Foundations for Scaling Analytics in Apache® Spark™

Joseph K. Bradley

September 19, 2016

Apache Spark™

databricks™

Who am I?

Apache Spark committer & PMC member

Software Engineer @ Databricks (ML team)

Machine Learning Department @ Carnegie Mellon

Talk outline

- Intro
  - Apache Spark
  - Machine Learning (and graphs) in Spark
- Original implementations: RDDs
- Future implementations: DataFrames

Apache Spark

- General engine for big data computing
- Fast & scalable
- Easy to use
- APIs in Python, Scala, Java & R

Open source

- Apache Software Foundation
- 1000+ contributors
- 250+ companies & universities

Spark SQL	Streaming	MLlib	GraphX

APACHE Spark™

It's big

- Spark beat Hadoop's Gray Sort record by 3x with 1/10 as many machines
- Largest cluster size of 8000 Nodes (Tencent)

MLlib: Spark's ML library

Goals

Scale-out

Standard library

Extensible API

Data utilities

Featurization

Statistics

Linear algebra

ML tasks

Classification

Regression

Recommendation

Clustering

Frequent itemsets

Workflow utilities

Model import/export

Pipelines

DataFrames

Cross validation

Challenges for big data

- Iterative algorithms
- Diverse algorithmic patterns
- Many data types

GraphX and GraphFrames

Goals
Scale-out
Standard library
Extensible API

Graph algorithms
Connected components
PageRank
Label propagation
…

Graph queries
Vertex degrees
Subgraphs
Motif finding
…

Challenges for big data

- Iterative algorithms
- Many (big) joins
- Many data types

Talk outline

- Intro
  - Apache Spark
  - Machine Learning (and graphs) in Spark
- Original implementations: RDDs
- Future implementations: DataFrames

