synthesize lessons learned into a simple toolkit for the community

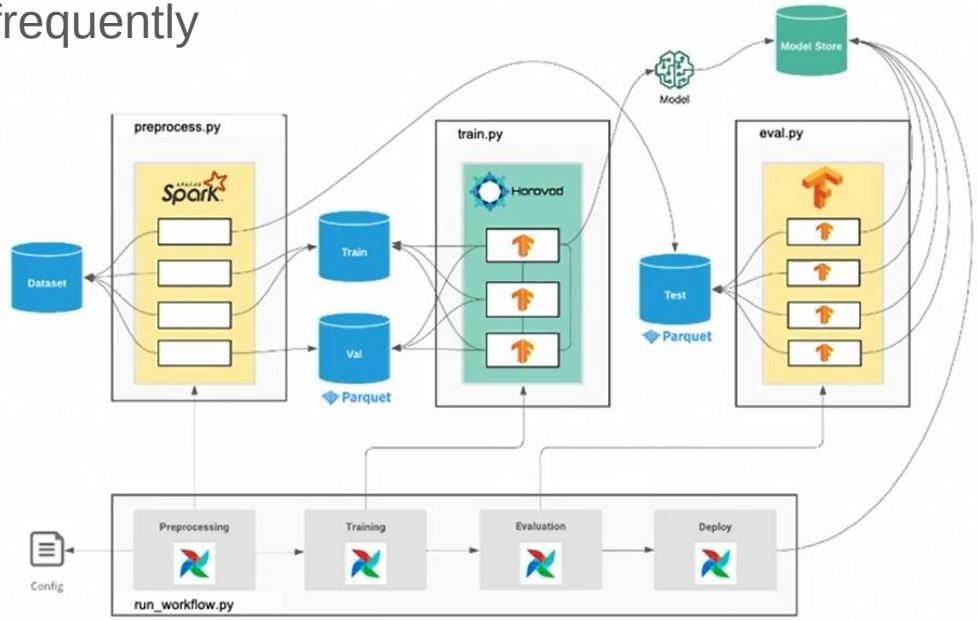
- AIR is built on Ray's existing scalable libraries
- Unified APIs for end-to-end ML
- Simplify ML Infra

Challenge:

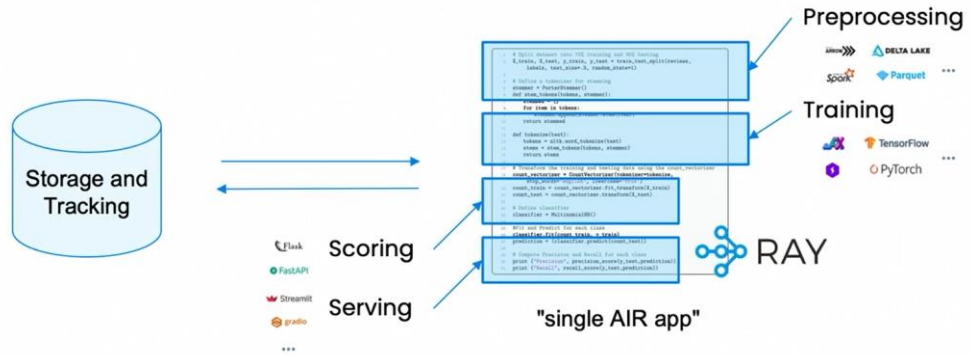
- Still not easy to take ML from development to production at scale
- ML field moves rather quickly, as such infrastructure changes frequently

Key Problems (all related to scalability)

- Scaling is hard, especially for data scientists
- Platforms solutions can limit flexibility
- Custom distributed apps are hard to build

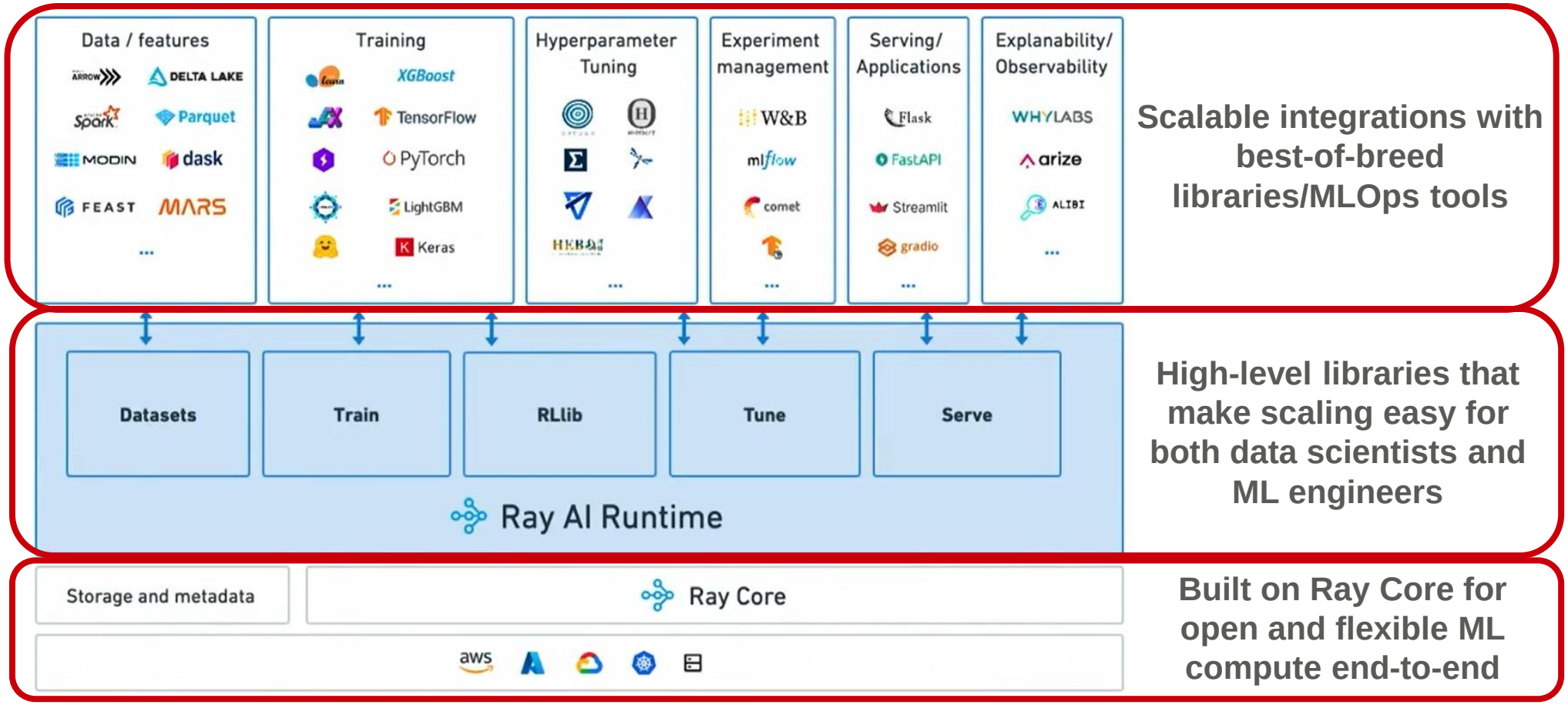


Introduction to Ray AI Runtime



What it provides:

- Scalable toolkit for end-to-end ML
- Primarily focuses on compute
- Leverages integrations for storage and tracking
- Helps ML practitioners with ease of use on today's powerful but complex ML infrastructure



FUTUREWEI Technologies

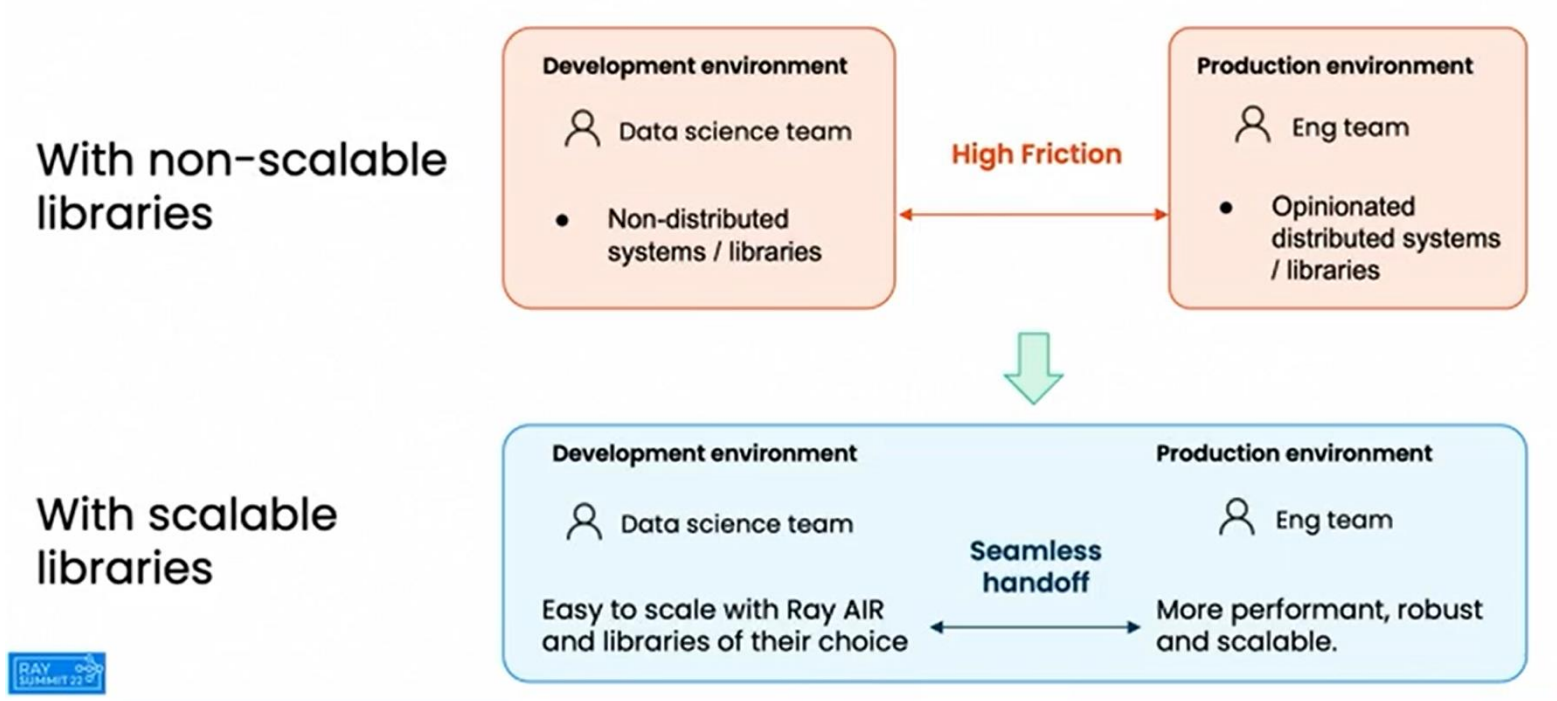
Introduction to Ray AI Runtime

Ray AIR:

- The importance of scalable library layer, which makes transferring from development to production environment frictionless even with scalable infrastructure

AIR integrations:

- Built-in integrations
- Integrations API to easily add integrations
- Custom scalable components can be built on Ray Core



Introduction to Ray AI Runtime

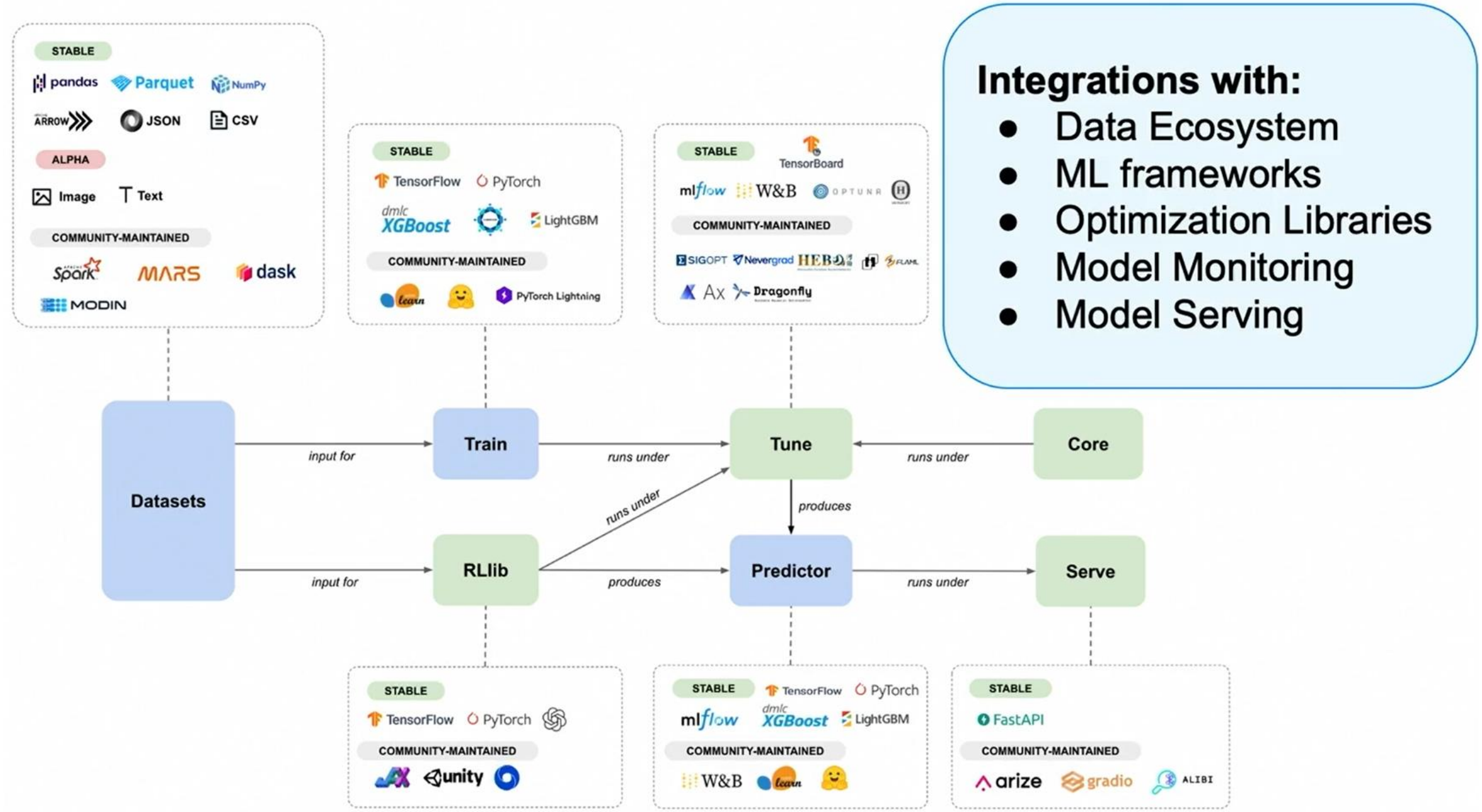
When to use Ray AIR:

- Scale a single type of workload (mostly for Data Scientist)
- Scale end-to-end ML applications (mostly for Data Scientist)
- Run ecosystem libraries using a unified API (mostly for ML Engineer)
- Build a custom ML platform (mostly for ML Engineer)

