Project Hydrogen: Unifying State-of-the-art Big Data and AI in Apache Spark

Tim Hunter

2018-10-03

databricks

About

- Tim Hunter
- Software engineer & Solutions Architect @ Databricks
- Ph.D. from UC Berkeley in Machine Learning
- Very early Spark user
- Contributor to MLlib
- Co-author of Deep Learning Pipelines, TensorFrames and GraphFrames

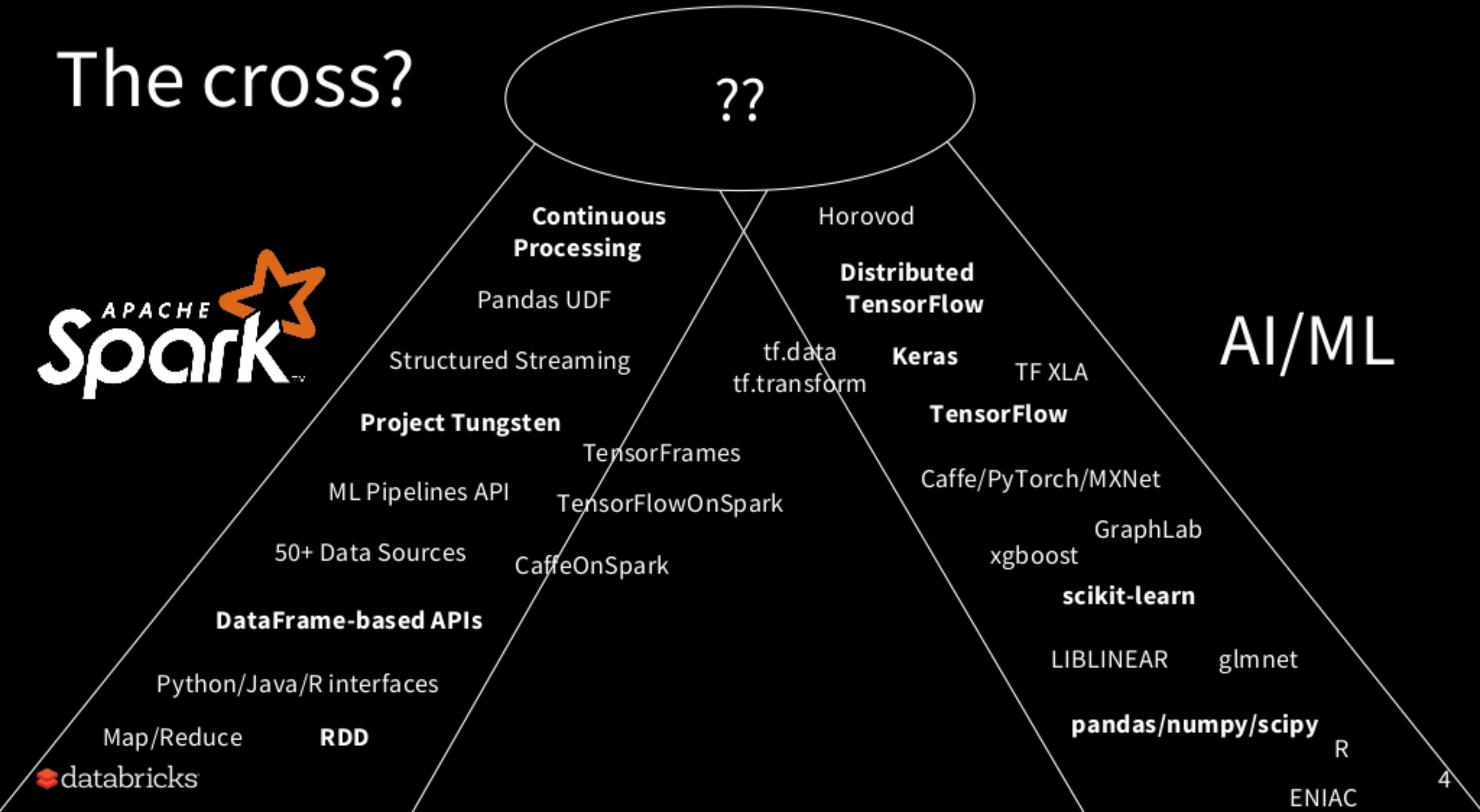
(talk co-authored with Xiangrui Meng)