Talk outline

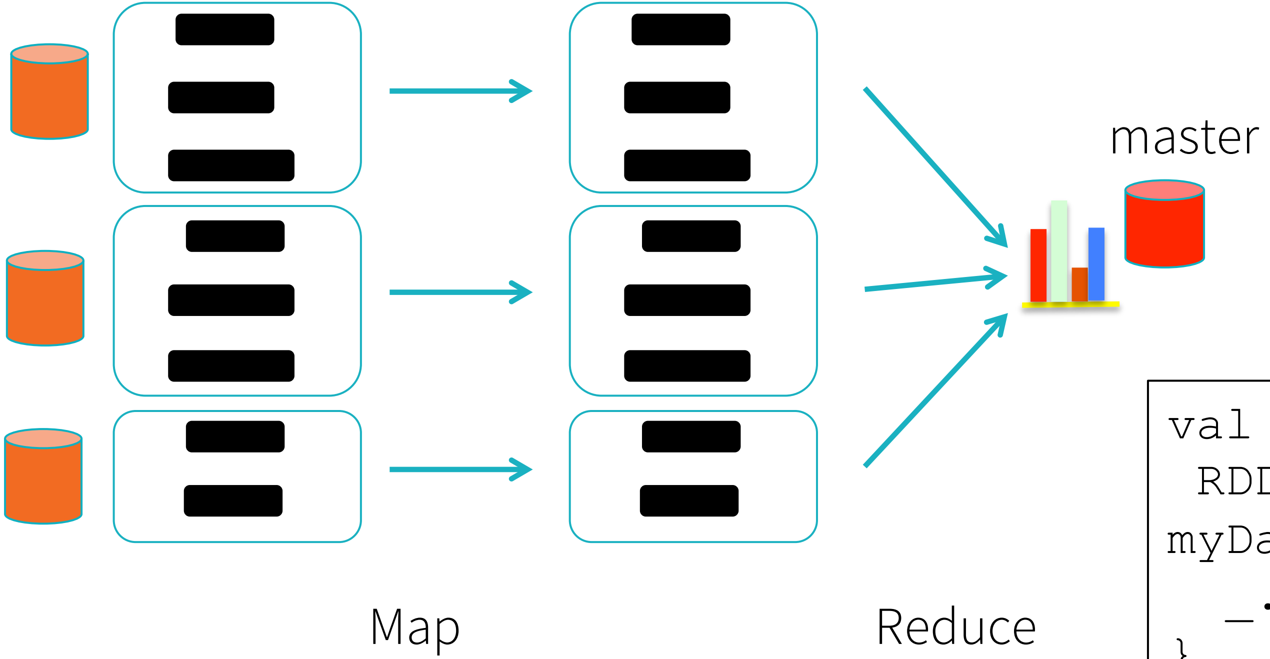
Intro

- Apache Spark
- Machine Learning (and graphs) in Spark

Original implementations: RDDs

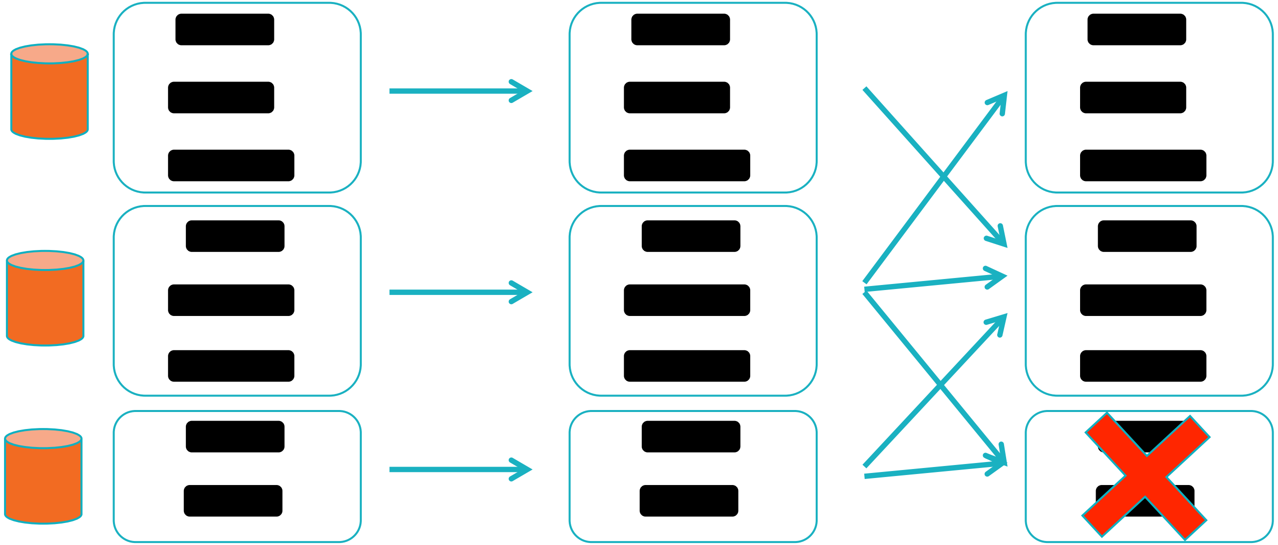
Future implementations: DataFrames

Resilient Distributed Datasets (RDDs)

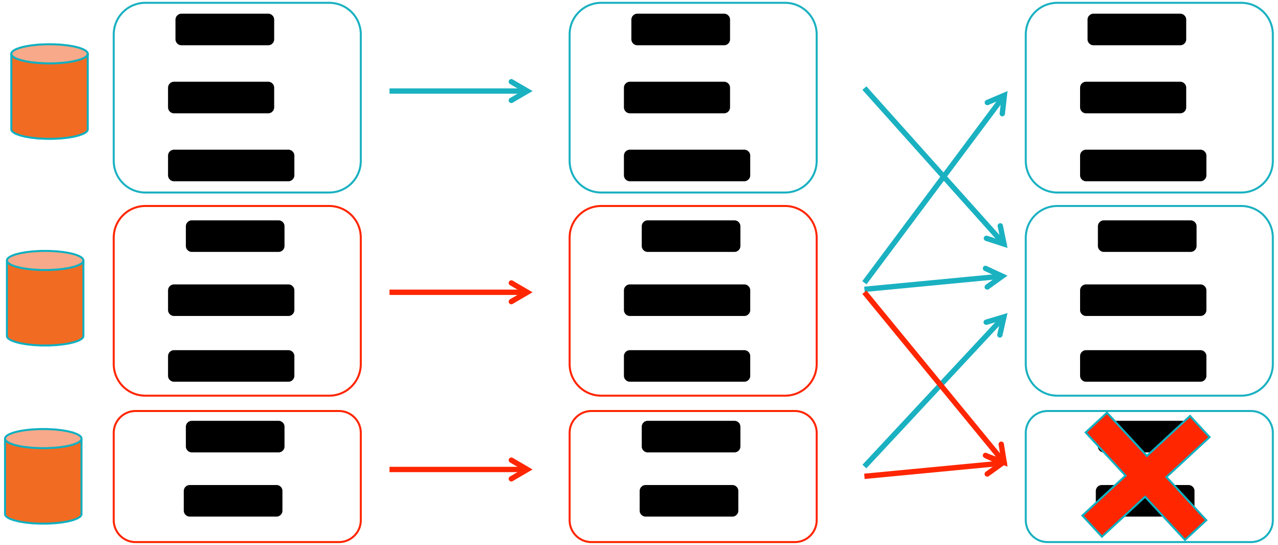



```
val myData:
 RDD[(String, Vector)]
myData.map {
  _._2 * 0.5
}
```