Available libraries in a nutshell

- Scalable Data Preparation and Loading (with Ray Data)
- Scalable Model Training (with Ray Train)
- Scalable Hyperparameter Tuning (with Ray Tune)
- Scalable Batch Prediction (with AIR's BatchPredictor)
- Scalable Online Inference (with Ray Serve)

Introduction to Ray AI Runtime



Distributed training with Ray on Kubernetes

Pain points:

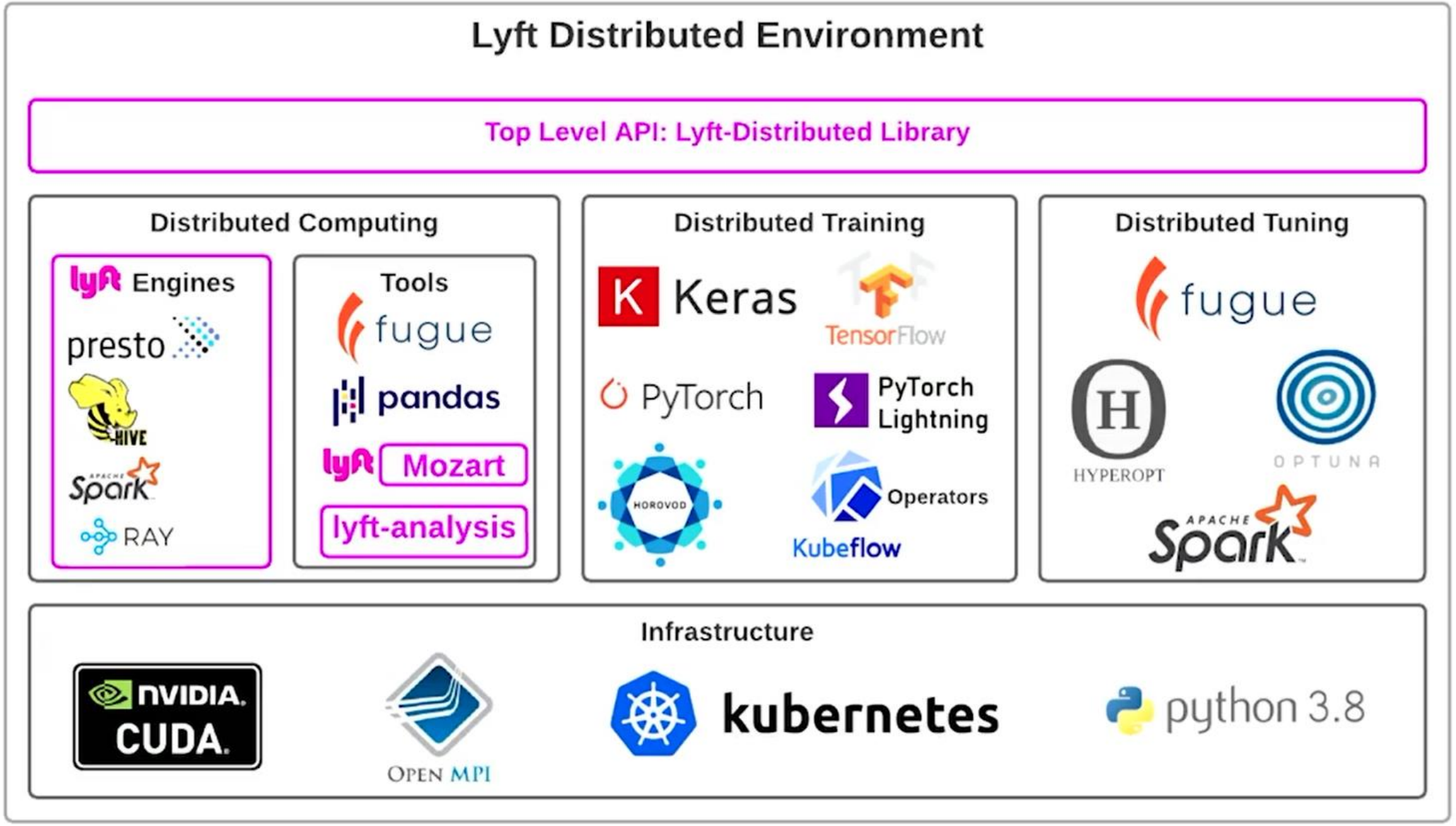
- Most of business decisions are empowered by ML models (i.e. price optimizations, safety, ETAs, fraud detection, mapping, incentives, etc.)
- Lot of well-written code is unscalable
- The scalable poorly-written logic
- Single vendor lock-in (majority of users are familiar with Spark only and use it almost exclusively, however Spark does well in pre- and post-processing, but cannot do the training step well)
- The cost of iterations (it increases drastically with the increase of data size and model complexity)
- Following good practices are sometimes penalized (i.e. require a huge amount of effort)

Design Principles:

- Ease and seamless to use, layered-cake approach
- Abstracting away the concern of scale and vendor lock-in
- Enabling fast iterations (save time and cost) → must provide a large collection of utility functions
- Making following good practices rewarding

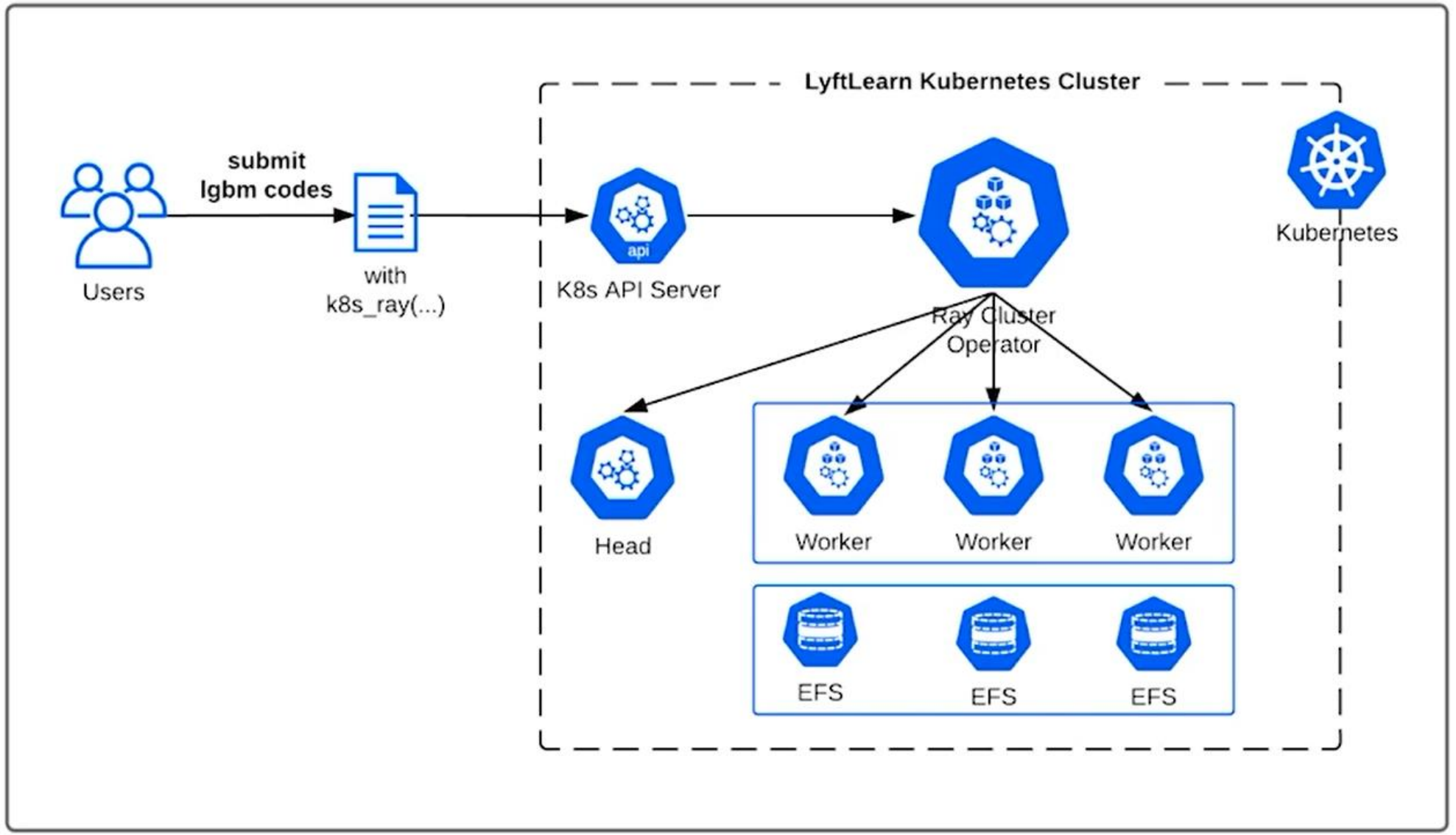
Distributed training with Ray on Kubernetes

Lyft Distributed Environment



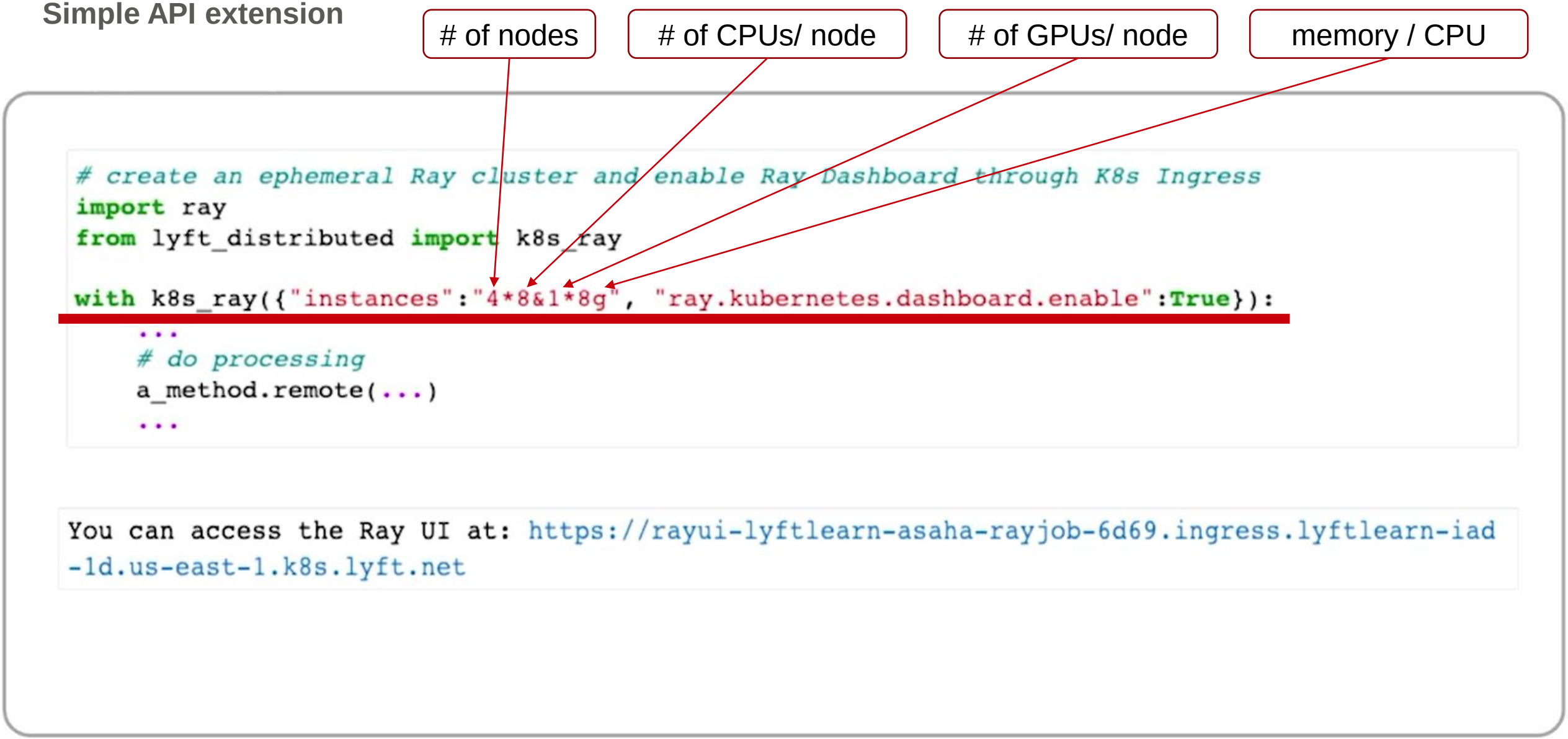
Distributed training with Ray on Kubernetes

Distributed LightGBM on Ray



Distributed training with Ray on Kubernetes

Simple API extension



Serverless Resilient Graph Analysis with Hierarchical Persistent Storage on Ray

Intro / Landscape:

- Graphs are relational data structures that can define information collectively by using the node-edge structures in a non-linear way
- Wide adoption at ByteDance: Thousands of graph computing jobs everyday (e.g. PageRank, Connected components, Betweenness centrality)
- All runs on super large graphs (i.e. 10s of billions of vertices and 100s of trillions of edges)
- Previous graph-processing system was based on Plato, an open-sourced Pregel-like system
    - Enhancements made:
        - More graph algorithms implemented
        - uint64 vertex IDs to support larger graphs

Major issues

- **High cost:** memory-bounded, nodes-consuming
- **High maintenance:** frequent failures with MPI, no states recovery
- **Hard to program:** mix of pull-push lambda functions

Motivations / Opportunities:

- Hierarchical storage (e.g. NVMe, SSD): In-memory extended by out-of-core
- Fault-tolerance on cloud: End-to-end failures recovery / Persist checkpoints
- Simplified APIs: One UDF with Python APIs

FUTUREWEI Technologies

Serverless Resilient Graph Analysis with Hierarchical Persistent Storage on Ray

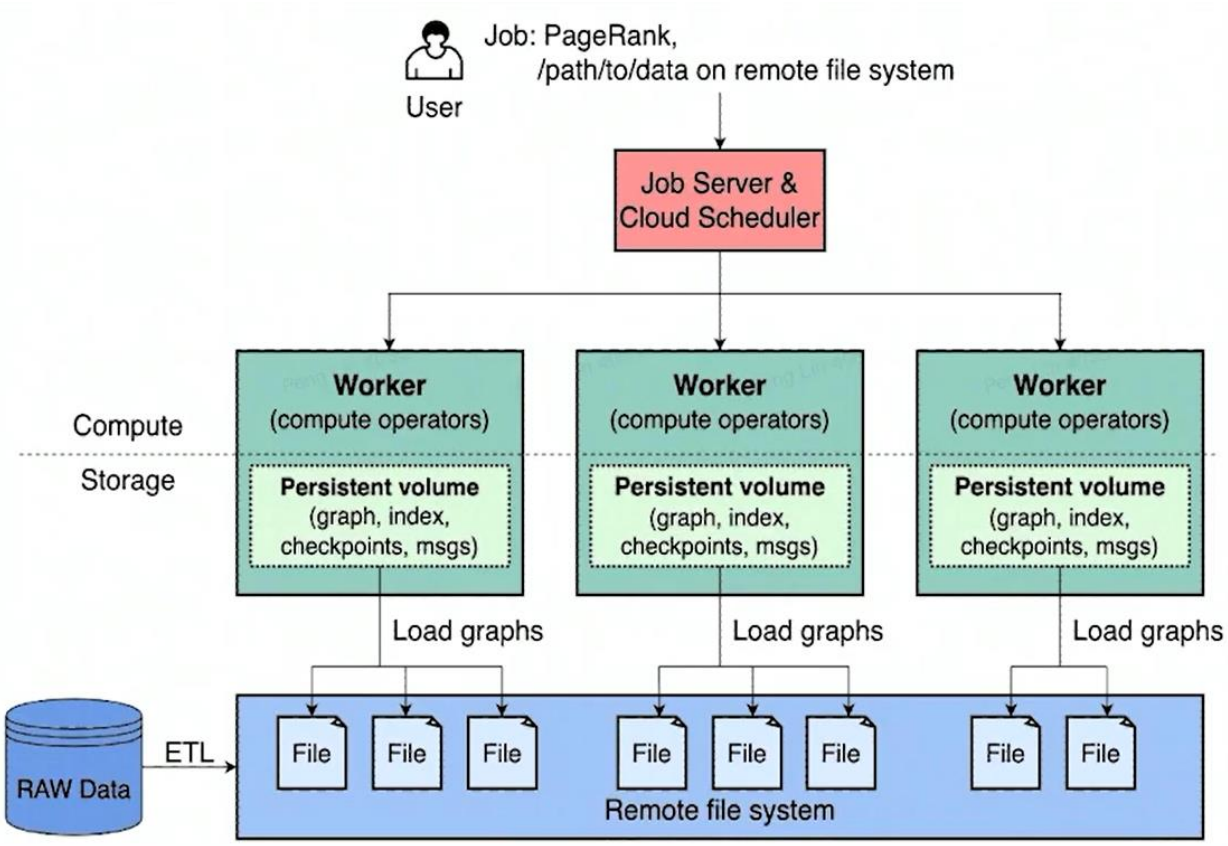
ByteGAP (Graph Analytics Platform)

