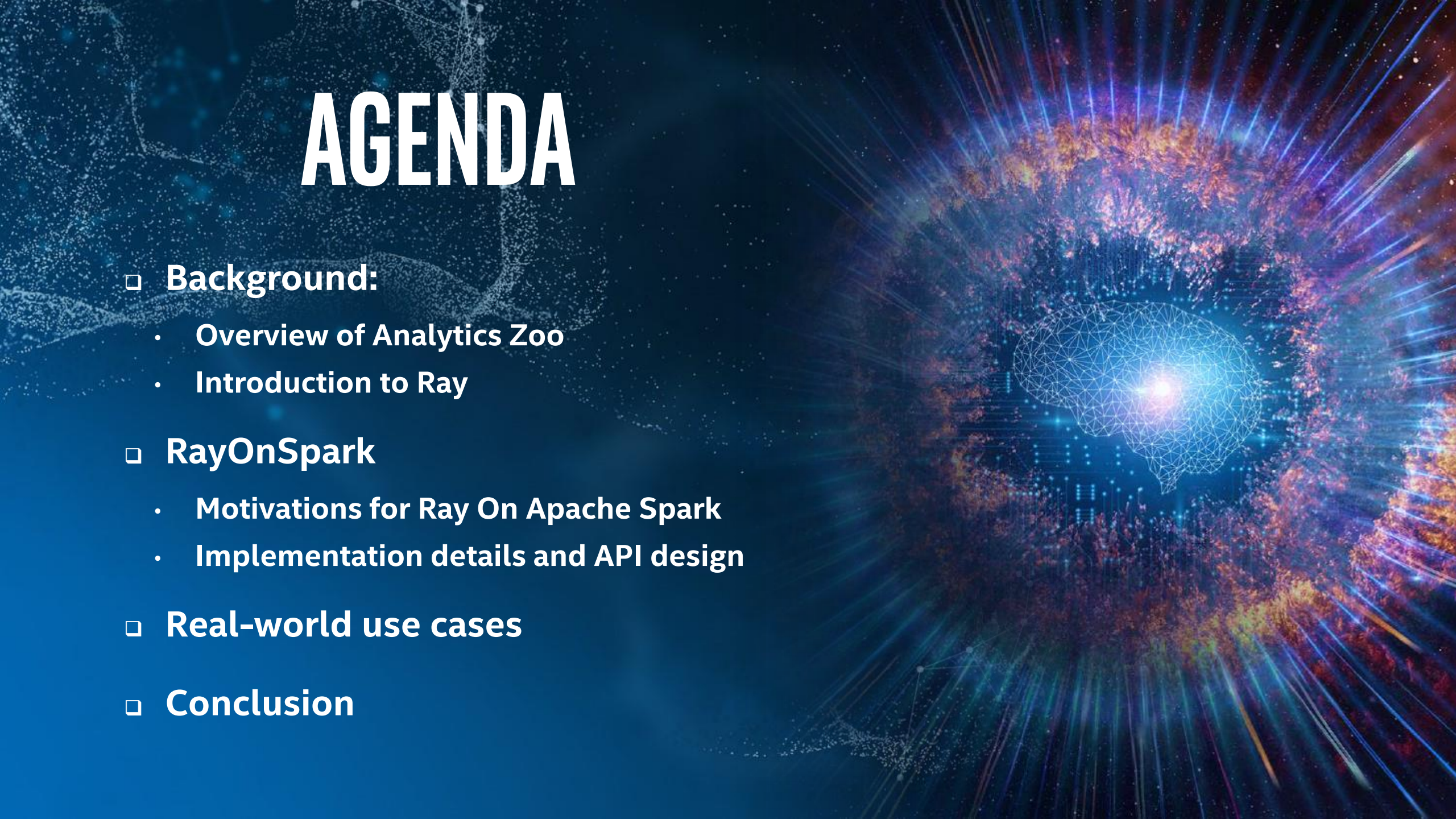
RUNNING EMERGING AI APPLICATIONS ON BIG DATA PLATFORMS WITH RAY ON APACHE SPARK

Kai Huang Jason Dai

AGENDA

- Background:
  - Overview of Analytics Zoo
  - Introduction to Ray
- RayOnSpark
  - Motivations for Ray On Apache Spark
  - Implementation details and API design
- Real-world use cases
- Conclusion



