Intro to Spark and Spark SQL

AMP Camp 2014

Michael Armbrust - @michaelarmbrust

What is Apache Spark?

Fast and general cluster computing system, interoperable with Hadoop, included in all major distros

Improves efficiency through:

- In-memory computing primitives
- General computation graphs

→ Up to 100× faster (2-10× on disk)

Improves usability through:

- Rich APIs in Scala, Java, Python
- Interactive shell

→ 2-5× less code

amplab UC BERKELEY databricks™

Spark Model

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets (RDDs)

- Collections of objects that can be stored in memory or disk across a cluster
- Parallel functional transformations (map, filter, …)
- Automatically rebuilt on failure

More than Map & Reduce

```
map
filter
groupBy
sort
union
join
leftOuterJoin
rightOuterJoin
```

```
reduce
count
fold
reduceByKey
groupByKey
cogroup
cross
zip
```

```
sample
take
first
partitionBy
mapWith
pipe
save
...
```

Example: Log Mining

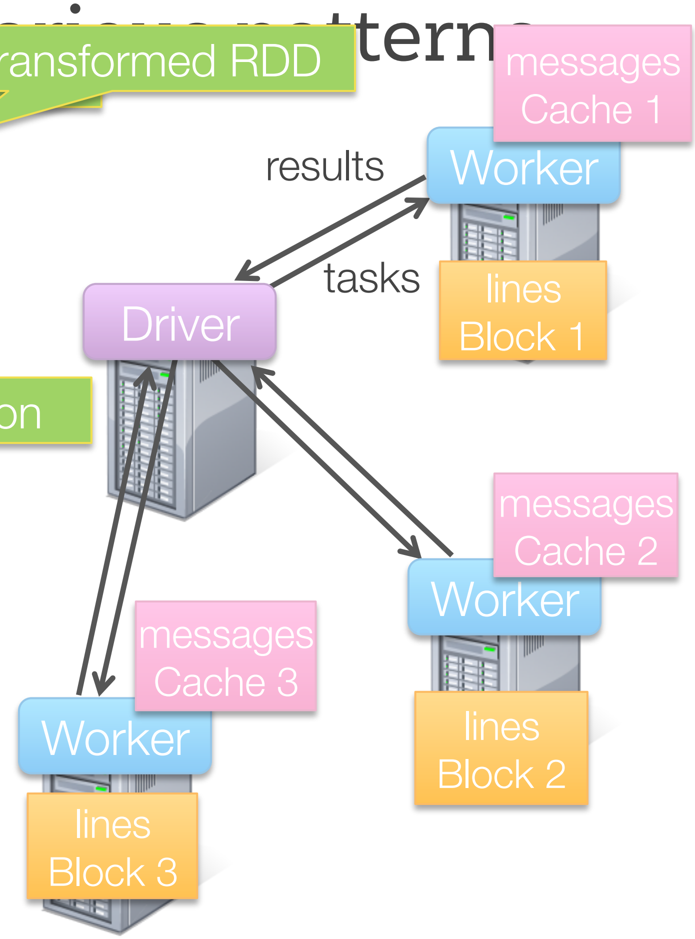
Load error messages from a log into memory, then interactively search for various patterns

Base RDD

Transformed RDD

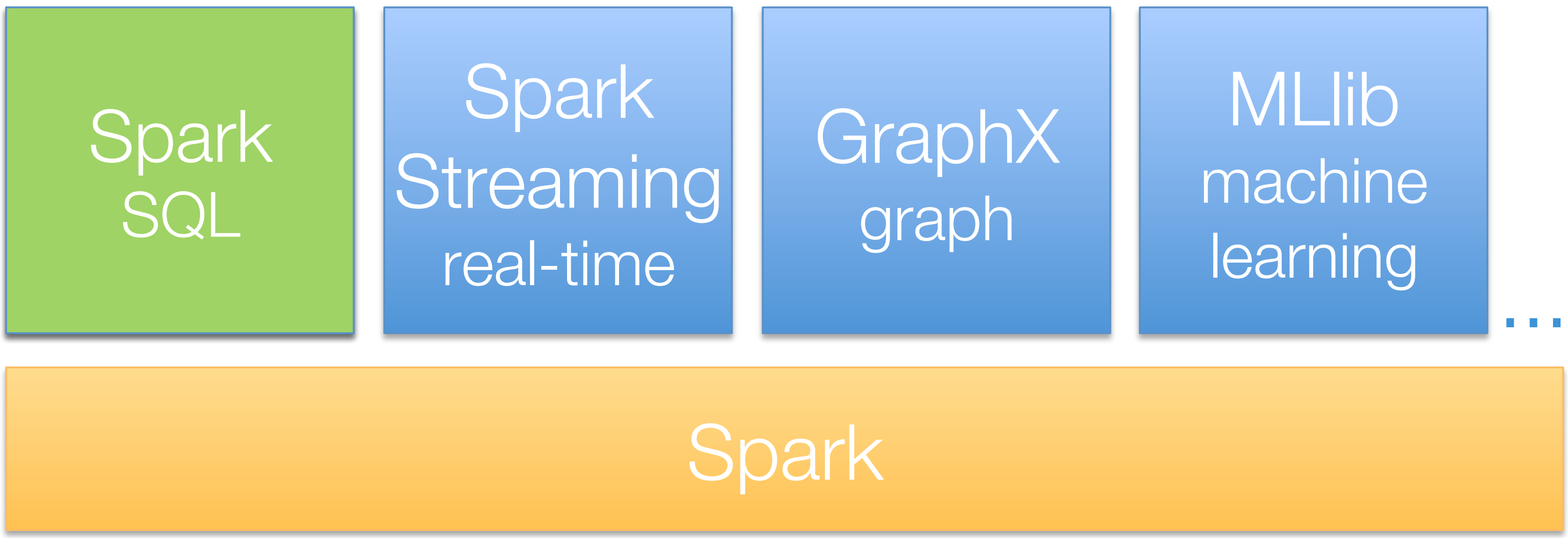
```
val lines = spark.textFile("hdfs://...")
val errors = lines.filter(_ startswith "ERROR")
val messages = errors.map(_.split("\t")(2))
messages.cache()

messages.filter(_ contains "foo").count()
messages.filter(_ contains "bar").count()
. . .
```