Two communities: big data and AI

Significant progress has been made by both big data and AI communities in recent years to push the cutting edge:

more complex big data scenarios	more complex deep learning scenarios

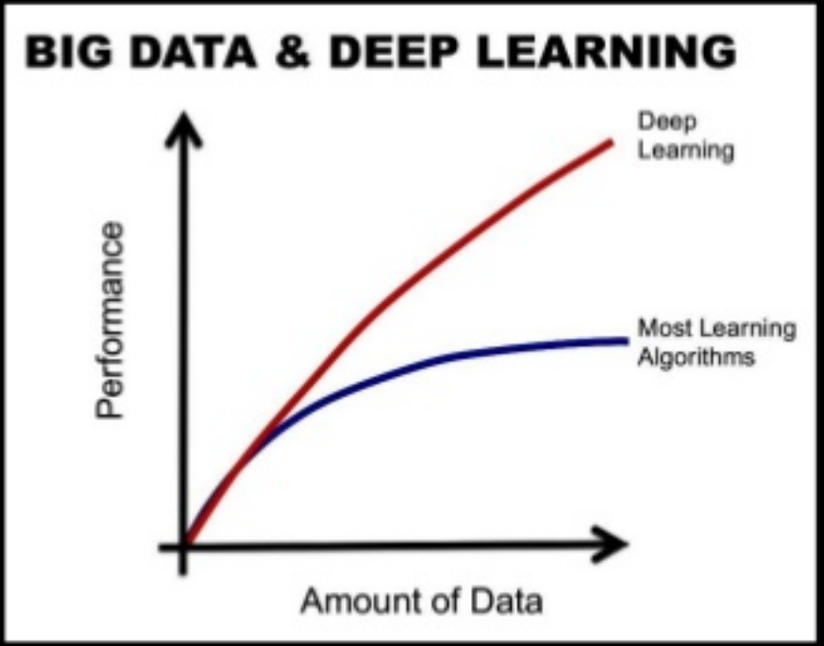
The cross?


??
APACHE Spark
Continuous Processing
Pandas UDF
Structured Streaming
Project Tungsten
ML Pipelines API
50+ Data Sources
DataFrame-based APIs
Python/Java/R interfaces
Map/Reduce
RDD
databricks
Horovod
Distributed TensorFlow
tf.data
tf.transform
Keras
TF XLA
TensorFlow
TensorFrames
TensorFlowOnSpark
CaffeOnSpark
Caffe/PyTorch/MXNet
GraphLab
xgboost
scikit-learn
LIBLINEAR
glmnet
pandas/numpy/scipy
R
ENIAC
AI/ML

AI needs big data

One cannot apply AI techniques without data. And DL models scale with amount of data. We have seen efforts from the AI community to handle different data scenarios:

- tf.data, tf.Transform
- spark-tensorflow-connector
- ...



source: Andrew Ng

Big data for AI

There are many efforts from the Spark community to integrate Spark with AI/ML frameworks:

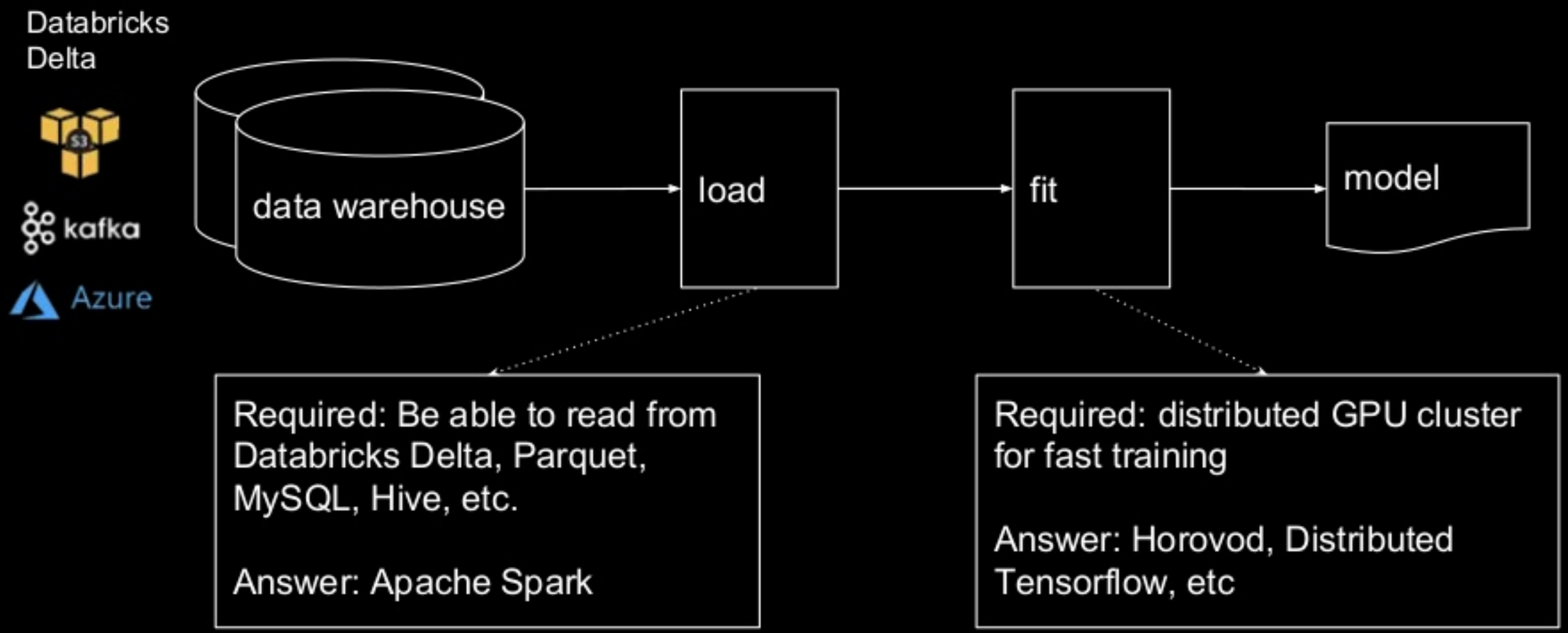
- (Yahoo) CaffeOnSpark, TensorFlowOnSpark
- (Intel) BigDL
- (John Snow Labs) Spark-NLP
- (Databricks) spark-sklearn, tensorframes, spark-deep-learning
- … 80+ ML/AI packages on spark-packages.org

