ORACLE®

JavaOne™

ORACLE®

CON3525: Java, Scala and Friends

Touring the Java Bedrock of Big Data

Dan McClary
Senior Principal Product Manager
Big Data and Hadoop, Oracle

So, Who Is This Talk For?

Curious about Hadoop

- Have used Hadoop, or want to
- Would like to know more about how the system functions

Hadoop-Curious

Curious about Internals

- Not particularly focused on Big Data
- Curious about what Big Data internals might be relevant to your projects

Internal-Curious

Program Agenda

1. A Quick Primer on Hadoop History
2. Core Hadoop and Java
3. Declarative Big Data Builds on Java
4. Scala and Spark Speed Things Up

Quick Refresher: Hadoop History

- Hadoop is
  - A framework for distributed processing of large data sets across clusters of computers using simple programming models.
  - Designed to scale up from single servers to thousands of machines, each offering local computation and storage.
- Created by Cutting and Cafarella @ Yahoo! in 2005
  - Part of the infrastructure for the Nutch search engine project
- 10 years later: synonymous with "Big Data"

“Core Hadoop” Components

Common Libraries

Hadoop Common

Programming Model

MapReduce

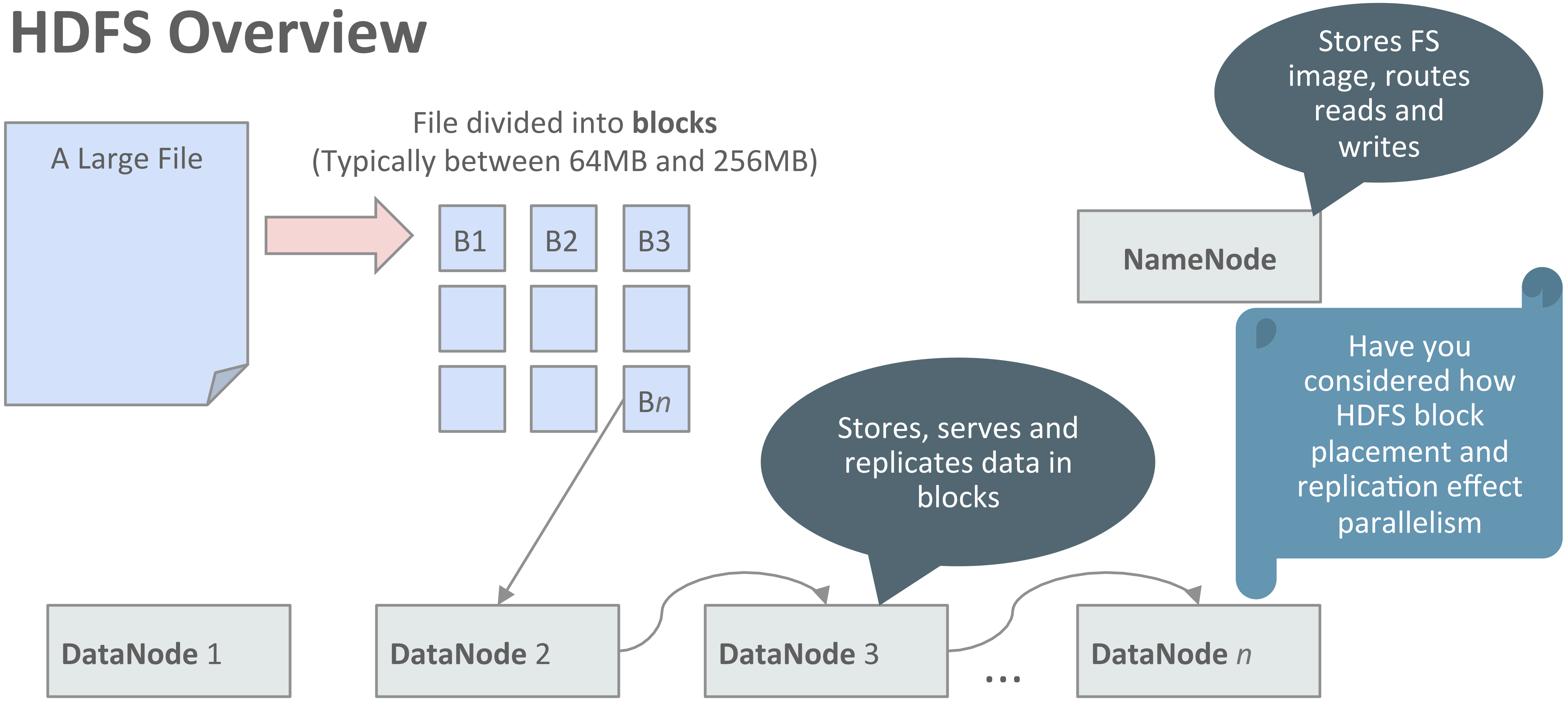
Resource Management

YARN (Yet Another Resource Negotiator)

Distributed Storage

HDFS (Hadoop Distributed File System)