Action



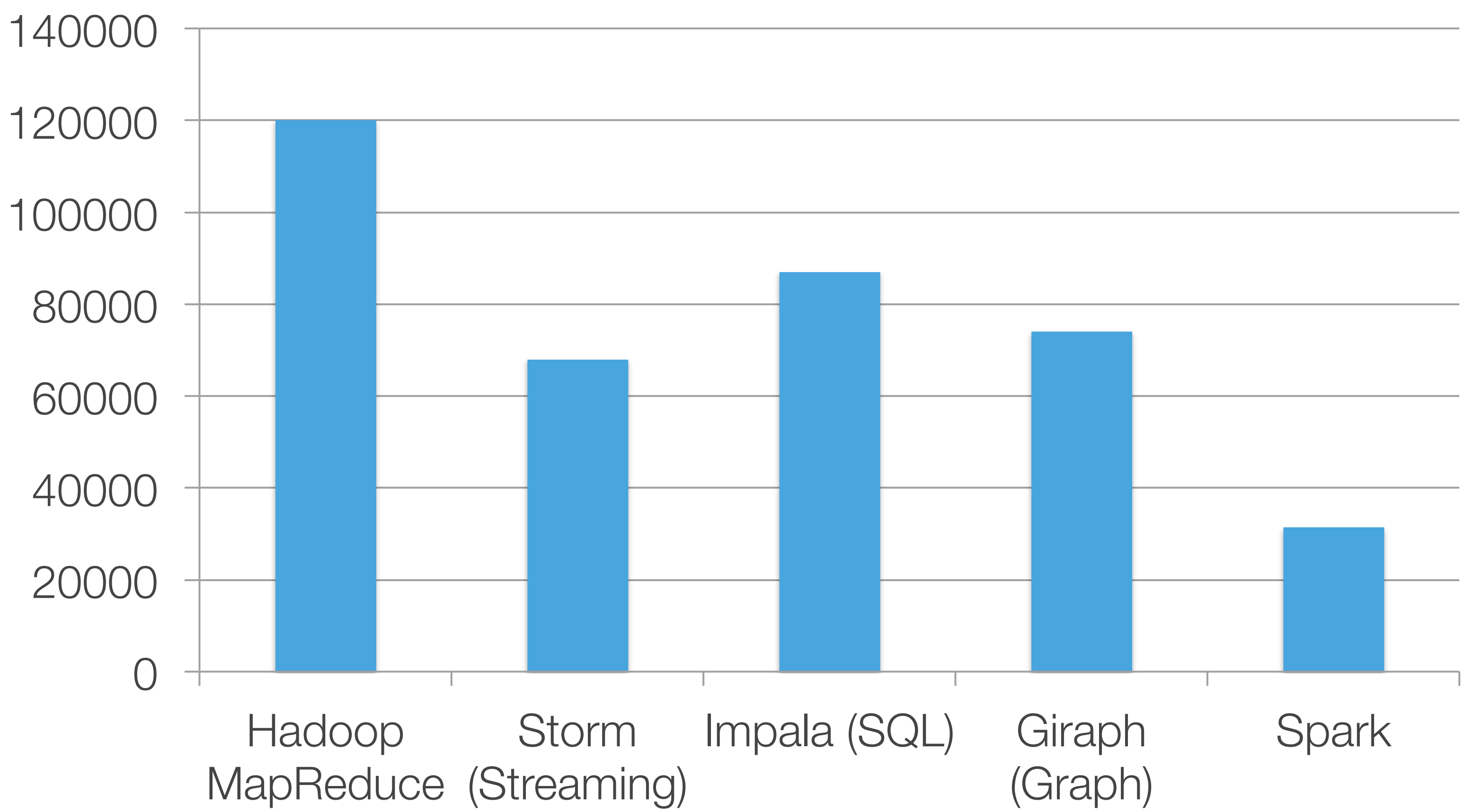
Result: scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)