AI ON BIG DATA

intel AI

BigDL

Distributed, High-Performance

Deep Learning Framework

for Apache Spark*

https://github.com/intel-analytics/bigdl

ANALYTICS ZOO

Unified Analytics + AI Platform

for TensorFlow*, PyTorch*, Keras*, BigDL, Ray* and Apache Spark*

https://github.com/intel-analytics/analytics-zoo

ACCELERATING DATA ANALYTICS + AI SOLUTIONS AT SCALE

ANALYTICS ZOO

Unified Data Analytics and AI Platform for distributed TensorFlow, Keras and PyTorch on Apache Spark/Flink & Ray

Models & Algorithms

- Recommendation
- Time Series
- Computer Vision
- NLP

Automated ML Workflow

- AutoML for Time Series
- Automatic Cluster Serving

Integrated Analytics & AI Pipelines

- Distributed TensorFlow & PyTorch on Spark
- RayOnSpark
- Spark Dataframes & ML Pipelines for DL
- InferenceModel

Compute Environment

- Laptop
- K8s Cluster
- Hadoop Cluster
- Spark Cluster

- DL Frameworks (TF/PyTorch/OpenVINO/…)
- Distributed Analytics (Spark/Flink/Ray/…)
- Python Libraries (Numpy/Pandas/sklearn/…)

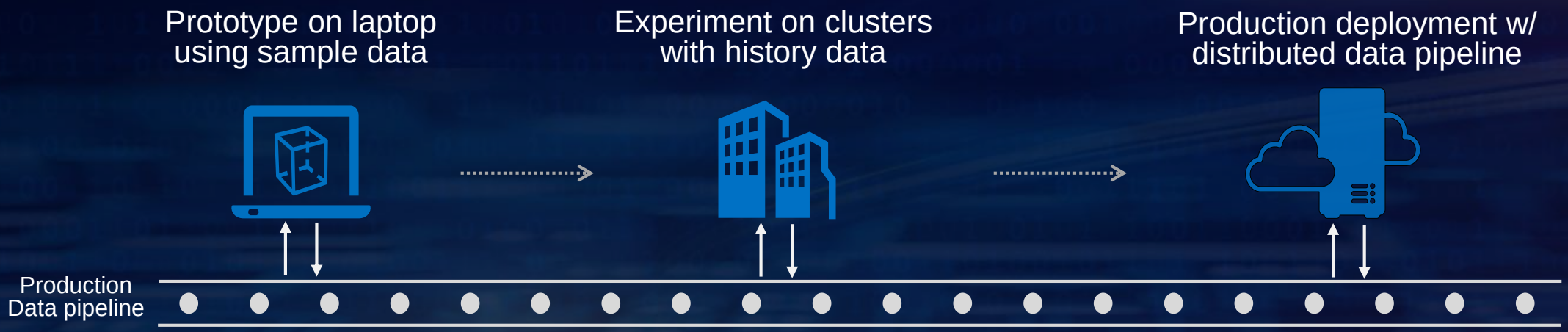
Powered by oneAPI

https://github.com/intel-analytics/analytics-zoo

UNIFIED DATA ANALYTICS AND AI PLATFORM

intel AI

Seamless Scaling from Laptop to Distributed Big Data Clusters



- Easily prototype end-to-end pipelines that apply AI models to big data.
- "Zero" code change from laptop to distributed cluster.
- Seamlessly deployed on production Hadoop/K8s clusters.
- Automate the process of applying machine learning to big data.

RAY

Ray is a fast and simple framework for building and running distributed applications.

- Ray Core provides easy Python interface for parallelism by using remote functions and actors.

```python
import ray
ray.init()

@ray.remote(num_cpus, …)
def f(x):
    return x * x

# Executed in parallel
ray.get([f.remote(i) for i in range(5)])
```

