Big Data and Internet Thinking

Chentao Wu
Associate Professor
Dept. of Computer Science and Engineering
wuct@cs.sjtu.edu.cn

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Download lectures

- ftp://public.sjtu.edu.cn
- User: wuct
- Password: wuct123456

- http://www.cs.sjtu.edu.cn/~wuct/bdit/

Schedule

- lec1: Introduction on big data, cloud computing & IoT
- lec2: Parallel processing framework (e.g., MapReduce)
- lec3: Advanced parallel processing techniques (e.g., YARN, Spark)
- lec4: Cloud & Fog/Edge Computing
- lec5: Data reliability & data consistency
- lec6: Distributed file system & objected-based storage
- lec7: Metadata management & NoSQL Database
- lec8: Big Data Analytics

Collaborators

Google
谷歌

阿里云
aliyun.com

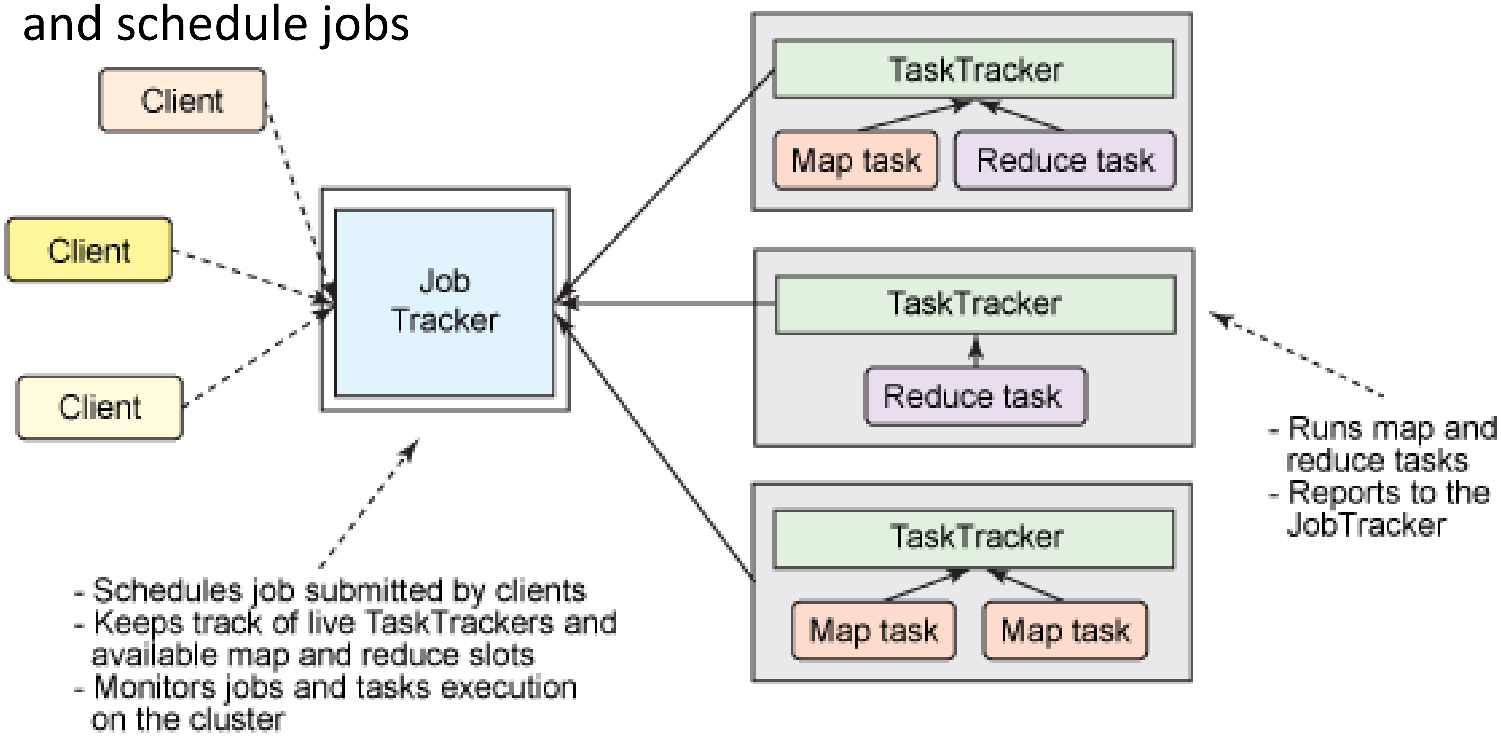
DELL EMC

Contents

1 **Introduction to Map-Reduce 2.0**

Classic Map-Reduce Task (MRv1)