Worker: implement graph algorithms and communicate through MPIs

Persistent Volume: hierarchical storage of DRAM/PMEM/SSD

Job Server/Cloud Scheduler: built on a serverless engine atop Kubernetes to provide:

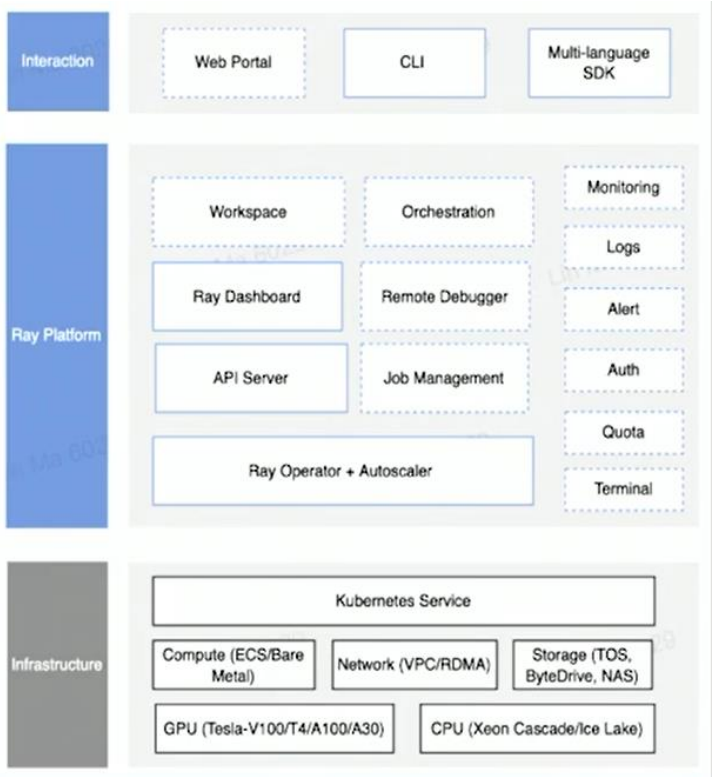
- Flexible cluster resource mgmt.
- Automatic deployment
- Decouple compute/storage
- E2E Fault tolerance
- Heterogeneous storage

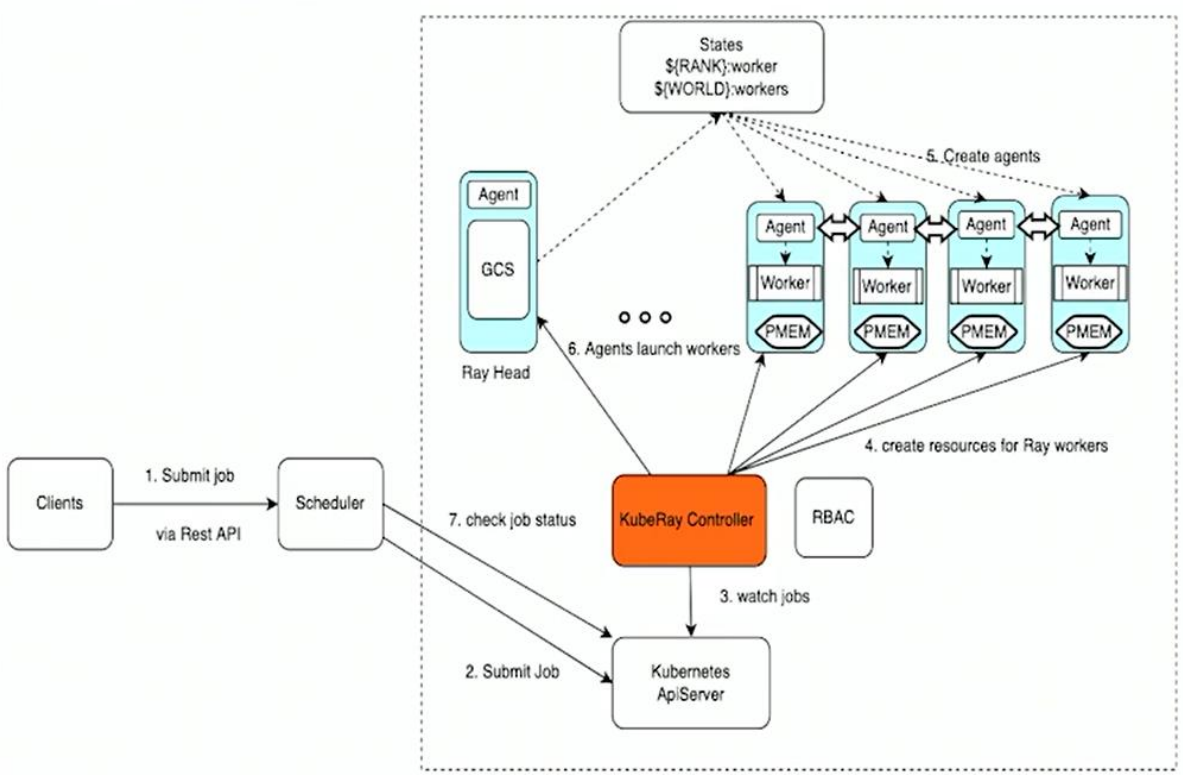


Serverless Resilient Graph Analysis with Hierarchical Persistent Storage on Ray

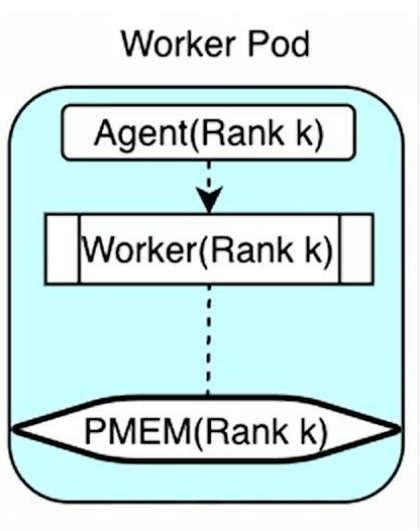
Deployed on Cloud using KubeRay

- RayElastic: newly-built in-house infrastructure components
- Autoscale: from single node to anyscale
- Powerful abstraction of distributed system & programming model
- Rich ecosystem of libraries and frameworks built on top of Ray





Serverless Resilient Graph Analysis with Hierarchical Persistent Storage on Ray



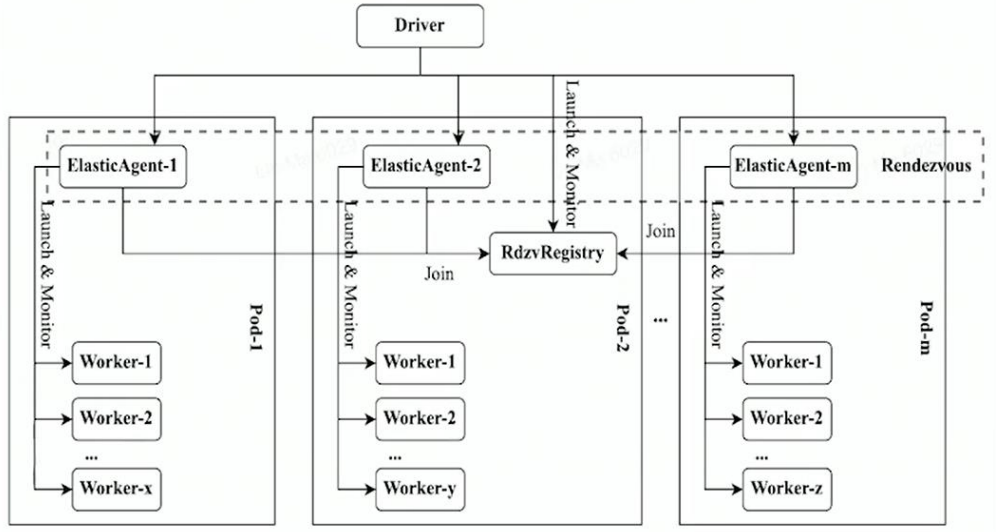
RayElastic: Ray-based Fault-Tolerance Control Plane

- Agent-based control plane for rank management and stateful fault tolerance
- Covers end-to-end fault-tolerance
- Handles failure end-to-end at 3 levels with synergy that MPI can not fully cover
- Relaunch agents/workers/storage of specific rank
- Ensure automatic recovery from any iteration for any workers of arbitrary ranks

General Single Program Multiple Data (SPMD) fault tolerance control plane as a **Ray Infrastructure Component**

ElasticAgents negotiate (rendezvous) and provide stable rank with arbitrarily-ranked node recovery, bind associated volume with states and sharded local data on it

RdzvRegistry keeps state of each agent, e.g. ranks, mapping of rank to persistent volume through pods



KubeRay: A Kubernetes Ray clustering solution

Goal:

- To have a compute platform that's capable of handling machine learning workloads with the scalability of Ray itself
- Kubernetes schedules pods & containers, but it's doesn't understand the notion of Ray cluster (i.e. head node, worker node and how they should be instantiated and started up)

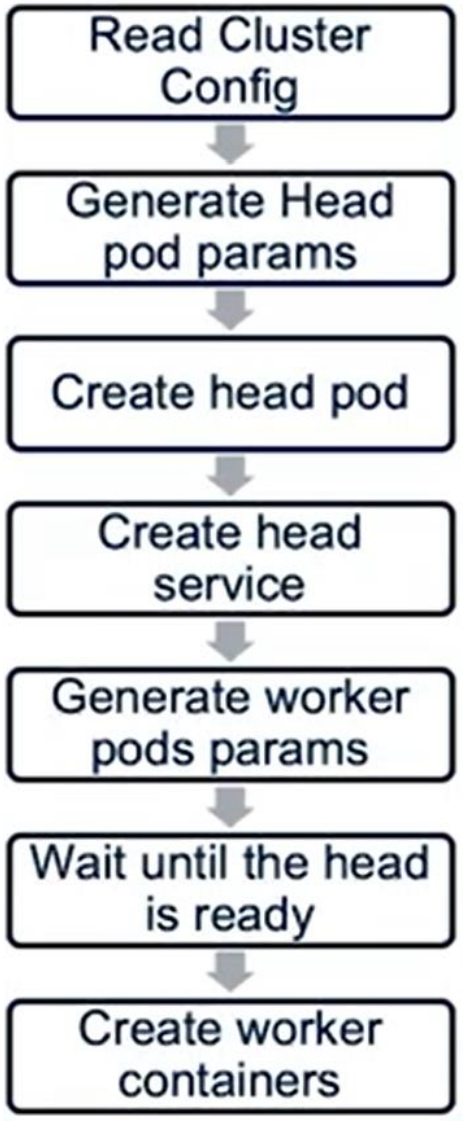
KubeRay:

- Extended Kubernetes with the knowledge of Ray clusters (i.e. creates/updates/deletes Ray clusters)

Why not just a K8s deployment?:

- The worker pods can have different resources (e.g. GPU)
- Each worker group can be performing a different functionality
- New worker groups can be added and removed
- For scaling down, specific idle worker pods need to be targeted for removal
- The head pod needs to start first and the workers need to connect to it afterwards

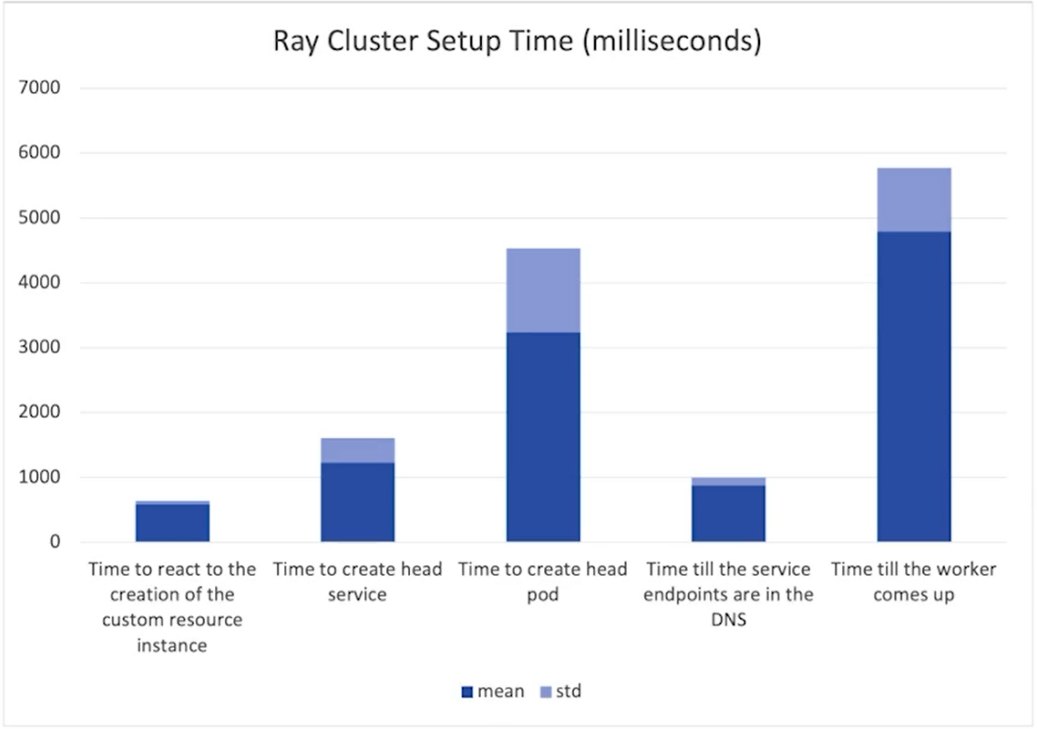
KubeRay Cluster creation



KubeRay: A Kubernetes Ray clustering solution

Performance Evaluation

- Total cluster creation takes about 10 seconds



Ray Cluster Management

The component(s) that is *managing* the lifecycle **and** *auto-scaling* a Ray cluster, must be capable of performing the following tasks:

Life-Cycle Management

1. *Create a Ray cluster*
2. *Delete a Ray cluster and cleanup all the relevant resources*
3. *Failure recovery (e.g. if a physical machine fails, create the Ray nodes on another machine(s))*
4. Creating additional nodes and deleting idle nodes.