Resilient Distributed Datasets (RDDs)



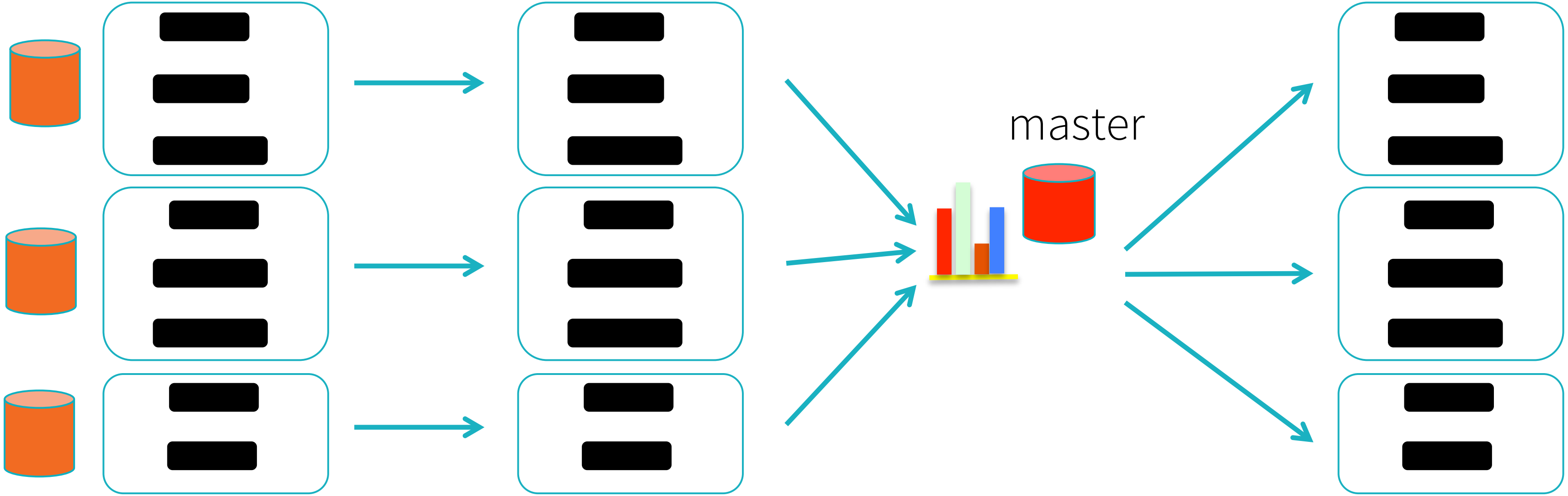
Resilient Distributed Datasets (RDDs)



Resiliency

- Lineage
- Caching & checkpointing

ML on RDDs



Compute gradient (Vector)
for each row (training example)

Aggregate
gradient

Broadcast
gradient

ML on RDDs: the good

Flexible: GLMs, trees, matrix factorization, etc.

Scalable: E.g., Alternating Least Squares on Spotify data (2014)

- 50+ million users x 30+ million songs
- 50 billion ratings

Cost ~ $10

- 32 r3.8xlarge nodes (spot instances)
- For rank 10 with 10 iterations, ~1 hour running time.