A General Stack



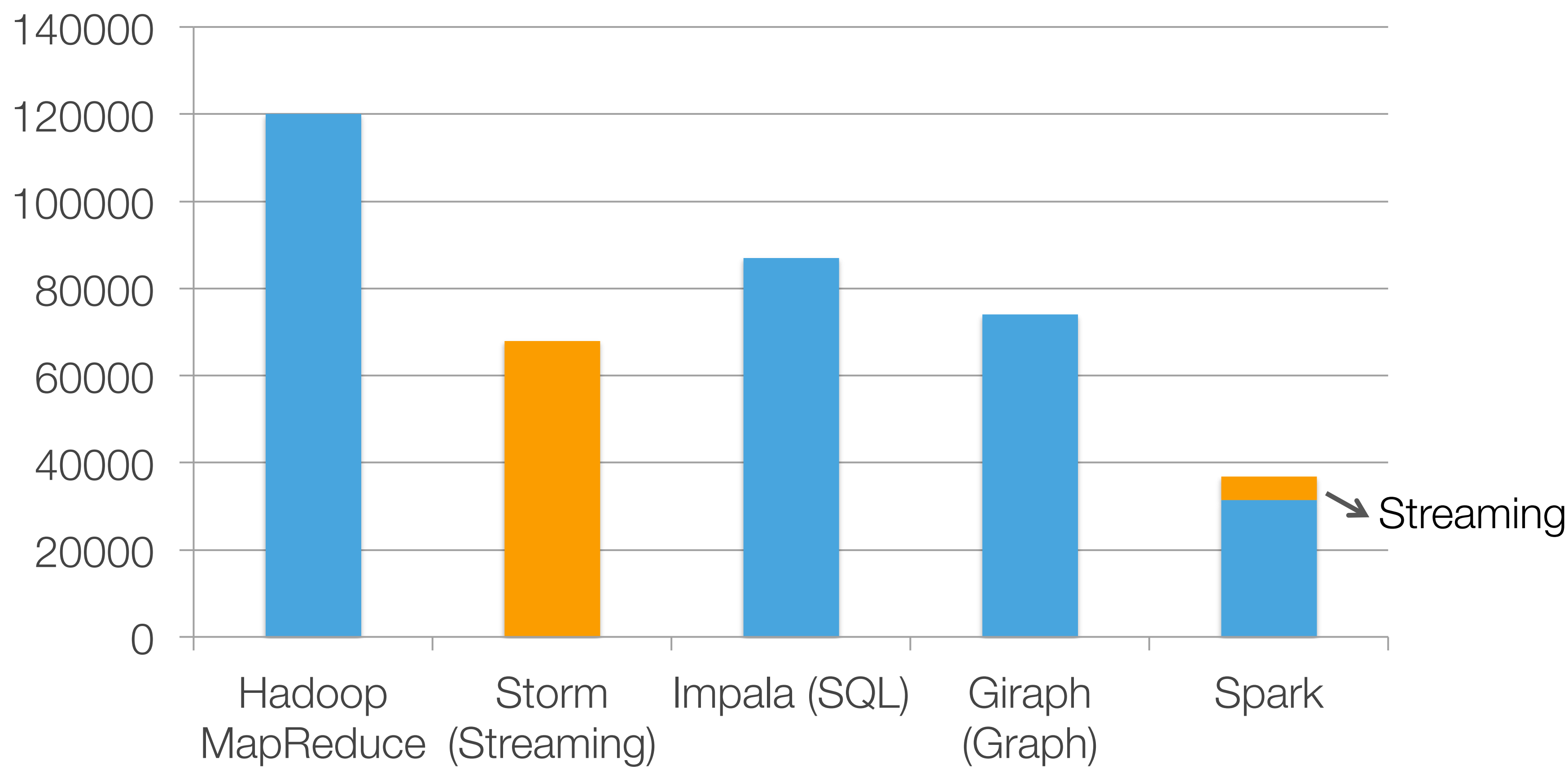
amplab UC BERKELEY databricks™

Powerful Stack – Agile Development



non-test, non-example source lines

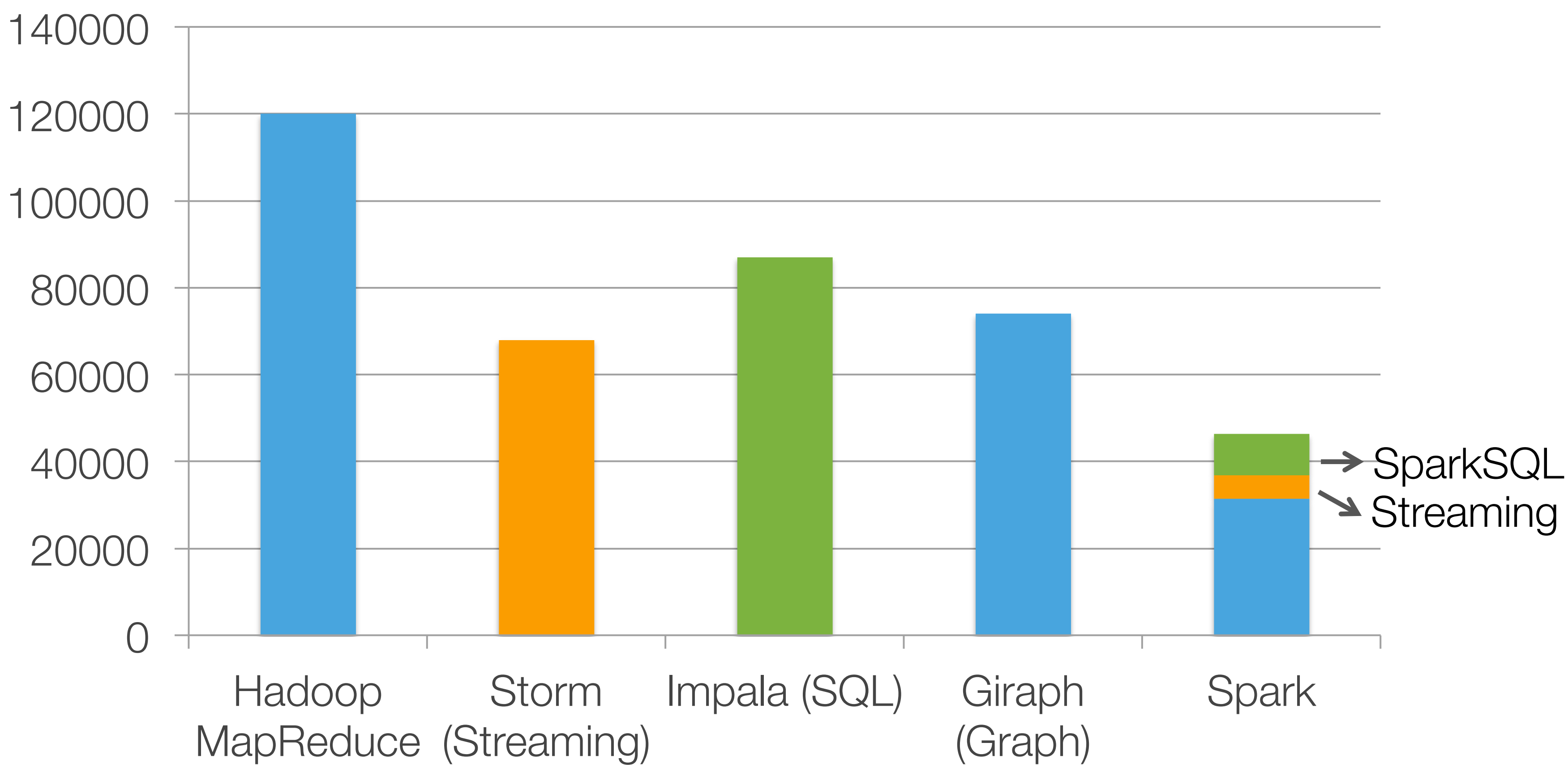
Powerful Stack – Agile Development



non-test, non-example source lines

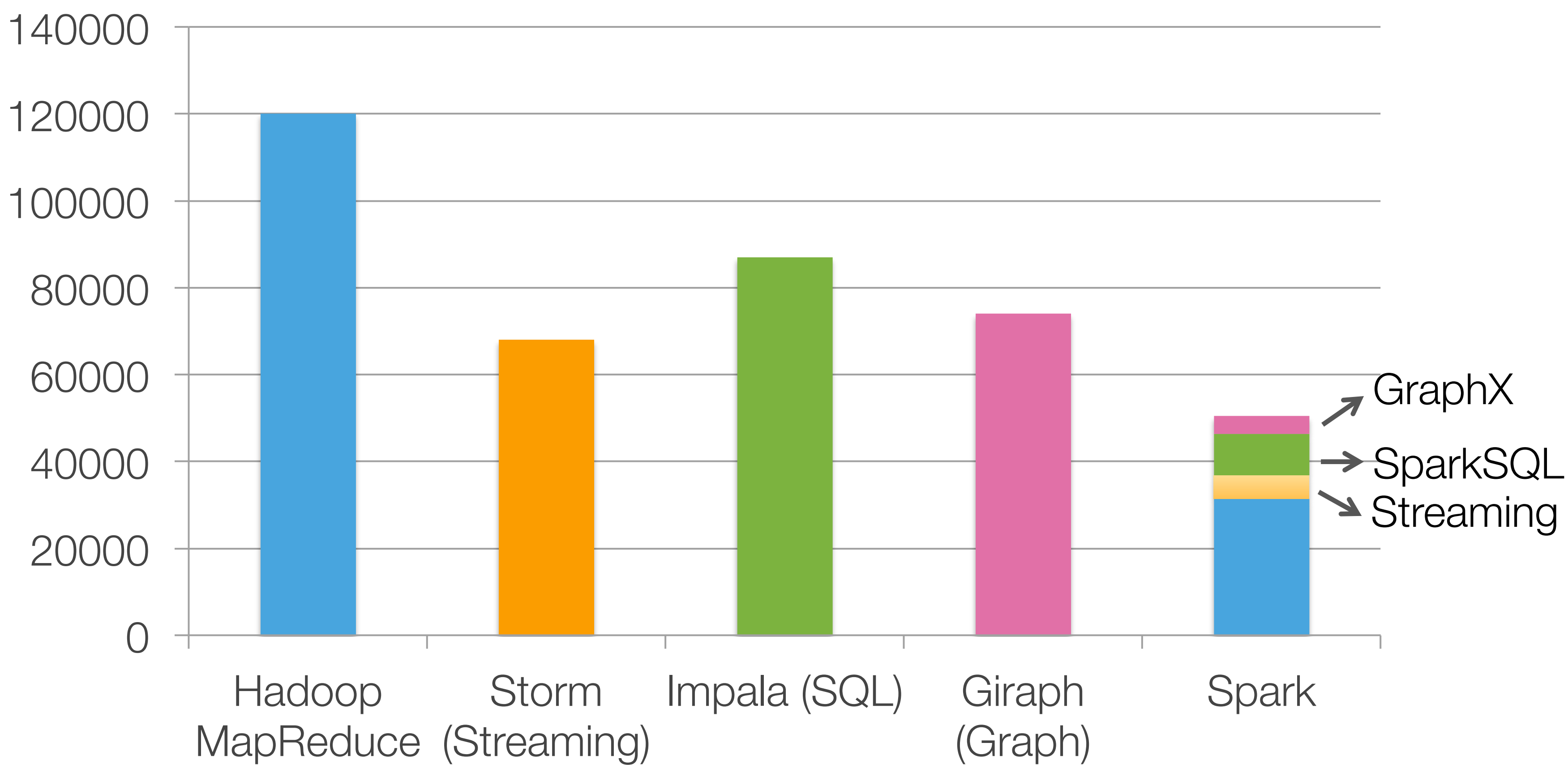
amplab UC BERKELEY databricks™

Powerful Stack – Agile Development



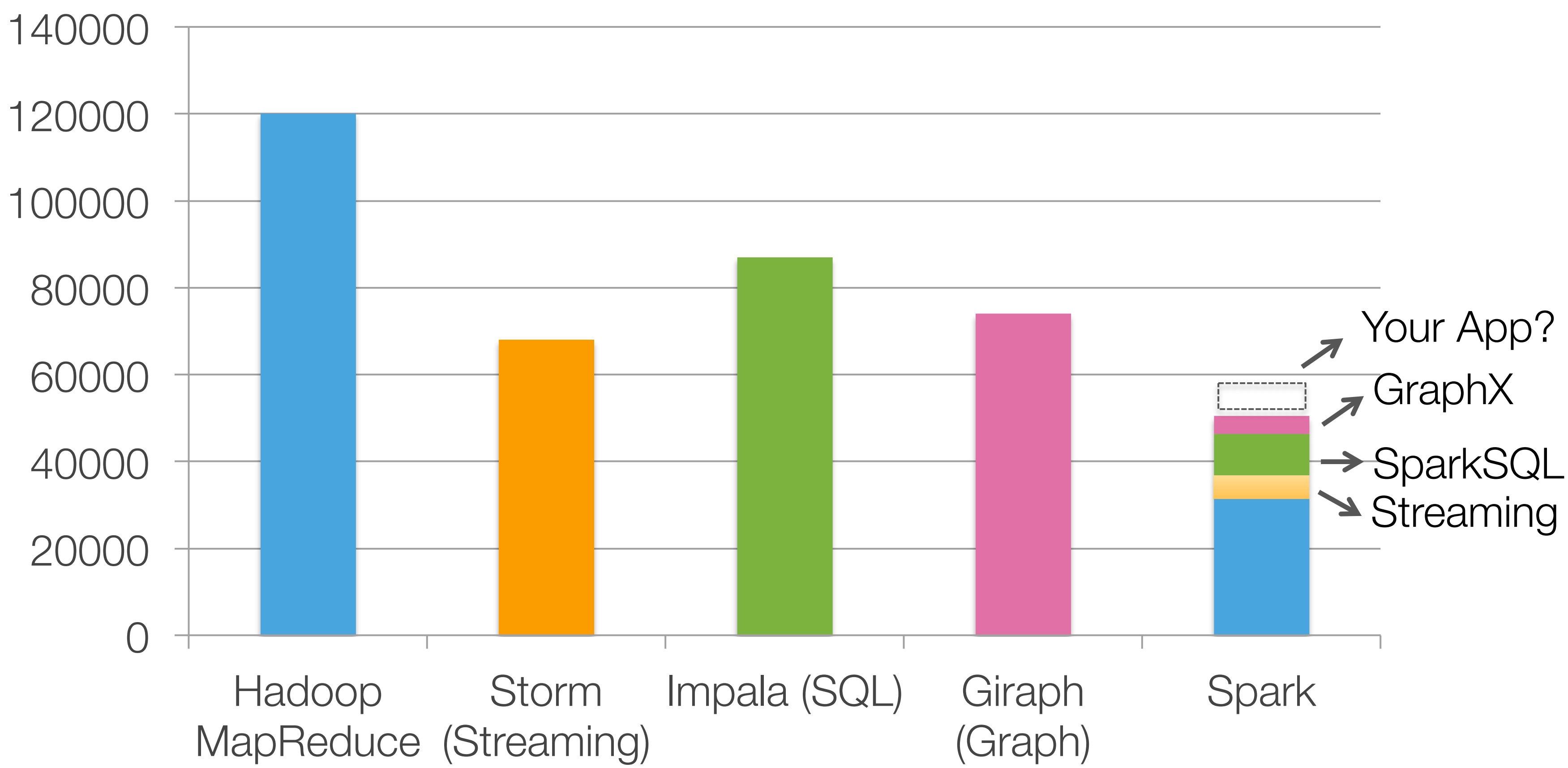
non-test, non-example source lines

Powerful Stack – Agile Development



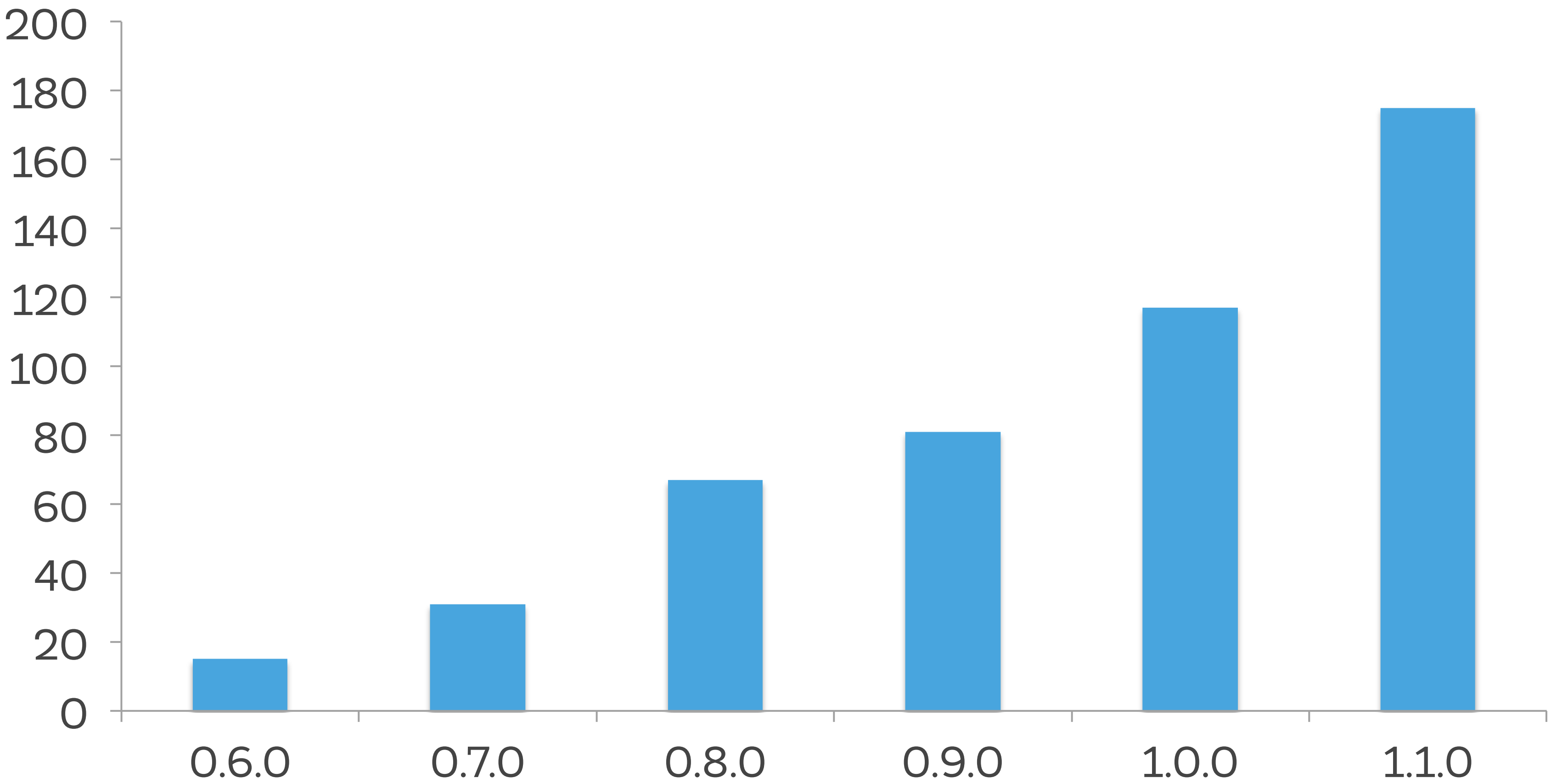
non-test, non-example source lines

