Tuning and Monitoring Deep Learning on Apache Spark

Tim Hunter

Databricks, Inc.

SPARK SUMMIT EAST 2017

About me

Software engineer at Databricks

Apache Spark contributor

Ph.D. UC Berkeley in Machine Learning

(and Spark user since Spark 0.2)

Outline

- Lessons (and challenges) learned from the field
- Tuning Spark for Deep Learning and GPUs
- Loading data in Spark
- Monitoring

Deep Learning and Spark

- 2016: the year of emerging solutions for combining Spark and Deep Learning
  - 6 talks on Deep Learning, 3 talks on GPUs
- Yet, no consensus:
  - Official MLlib support is limited (perceptron-like networks)
  - Each talk mentions a different framework

SPARK SUMMIT EAST 2017

Deep Learning and Spark

- Deep learning frameworks with Spark bindings:
  - Caffe (CaffeOnSpark)
  - Keras (Elephas)
  - mxnet
  - Paddle
  - TensorFlow (TensorFlow on Spark, TensorFrames)
- Running natively on Spark
  - BigDL
  - DeepDist
  - DeepLearning4J
  - MLlib
  - SparkCL
  - SparkNet
- Extensions to Spark for specialized hardware
  - Blaze (UCLA & Falcon Computing Solutions)
  - IBM Conductor with Spark

(Alphabetical order!)

One Framework to Rule Them All?

- Should we look for The One Deep Learning Framework?

Databricks' perspective

- Databricks: a Spark vendor on top of a public cloud
- Now provides GPU instances
- Enables customers with compute-intensive workloads
- This talk:
  - lessons learned when deploying GPU instances on a public cloud
  - what Spark could do better when running Deep Learning applications

ML in a data pipeline

- ML is small part in the full pipeline:

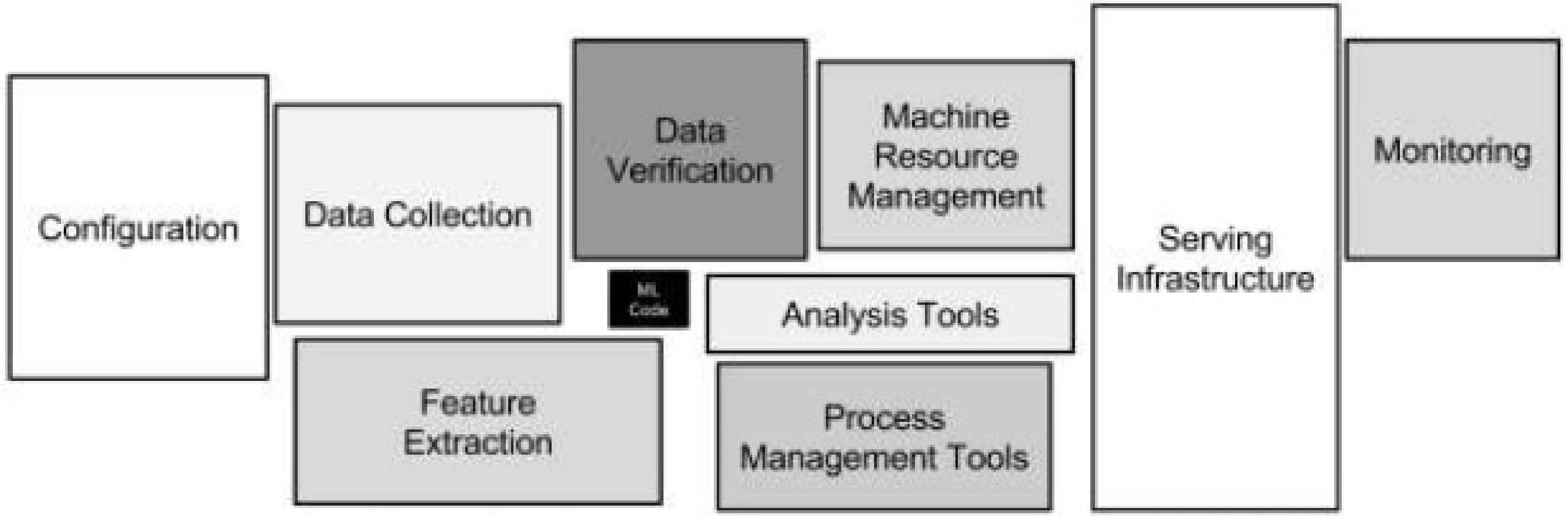
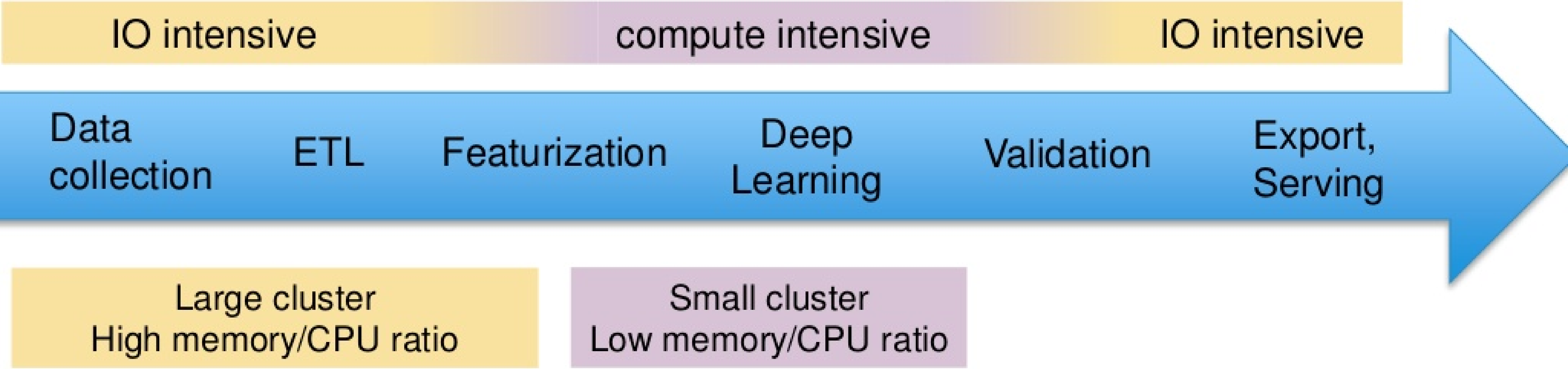