ML on RDDs: the challenges

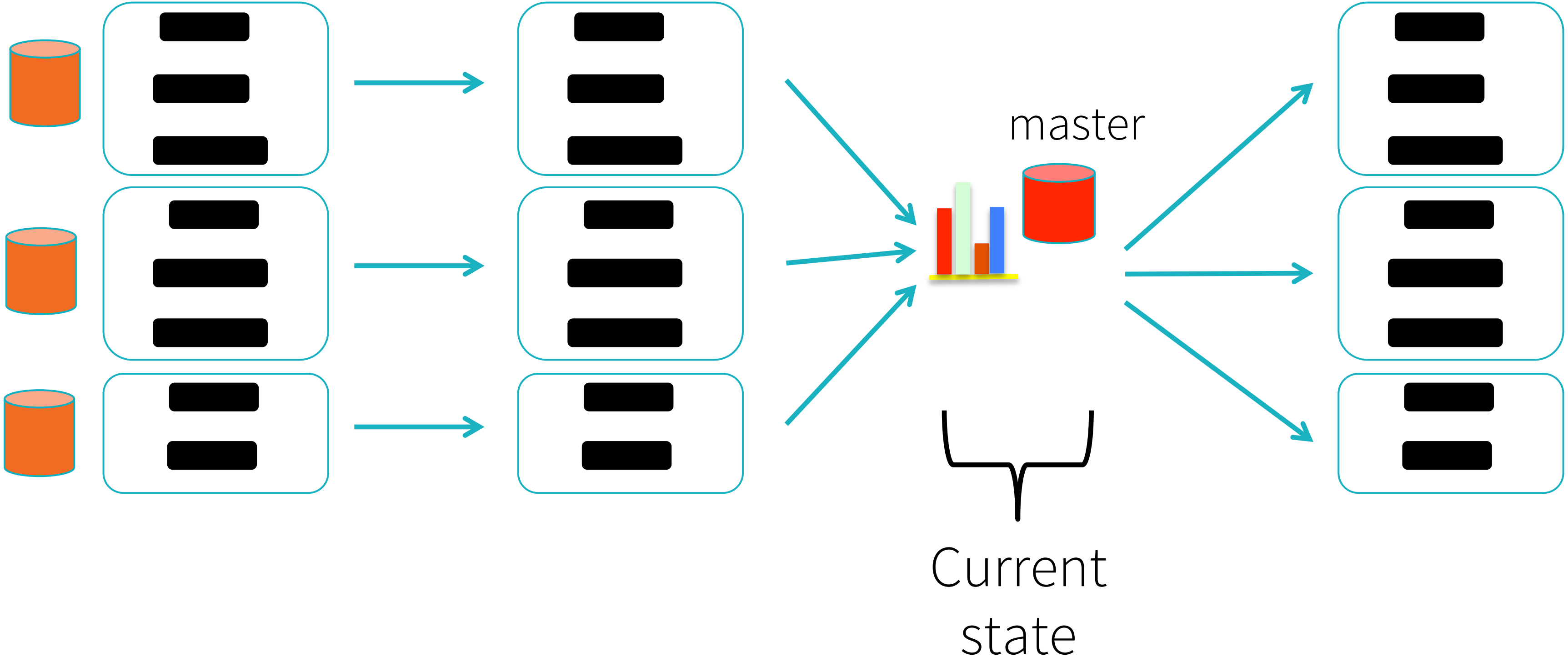
Maintaining state

Python API

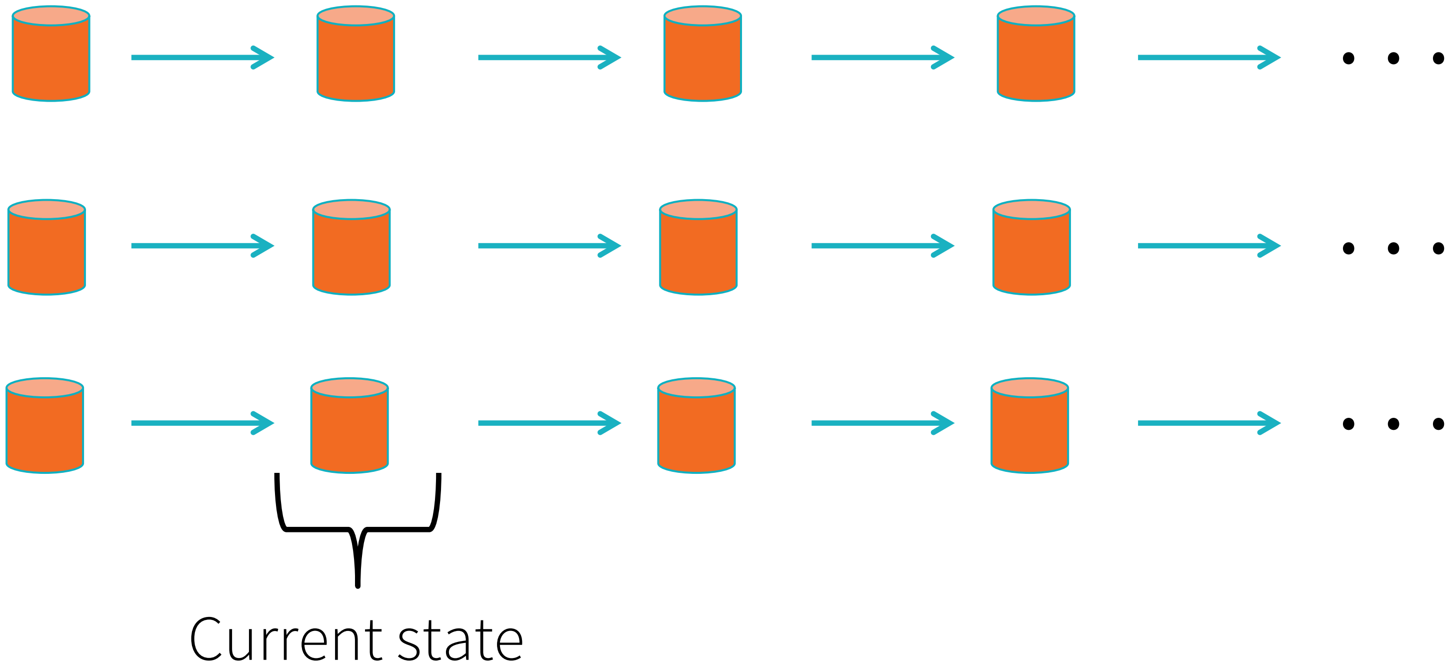
Iterator model

Data partitioning

Maintaining state on master



Maintaining state in RDDs



Maintaining state

Cons of master

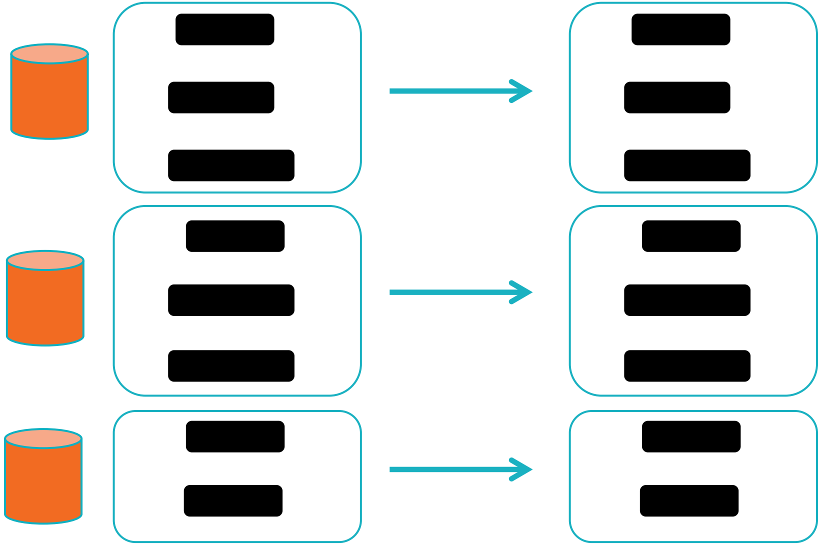
- Single point of failure.
- Cannot support large state (1 billion parameters)

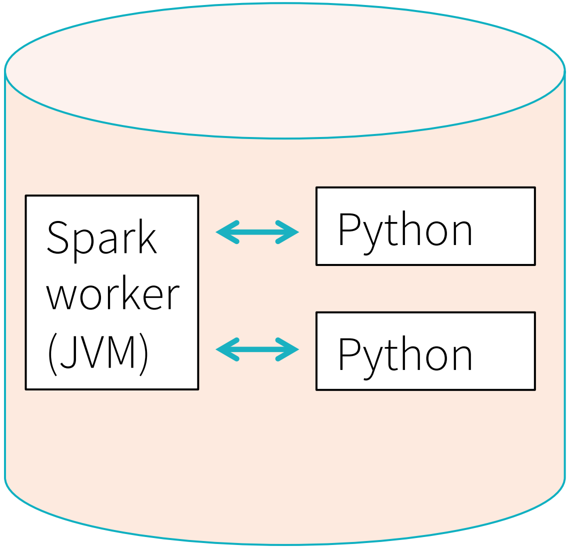
Cons of RDDs

- More complex