Powerful Stack – Agile Development



non-test, non-example source lines

Community Growth



Not Just for In-Memory Data

WIRED

Startup Crunches 100 Terabytes of Data in a Record 23 Minutes

	Hadoop Record	Spark 100TB	Spark 1PB
Data Size	102.5TB	100TB	1000TB
Time	72 min	23 min	234 min
# Cores	50400	6592	6080
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Environment	Dedicate	Cloud (EC2)	Cloud (EC2)

amplab UC BERKELEY databricks™

Spark SQL Overview

- Newest component of Spark initially contributed by databricks (< 1 year old)
- Tightly integrated way to work with structured data (tables with rows/columns)
- Transform RDDs using SQL
- Data source integration: Hive, Parquet, JSON, and more

Relationship to SHARK

Shark modified the Hive backend to run over Spark, but had two challenges:

> Limited integration with Spark programs
> Hive optimizer not designed for Spark

Spark SQL reuses the best parts of Shark:

Borrows

- Hive data loading
- In-memory column store

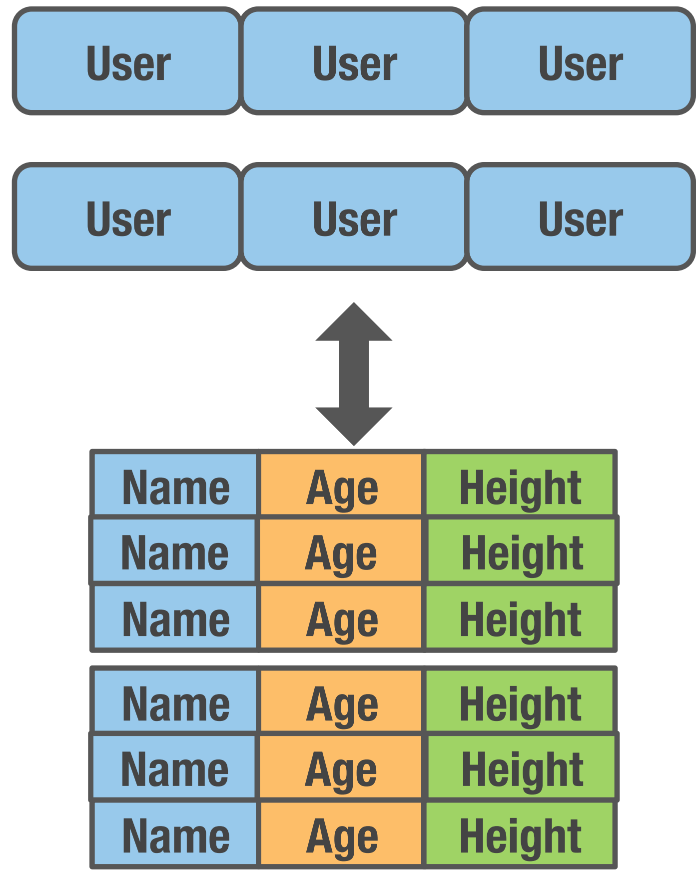
Adds

- RDD-aware optimizer
- Rich language interfaces

Adding Schema to RDDs

Spark + RDDs

Functional transformations on partitioned collections of *opaque objects*.