HDFS Overview



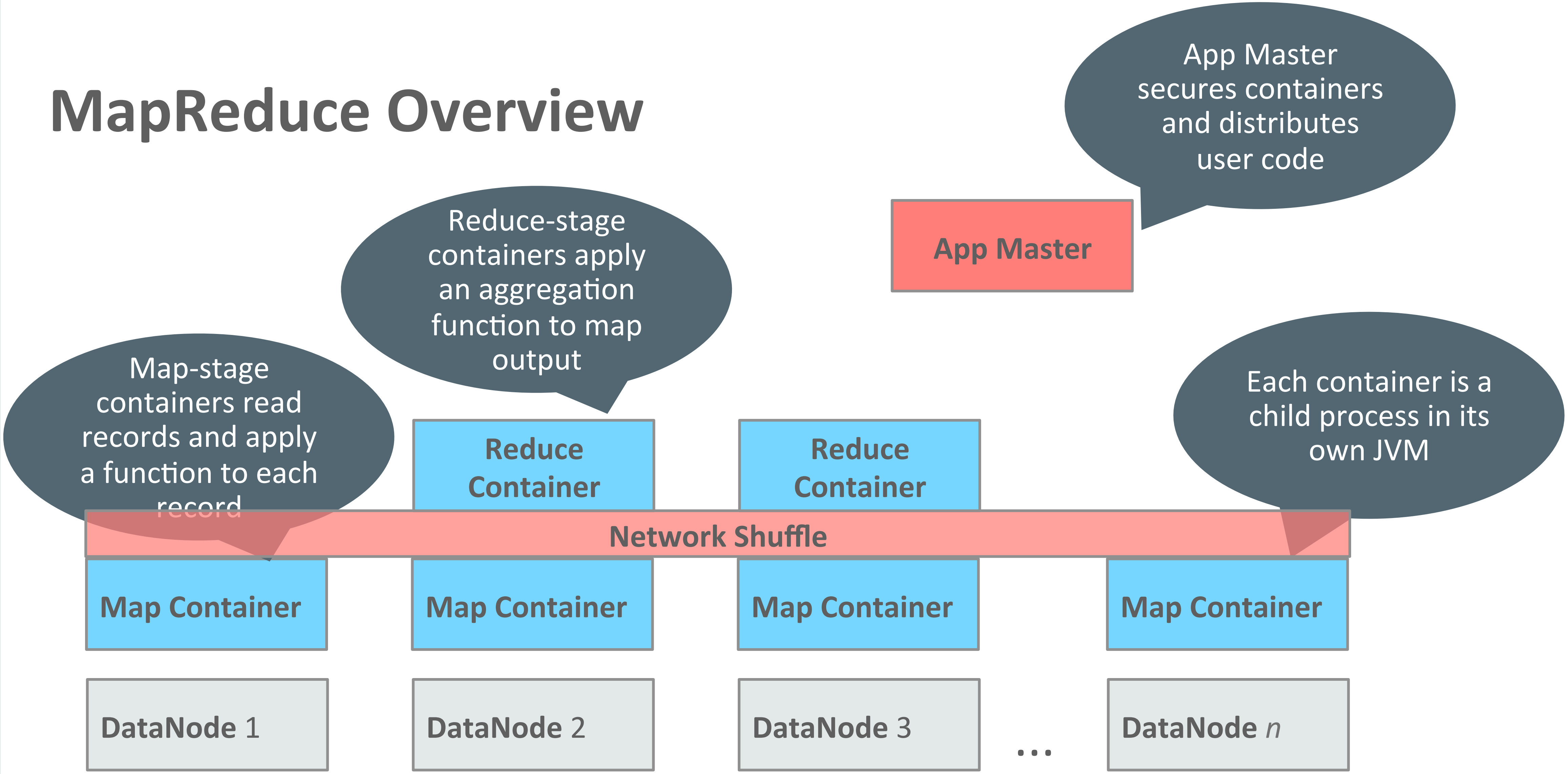
HDFS Overview

- For Java Programmers
  - **Not** a child of nio.file.FileSystem
  - Expanded approach to a FileSystem class
- Subclasses for
  - S3
  - FTP
  - Local
  - And more

Can you think of ways to leverage AbstractFilesystem subclasses to access more data?

- For System Users
  - Presents as a Unix filesystem
  - Standard operations
    - ls
    - mkdir
    - chmod/chown
  - Repair functions
    - fsck
    - Safe Mode: when block replication inconsistency is too high

MapReduce Overview



How Do We Coordinate All These Nodes?

- **Dynamic Proxies!**
- Define a single interface between server and client
  - `@ProtocolInfo(protocolName = "org.apache.hadoop.mapreduce.v2.api.MRC` `protocolVersion = 1)`
  - `@ProtocolInfo(protocolName = "org.apache.hadoop.hdfs.server.protocol` `protocolVersion = 1)`
  - Server sets up with `RPC.Builder(setInstance(new ProtocolKlass()))`
  - Client runs `RPC.getProxy(ProtocolKlass.class,…)`
- Simple RPC interface which maintains backward wire-compatibility
- Core to: HDFS, Common, YARN, MapReduce Client, etc.

Dynamic Proxies provide a straightforward way to define stable RPC interfaces over time.

What About User Code?

```
public void map(Object key,
      Text value,
       Context context) throws IOException,
              InterruptedException
{
      StringTokenizer itr = new
StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
      }
    }
```

```
public void reduce(
      Text key,
      Iterable<IntWritable> values,
       Context context) throws IOException,
      InterruptedException
{
      int sum = 0;
      for (IntWritable val : values) {
       sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
   }
```

What's Up With Those Types?

- Hadoop introduces new types for serialization
- **Writables**
  - Text, IntWritable, FloatWritable, etc.
- Why? We've had serialization for a long time.
- Answer: Java serialized objects store type
- The other nodes in the job know the type
  - We just need to serialize to transfer the data
- Writables are more compact
  - .get() and .set(val) methods allow us to minimize creation overhead