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Hidden technical debt in Machine Learning systems
Sculley et al., NIPS 2016

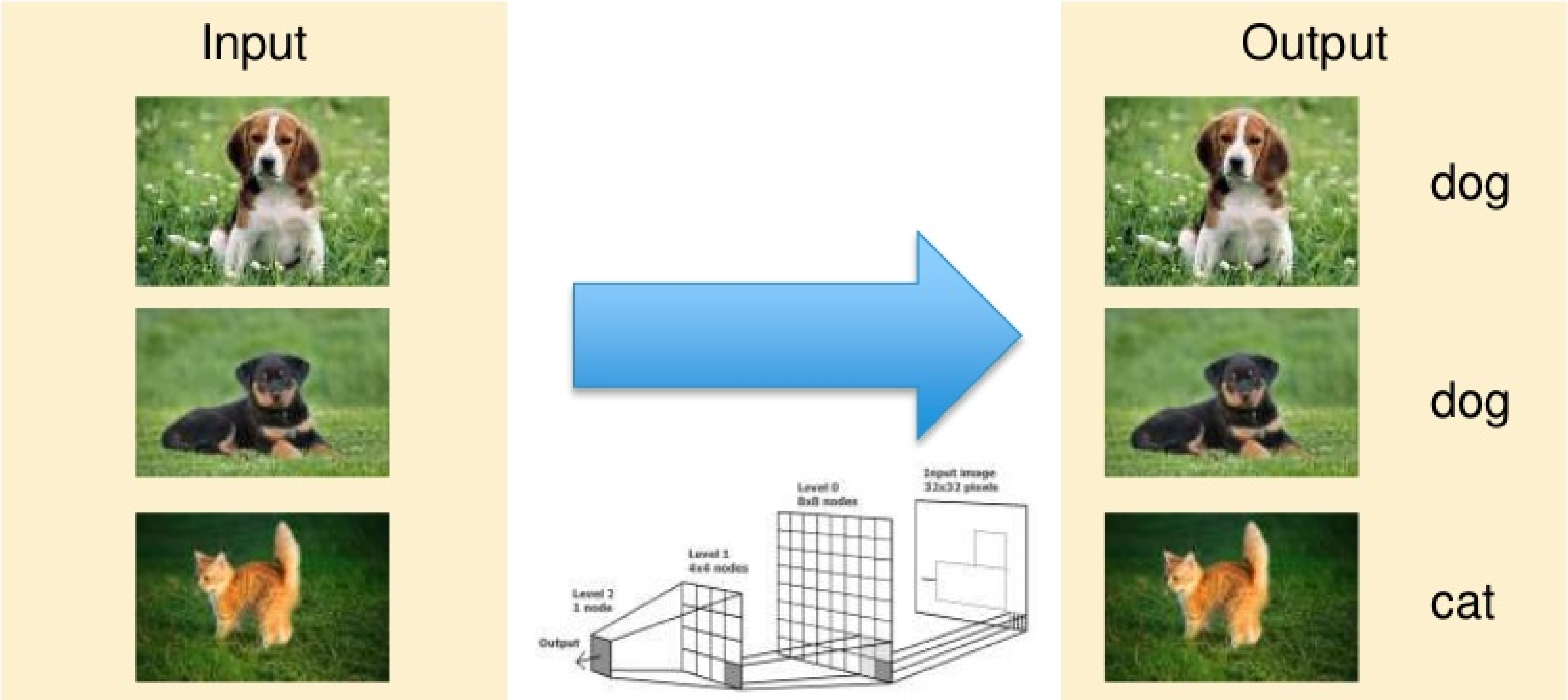
DL in a data pipeline (1)