SQL + SchemaRDDs

Declarative transformations on partitioned collections of *tuples*.

SchemaRDDs: More than SQL

Unified interface for structured data

Image credit: http://barrymieny.deviantart.com/

Getting Started: Spark SQL

`SQLContext/HiveContext`

- Entry point for all SQL functionality
- Wraps/extends existing spark context

```
from pyspark.sql import SQLContext
sqlCtx = SQLContext(sc)
```

Example Dataset

A text file filled with people's names and ages:

```
Michael, 30
Andy, 31
…
```

RDDs as Relations (Python)

```
# Load a text file and convert each line to a dictionary.
lines = sc.textFile("examples/…/people.txt")

parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))

# Infer the schema, and register the SchemaRDD as a table
peopleTable = sqlCtx.inferSchema(people)
peopleTable.registerAsTable("people")
```

amplab UC BERKELEY databricks™

RDDs as Relations (Scala)

```scala
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
import sqlContext._

// Define the schema using a case class.
case class Person(name: String, age: Int)
// Create an RDD of Person objects and register it as a table.
val people =
  sc.textFile("examples/src/main/resources/people.txt")
    .map(_.split(","))
    .map(p => Person(p(0), p(1).trim.toInt))

people.registerAsTable("people")
```

RDDs as Relations (Java)

```
public class Person implements Serializable {
  private String _name;
  private int _age;
  public String getName() { return _name;  }
  public void setName(String name) { _name = name; }
  public int getAge() { return _age; }
  public void setAge(int age) { _age = age; }
}

JavaSQLContext ctx = new org.apache.spark.sql.api.java.JavaSQLContext(sc)
JavaRDD<Person> people = ctx.textFile("examples/src/main/resources/
people.txt").map(
  new Function<String, Person>() {
    public Person call(String line) throws Exception {
      String[] parts = line.split(",");
      Person person = new Person();
      person.setName(parts[0]);
      person.setAge(Integer.parseInt(parts[1].trim()));
      return person;
    }
  });
JavaSchemaRDD schemaPeople = sqlCtx.applySchema(people, Person.class);
```

Querying Using SQL

```python
# SQL can be run over SchemaRDDs that have been registered
# as a table.
teenagers = sqlCtx.sql("""
  SELECT name FROM people WHERE age >= 13 AND age <= 19""")

# The results of SQL queries are RDDs and support all the normal
# RDD operations.
teenNames = teenagers.map(lambda p: "Name: " + p.name)
```

Existing Tools, New Data Sources

Spark SQL includes a server that exposes its data using JDBC/ODBC

- Query data from HDFS/S3,
- Including formats like Hive/Parquet/JSON*
- Support for caching data in-memory

* Coming in Spark 1.2

Caching Tables In-Memory

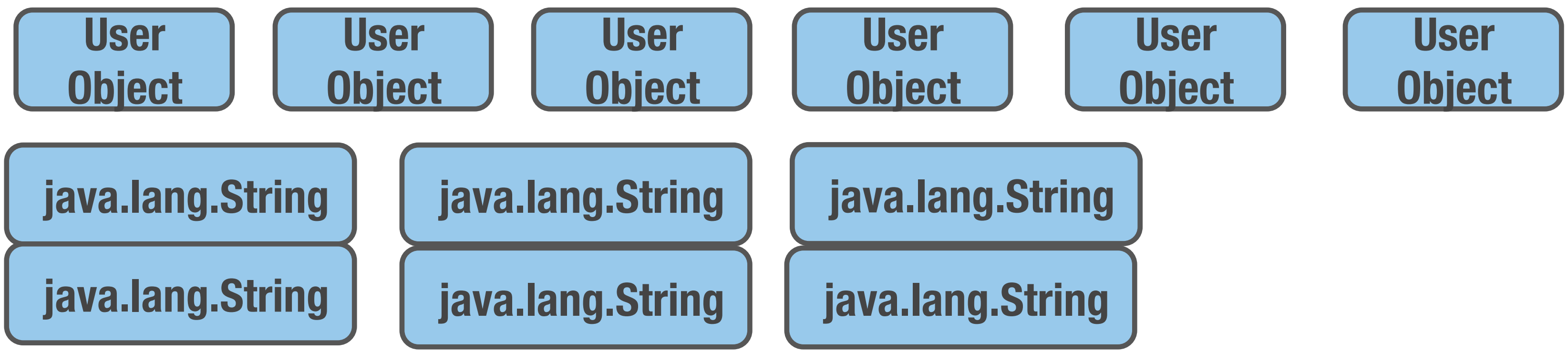
Spark SQL can cache tables using an in-memory columnar format:

- Scan only required columns
- Fewer allocated objects (less GC)
- Automatically selects best compression

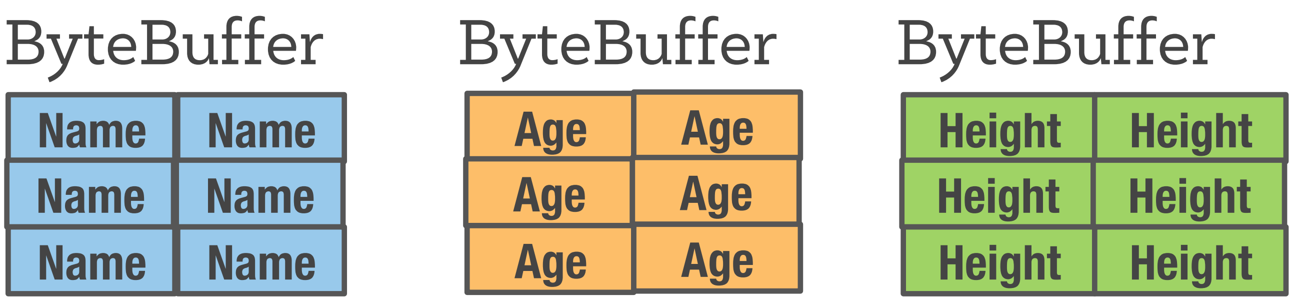
```
cacheTable("people")

schemaRDD.cache() – *Requires Spark 1.2
```

Caching Comparison

Spark MEMORY_ONLY Caching



SchemaRDD Columnar Caching



Language Integrated UDFs

```
registerFunction(“countMatches”,
  lambda (pattern, text):
    re.subn(pattern, '', text)[1])

sql("SELECT countMatches(‘a’, text)…")
```