The status quo: two simple stories

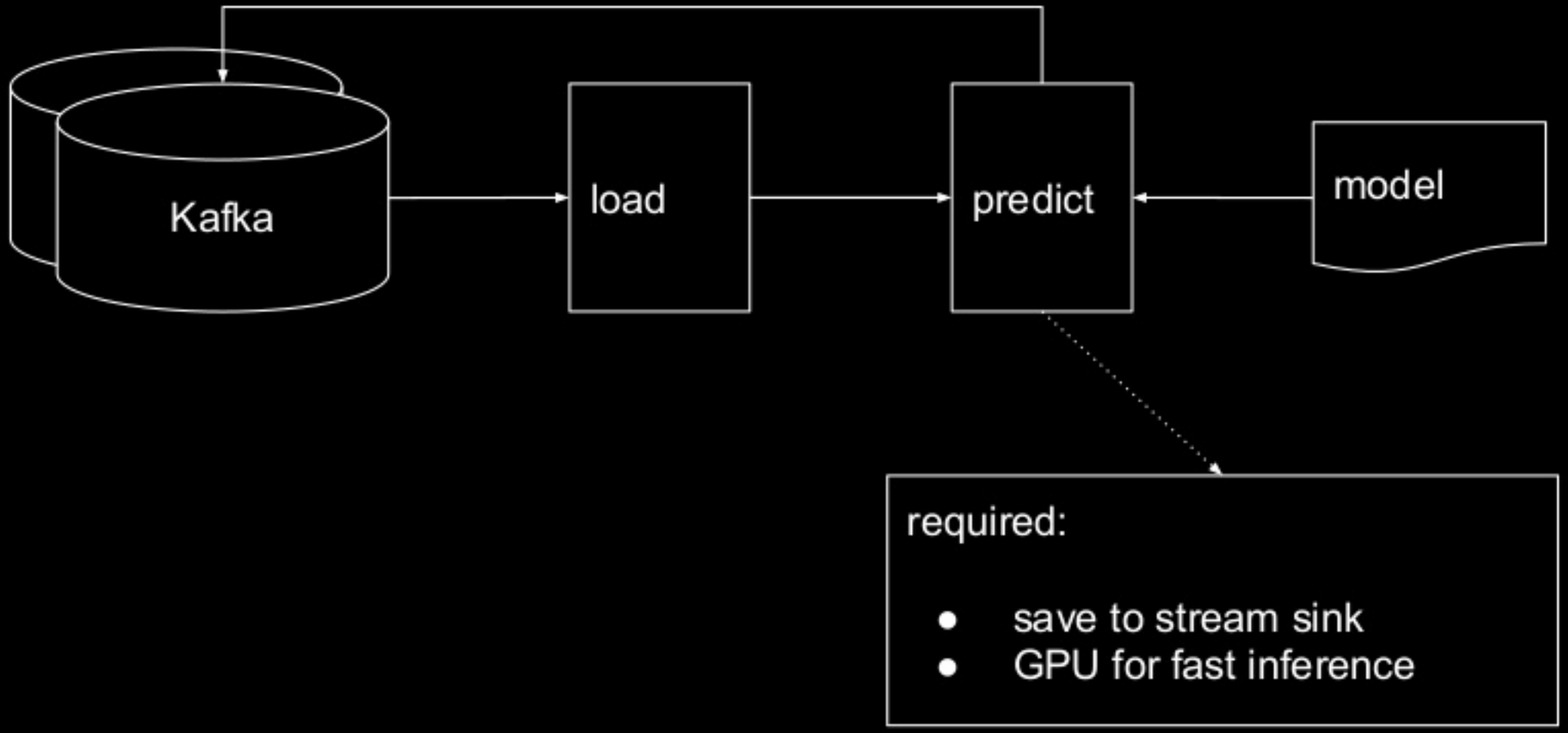
As a data scientist, I can:

- build a pipeline that fetches training events from a production data warehouse and trains a DL model in parallel;
- apply a trained DL model to a distributed stream of events and enrich it with predicted labels.

Distributed training



Streaming model inference

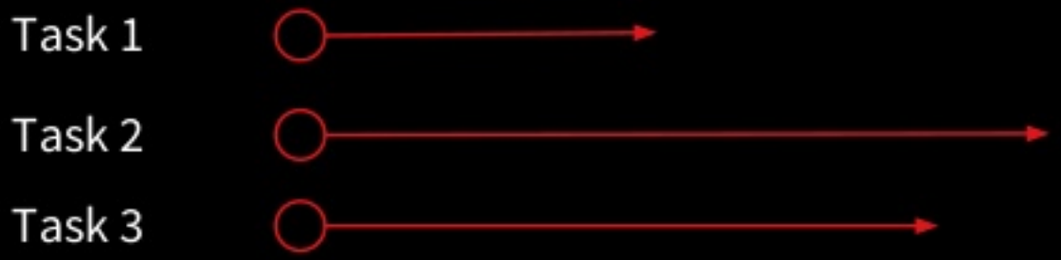


Different execution models

Spark

Tasks are independent of each other

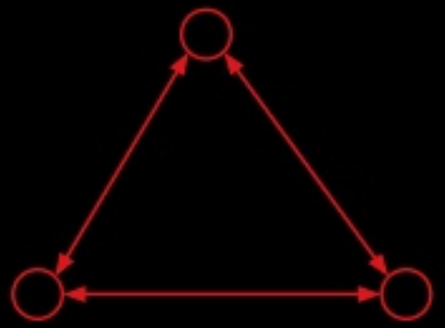
Embarrassingly parallel & massively scalable



Distributed Training

Complete coordination among tasks

Optimized for communication



Different execution models

Spark

Tasks are independent of each other

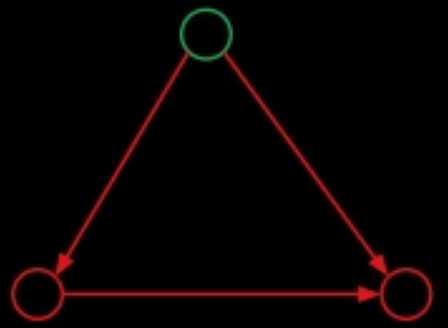
Embarrassingly parallel & massively scalable

If one crashes…

Distributed Training

Complete coordination among tasks

Optimized for communication



Different execution models

Spark

Tasks are independent of each other

Embarrassingly parallel & massively scalable

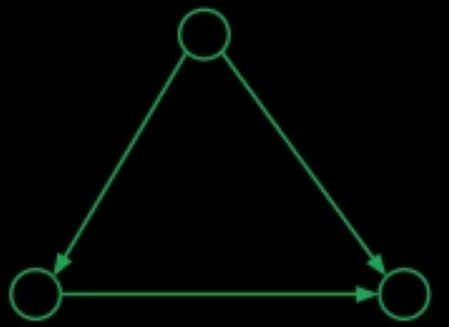
If one crashes, rerun that one

Distributed Training

Complete coordination among tasks

Optimized for communication

If one crashes, must rerun all tasks



Project Hydrogen: Spark + AI

Barrier Execution Mode

Optimized Data Exchange

Accelerator Aware Scheduling

Project Hydrogen: Spark + AI

Barrier Execution Mode

Optimized Data Exchange

Accelerator Aware Scheduling

Barrier execution mode

We introduce gang scheduling to Spark on top of MapReduce execution model. So a distributed DL job can run as a Spark job.