- Training tasks:



DL in a data pipeline (2)

- Specialized data transform (feature extraction, …)



Saulius Garalevicius - CC BY-SA 3.0

Recurring patterns

- Spark as a scheduler
  - Embarrassingly parallel tasks
  - Data stored outside Spark
- Embedded Deep Learning transforms
  - Data stored in DataFrame/RDDs
  - Follows the RDD guarantees
- Specialized computations
  - Multiple passes over data
  - Specific communication patterns

Using GPUs through PySpark

- A popular choice when a lot of independent tasks
- A lot of DL packages have a Python interface: TensorFlow, Theano, Caffe, mxnet, etc.

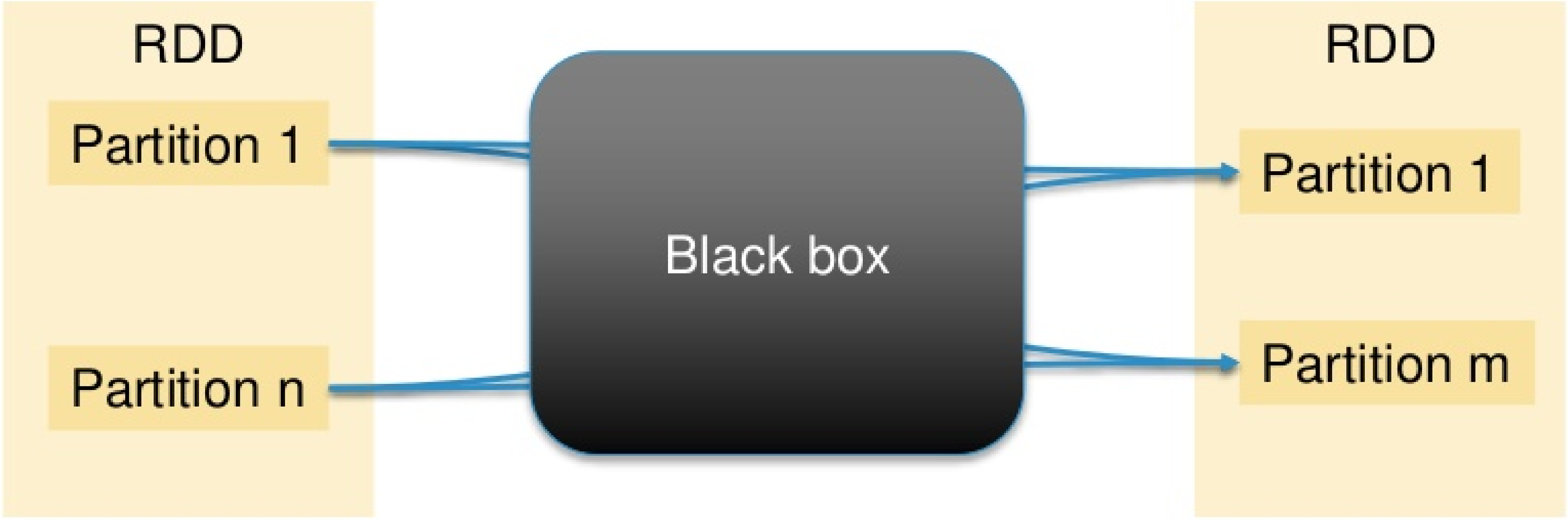
- Lifetime for python packages: the process
- Requires some configuration tweaks in Spark

PySpark recommendation

- spark.executor.cores = 1
  - Gives the DL framework full access over all the resources
  - This may be important for some frameworks that attempt to optimize the processor pipelines

Cooperative frameworks

- Use Spark for data input
- Examples:
  - IBM GPU efforts
  - Skymind's DeepLearning4J
  - DistML and other Parameter Server efforts