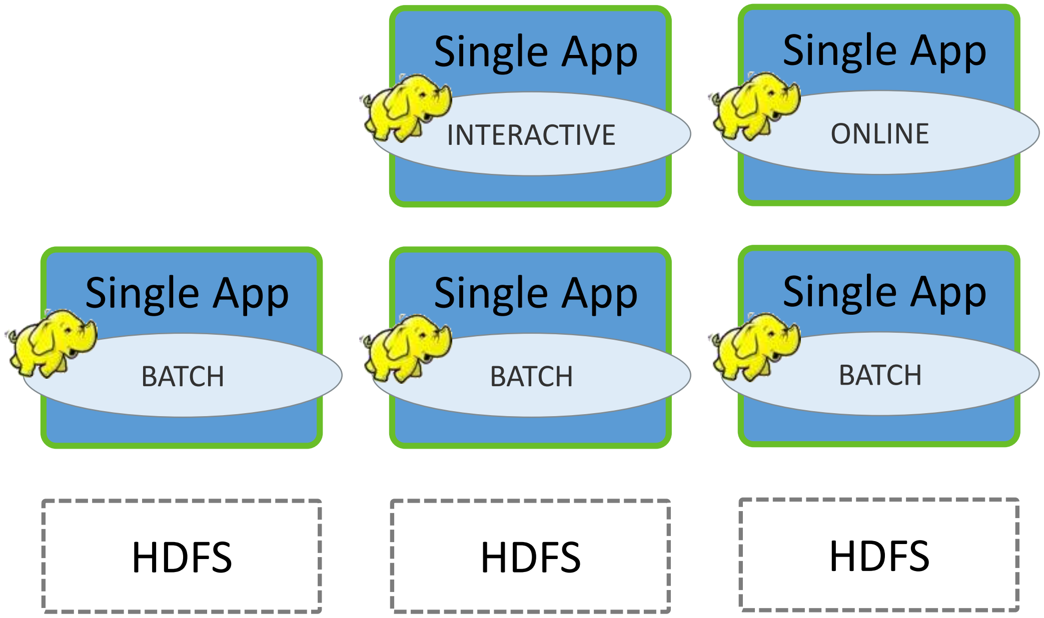
- MapReduce 1 (“classic”) has three main components
  - API→for user-level programming of MR applications
  - Framework→runtime services for running Map and Reduce processes, shuffling and sorting, etc.
  - Resource management→infrastructure to monitor nodes, allocate resources, and schedule jobs



MRv1: Batch Focus

HADOOP 1.0

Built for Web-Scale Batch Apps

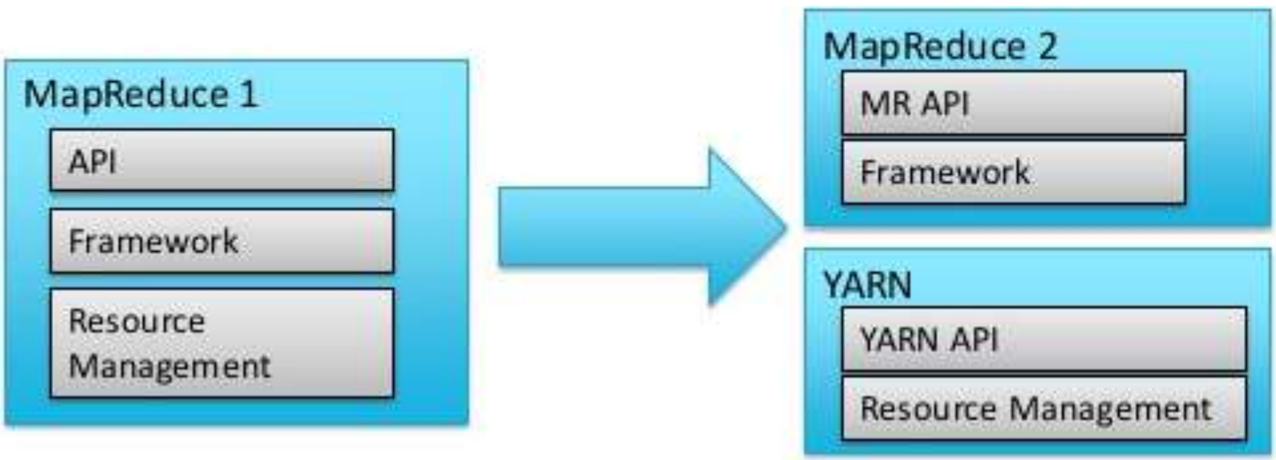


All other usage patterns MUST leverage same infrastructure

Forces Creation of Silos to Manage Mixed Workloads

YARN (MRv2)