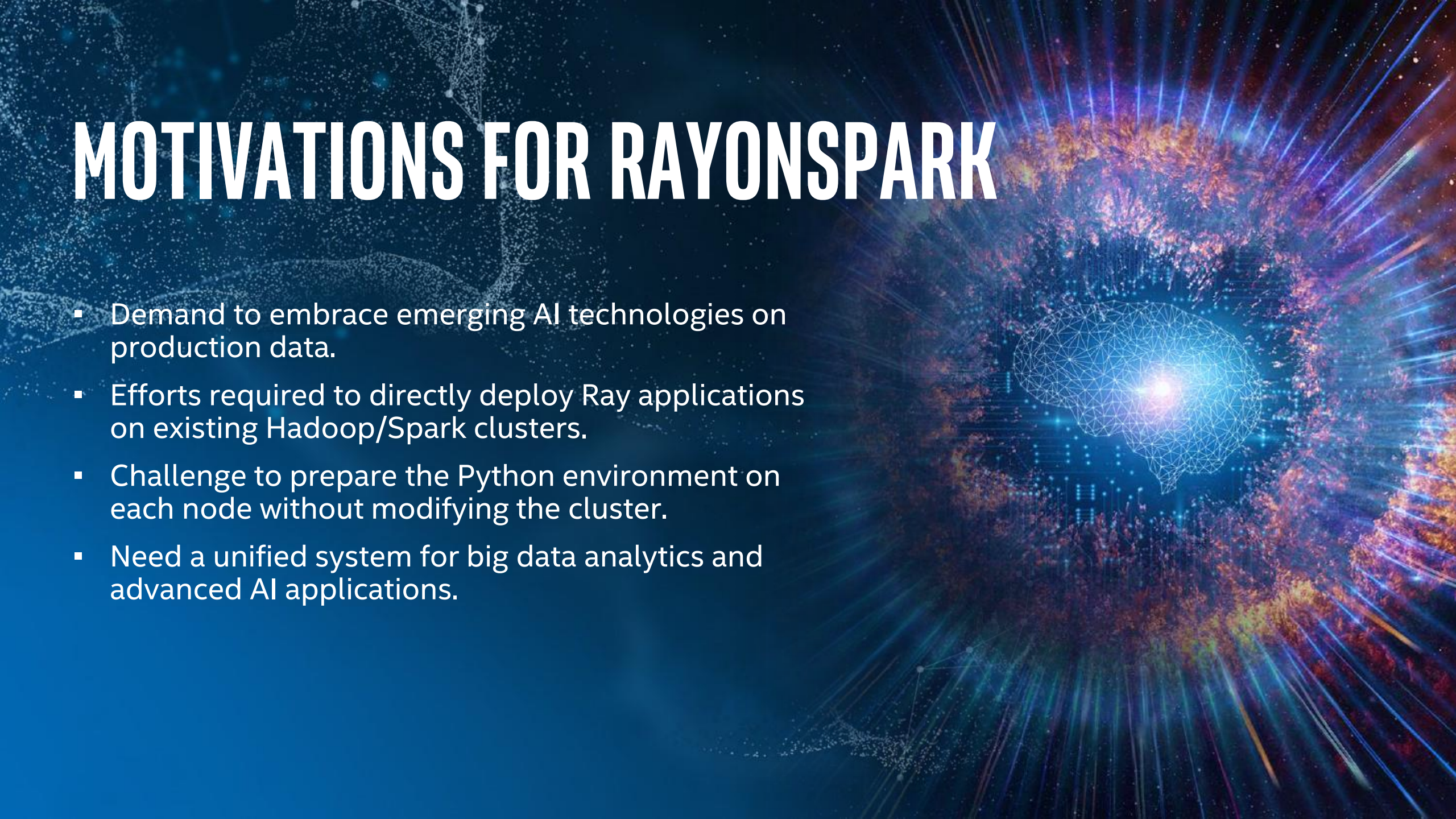
```python
@ray.remote(num_cpus, …)
class Counter(object):
    def __init__(self):
        self.n = 0
    def increment(self):
        self.n += 1
        return self.n

counters = [Counter.remote() for i in range(5)]
# Executed in parallel
ray.get([c.increment.remote() for c in counters])
```

RAY

Ray is packaged with several high-level libraries to accelerate machine learning workloads.