Auto-scaling

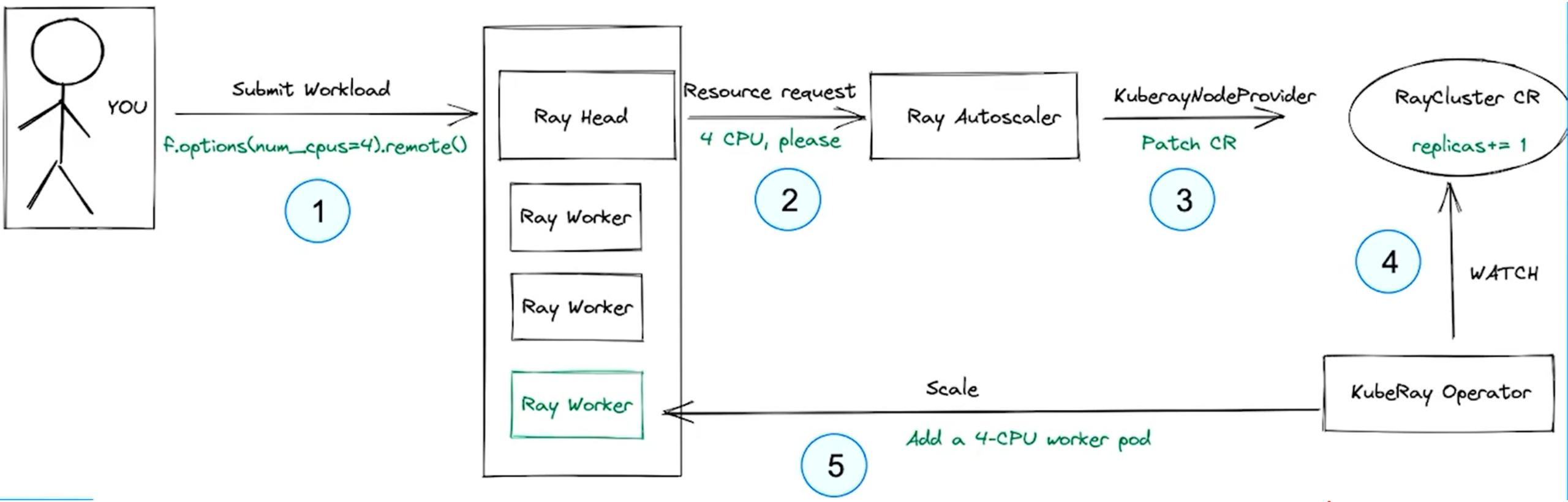
5. *Metrics collection: determine the CPU, Mem, GPU, TPU utilization in the cluster (currently there is only one threshold supported). And determine the idle nodes.*
6. *Making a scaling up/down **decision** based on the metrics collected*

FUTUREWEI Technologies

KubeRay: A Kubernetes Ray clustering solution

AutoScaling – Automatic scaling based on Ray application semantics!

- Autoscaler reads load metrics from Ray
- KuberayNodeProvider patches RayCluster CR scale to size your cluster



FUTUREWEI Technologies

Highly Available Architectures for Online Serving in Ray

Highly Available Serving Requirements

- High throughput, low latency
- Observable
- Available
  - The system should continue to serve traffic without significant drop in error rate (e.g. 99.5% success rate even when any component goes down)
  - Component loosely defined
    - Application error (code OOM, randomly segfaults)
    - Internal component (RayServe system component went down, Ray went down)
    - External dependencies (node upgrade, node went down, AZ unavailable, network failure)

Challenges / HA Architecture in Ray 2.0:

- Application Error
  - E.g. Nondeterministic Segfault
  - **Solved by Serve actively health check and restart**
- Internal Component
  - E.g. Component X went down
  - **Solved by component by component fault tolerance implementation**
    - **GCS Fault Tolerance**
- Dependencies Issue
  - E.g. Node crash
  - **Solved by the Ray Kubernetes operator**

FUTUREWEI Technologies

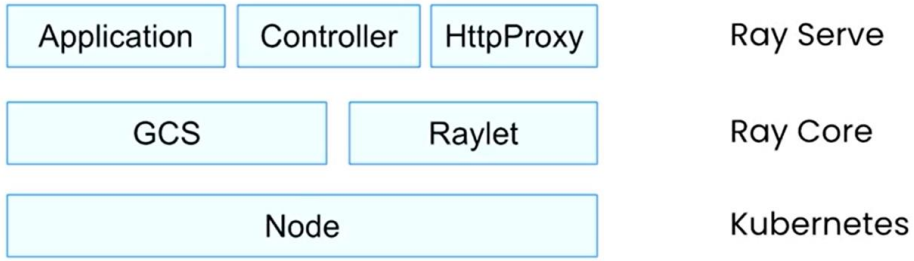
Highly Available Architectures for Online Serving in Ray

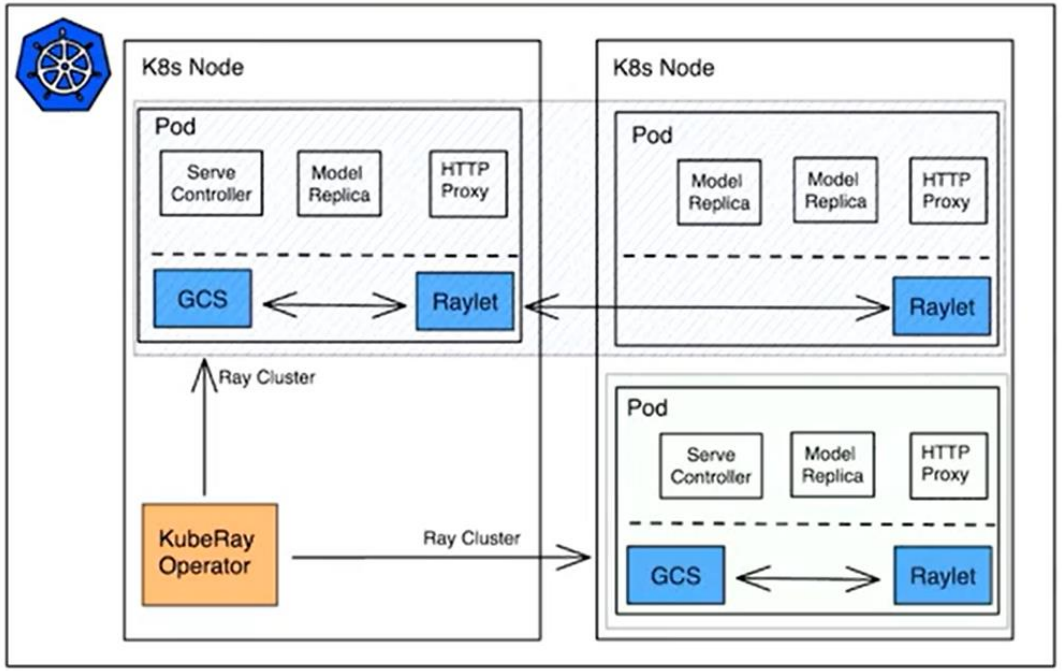
Architecture Overview

- **Layered Architecture**
  - Three layers, many components
    - Ray Serve: user code, serve controller, http proxy
    - Ray Core: raylet, gcs
    - Kubernetes: nodes, cluster operator
  - Many types of failures
    - We want to handle them all while still being able to serve traffic

Future Roadmap

- ***Bring HA architecture to GA***
  - Alpha release and ready for dogfooding
- ***Bring HA architecture to other Ray Libraries***
  - E.g. training tuning, data processing
  - Currently the Ray FT availability is tightly scoped to support Serve
- ***Better HA Redis Integration***
  - SPOF is now shifted to HA Redis owned by you





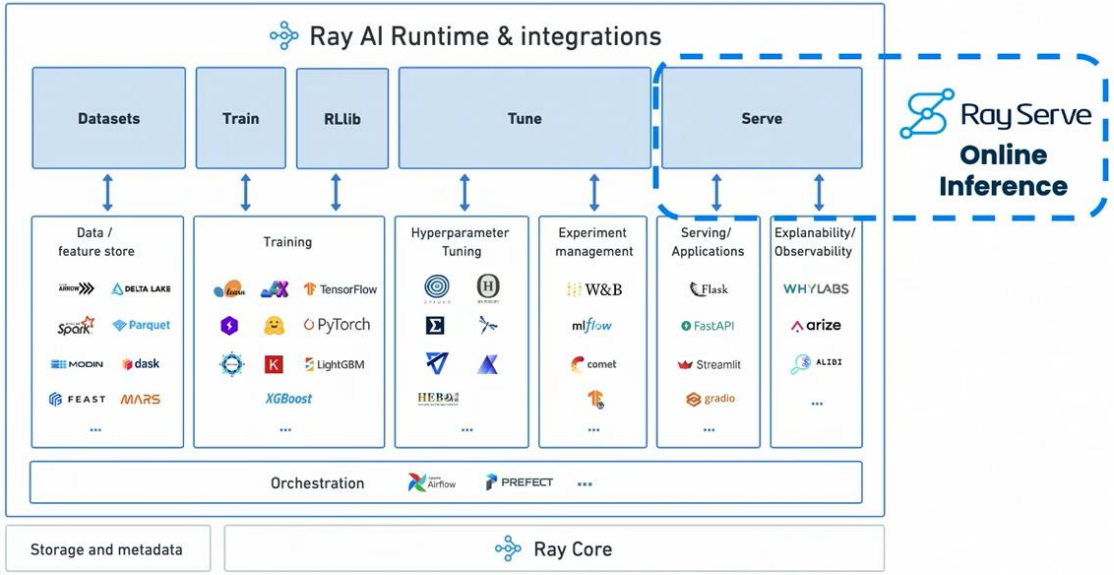
Availability while recovering

Component	Deploy/Reconfig	**Serving**	Recover time
Serve Replicas	✓	✓	Seconds
HttpProxy	✓	✓	Seconds
ServeController	✗	✓	Seconds
Raylet	✓	✓	Minutes
GCS	✗	✓	Minutes
Node	✓	✓	Minutes

State of Ray Serve in 2.0

Overview

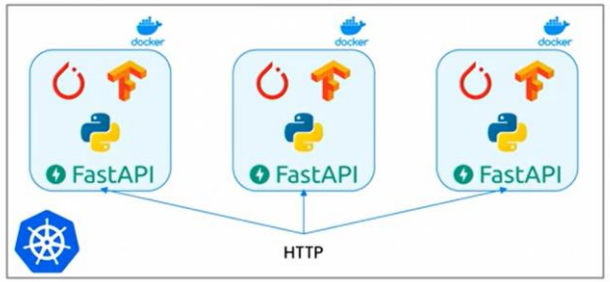
- Ray Serve is part of Ray AIR, which is a collection of libraries that offers a unified API and runtime to build end-to-end AI applications
- **Flexible, scalable, efficient** compute for online inference
- It's built on top of Ray, so it inherits its benefits
- First-class support for multi-model inference
- Python-native: users can easily mix business logic & machine learning using all the favorite tools and libraries to build online inference APIs



Requirements for Online Inference (really about the infrastructure and compute layer)

- Minimize **latency**, maximize **throughput**
- Fine-grained resources: **fractional CPUs & GPUs**
- **Programmatic API** for development and testing
- Rock-solid story for **production**

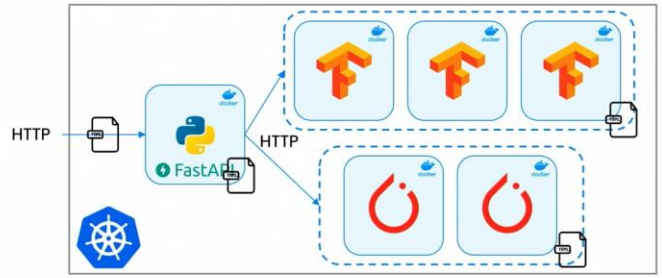
Basic Solution: Multi-model Monolith



❌ Unable to scale models independently!

😐 High latency, low throughput, and costly

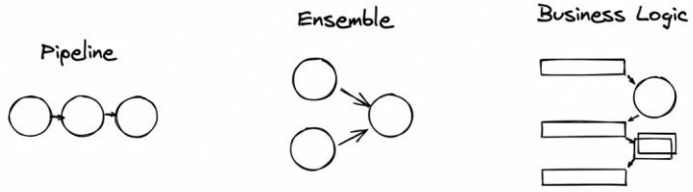
Complex Solution: Models as Containers



✅ Scale models independently, use different resources

❌ Requires deep Kubernetes knowledge to build an app

Ray Serve is built for Multi-model Inference



Ray Serve

Write a single Python program

Use your favorite tools & libraries

Scale across CPUs and GPUs

State of Ray Serve in 2.0

Overview

- Whole end-to-end serving function chain is a single Python program
- Developed and tested locally (like on a laptop)
- Ultimately be deployed on a cluster & yet still updated as a single app

Goal: Make it easy to put scalable ML in production

- Great UX for **flexible model composition**
- Improved efficiency and save costs with **advanced autoscaling**
- **Production hardening**, focused on Kubernetes

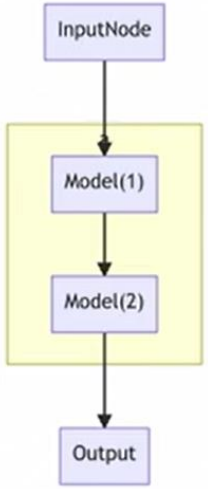
Model Composition API Requirements

- Flexible to satisfy diverse use cases
  - Different models, frameworks, and business logic
- Ability to develop, test, and debug locally
- Scalable and efficient when running in production

Solution: Model Composition API

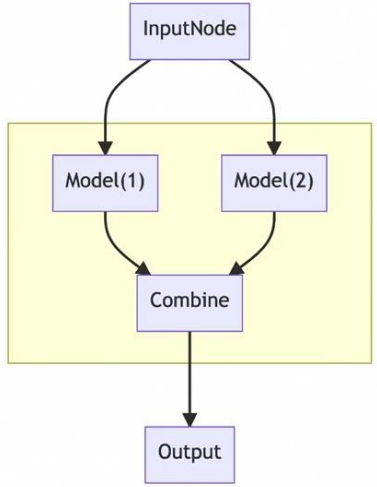
- First-class API to build *graphs* of Serve *deployments*
- Full flexibility of Ray Serve
  - Author, configure, scale each model independently
- Orchestrate computation using regular Python code

Model Composition Pattern: *Chaining*




```
m1 = Model.bind(model_v1)
m2 = Model.bind(model_v2)

with InputNode() as input:
    m1_out = m1.forward.bind(input)
    m2_out = m2.forward.bind(m1_out)
```