Cooperative frameworks

- Bypass Spark for asynchronous / specific communication patterns across machines
- Lose benefit of RDDs and DataFrames and reproducibility/determinism
- But these guarantees are not requested anyway when doing deep learning (stochastic gradient)
- “reproducibility is worth a factor of 2” (Leon Bottou, quoted by John Langford)

Streaming data through DL

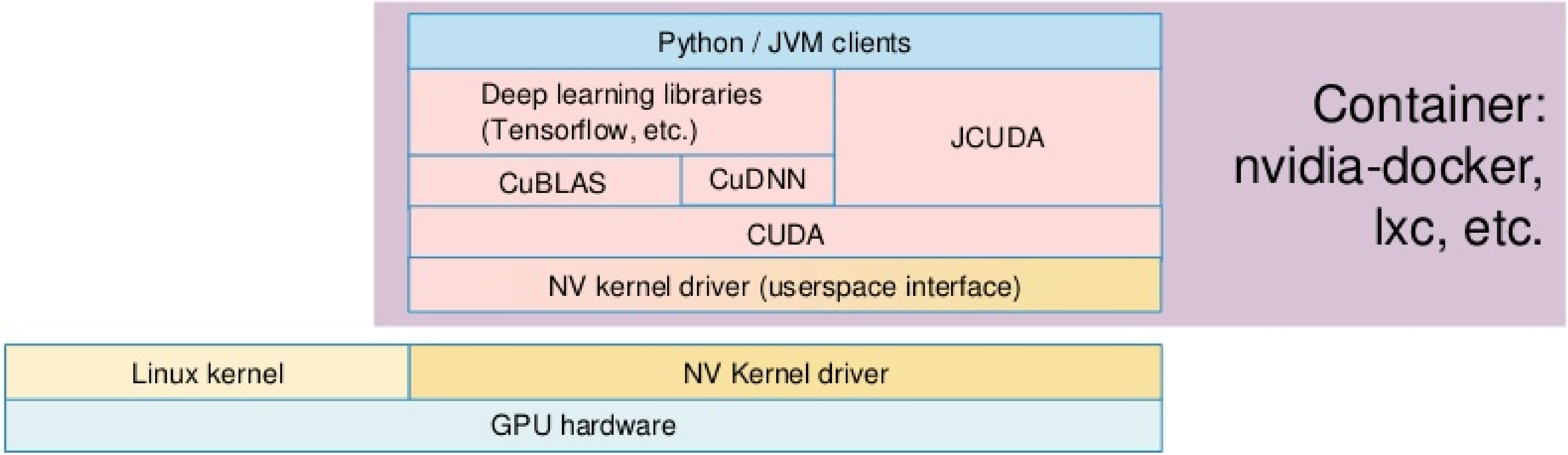
- The general choice:
  - Cold layer (HDFS/S3/etc.)
  - Local storage: files, Spark's on-disk persistence layer
  - In memory: Spark RDDs or Spark Dataframes
- Find out if you are I/O constrained or processor-constrained
  - How big is your dataset? MNIST or ImageNet?
- If using PySpark:
  - All frameworks heavily optimized for disk I/O
  - Use Spark's broadcast for small datasets that fit in memory
  - Reading files is fast: use local files when it does not fit
- If using a binding:
  - each binding has its own layer
  - in general, better performance by caching in files (same reasons)