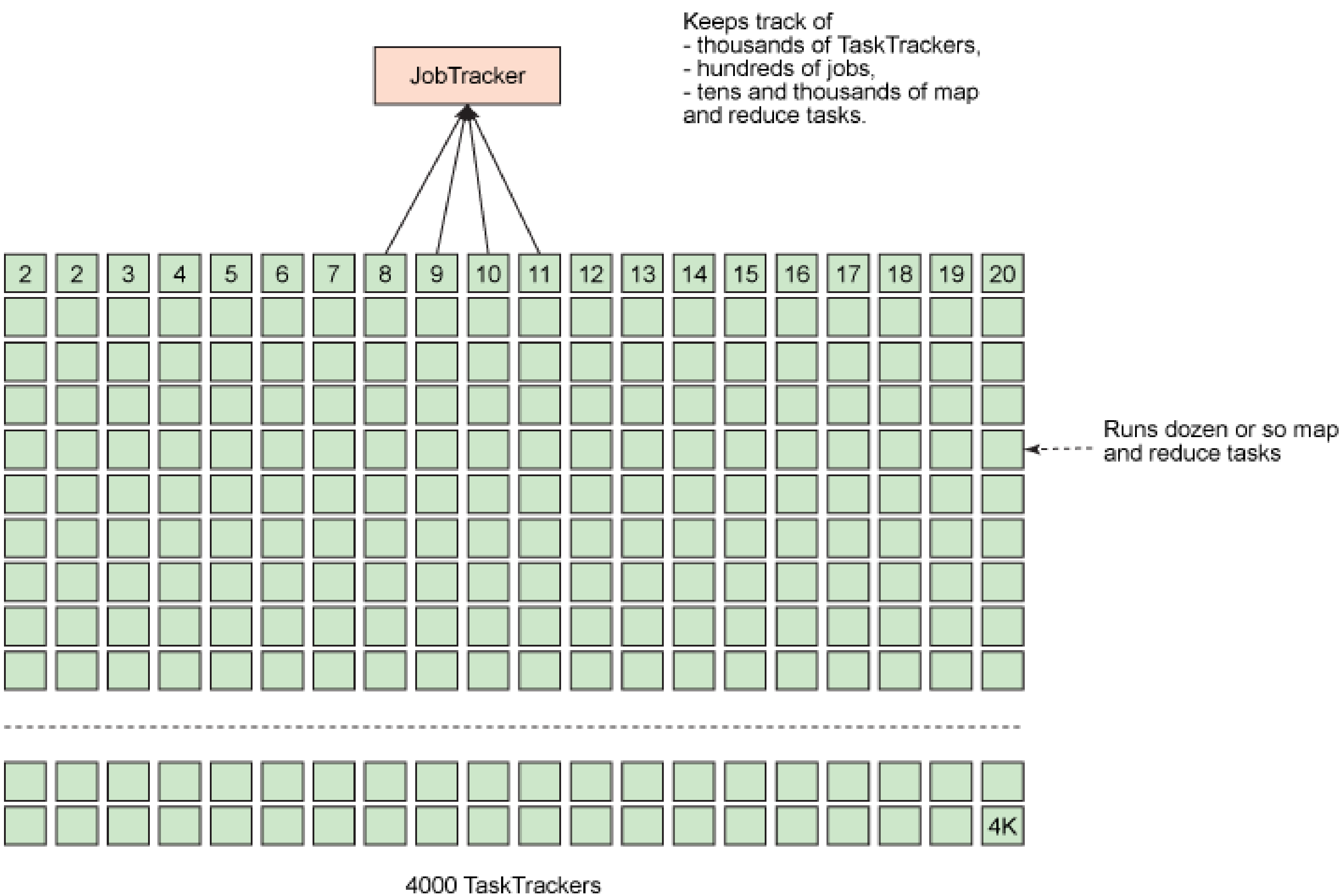
- MapReduce 2 move resource management to YARN
  - MapReduce originally architecture at Yahoo in 2008
  - “alpha” in Hadoop 2 (pre-GA)
  - YARN promoted to sub-project in Hadoop in 2013 (Best Paper in SOCC 2013)



Why YARN is needed? (1)