- Lineage becomes a problem → cache & checkpoint

Unstated con: Developers have to choose 1 option!

Python API (RDD-based)





Data stored as Python objects
→ Serialization overhead

Iterator model

```
val rdd0: RDD[(String, Vector)] = …
val rdd1 = rdd0.map { (name, data) =>
   (name.trim, normalizeVec(data))
}
val rdd2 = rdd1.map {
   ...
}
```

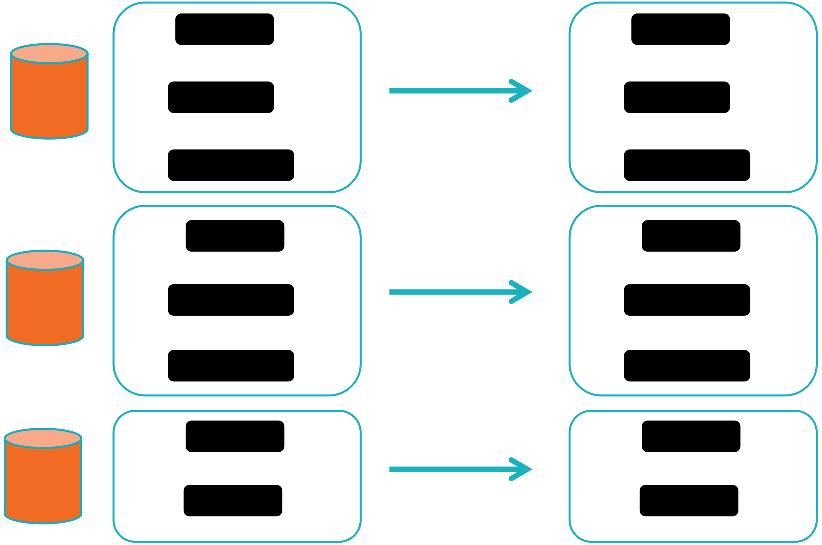
Arbitrary data types

Black box lambda functions

Iterative processing (especially in ML!)