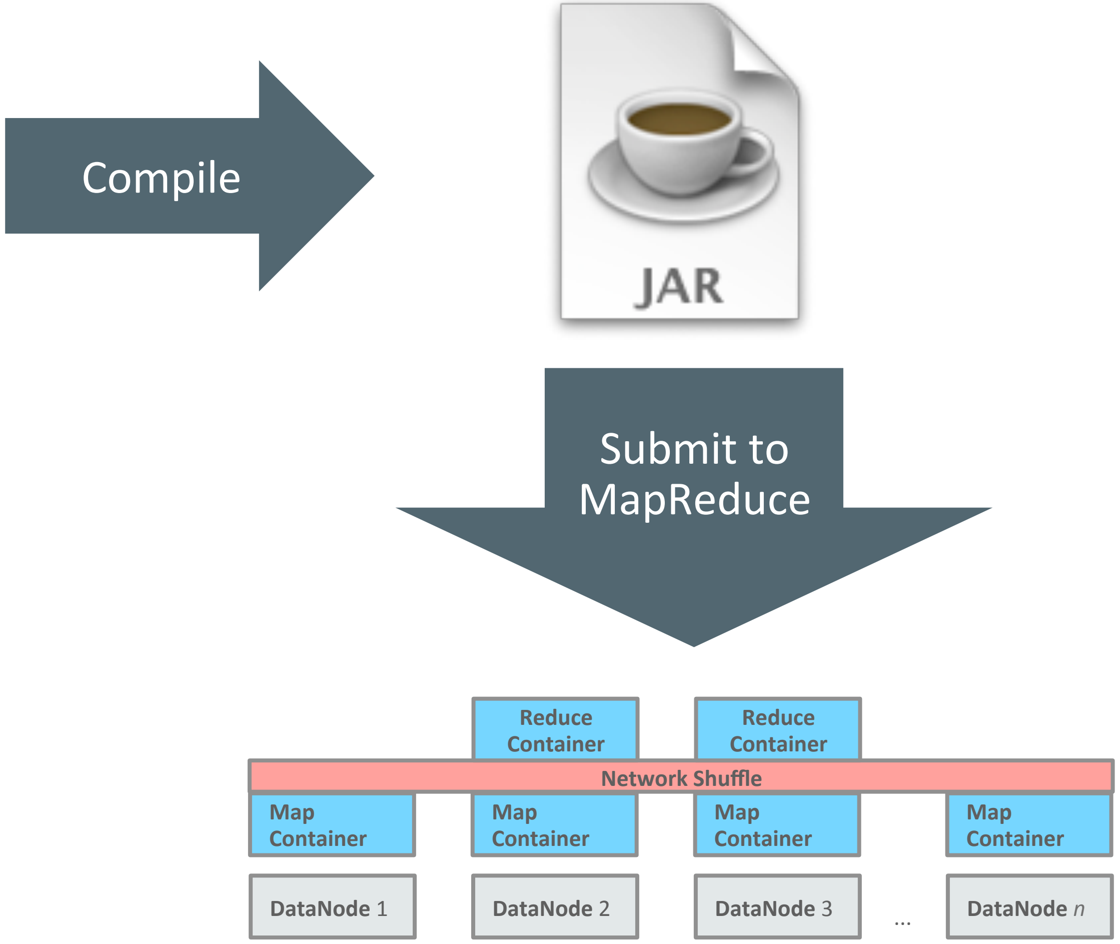
Are you paying attention to your Writeables?

The Demand for More Languages

- Not everyone is comfortable writing distributed Java code
  - For joining data
  - For multiple passes on intermediate data
- Data Analysts and Engineers demand declarative languages
- Enter Pig and Hive
  - Apache Pig (2007), developed as a dataflow language inside Yahoo!
  - Apache Hive (2009), developed as SQL-on-MapReduce at Facebook

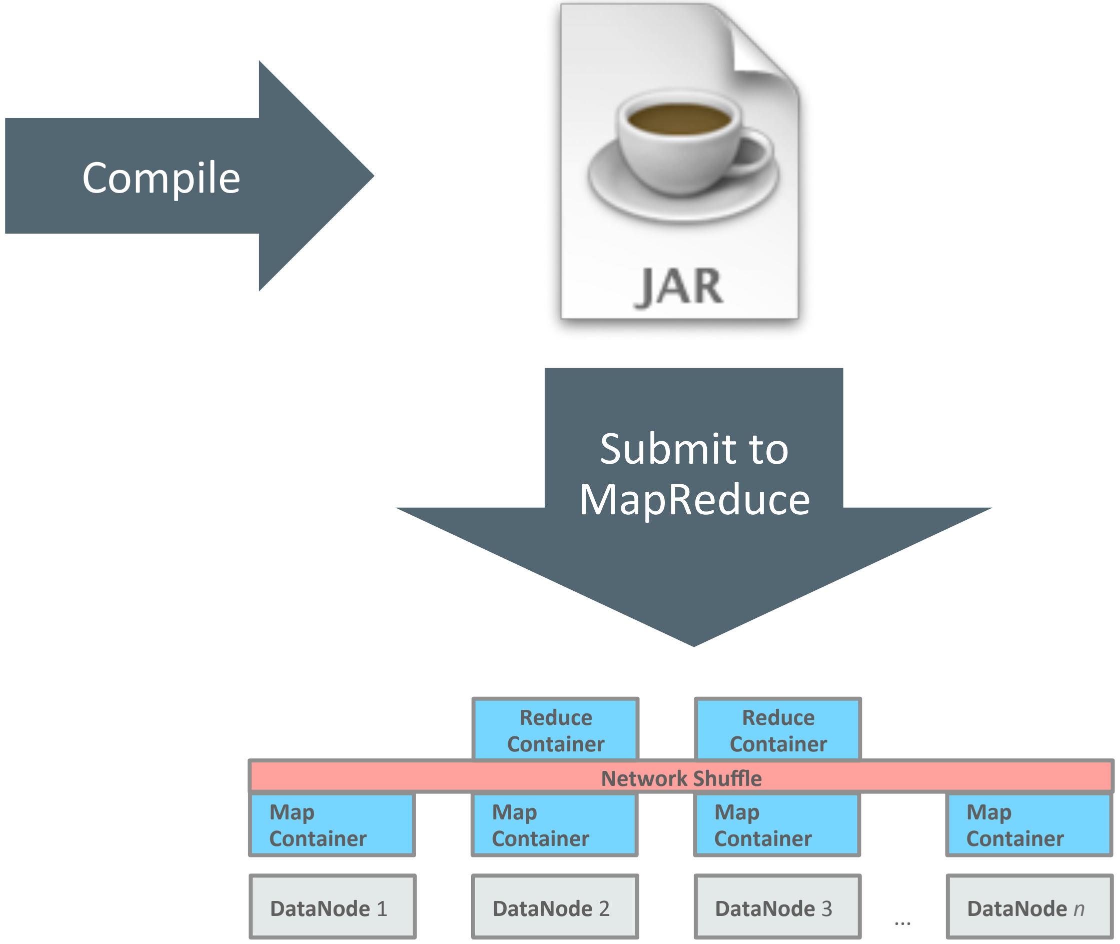
Pig Code and Deployment

```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet'
  AS (line:chararray);

words = FOREACH input_lines GENERATE
  FLATTEN(TOKENIZE(line)) AS word;

filtered_words = FILTER words BY word MATCHES '\\w+';

word_groups = GROUP filtered_words BY word;

word_count = FOREACH word_groups GENERATE
  COUNT(filtered_words) AS count, group AS word;

ordered_word_count = ORDER word_count BY count DESC;
 STORE ordered_word_count
  INTO '/tmp/number-of-words-on-internet';
```