SQL and Machine Learning

```
training_data_table = sql("""
  SELECT e.action, u.age, u.latitude, u.logitude
    FROM Users  u
    JOIN Events e ON u.userId = e.userId""")

def featurize(u):
   LabeledPoint(u.action, [u.age, u.latitude, u.longitude])


// SQL results are RDDs so can be used directly in Mllib.
training_data = training_data_table.map(featurize)
model = new LogisticRegressionWithSGD.train(training_data)
```

Machine Learning Pipelines

```
// training:{eventId:String, features:Vector, label:Int}
val training = parquetFile("/path/to/training")
val lr = new LogisticRegression().fit(training)

// event: {eventId: String, features: Vector}
val event = parquetFile("/path/to/event")
val prediction =
  lr.transform(event).select('eventId, 'prediction)

prediction.saveAsParquetFile("/path/to/prediction”)
```

amplab UC BERKELEY databricks™

Reading Data Stored in Hive

```
from pyspark.sql import HiveContext
hiveCtx = HiveContext(sc)

hiveCtx.hql("""
  CREATE TABLE IF NOT EXISTS src (key INT, value STRING)""")

hiveCtx.hql("""
  LOAD DATA LOCAL INPATH 'examples/…/kv1.txt' INTO TABLE src""")

# Queries can be expressed in HiveQL.
results = hiveCtx.hql("FROM src SELECT key, value").collect()
```

Parquet Compatibility

Native support for reading data in Parquet:

- Columnar storage avoids reading unneeded data.
- RDDs can be written to parquet files, preserving the schema.
- Convert other slower formats into Parquet for repeated querying

Using Parquet

```
# SchemaRDDs can be saved as Parquet files, maintaining the
# schema information.
peopleTable.saveAsParquetFile("people.parquet")

# Read in the Parquet file created above.  Parquet files are
# self-describing so the schema is preserved. The result of
# loading a parquet file is also a SchemaRDD.
parquetFile = sqlCtx.parquetFile("people.parquet”)

# Parquet files can be registered as tables used in SQL.
parquetFile.registerAsTable("parquetFile”)
teenagers = sqlCtx.sql("""
  SELECT name FROM parquetFile WHERE age >= 13 AND age <= 19""")
```

{JSON} Support

- Use **`jsonFile`** or **`jsonRDD`** to convert a collection of JSON objects into a SchemaRDD
- Infers and unions the schema of each record
- Maintains nested structures and arrays

{JSON} Example

```
# Create a SchemaRDD from the file(s) pointed to by path
people = sqlContext.jsonFile(path)

# Visualized inferred schema with printSchema().
people.printSchema()
# root
#  |-- age: integer
#  |-- name: string

# Register this SchemaRDD as a table.
people.registerTempTable("people")
```

Data Sources API

Allow easy integration with new sources of structured data:

```
CREATE TEMPORARY TABLE episodes
USING com.databricks.spark.avro
OPTIONS (
  path ”./episodes.avro”
)
```

https://github.com/databricks/spark-avro

amplab UC BERKELEY databricks™

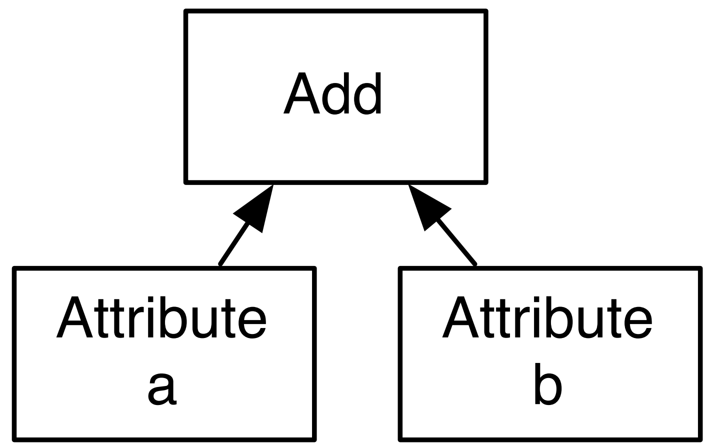
Efficient Expression Evaluation

Interpreting expressions (e.g., 'a + b') can very expensive on the JVM:

- Virtual function calls
- Branches based on expression type
- Object creation due to primitive boxing
- Memory consumption by boxed primitive objects

Interpreting "a+b"

1. Virtual call to Add.eval()
2. Virtual call to a.eval()
3. Return boxed Int
4. Virtual call to b.eval()
5. Return boxed Int
6. Integer addition
7. Return boxed result



Using Runtime Reflection

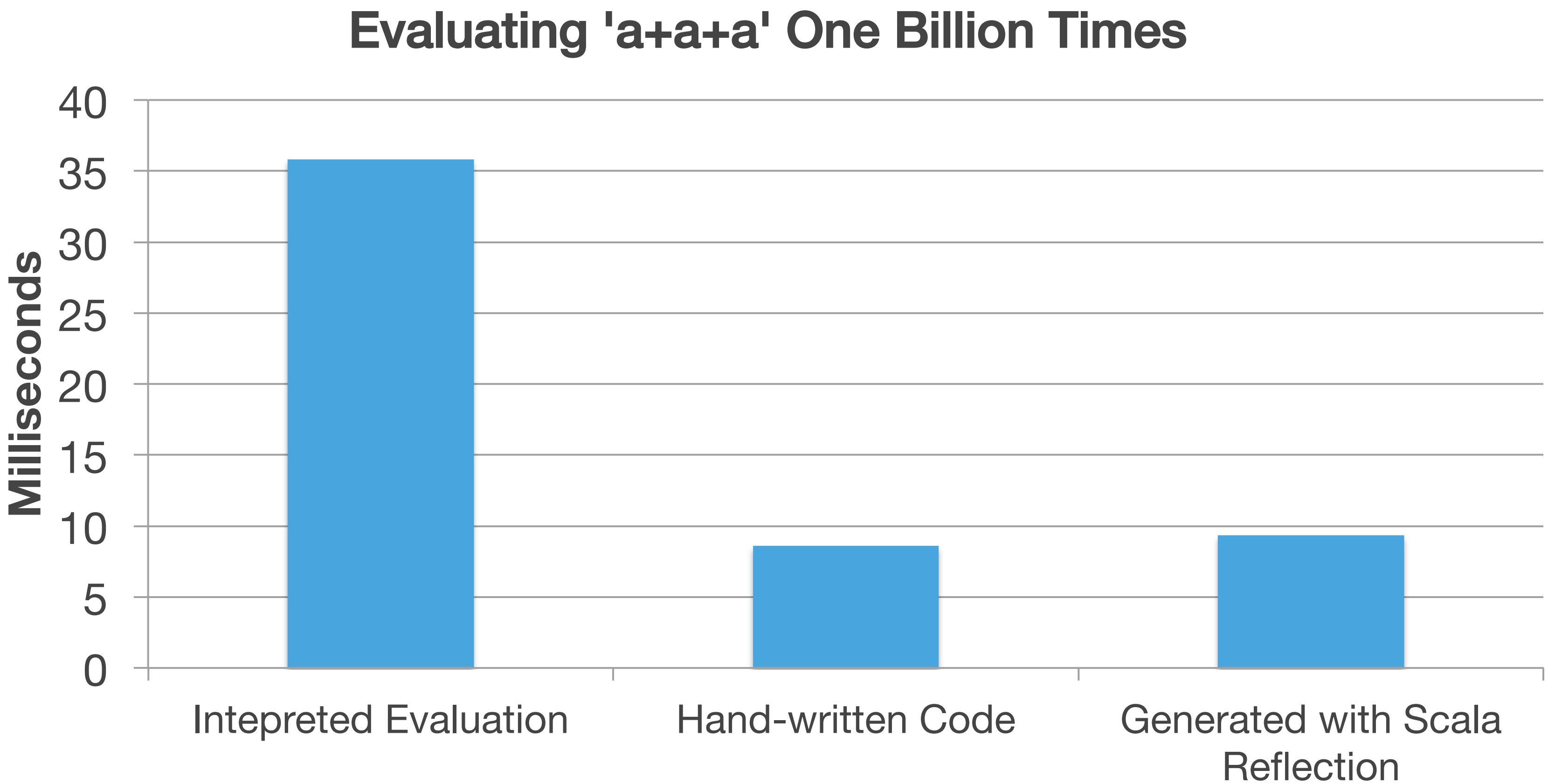
```
def generateCode(e: Expression): Tree = e match {
  case Attribute(ordinal) =>
    q"inputRow.getInt($ordinal)"
  case Add(left, right) =>
    q"""
      {
        val leftResult = ${generateCode(left)}
        val rightResult = ${generateCode(right)}
        leftResult + rightResult
      }
     """
}
```