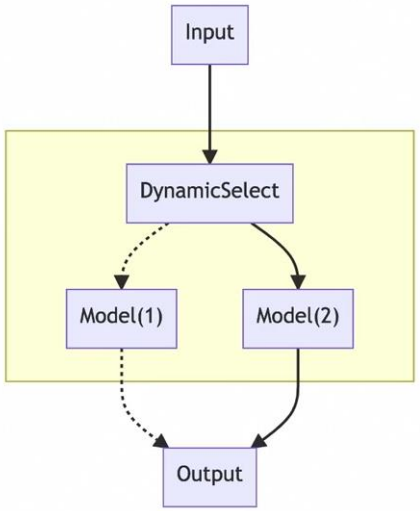
Model Composition Pattern: *Ensemble*




```
m1 = Model.bind(model_v1)
m2 = Model.bind(model_v2)

with InputNode() as input:
    m1_out = m1.forward.bind(input)
    m2_out = m2.forward.bind(input)
    combined = combine.bind(m1_out, m2_out)
```

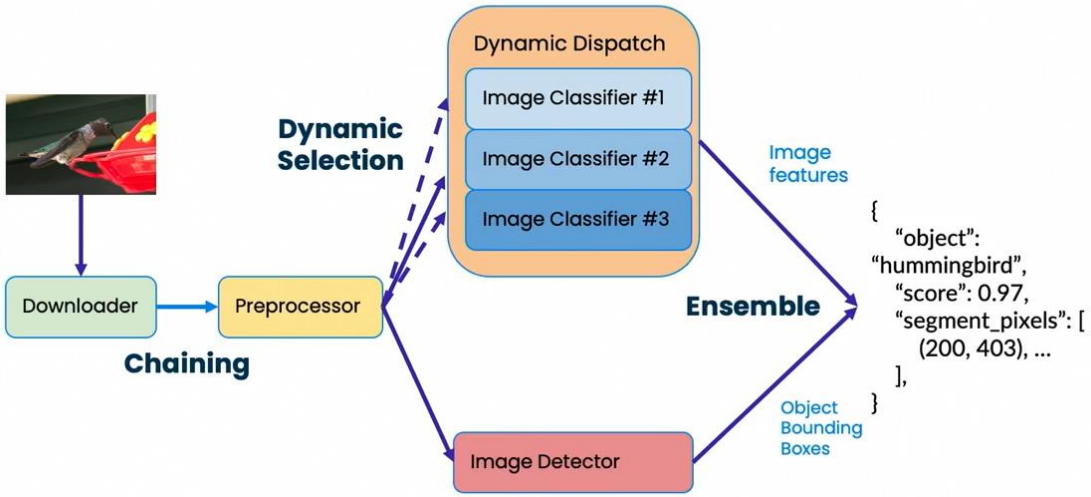
State of Ray Serve in 2.0

Model Composition Pattern: *Dynamic Selection*




```
m1 = Model.bind(model_v1)
m2 = Model.bind(model_v2)
selector = Selector.bind(m1, m2)

with InputNode() as input:
    dynamic_output = selector.forward.bind(input)
```

Deployment Graph API Enables *Flexible Model Composition*



Ray Serve Model Composition API

- *Write* your models as **ordinary classes**
- *Flexibly* compose models & logic w/ **Python code**
- *Run, test, and debug* **on your laptop**
- *Deploy to production* – configure and **scale models independently**

Autoscaling for ML Models

- **Problem**: ML models are compute intensive
  - Not all models are always used
  - Hard to tune hardware utilization
  - Needs to work for multi-model
- **Solution**: Advanced autoscaling for Serve
  - Supports scale-to-zero
  - Uses request queue lengths, no profiling
  - Fully compatible with model composition API

State of Ray Serve in 2.0

Production Hardening

→ Online inference means solving operational problems:
- \+ Updates without downtime
- \+ Handling failures gracefully
- \+ Monitoring, observability, alerting

Operational Benefits

Flexibility, User Experience, Efficiency

Production Hardening: GCS Fault Tolerance

Ray can now **recover from GCS failures** in version 2.0

→ Tasks and actors continue to run
→ A new GCS is started and the cluster is recovered
- \+ Handled automatically by k8s operator

Ray Serve applications **continue to serve traffic**

Ray Serve in 2.0

→ **Goal**: Make it easy to put scalable ML in production

+ Great UX for **flexible model composition**

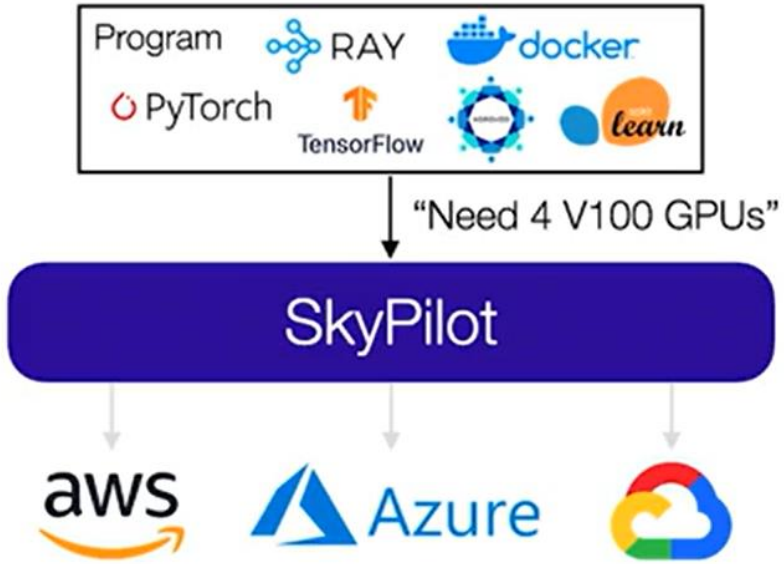
+ Improved efficiency and save costs with **advanced autoscaling**

+ **Production hardening**, focus on Kubernetes

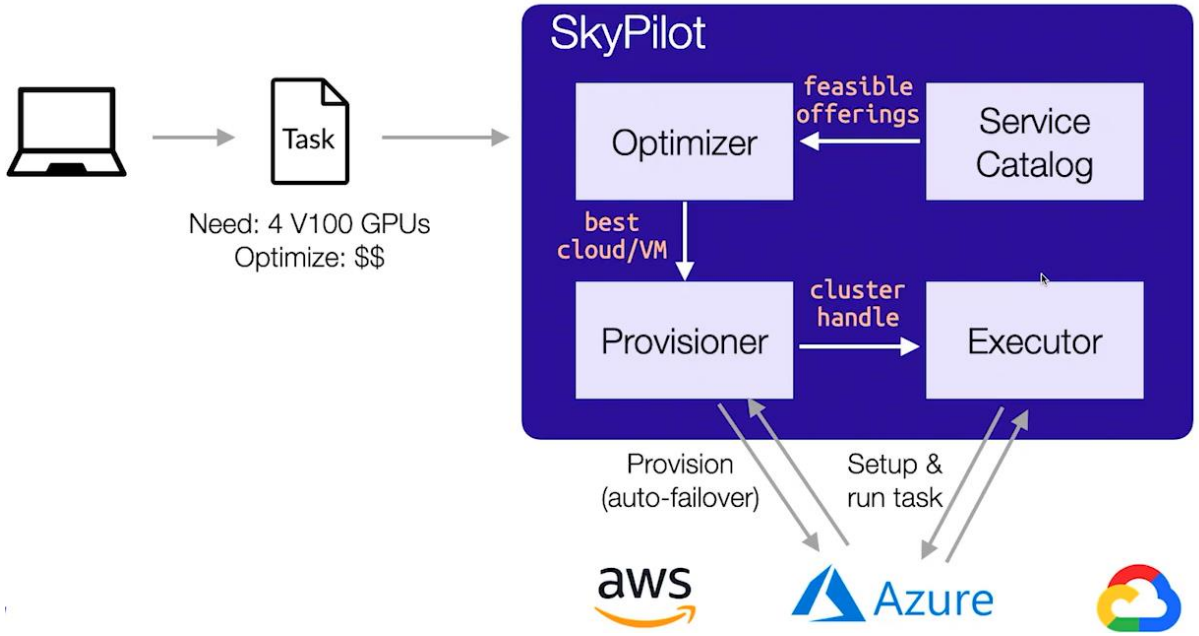
SkyPilot: Infra-less Machine Learning on Any Cloud

First intercloud broker for the Sky (formerly was called SkyML)

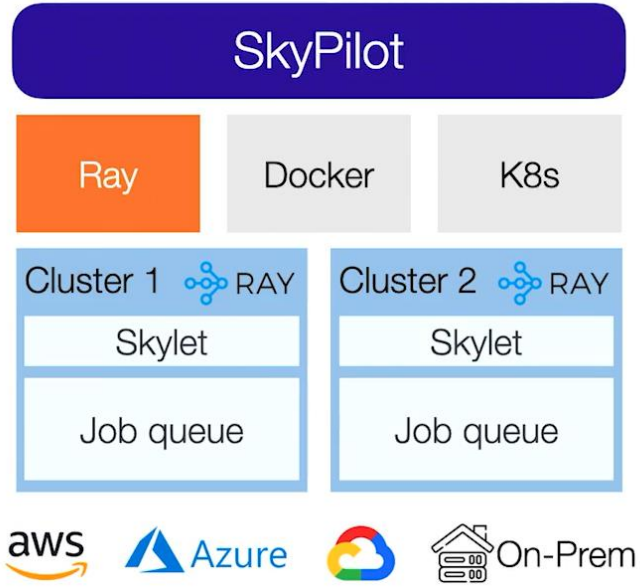
- It enable **ease of use** and **cost saving**
- First, users submits a program to SkyPilot, which can be written in any of the supported ML frameworks
- User also submits the resource requirements (e.g. number and type of GPUs)
- Based on these information SkyPilot does some optimization and decides on the best cloud to use



Life of a Task



Powered by Ray



Ray Clusters: cluster lifecycle

Ray Task: remote execution

Ray Placement: resource allocation

Ray Job: job management

SkyPilot: Infra-less Machine Learning on Any Cloud

SkyPilot's ease-of-use solution to the complexities of running ML in the Cloud

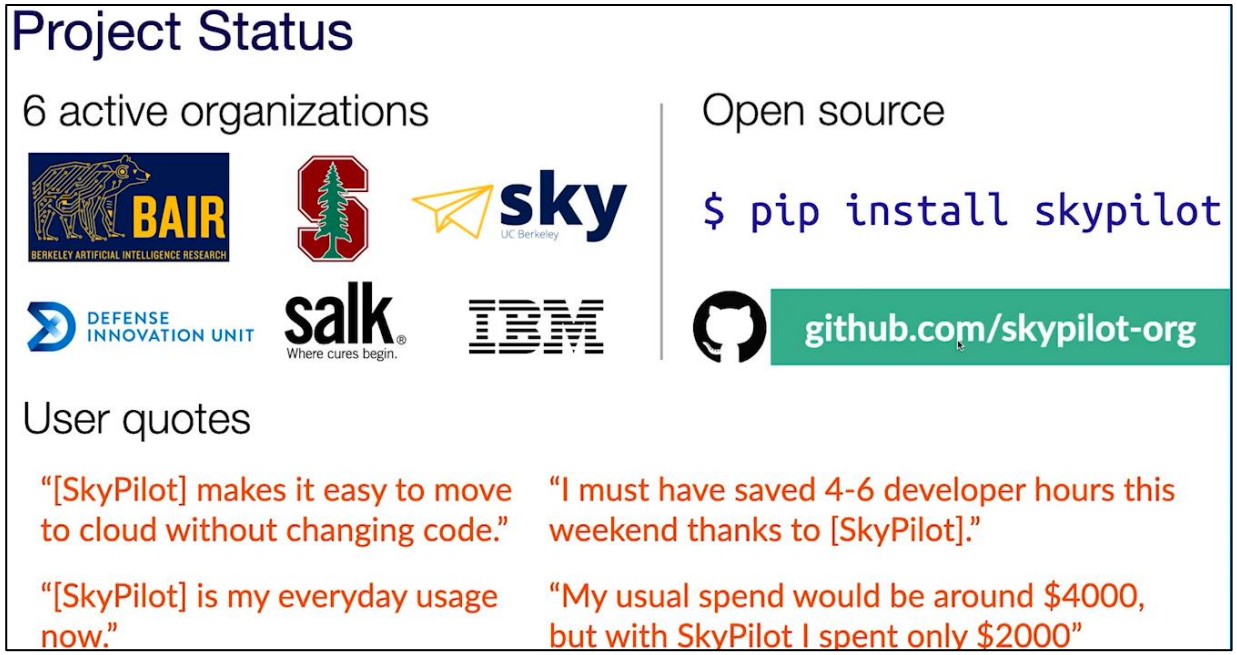
- **Too many choices** → Optimizer, Benchmark to decide on the best location based on the hardware requirements and availability
- **Resource (un)availability** → Auto-failover in the Sky
- **Varying APIs across clouds** → Unified task interface
- **Infra management burden** → Job queue, Auto-stop, Storage

SkyPilot Storage (unified storage interface)

- Make data available whenever a job runs
- Supports standard POSIX API

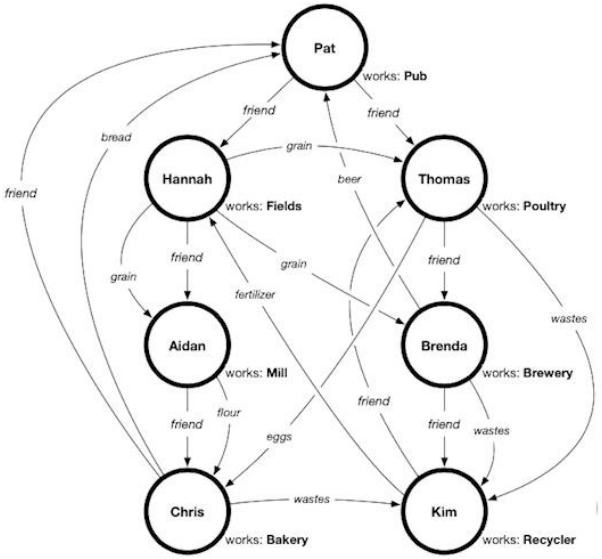
How cost is saved in the Clouds

- Auto-pick cheapest cloud/VM for requested resource
- Also takes into account data transfer costs
- Also provides a benchmark feature
- Also possible to specify candidate resources
- Managed spot-instances can further reduce cost (also supports auto-recovery from preemptions to overcome availability challenge posed by spot instances)



FUTUREWEI Technologies

Graph use at scale with Ray, for AI in Manufacturing



Foundations of Scalable Graph Compute

- **Graph Thinking**: For complex contexts, network views bring the data closer to people who can make sense of it
  - Acknowledge the complexity of the context
  - Identify emergent patterns
  - Make informed decisions
- "**Normalization**" of graph data into relational databases as a primary mode of interaction will make the **perception of patterns lost** for domain experts
- Recommended literature:
  - "Graph Thinking"
  - "Why interest in graph databases and graph analytics are growing"
- **Industry Analysis**
  - Started to pick interest since Feb 2021 (Gartner)

Summary: "Cheat Sheet"

examples

Business Contexts	Modes of Learning	
Simple: establish facts, categorize needs, apply rules	**Novices:** disconnected facts; rules which must be followed	*reporting: "known knowns"* probably uses a **spreadsheet**
Complicated: assess facts, develop analysis, select responses	**Analysts:** bets on predictions about the future	*prediction: "known unknowns"* probably uses a **data lake**
Complex: probe situations, sense patterns, informed decisions	**Expert Leaders:** descerns when to break the rules, and why	*prescription: "unknown unknowns"* probably needs a **large graph**

Graph use at scale with Ray, for AI in Manufacturing

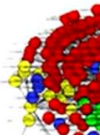
Primary sources

"A Brief History of Knowledge Graph's Main Ideas: A tutorial"
Claudio Gutierrez, Juan F. Sequeda
CACM (2021)

"Knowledge Graphs"
Aidan Hogan, et al.
arXiv (2020)

"Ontology Development 101: A Guide to Creating Your First Ontology"
Natalya F. Noy, Deborah L. McGuinness
Stanford KSL-01-05 (2001)

MS 514: "Existential Graphs"
Charles Sanders Peirce
New Elements of Mathematics (1882)



Foundations of Scalable Graph Compute

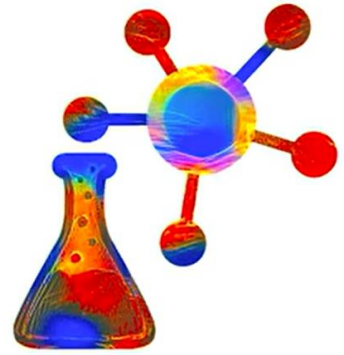
- **"Knowledge Graph" (KG)**
  - Each **entity** within a graph has a name and some attributes
  - Some attributes are relations which **link** to other entities
  - Entities, relations, attributes, etc., may have **probabilities**
  - Other attributes represent **values**
  - **Controlled vocabularies** describe the possible kinds of entities, relations, and values
  - Mix and match vocabularies, or **extend** per use case
- **Graph Neural Networks**
  - A substantial thrust in AI toward *graph neural networks*: *geometric deep learning* is an umbrella term for emerging techniques that attempt to generalize deep learning models in non-Euclidean domains such as graphs and manifolds, and *motif mining* operates on complex graph patterns:
    - "Machine Learning on Graphs: A Model and Comprehensive Taxonomy"
      Ines Chami, et al. (2021)
    - "Geometric deep learning: going beyond Euclidean data"
      Michael Bronstein, et al. (2016)
    - "Motif Prediction with Graph Neural Networks"
      Maciej Besta, et al. (2021)
    - PyG, DGL, GraphGym, etc.