Hive Follows Suit, But With SQL

```sql
SELECT word, COUNT(*)
  FROM doc
  LATERAL VIEW
    explode(split(text, ' ')) myTable
    as word
  GROUP BY word;
```



Hive is Just a SQL-Layer, What's Java Got To Do With It?

- Java types for columns!
  - Maps
  - Arrays
  - Structs (named maps of variable type)
- Built on the back of DataNucleus
  - Hive has to put metadata somewhere
    - Usually MySQL, but sometimes Derby
  - Metadata operations use DataNucleus to communicate with backing store

Is That Good Enough?

Map #1:
Read records from *sales* and *store_locations*
Emit all records as (**join_key**, **record**)

Reduce #1:
Read (**join_key**, **record**) pairs
If $\text{join\_key}_A == \text{join\_key}_B$, emit $\text{record}_A+\text{record}_B$

Checkpoint

Forced checkpointing:
Must write and re-read

Good for fault-tolerance
Bad for performance

Map #2:
Read joined records
Emit all records as (**location**, **sales**)

Reduce #2:
Sum all (location, sales) pairs
Emit (location, sum(sales))

JavaOne
ORACLE

Apache Spark Takes Big Data to the Next Level

Spark

Lightning Fast Cluster Computing

Scala

What Does Spark Offer?

- More generic DAGs
  – Avoid the checkpointing overhead
- In-memory storage for iterative operations
  – Radical speed-up for iterative operations
- Richer abstractions and functions
  – Inherit collection behaviors from Scala
  – Program in Java, Scala, or Python
- REPLs for interactive use
  – Scala, Python