- It starts all tasks together.
- It provides sufficient info and tooling to run a hybrid distributed job.
- It cancels and restarts all tasks in case of failures.

Umbrella JIRA: SPARK-24374 (ETA: Spark 2.4, 3.0)

RDD.barrier()

RDD.barrier() tells Spark to launch the tasks together.

```
rdd.barrier().mapPartitions { iter =>
  val context = BarrierTaskContext.get()
  ...
}
```

context.barrier()

context.barrier() places a global barrier and waits until all tasks in this stage hit this barrier.

```
val context = BarrierTaskContext.get()
... // write partition data out
context.barrier()
```

context.getTaskInfos()

context.getTaskInfos() returns info about all tasks in this stage.

```
if (context.partitionId == 0) {
  val addrs = context.getTaskInfos().map(_.address)
  ... // start a hybrid training job, e.g., via MPI
}

context.barrier() // wait until training finishes
```

Cluster manager support

In Spark standalone mode, users need passwordless SSH among workers to run a hybrid MPI job. The community is working on the support of other cluster managers.

YARN	Kubernetes	Mesos
SPARK-24723	SPARK-24724	SPARK-24725

Project Hydrogen: Spark + AI

Barrier Execution Mode

Optimized Data Exchange

Accelerator Aware Scheduling

Optimized data exchange (SPIP)

None of the integrations are possible without exchanging data between Spark and AI frameworks. And performance matters. We proposed using a standard interface for data exchange to simplify the integrations without introducing much overhead.

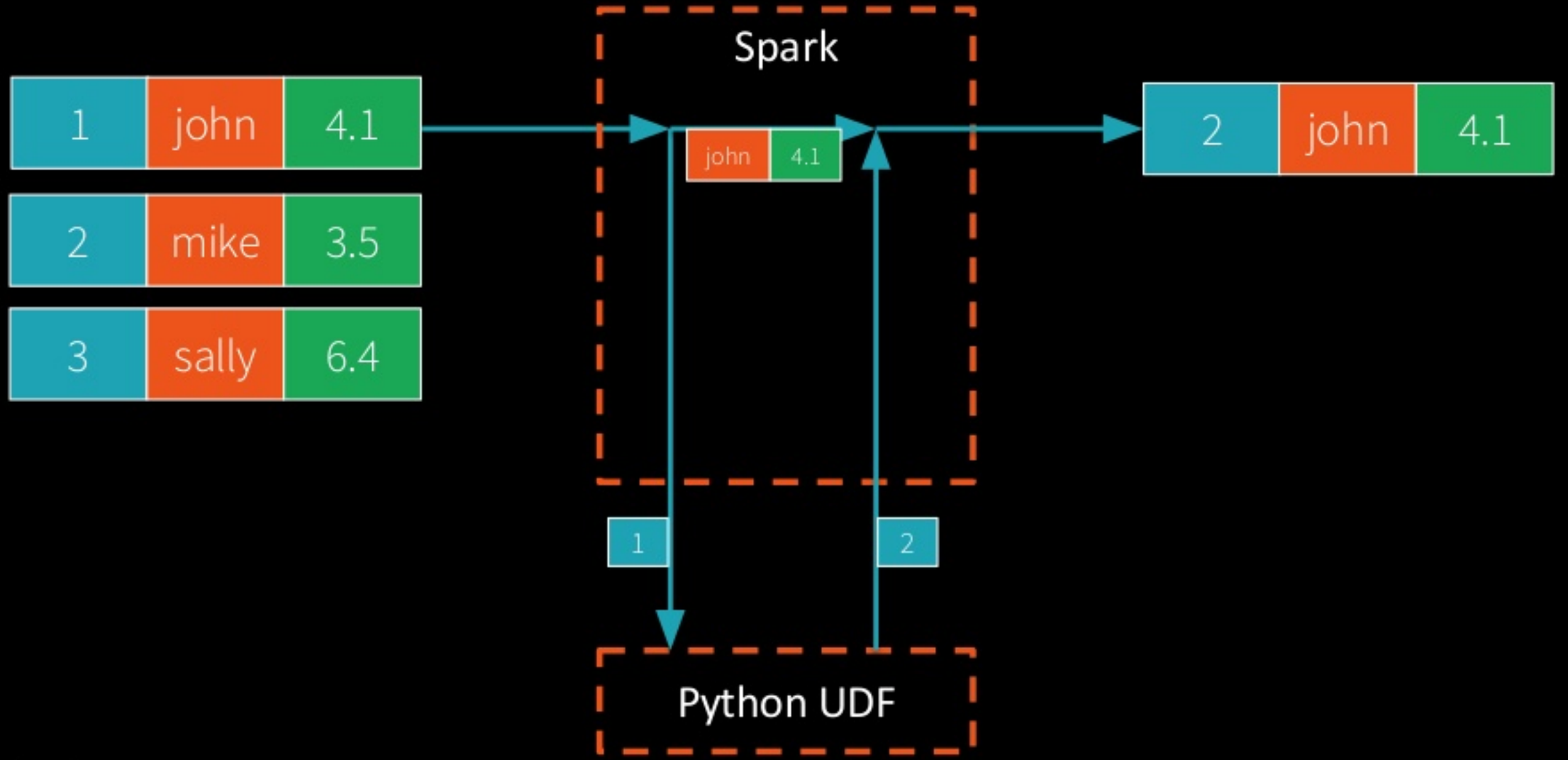
SPIP JIRA: SPARK-24579 (pending vote, ETA 3.0)