Detour: simplifying the life of users

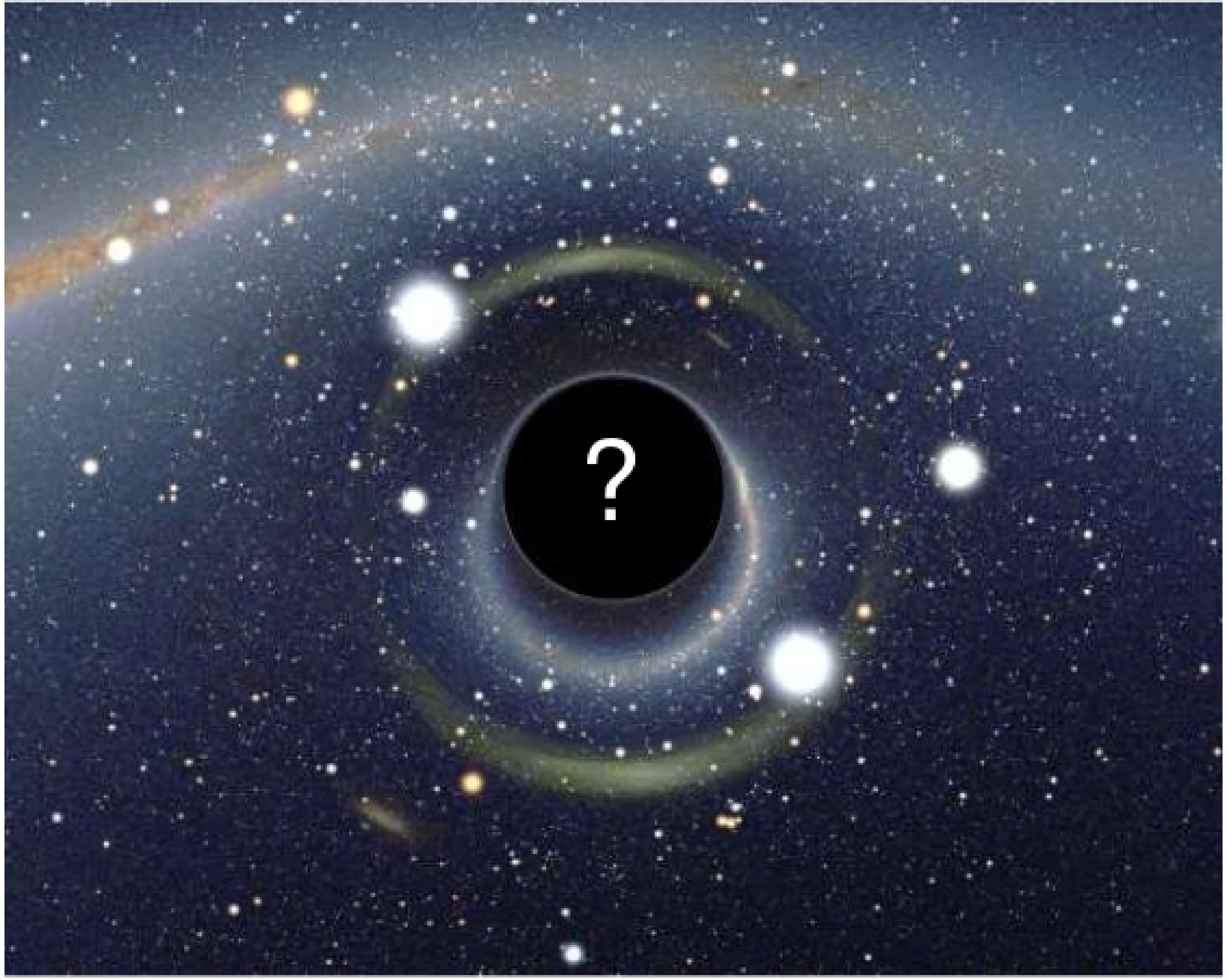
- Deep Learning commonly used with GPUs
- A lot of work on Spark dependencies:
  - Few dependencies on local machine when compiling Spark
  - The build process works well in a large number of configurations (just scala + maven)
- GPUs present challenges: CUDA, support libraries, drivers, etc.
  - Deep software stack, requires careful construction (hardware + drivers + CUDA + libraries)
  - All these are expected by the user
  - Turnkey stacks just starting to appear

Simplifying the life of users

- Provide a Docker image with all the GPU SDK
- Pre-install GPU drivers on the instance



Monitoring



Monitoring

- How do you monitor the progress of your tasks?
- It depends on the granularity
  - Around tasks
  - Inside (long-running) tasks

Monitoring: Accumulators

- Good to check throughput or failure rate
- Works for Scala
- Limited use for Python (for now, SPARK-2868)
- No “real-time” update

```
batchesAcc = sc.accumulator(1)

def processBatch(i):
    global acc
    acc += 1
    # Process image batch here

images = sc.parallelize(…)
images.map(processBatch).collect()
```

SPARK SUMMIT EAST 2017

Monitoring: external system

- Plugs into an external system
- Existing solutions: Grafana, Graphite, Prometheus, etc.
- Most flexible, but more complex to deploy

Conclusion

- Distributed deep learning: exciting and fast-moving space
- Most insights are specific to a task, a dataset and an algorithm: nothing replaces experiments
- Easy to get started with parallel jobs
- Move in complexity only when enough data/insufficient patience

Challenges to address

- For Spark developers
  - Monitoring long-running tasks
  - Presenting and introspecting intermediate results
- For DL developers:
  - What boundary to put between the algorithm and Spark?
  - How to integrate with Spark at the low-level?

Thank You.

Check our blog post on Deep Learning and GPUs!

https://docs.databricks.com/applications/deep-learning/index.html

Office hours: today 3:50 at the Databricks booth

SPARK SUMMIT EAST 2017