Up to **10×** faster on disk, **100×** in memory

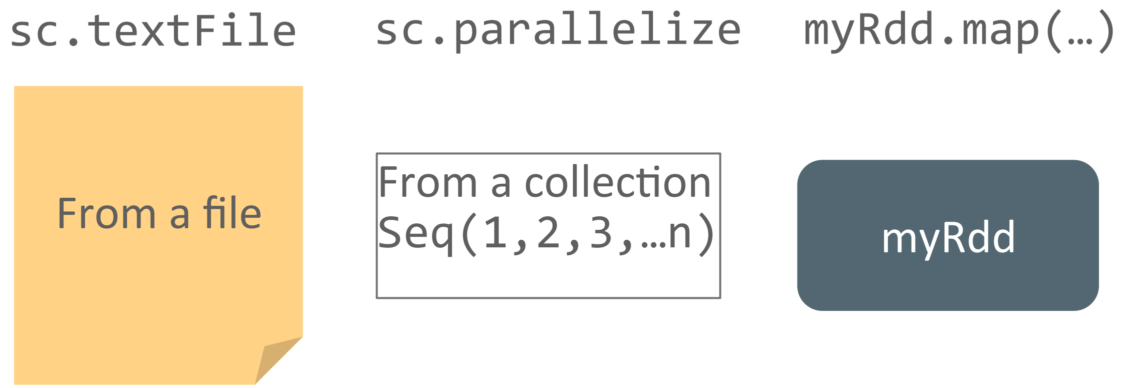
2-5× less code

JavaOne ORACLE

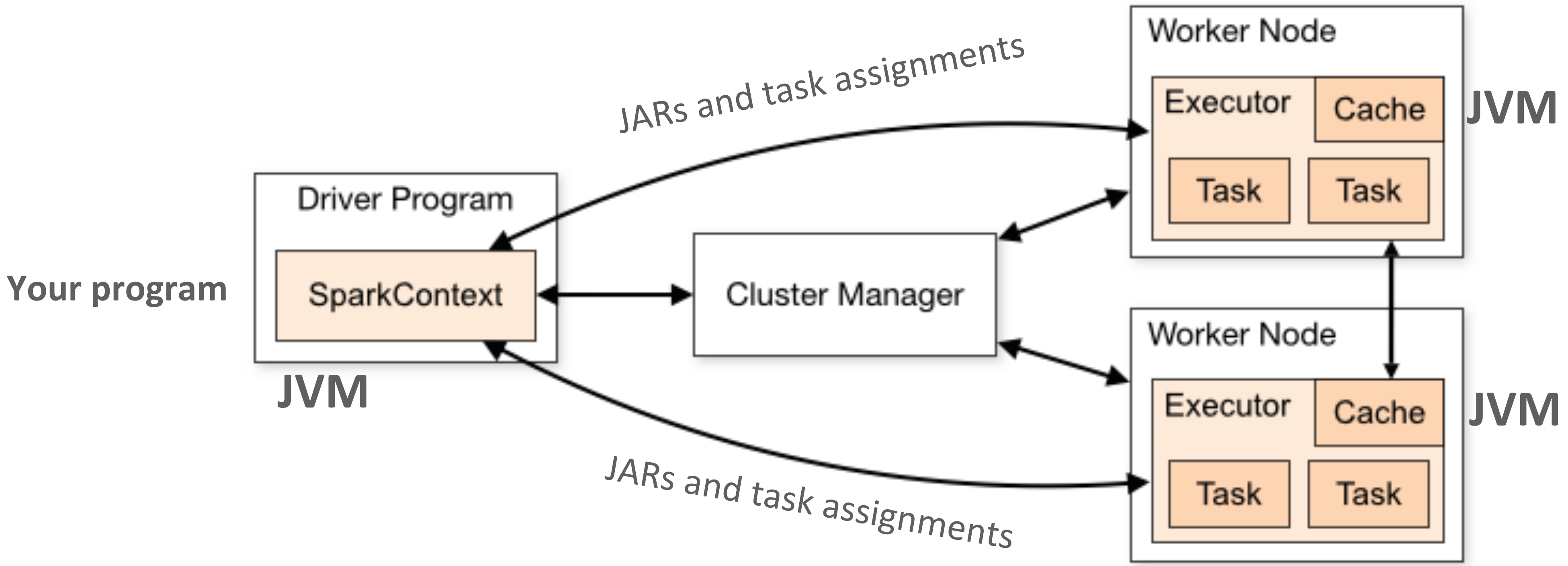
Core Abstraction: Resilient Distributed Datasets (RDDs)

- RDDs are **immutable**, distributed collections
  – Doesn't need to exist in physical storage
- Handle to an RDD ➔ How to compute from known-good storage
- Lazy and ephemeral
  – Partitions are materialized on-demand