- Tune: Scalable Experiment Execution and Hyperparameter Tuning
- RLlib: Scalable Reinforcement Learning
- RaySGD: Distributed Training Wrappers
- https://github.com/ray-project/ray/

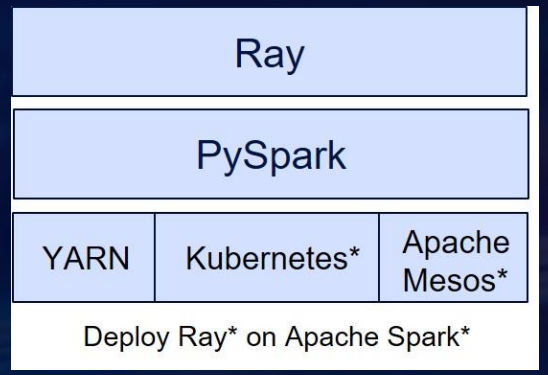
MOTIVATIONS FOR RAYONSPARK



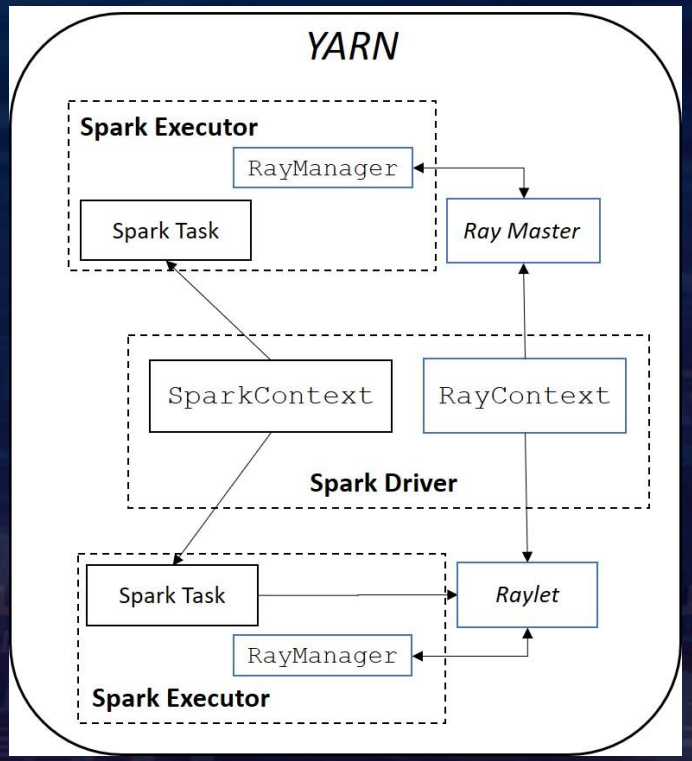
- Demand to embrace emerging AI technologies on production data.
- Efforts required to directly deploy Ray applications on existing Hadoop/Spark clusters.
- Challenge to prepare the Python environment on each node without modifying the cluster.
- Need a unified system for big data analytics and advanced AI applications.

IMPLEMENTATION OF RAYONSPARK

RayOnSpark allows Ray applications to seamlessly integrate into Spark data processing pipelines.



- Leverage conda-pack and Spark for runtime Python package distribution.
- *RayContext* on Spark driver launches Ray across the cluster.
- Ray processes exist alongside Spark executors.
- For each Spark executor, a Ray Manager is created to manage Ray processes.
- Able to run in-memory Spark RDDs or DataFrames in Ray applications.



Launch Ray* on Apache Spark*

RAYONSPARK API

Three-step programming with minimum code changes:

- Initiate or use an existing *SparkContext*.
- Initiate *RayContext*.
- Shut down *SparkContext* and *RayContext* after tasks finish.

More instructions at: https://analytics-zoo.github.io/master/#ProgrammingGuide/rayonspark/

```
import ray
from zoo import init_spark_on_yarn
from zoo.ray import RayContext
```

RayOnSpark code

```
sc = init_spark_on_yarn(hadoop_conf, conda_name,
num_executors, executor_cores,…)
ray_ctx = RayContext(sc, object_store_memory,…)
ray_ctx.init()
```

Pure Ray code

```
@ray.remote
class Counter(object):
      def __init__(self):
          self.n = 0
      def increment(self):
          self.n += 1
          return self.n
counters = [Counter.remote() for i in range(5)]
ray.get([c.increment.remote() for c in counters])
```

```
ray_ctx.stop()
sc.stop()
```

USE CASES OF RAYONSPARK

- Scalable AutoML for time series prediction.
  - Automate the feature generation, model selection and hyperparameter tuning processes.
  - See more at: https://github.com/intel-analytics/analytics-zoo/tree/master/pyzoo/zoo/automl.