User-Defined Functions (UDFs)

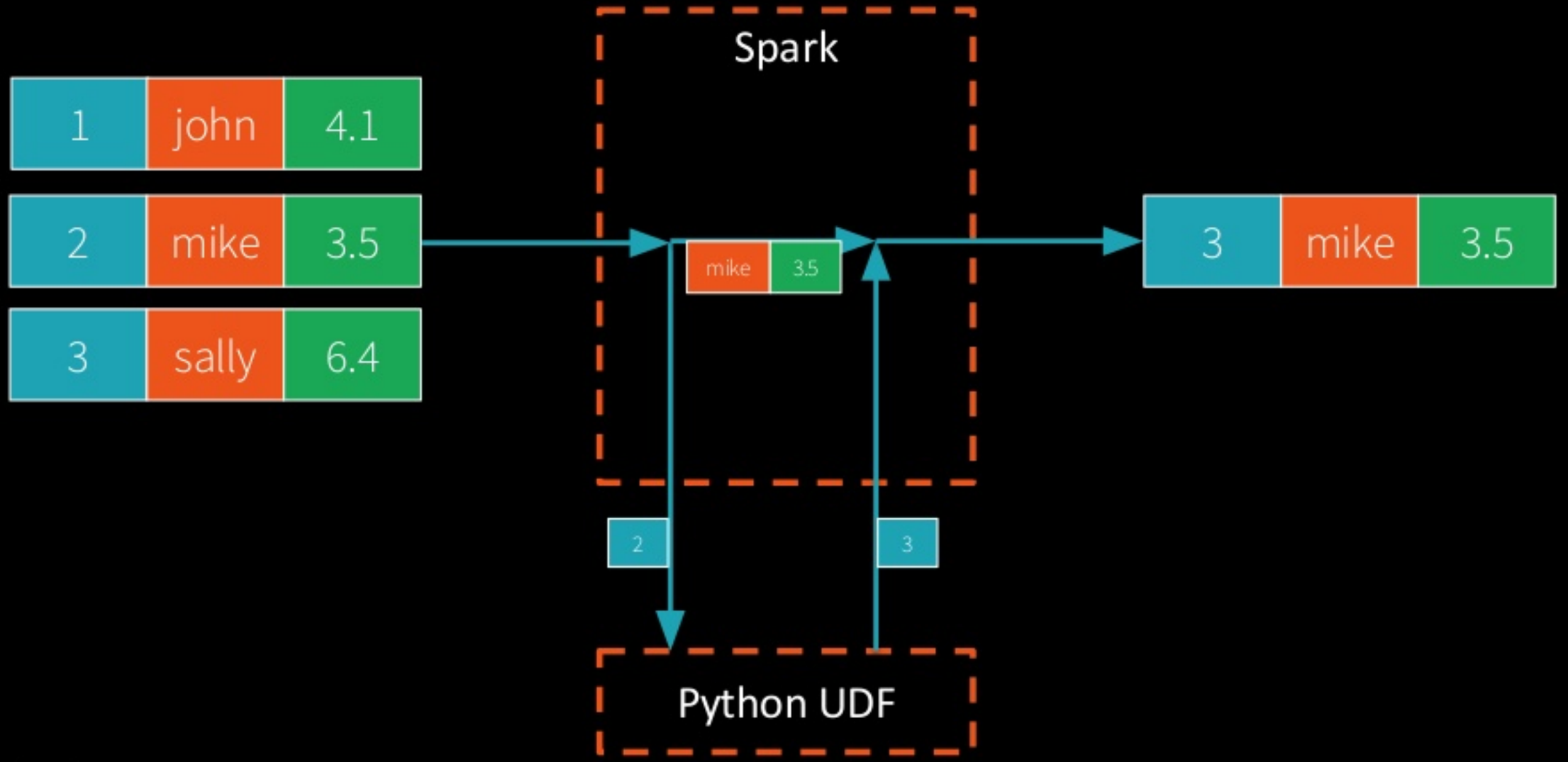
Allows executing arbitrary code, often used for integration with ML frameworks