How to Make an RDD



Spark Cluster Architecture



Scala Makes RDDs Expressive

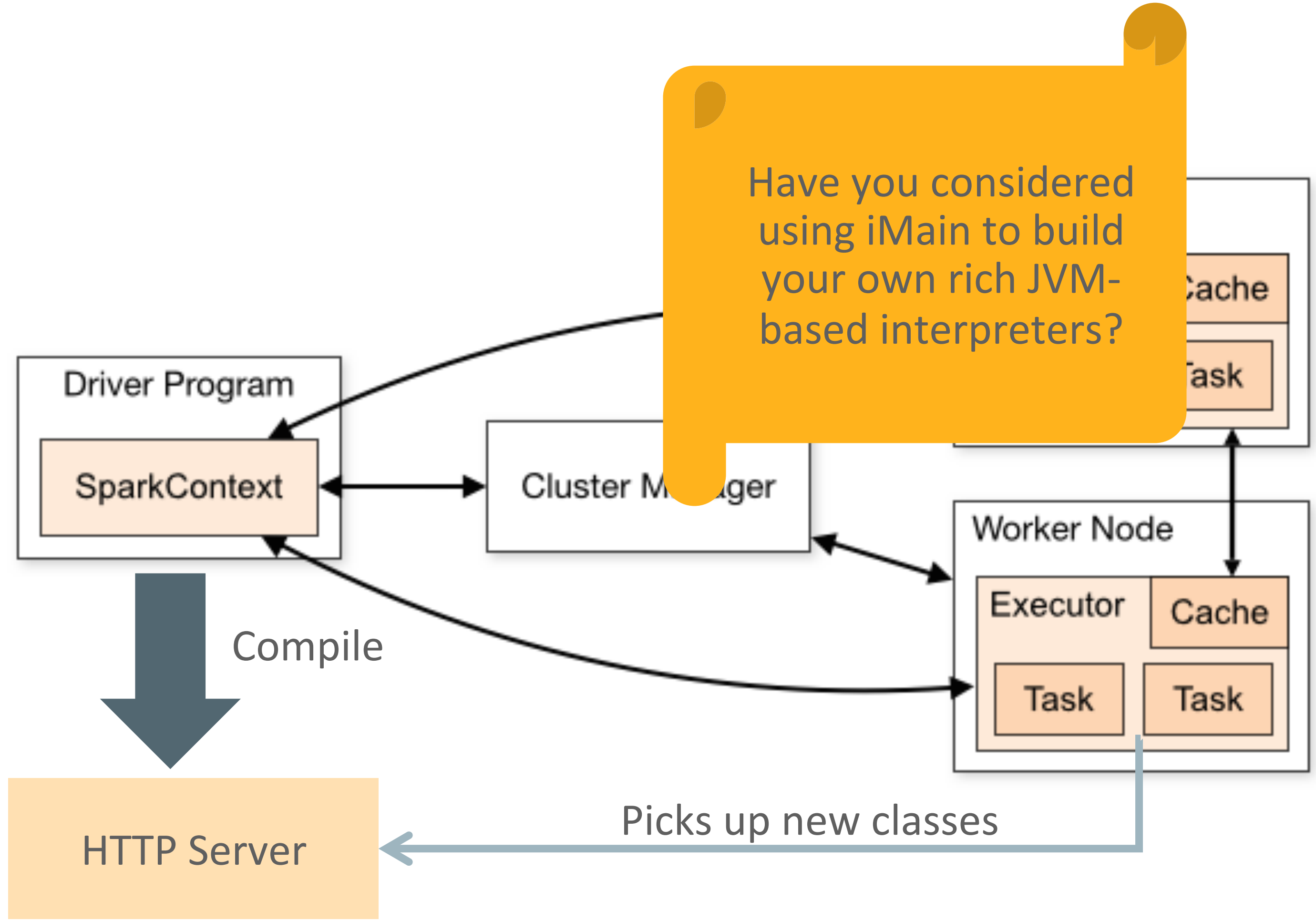
- RDDs support most of the operations of Scala collections
  - Specifically Traversable and Iterable
- Map, Reduce, and…
  - flatMap, filter, groupBy
  - count, max, min
  - fold, reduce, take, collect
- **Why?**