Graph Data Science

The **kglab** library, an abstraction layer in Python for building knowledge graphs, integrated with popular graph libraries:

- repo github.com/DerwenAI/kglab
- docs derwen.ai/docs/kgl/
- tutorial derwen.ai/docs/kgl/tutorial/
- forum linkedin.com/groups/6725785/
- glossary derwen.ai/docs/kgl/glossary/
- survey forms.gle/FMHgtmxHYWocprMn6

Since September 2020



Graph use at scale with Ray, for AI in Manufacturing

Open issues

- How can we work with probabilistic graphs?
- How can we work with semantic technologies?
- How to work with graph algorithms efficiently?
- How to do visualization?
- Graph data serialization? The formats currently used are fairly poor. → We really talk about column stores when we represent graph data. → Moving to Parquet can lead to several orders of magnitude improvement in terms of serialization.
- Probabilistic subgraphs / statistical relational learning (SRL)

Common ground among AI in Manufacturing

- Key use-case areas
  - Social networks, advertising, finance, etc.
  - Manufacturing and pharma in major focus by the presenter's groups → Common themes in these industry use-cases:
    - Data objects as "shapes" (motifs)
    - Grappling with complexity and uncertainty
    - Cycle detection (a common data prep need)
    - Disambiguation (NP-hard problems at scale)
    - Details are more important than aggregate counts

FUTUREWEI Technologies

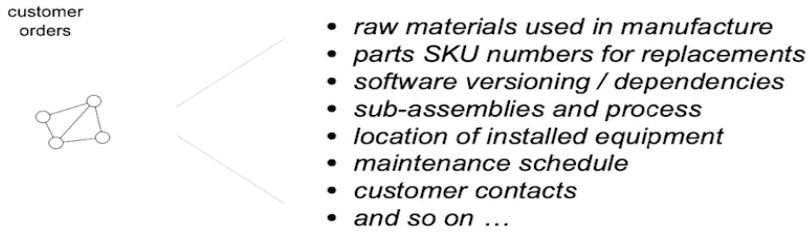
Graph use at scale with Ray, for AI in Manufacturing

KG Use Cases – typical rollout 1.

While there is much discussion about **Enterprise Knowledge Graphs**, projects defined top-down are still quite rare.

In industry we typically see a **"middle-out"** approach where one business unit gains initial success, then federates with other business units.

For example, suppose Acme Manufacturing begins to analyze their customer orders in a graph constructed from *Bill Of Materials* (BOM):

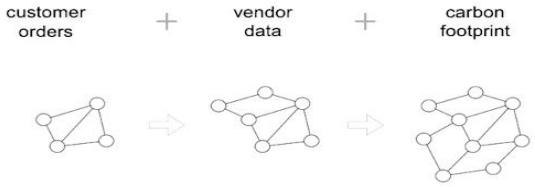


- *raw materials used in manufacture*
- *parts SKU numbers for replacements*
- *software versioning / dependencies*
- *sub-assemblies and process*
- *location of installed equipment*
- *maintenance schedule*
- *customer contacts*
- *and so on …*

KG Use Cases – typical rollout 2.

Customer data plus vendor data begins to provide more global perspectives about a business, as a kind of "enterprise digital twin"

Now suppose that Acme Manufacturing extends their knowledge graph by integrating data about the *sustainability* of sources procured across their supply network of vendors:

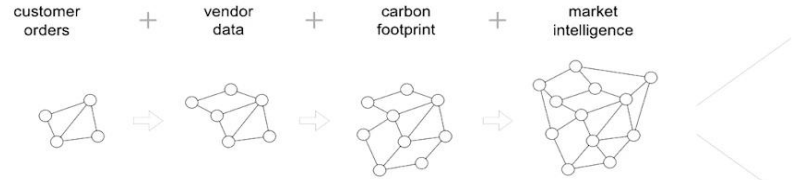


- *ESG factors*
- *relative carbon footprint*
- *regulatory compliance*
- *sustainable practices*
- *strategic investments*
- *environmental impact reports*
- *material risks*
- *and so on …*

KG Use Cases – typical rollout 3.

For example, tracing carbon footprint through a complex supply network involving thousands of vendors extends the "enterprise digital twin" notion so that long-term decisions and strategies can be developed.

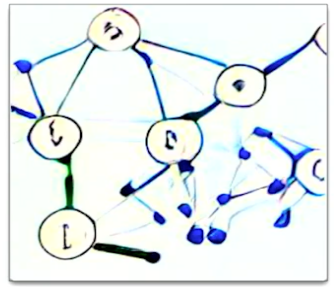
Now suppose that Acme Manufacturing extends their knowledge graph by integrating data sources for *market intelligence* analysis as well:



- *competitive intelligence*
- *shipping manifests*
- *new IP filings*
- *new EIR filings*
- *analysis of lost bids*
- *vendor M&A activity*
- *market estimates*
- *and so on …*

KG Use Cases – common hurdles

- Enterprise IT staff are familiar with relational databases yet unfamiliar with graph use cases, and often try to discourage the latter
- Graph practices in industry generally cut across company divisions and lines of executive responsibility – e.g., where a KG integrates data from logistics, research, and market analysis to provide a "business 360" for top execs – and this tends to alarm mid-level executives
- Vendors are tenaciously concerned with database features and corresponding lock-in for licensing strategies that don't scale well, while generally less concerned with horizontal scale-out for graph computing



Graph use at scale with Ray, for AI in Manufacturing

Identified problem areas

- **balkanization**: disparate “camps”, e.g., SPARQL vs. Cypher
- over-emphasis on **database transactions**; but lack of emphasis on enterprise ETL, data integration, horizontal scale-out, costs at scale, etc.
- good available vendors (e.g., **Graphistry**) albeit “nice work when you can get it” – since the provably hard problems in graph-based data tend to **occur during data prep**, far upstream from applications
- **enterprise licensing** for commercial graph databases at +1B node scale tends to exceed the already **high costs of cloud computing** required for the same use cases, and meanwhile these tend to have **serious performance bottlenecks** beyond 100M node scale

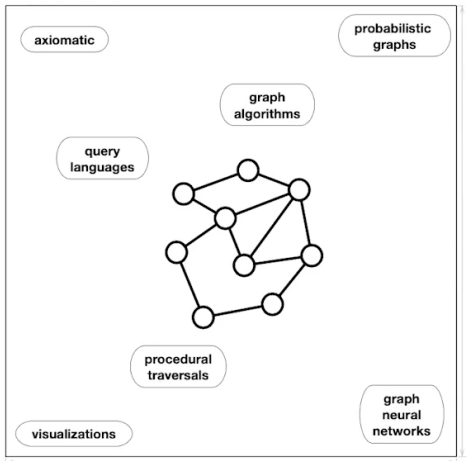
Balkanized: inference with graphs

While there are many useful graph technologies, the various camps don’t talk much with each other …

Nor does their software.

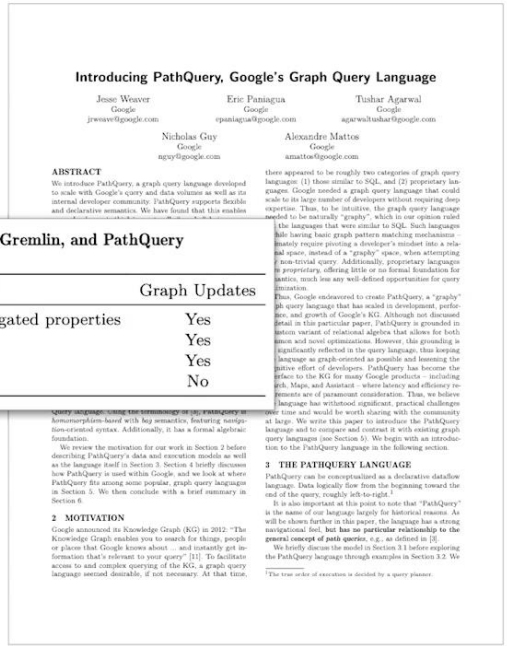
For example, graph DBs focus mostly on the top/left quadrant.

This is one of the major hurdles still to overcome.



Graphs by query language

“Introducing PathQuery”
arxiv.org/abs/2106.09799



Introducing PathQuery, Google’s Graph Query Language

Jesse Weaver, Google, jrweave@google.com
Eric Paniagua, Google, epaniagua@google.com
Tushar Agarwal, Google, agarwaltushar@google.com
Nicholas Guy, Google, nguy@google.com
Alexandre Mattos, Google, amattos@google.com

ABSTRACT

Table 2: Comparison of common language features for SPARQL, Cypher, Gremlin, and PathQuery

Language	Data Model	Pattern Composition	Path Queries[11]	Graph Updates
SPARQL	edge-labelled graph (RDF)	SQL-ish keywords	regular, plus negated properties	Yes
Cypher	property graph	SQL-ish keywords	regular	Yes
Gremlin	property graph	path structures	regular	Yes
PathQuery	edge-labelled graph	path structures	bounded	No

Expect to see more projects like **PathQuery** for managing large, distributed graphs on contemporary hardware.

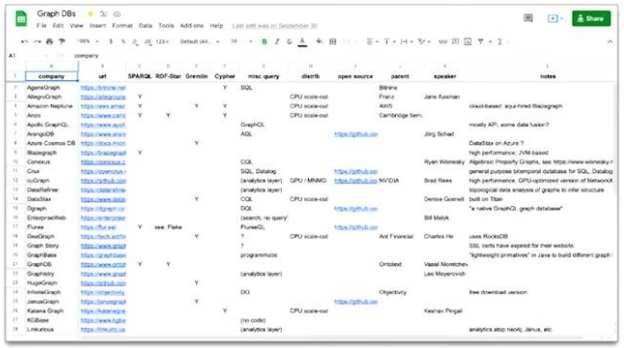
Graphs Database vs. Graph Compute

Currently, **50+** *graph database* vendors. Here’s a public **curated list** on KGC (ask to make edits!)

Even so, industry customers ask for *scalable graph compute* capabilities in lieu of database features.

For system of record, enterprise firms already have SAP, IBM, Oracle, etc.

Furthermore, database-centric practices are giving way to **streaming, event sourcing, etc.**

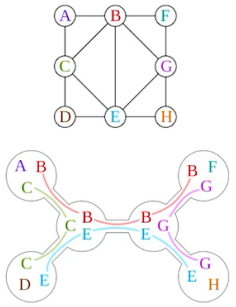


Graph use at scale with Ray, for AI in Manufacturing

Difficulties in graph data preparation

Many of the **provably hard problems** in graph data science tend to occur during *data preparation*, far upstream from vendor OSFA solutions. Consider that costs for these tend to be O(N^2) or greater complexity, where N may be measured in billions:

- detecting and resolving *cycles* where they cannot be allowed by definition in a use case
- *similarity analysis* and *deduplication*, especially across wide graph diameters (high dimensional problems)
- using **graph theory** to help mitigate, e.g., use of **chordal graphs**, **topological transforms**, etc., to avoid highlighting trivial connections
- performing *transitive closures* (axiomatic inheritance)



Identified projects – pro's

A few projects sponsored by technology organizations in China were identified as being mostly in-category based on objectives for scale, performance, and cost:

- **GeaBase** by Ant Financial, Alibaba
- **NebulaGraph** by Vesoft
- **Grasper** by CUHK

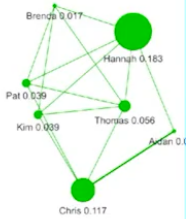
Identified projects – con's

Concerns which limit their use include:

- these tend toward ad-hoc implementations which tend to be less compliant with open standards required by use cases
- security concerns: **"China could be reviewing security bugs before tech companies issue patches, DHS official says"** (2022-08-10)
- intellectual property concerns: **"China's New Data Security and Personal Information Protection Laws: What They Mean for Multinational Companies"** (2021-11-03)
- US federal agencies prefer **open source** projects run by US-led organizations

Needs based on industry use cases

- *horizontal scale-out* on commodity hardware, for +1T node graphs
- overcome ~100M node *performance knee* for load/store operations
- perform *W3C/SPARQL* and *LPG/Cypher* queries on the same data
- support advanced *graph algorithms* for data prep (beyond queries)
- *eventual consistency*, with data integration features similar to Git (commits, PRs, merges, etc., for bulk data operations)
- *multi-tenancy* with strong partitioning, secure authentication/access
- *non-repudiation* features (provably restore to a prior known state)
- probabilistic graphs to model *uncertainty* for nodes, edges, properties
- support for managing *subgraphs* (e.g., **motifs**)

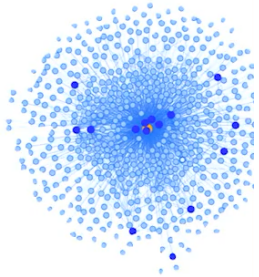


Graph use at scale with Ray, for AI in Manufacturing

Graph technologies, optimally

What would be optimal graph technology for large scale use cases in industry (which aren't social networks) and how could Ray help support this?