→ Boxed types
→ JVM object creation & GC

Data partitioning: numPartitions



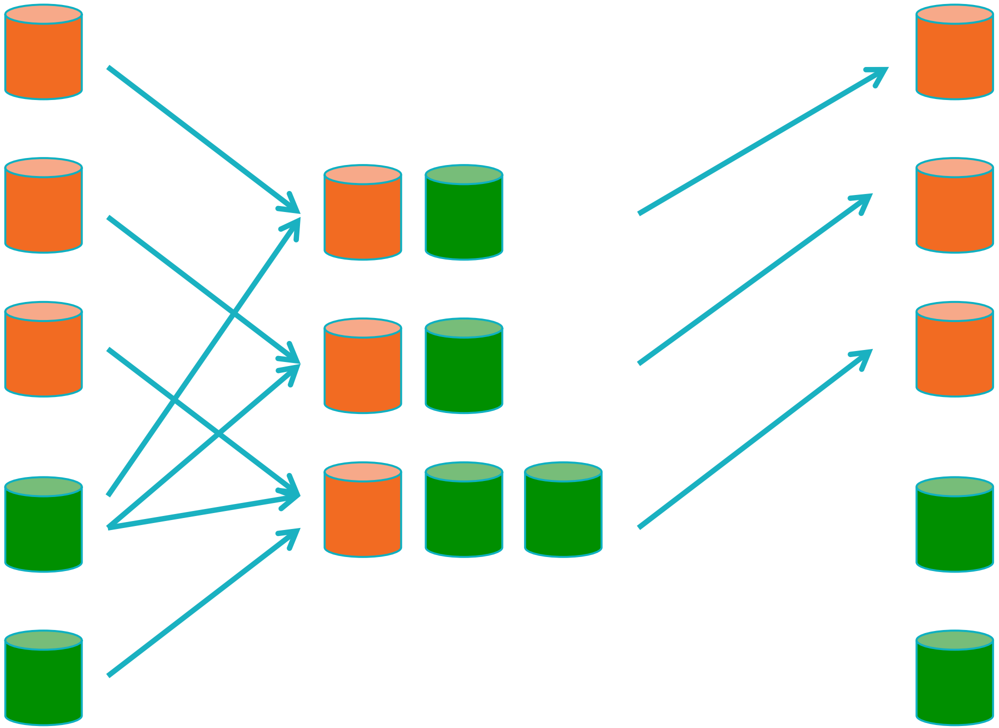
Selecting numPartitions can be critical.

- Each task has overhead.
- Overhead / parallelism trade-off.

Different numPartitions for different jobs:

- SQL: 200+ is reasonable
- ML: 1 per compute core

Data partitioning: co-partitioning



<u>Algorithm</u>

- Join
- Map
- Iterate

Co-partitioning is critical for

- ALS (matrix factorization)
- Graph algorithms

ML on RDDs: the challenges

Maintaining state (& lineage)

Python API

Iterator model

Data partitioning

Talk outline

Intro

- Apache Spark
- Machine Learning (and graphs) in Spark

Original implementations: RDDs

Future implementations: DataFrames

Talk outline

- Intro
  - Apache Spark
  - Machine Learning (and graphs) in Spark
- Original implementations: RDDs
- **Future implementations: DataFrames**

databricks

Spark DataFrames & Datasets

```
data.groupBy("dept").avg("age")
```

dept	age	name
Bio	48	H Smith
CS	34	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