1. Scala closures are serializable!
2. We can efficiently send closures to a distributed set of workers

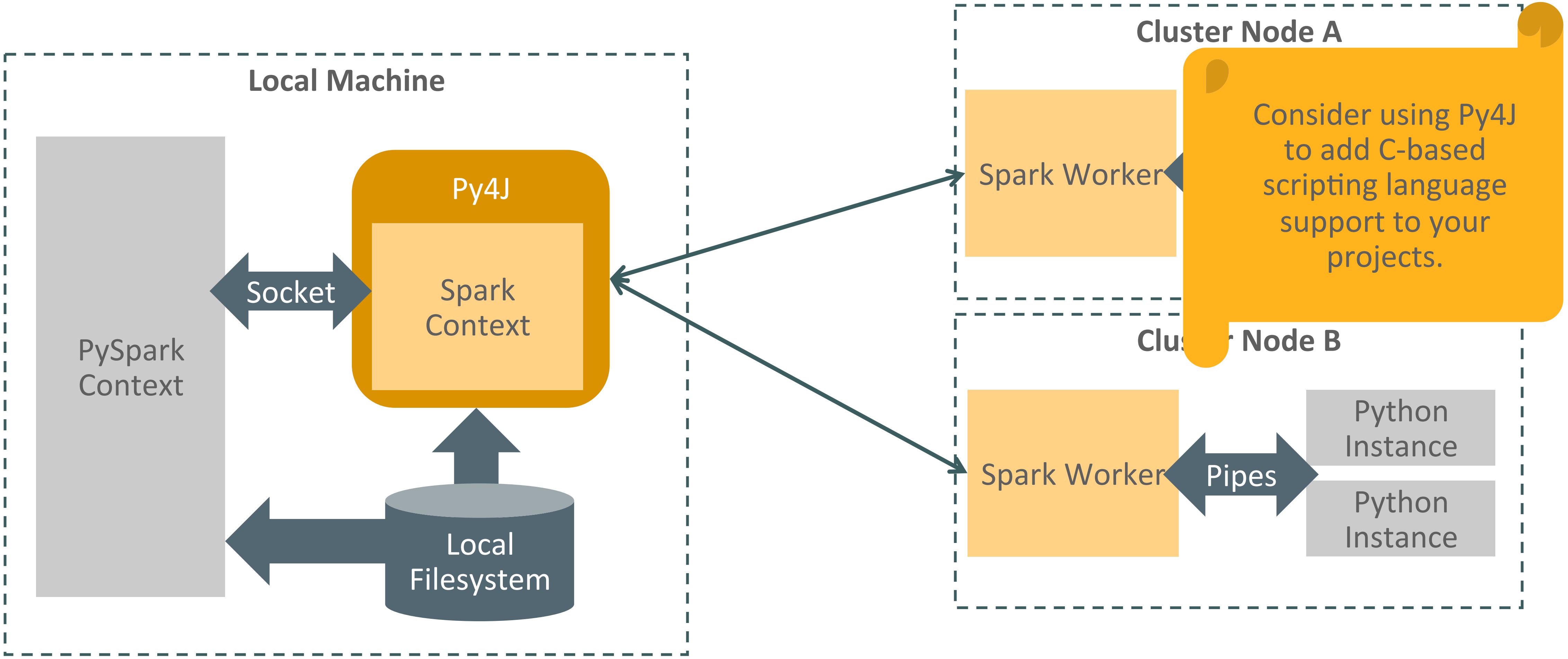
How many fewer lines of code would it take to rewrite your MapReduce jobs in Spark?

How Does it Become Interactive?

- Have you ever looked at scala.tools.nsc.interpreter?
- The **IMain** class →
  - The Scala interpreter
- Provides compile methods!
  - Compile code on the fly
  - Generate a class file
- Compile REPL definitions
  - Let executors pick them up



But What About Python?



Where does it go from here?

- Right now
  - Streaming With Spark
  - Massively Scalable Message Transport with Kafka
  - Interactive Data Science with Zeppelin
- On the horizon
  - Twill: Simpler distributed systems
  - Kylin: OLAP for Big Data
  - DeepLearning4J: Scalable Deep Learning on GPUs or Spark

Integrated Cloud

Applications & Platform Services

JavaOne™

ORACLE®