1. Critical paths for graph query engines at scale:
   - SPARQL: *distributed parallel iterator for triples*
   - Cypher: *parallel enumeration of unbounded paths*
2. Distributed *hash-maps* with fast scans and LRU memory spill to disk
3. Sparse matrix factorization
4. Other solvers, e.g., *hinge-loss MRF* optimization for PSL
5. More attention to security (component libraries)



Impl – features

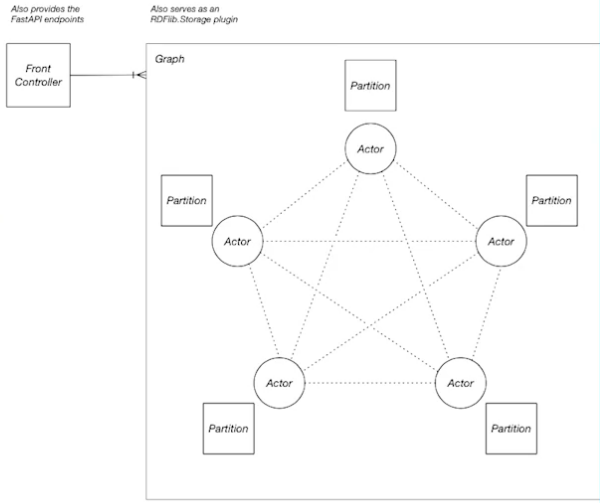
Leveraging open source libraries for:

- partition code written in C++, using HPC libraries
- partitions managed by Ray **actors**, over a Cython layer
- reuses **kglab** for integrating **RDFlib**, **Morph**, etc.
  - SPARQL distrib triple iterators use Ray **datasets**
  - **openCypher** to parse LPG queries
  - sparse matrix factorization (Cypher paths) in **Dask**
- **PSL** for probabilistic graph support
- **Redis** for full-text indexing and search

Impl – persistence

Leveraging open source libraries for:

- **Arrow/Parquet** for persistence and cross-app serialization
- **Pulsar** message queue and event sourcing for CRUD operations (*eventual consistency* in multi-tenant **named graphs)**
- **BookKeeper** for distributed ledger of cloud storage checkpoint metadata



Impl – deployment

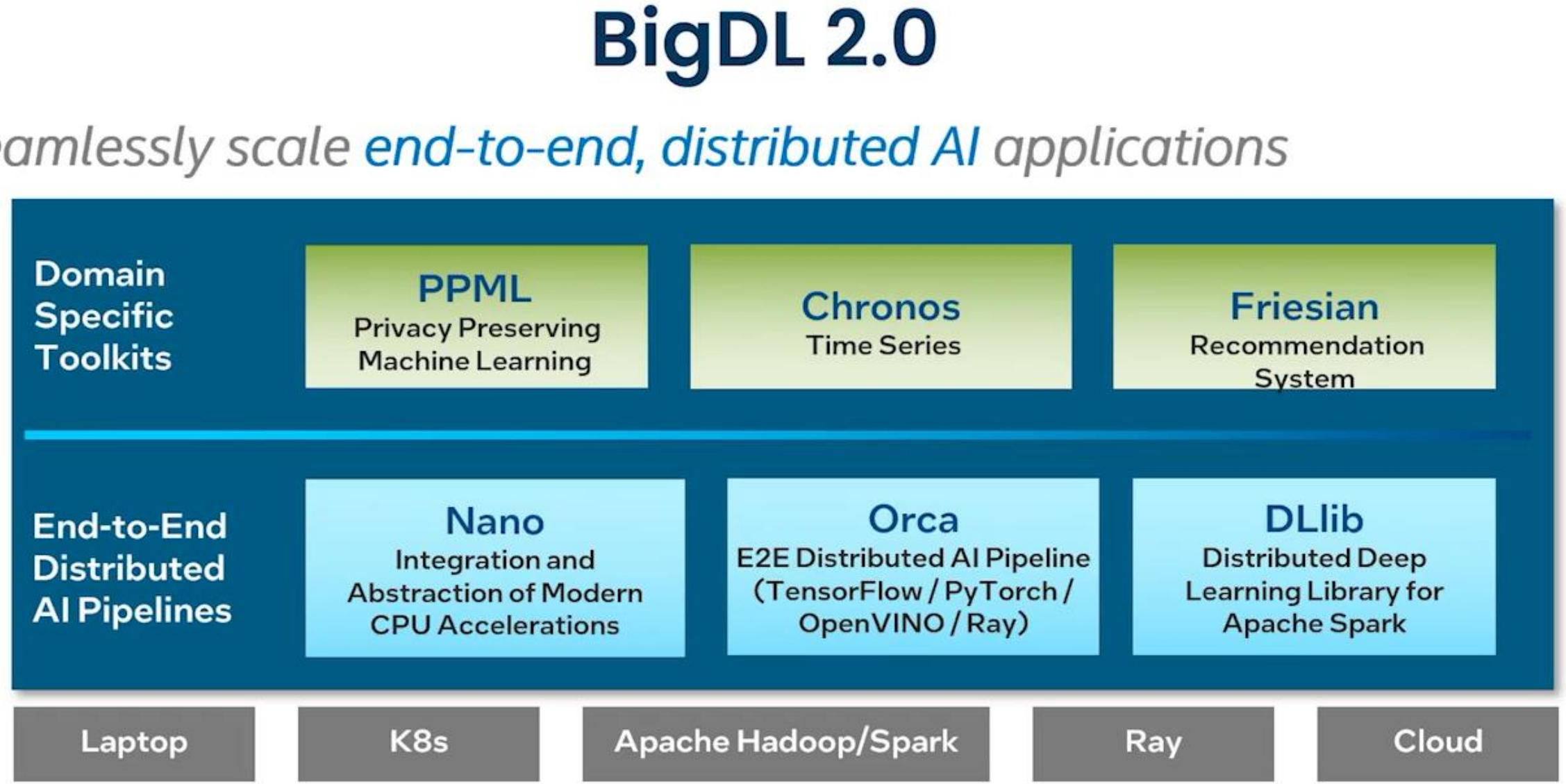
Leveraging open source libraries for:

- Ray-on-K8s, using Ray **placement groups**
- secure API for multi-tenant **named graphs**, **Keycloak**, **Nginx**, **FastAPI**, **Pydantic**
- **minikube** for local dev/test and small use cases

How could Ray work better for industry use?

- play nicer with Cython (see **examples**)
- more attention on **CVE scans** for Docker Hub images
- better scripts for custom build of Ray-on-K8s **container** layers
- Ray-based replacement for HPC C++ *hash-maps* with fast scans and LRU memory spills to disk
- better support for large scale matrix factorization (or simply keep this in Dask?)
- support for **Bulk Synchronous Parallel** leveraging Ray actors
- solvers: there so many optimization problems at scale in industry which **aren't** ML

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster



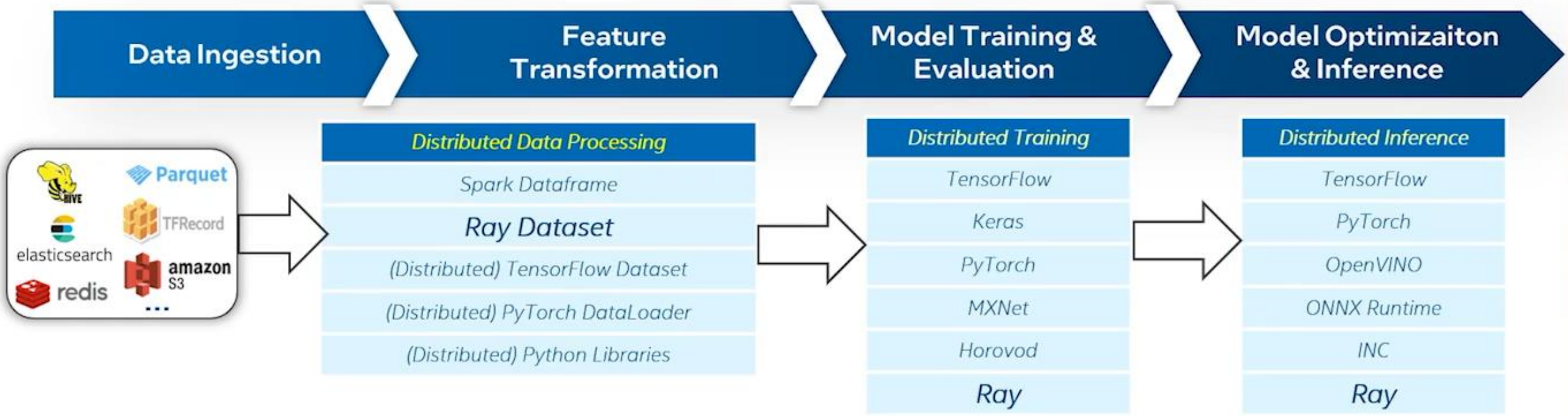
BigDL 2.0 (https://github.com/intel-analytics/BigDL/) combines the *original BigDL* and *Analytics Zoo* projects

* *"BigDL 2.0: Seamless Scaling of AI Pipelines from Laptops to Distributed Cluster", 2022 Conference on Computer Vision and Pattern Recognition (CVPR 2022)*
* *"BigDL: A Distributed Deep Learning Framework for Big Data", in Proceedings of ACM Symposium on Cloud Computing 2019 (SOCC'19)*

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

BigDL-Orca: Building End-to-End Distributed AI Pipeline

E2E, distributed, in-memory pipeline

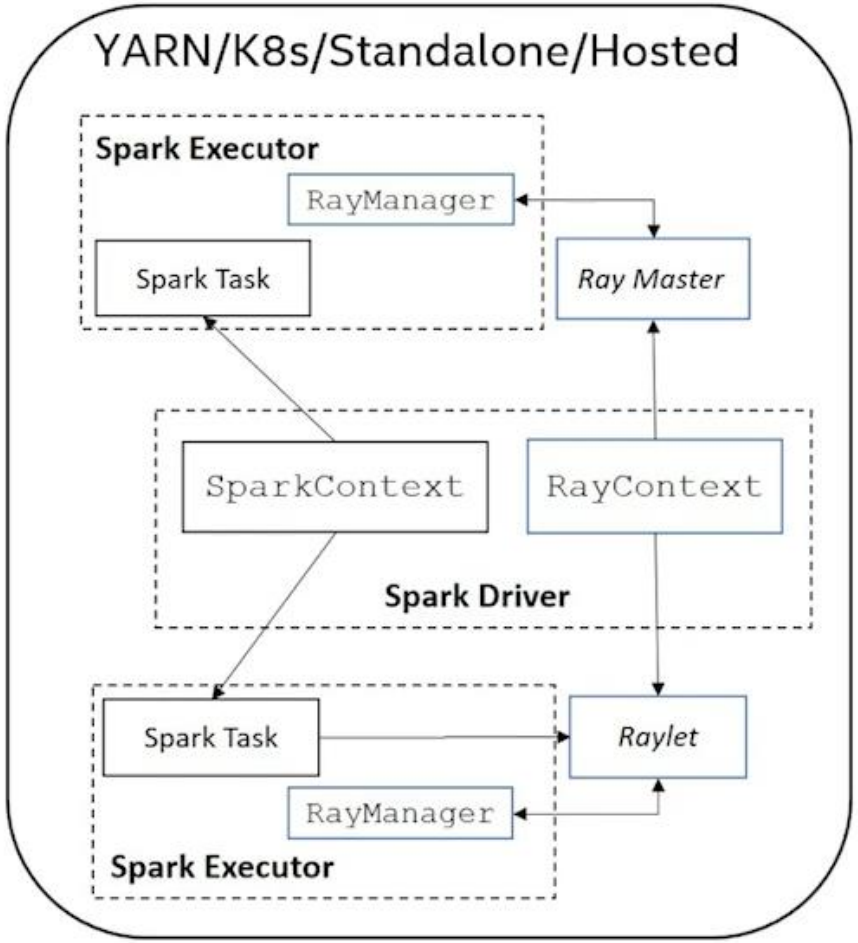


FUTUREWEI Technologies

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

Seamless Pipeline between Spark and Ray using Orca

RayOnSpark



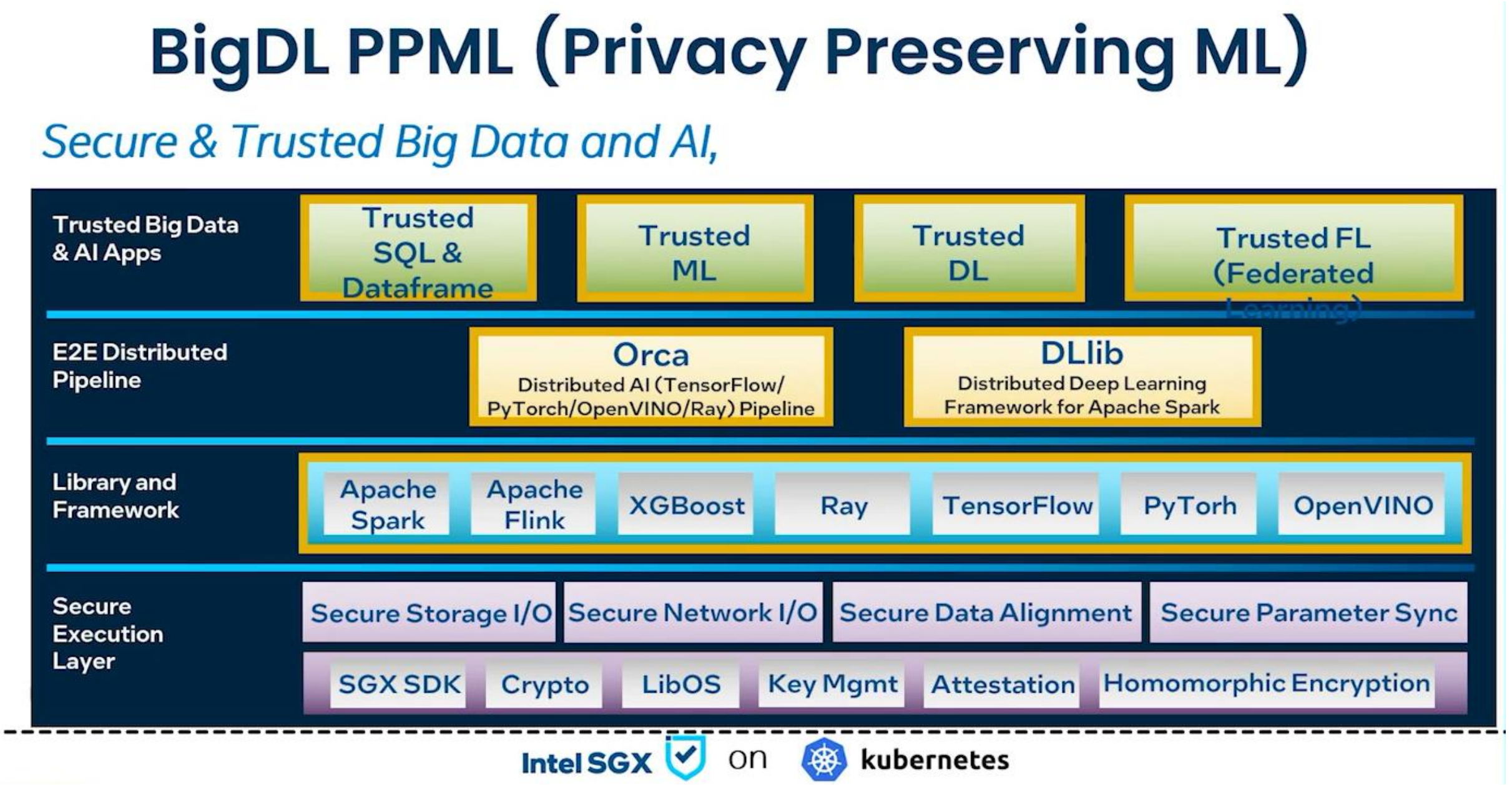
Connecting Spark Dataframe & Ray Dataset

```
...

#spark dataframe processing
df = spark.read.csv(...)
train_df = data_process(df)

#connecting spark dataframe to ray dataset
from bigdl.orca.data \
   import spark_df_to_ray_dataset
train_data = spark_df_to_ray_dataset(train_df)

#xgboost on ray
from xgboost_ray import RayDMatrix, train
dtrain = RayDMatrix(train_data, ...)
bst = train(config, dtrain, ...)
```