Example: prediction on data using TensorFlow

Row-at-a-time Data Exchange



Row-at-a-time Data Exchange



Row-at-a-time Data Exchange

Profile UDF

lambda x: x + 1

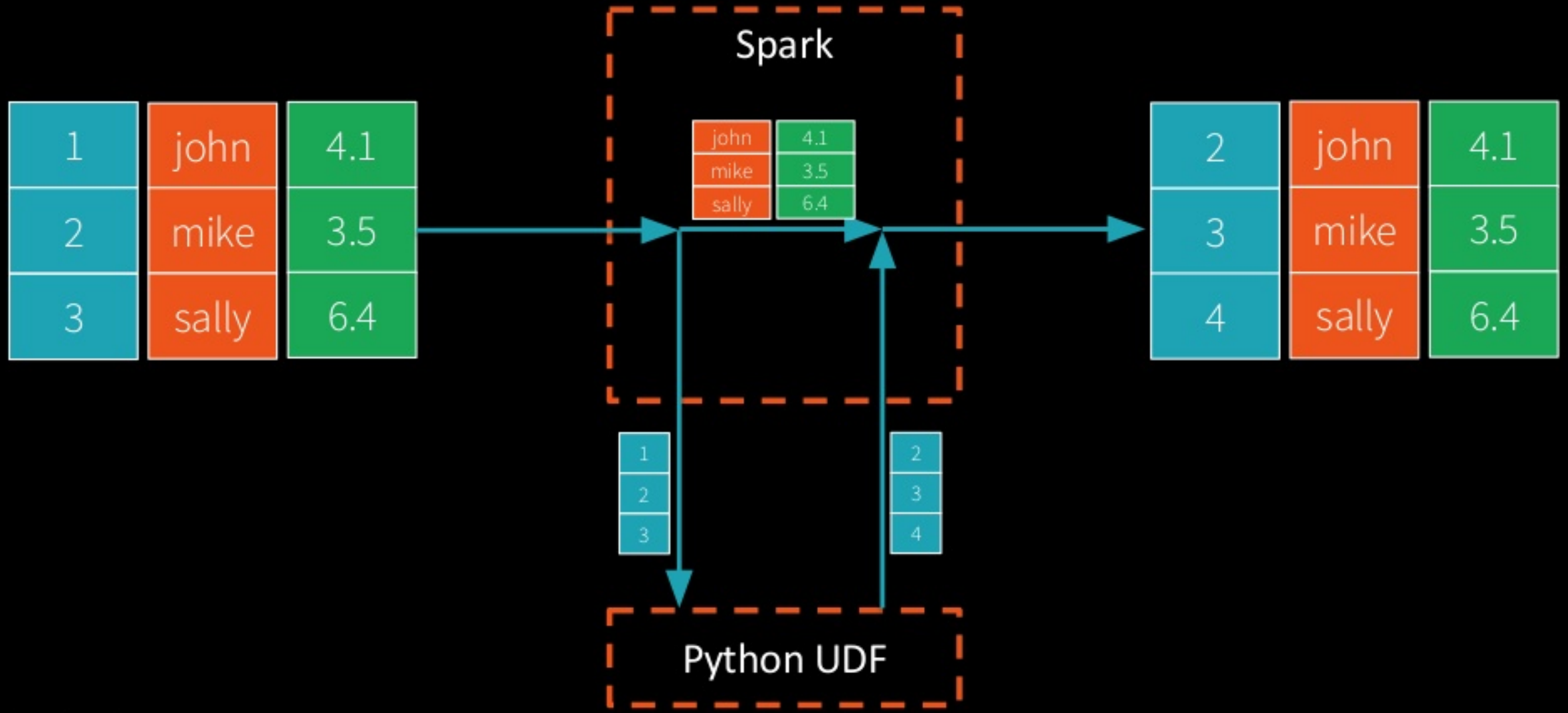
8 Mb/s

```
8787091 function calls in 4.084 seconds

Ordered by: internal time

 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  20973    1.296    0.000    3.820    0.000 serializers.py:223(_batched)
2097152    0.800    0.000    2.004    0.000 worker.py:107(<lambda>)
2097152    0.761    0.000    1.204    0.000 worker.py:72(<lambda>)
2097152    0.443    0.000    0.443    0.000 <ipython-input-2-853f857cd265>:14(<lambda>)
2097152    0.214    0.000    0.214    0.000 {method 'append' of 'list' objects}
  20972    0.153    0.000    0.153    0.000 {built-in method _pickle.loads}
  20972    0.086    0.000    0.086    0.000 {built-in method _pickle.dumps}
```