Code Generating "a + b"

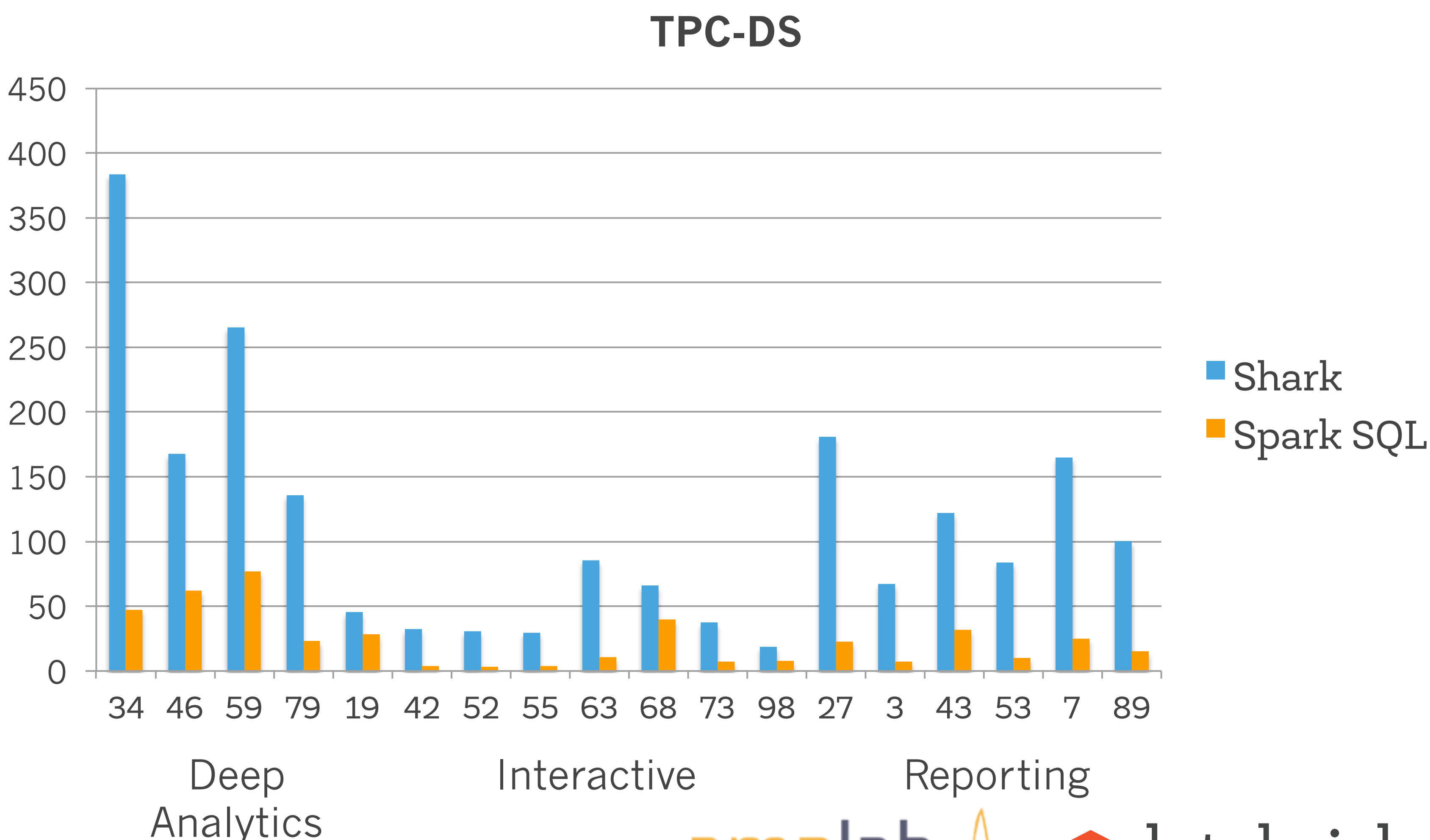
```
val left: Int = inputRow.getInt(0)
val right: Int = inputRow.getInt(1)
val result: Int = left + right
resultRow.setInt(0, result)
```

- Fewer function calls
- No boxing of primitives

Performance Comparison



Performance vs. SHARK



amplab UC BERKELEY databricks™

What's Coming in Spark 1.2?

- MLlib pipeline support for SchemaRDDs
- New APIs for developers to add external data sources
- Full support for Decimal and Date types.
- Statistics for in-memory data
- Lots of bug fixes and improvements to Hive compatibility

Questions?

databricks™

ampcamp

big data bootcamp