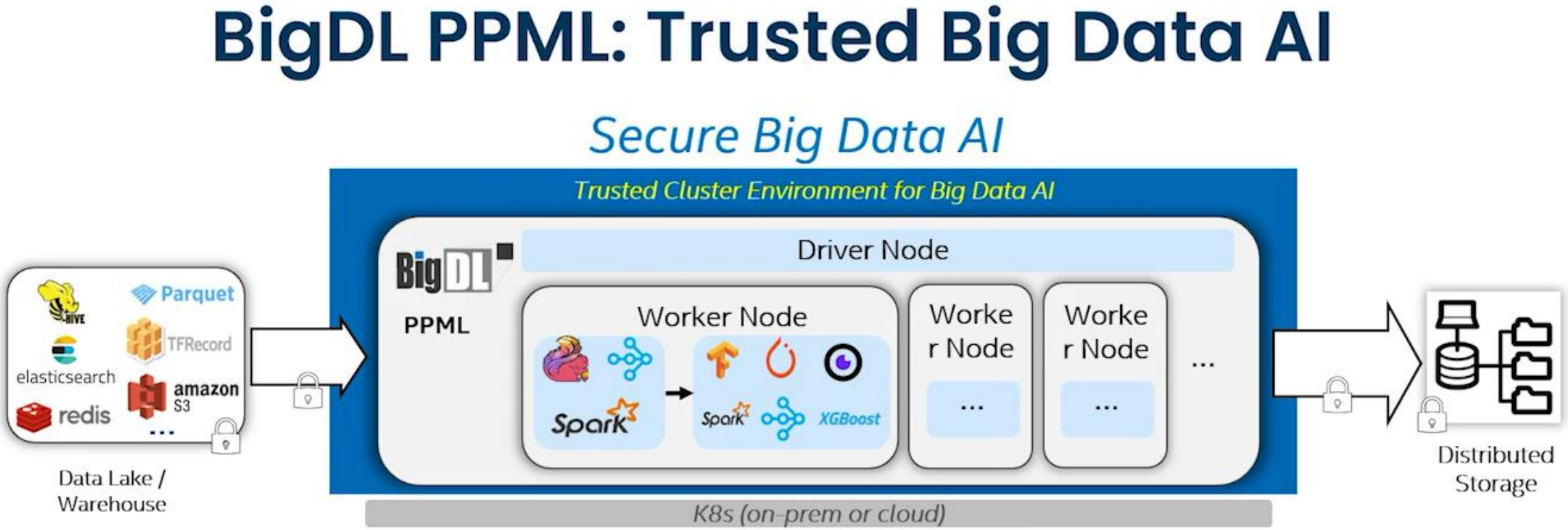
BigDL-Nano: Automatic Integration of Modern CPU Accelerations for TensorFlow & PyTorch

Accelerations	Tuning / Configuration / Tool	Action Needed
Better Memory Allocator	tcmalloc, jemelloc	Download, set environment variables correctly
Proper environment variables	Intel OpenMP (OMP/KMP)	Install, set environment variables correctly
AVX512, BF16	IPEX, Intel Optimized TensorFlow, oneDNN	Install, change some code to use the extensions
Multi-processing	Torch distributed, TF Distributed Strategy, Horovod, etc.	Change some code to use the functionality
Inference Engine	OpenVINO, ONNX Runtime, etc.	Install, Export your model to onnx file, use ort/IE API
Quantization	INC, NNCF/POT	Install, quantize through tool APIs and save/load the quantized model
…	…	…

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster



BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

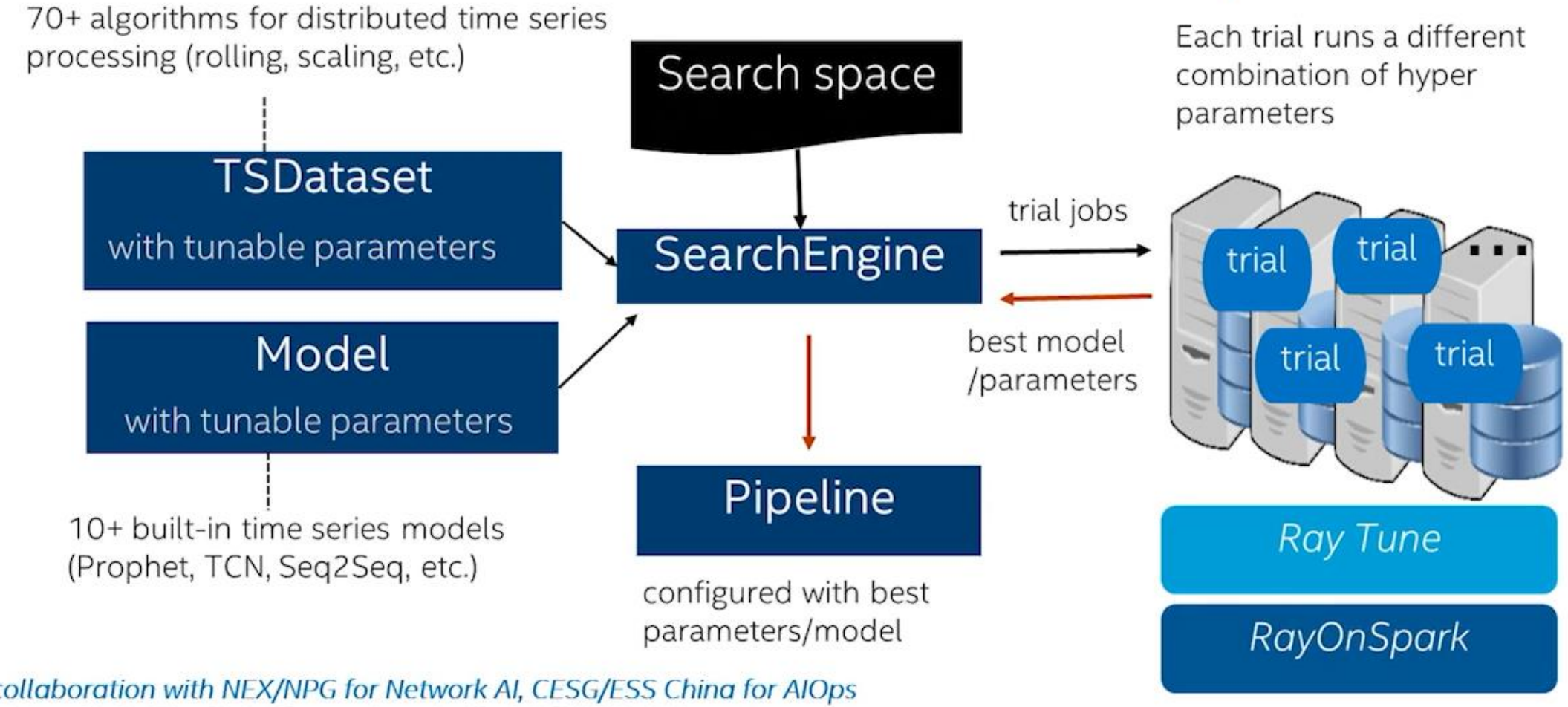


- Standard, distributed AI applications on encrypted data
- Hardware (Intel SGX/TDX) protected computation (and memory)
- End-to-end security enabled for the entire workflow
  - *Provision and attestation of "trusted cluster environment" on K8s (of SGX nodes)*
  - *Secrete key management through KMS for distributed data decryption/encryption*
  - *Secure distributed compute and communication (via SGX, encryption, TLS, etc.)*

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

BigDL-Chronos

Application framework for scalable time series analysis w/ AutoML



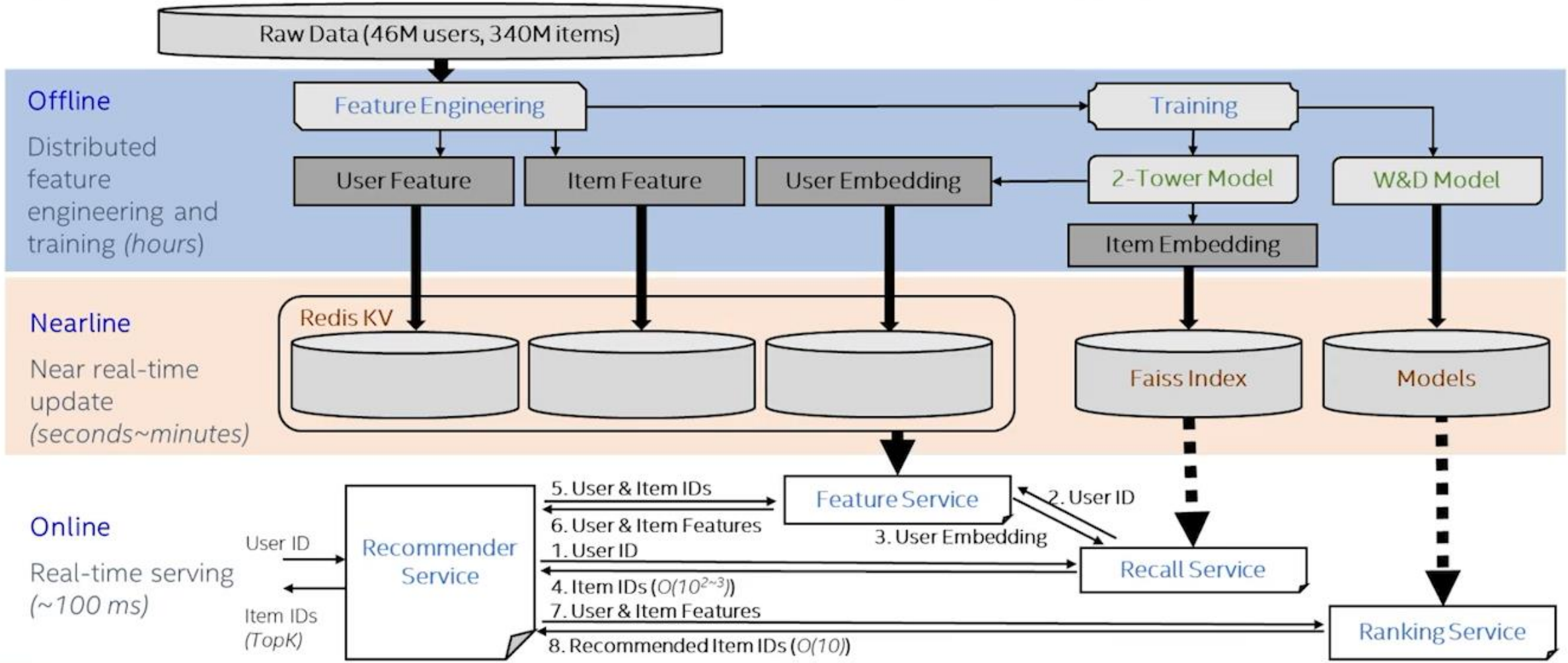
*Joint-collaboration with NEX/NPG for Network AI, CESG/ESS China for AIOps

* "Scalable AutoML for Time Series Forecasting using Ray", USENIX OpML'20

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

BigDL-Friesian

Application framework for large-scale E2E recommender solution



BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

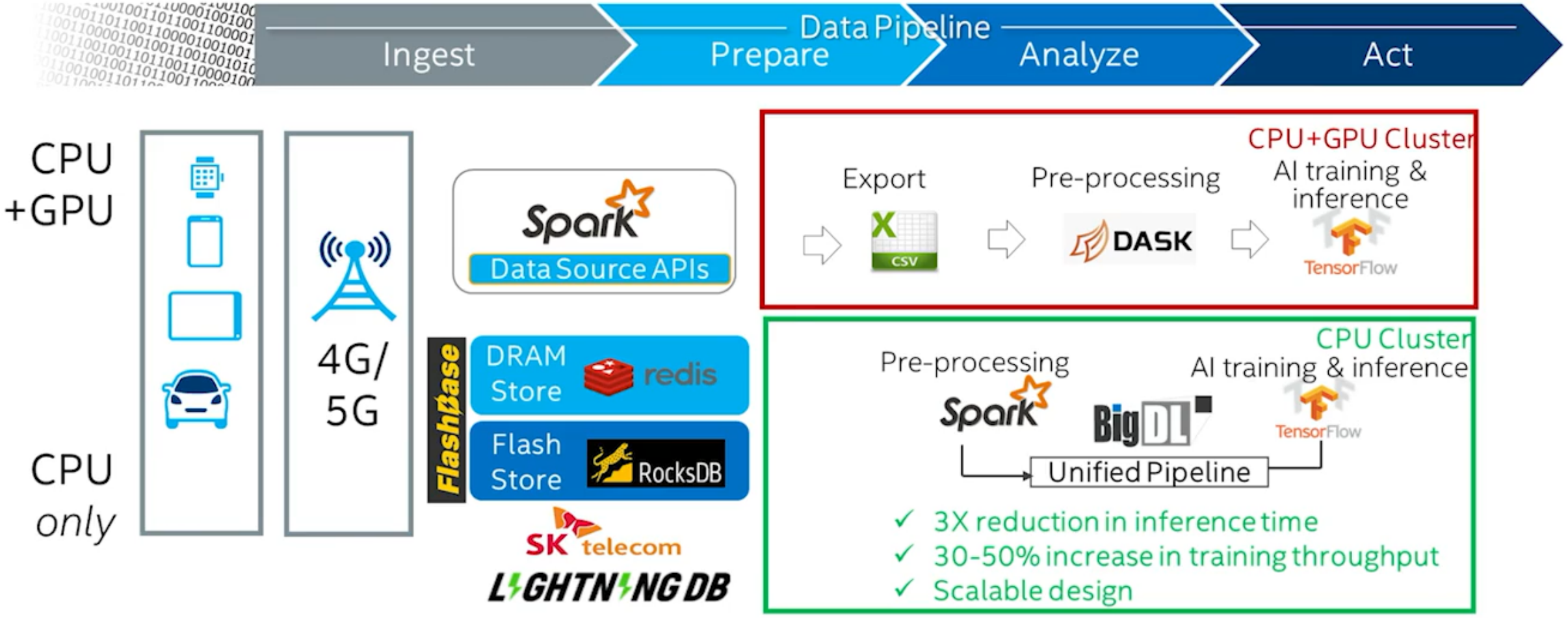
"AI at Scale" in Mastercard with BigDL

- Building distributed AI applications directly on Enterprise Data Warehouse platform
- Supporting up to 2.2 billion users, 100s of billions of records, and distributed training on several hundred Intel Xeon servers

https://www.intel.com/content/www/us/en/developer/articles/technical/ai-at-scale-in-mastercard-with-bigdl0.html

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

Network Quality Prediction in SK Telecom with BigDL



https://networkbuilders.intel.com/solutionslibrary/sk-telecom-intel-build-ai-pipeline-to-improve-network-quality

BigDL 2.0: Seamlessly scale E2E distributed AI from laptop to cluster

Fast Food Recommendation in Burger King with BigDL and Ray

* https://medium.com/riselab/context-aware-fast-food-recommendation-at-burger-king-with-rayonspark-2e7a6009dd2d
* *"Context-aware Fast Food Recommendation with Ray on Apache Spark at Burger King", Data + AI Summit Europe 2020*

THANK YOU!!!