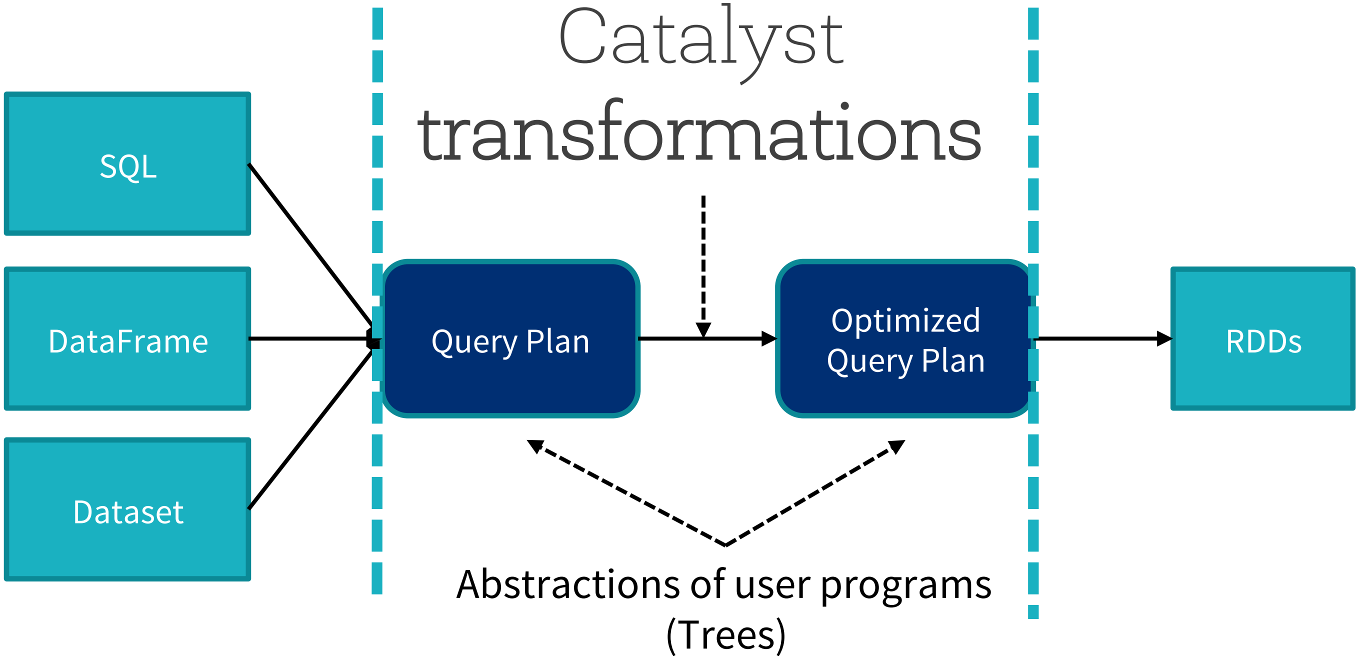
Data grouped into named columns

DSL for common tasks

- Project, filter, aggregate, join, …
- Statistics, n/a values, sketching, …
- User-Defined Functions (UDFs) & Aggregation (UDAFs)

Datasets: Strongly typed DataFrames

Catalyst query optimizer



Project Tungsten

Memory management

- Off-heap (Java Unsafe API)
- Avoid JVM GC
- Compressed format

Code generation

- Rewrite chain of iterators into single code blocks
- Operate directly on compressed format