91.8% in data exchange!!!

Source: Li Jin, Pandas UDF: Scalable Analysis with Python and PySpark

Vectorized Data Exchange



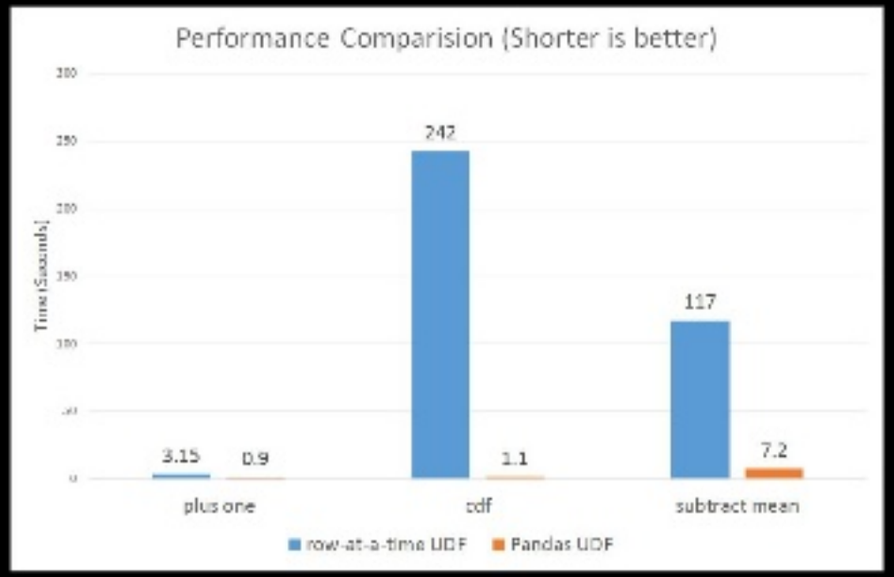
Vectorized computation

Though Spark uses a columnar storage format, the public user access interface is row-based.

```
df.toArrowRDD.map { batches =>
  ... // vectorized computation
}
```

Generalize Pandas UDF to Scala/Java

Pandas UDF was introduced in Spark 2.3, which uses Arrow for data exchange and utilizes Pandas for vectorized computation.



Demo

Project Hydrogen: Spark + AI

Barrier Execution Mode

Optimized Data Exchange

Accelerator Aware Scheduling

Accelerator-aware scheduling (SPIP)

To utilize accelerators (GPUs, FPGAs) in a heterogeneous cluster or to utilize multiple accelerators in a multi-task node, Spark needs to understand the accelerators installed on each node.

SPIP JIRA: SPARK-24615 (pending vote, ETA: 3.0)

Request accelerators

With accelerator awareness, users can specify accelerators constraints or hints (API pending discussion):

```
rdd.accelerated
   .by(“/gpu/p100”)
   .numPerTask(2)
   .required // or .optional
```

Multiple tasks on the same node

When multiples tasks are scheduled on the same node with multiple GPUs, each task knows which GPUs are assigned to avoid crashing into each other (API pending discussion):

```
// inside a task closure
val gpus = context.getAcceleratorInfos()
```

Cluster manager support

In Spark standalone mode, we can list accelerators in worker conf and aggregate the information for scheduling. We also need to provide support for other cluster managers:

Timeline

Spark 2.4

- barrier execution mode (basic scenarios)
- (Databricks) horovod integration w/ barrier mode

Spark 3.0

- barrier execution mode
- optimized data exchange
- accelerator-aware scheduling

Acknowledgement

- Many ideas in Project Hydrogen are based on previous community work: TensorFrames, BigDL, Apache Arrow, Pandas UDF, Spark GPU support, MPI, etc.

- We would like to thank many Spark committers and contributors who helped the project proposal, design, and implementation.

Thank you!