- Data parallel pre-processing and distributed training pipeline of deep neural networks.
  - Use PySpark or Ray for parallel data loading and processing.
  - Use RayOnSpark to implement thin wrappers to automatically setup distributed environment.
  - Run distributed training with either framework native modules or Horovod (from Uber) as the backend.
  - Users only need to write the training script on the single node and make minimum code changes to achieve distributed training.

PROJECT ORCA

Easily scaling out AI pipelines.

- https://github.com/intel-analytics/analytics-zoo/tree/master/pyzoo/zoo/orca

- *Project Orca* allows you to easily scale out your single node Python notebook across large clusters, by providing:
  - Data-parallel preprocessing for Python AI (supporting common Python libraries such as Pandas, Numpy, PIL, TensorFlow Dataset, PyTorch DataLoader, etc.)
  - Sklearn-style APIs for transparently distributed training and inference (supporting TensorFlow, PyTorch, Keras, MXNet, Horovod, etc.)

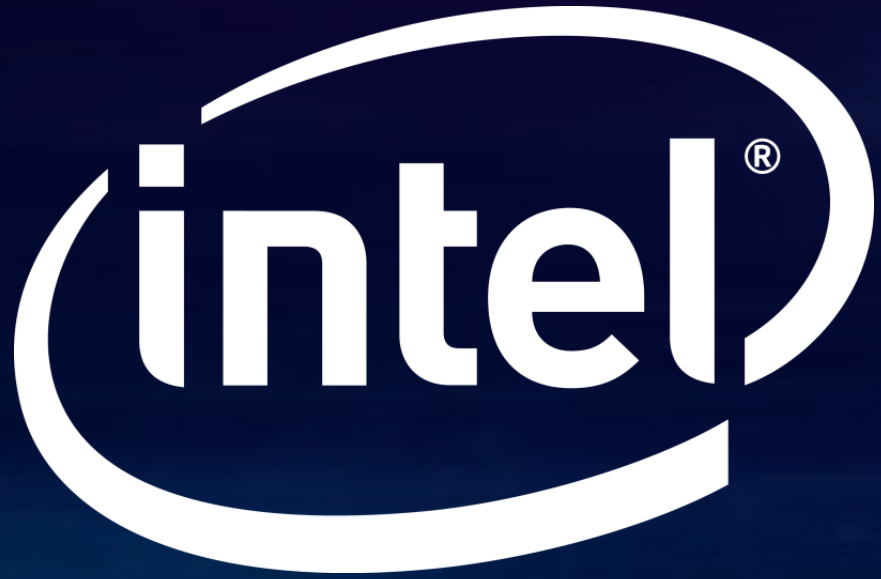
RECOMMENDATION SYSTEM AT

- Burger King performs Spark ETL tasks first, followed by distributed MXNet training.
- Similar to RaySGD, we implement a lightweight shim layer around native MXNet modules for easy deployment on YARN cluster.
- The entire pipeline runs on a single cluster. No extra data transfer needed.
- Check our blog at: https://medium.com/riselab/context-aware-fast-food-recommendation-at-burger-king-with-rayonspark-2e7a6009dd2d

```
from zoo.orca.learn.mxnet import Estimator

mxnet_estimator = Estimator(train_config, model, loss, metrics,
                            num_workers, num_servers)

mxnet_estimator.fit(data=train_rdd, validation_data=val_rdd,
                    epochs=…, batch_size=…)
```

CONCLUSION

- RayOnSpark: Running Emerging AI Applications on Big Data Clusters with Ray and Analytics Zoo https://medium.com/riselab/rayonspark-running-emerging-ai-applications-on-big-data-clusters-with-ray-and-analytics-zoo-923e0136ed6a
- More information for Analytics Zoo at:
  - https://github.com/intel-analytics/analytics-zoo
  - https://analytics-zoo.github.io/
- We are working on full support and more out-of-box solutions for scaling Python AI pipelines based on Ray and Spark in Project Orca.


intel

Legal Notices and Disclaimers

- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit intel.com/performance.
- Intel does not control or audit the design or implementation of third-party benchmark data or websites referenced in this document. Intel encourages all of its customers to visit the referenced websites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.
- Optimization notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com/benchmarks.
- Intel, the Intel logo, Intel Inside, the Intel Inside logo, Intel Atom, Intel Core, Iris, Movidius, Myriad, Intel Nervana, OpenVINO, Intel Optane, Stratix, and Xeon are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
- *Other names and brands may be claimed as the property of others.
-