DataFrames in ML and Graphs

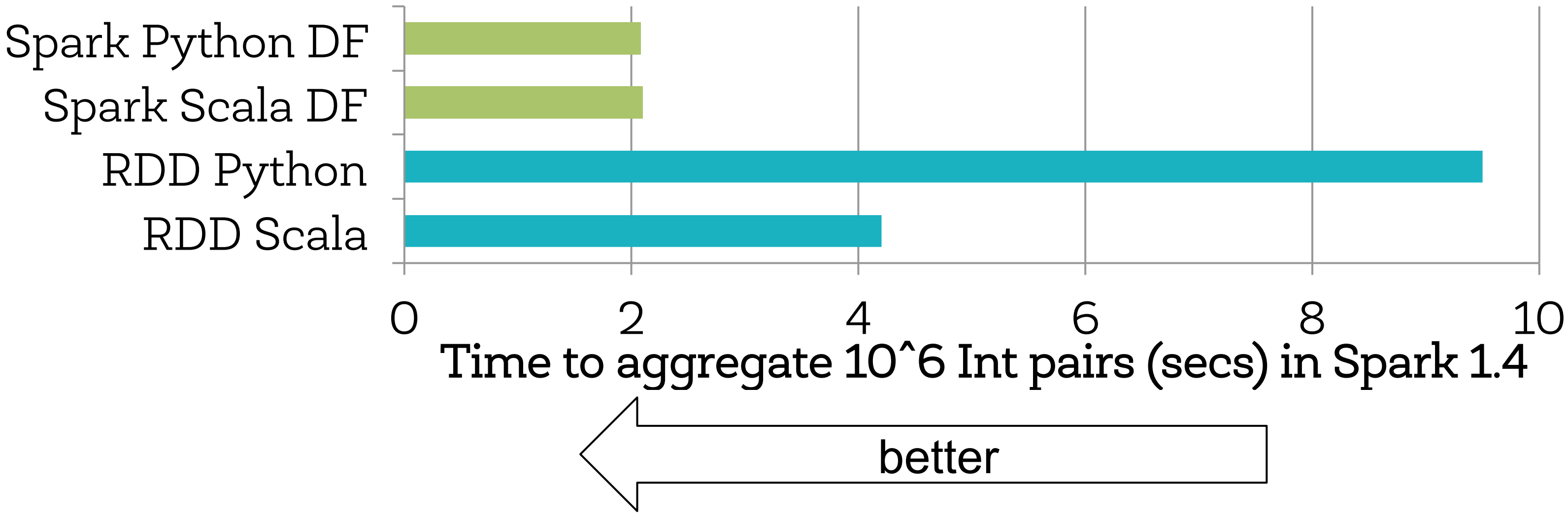
API

- DataFrame-based API in MLlib (spark.ml package)
- GraphFrames (Spark package)

Transformation & prediction

Training

Python API


Spark Python DF
Spark Scala DF
RDD Python
RDD Scala
0
2
4
6
8
10
Time to aggregate 10^6 Int pairs (secs) in Spark 1.4
better

Transformation/prediction with DataFrames

User-Defined Types (UDTs)

- Vector (sparse & dense)
- Matrix (sparse & dense)

User-Defined Functions (UDFs)

- Feature transformation
- Model prediction

Whole-stage code generation

- Fuse across multiple operators

databricks

Future work: model training

Goal: Port all ML/graph algorithms to run on DataFrames for better speed & scalability.

Currently:

- Belief propagation
- Connected components

Catalyst in ML

What's missing?

- Concept of iteration
- Handling caching and checkpointing across *many* iterations
- ML/Graph-specific optimizations for Catalyst query planner

Tungsten in ML

Partly done

- Vector/Matrix UDTs
- UDFs for some operations

What's missing?

- Code generation for critical paths
- Closer integration of Vector/Matrix types with Tungsten

OOMing

DataFrames automatically spill to disk

→ Classic pain point of RDDs

`java.lang.OutOfMemoryError`

Goal: Smoothly scale, without custom per-algorithm optimizations

To summarize…

MLlib on RDDs

- Required custom optimizations

MLlib with a DataFrame-based API

- Friendly API
- Improvements for prediction

In the future

- Potential for even greater scaling for training
- Simpler for non-experts to write new algorithms

Get started

Get involved

- JIRA http://issues.apache.org
- mailing lists http://spark.apache.org
- Github http://github.com/apache/spark
- Spark Packages http://spark-packages.org

Try out Apache Spark 2.0 in Databricks Community Edition

http://databricks.com/ce

Learn more

- New in Apache Spark 2.0
  http://databricks.com/blog/2016/06/01
- MOOCs on EdX http://databricks.com/spark/training

Many thanks to the community for contributions & support!

Databricks

We're hiring!

Founded by the creators of Apache Spark

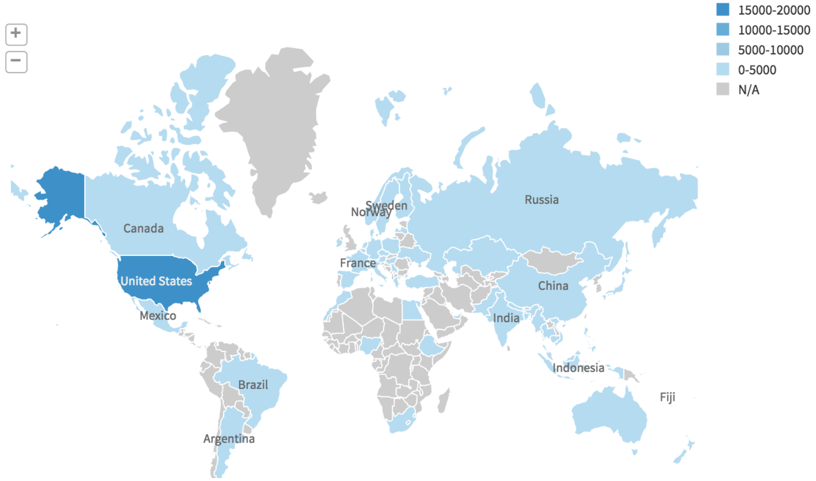
Offers hosted service

- Spark on EC2
- Notebooks
- Visualizations
- Cluster management
- Scheduled jobs

Mobile Devices by Geography (Sample Data)

This is a world map of number of mobile phones by country from a sample dataset

```
> select m.ClientID, c.CountryCode3, m.DeviceMake
  from mobile_sample m
    join countrycodes c
      on m.Country = c.Country
```



Thank you!

FB group: Databricks at CMU

databricks.com/careers

databricks™