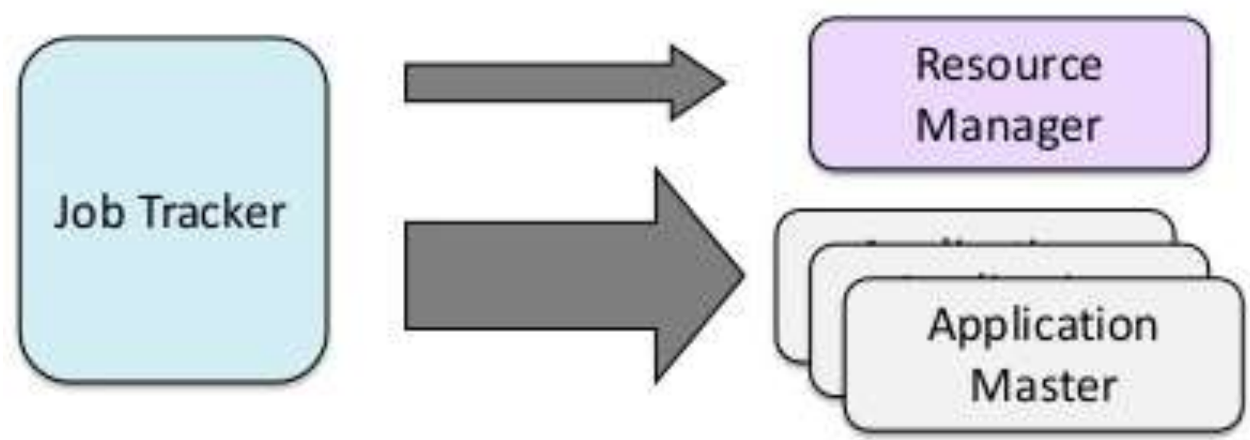
- MapReduce 1 resource management issues
  - Inflexible "slots" configured on nodes → map or reduce, not both
    - Underutilization of cluster when more map or reduce tasks are running
  - Cannot share resources with non-MR applications running on Hadoop cluster (e.g., impala, apache giraph)
  - Scalability → one Job Tracker per cluster – limit of about 4000 nodes per cluster

Busy JobTracker on a large Apache Hadoop cluster (MRv1)



Why YARN is needed? (2)

- YARN Solutions
  - No slots
    - Nodes have “resources” → memory and CPU cores – which are allocated to applications when requested
  - Supports MR and non-MR applications running on the same cluster
  - Most Job Tracker functions moved to Application Master → one cluster can have many Application Masters

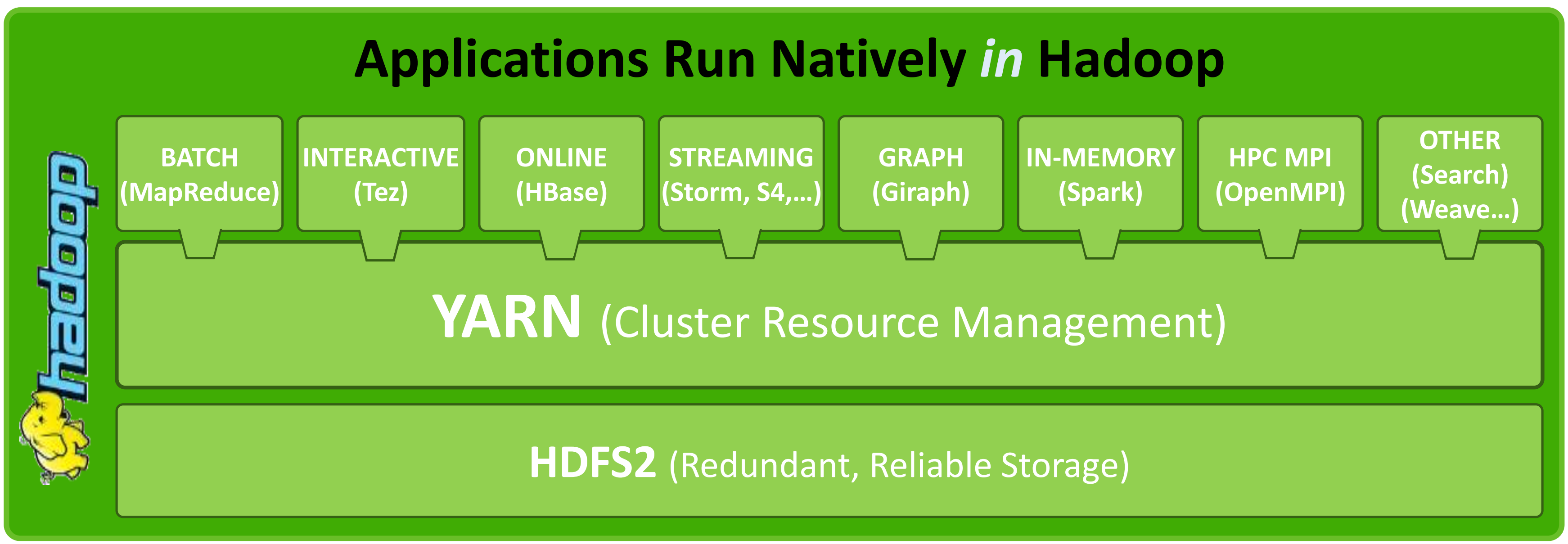


YARN: Taking Hadoop Beyond Batch

Store ALL DATA in one place…

Interact with that data in MULTIPLE WAYS

with Predictable Performance and Quality of Service



YARN: Efficiency with Shared Services

YAHOO!

Yahoo! leverages YARN

40,000+ nodes running YARN across over 365PB of data

~400,000 jobs per day for about 10 million hours of compute time

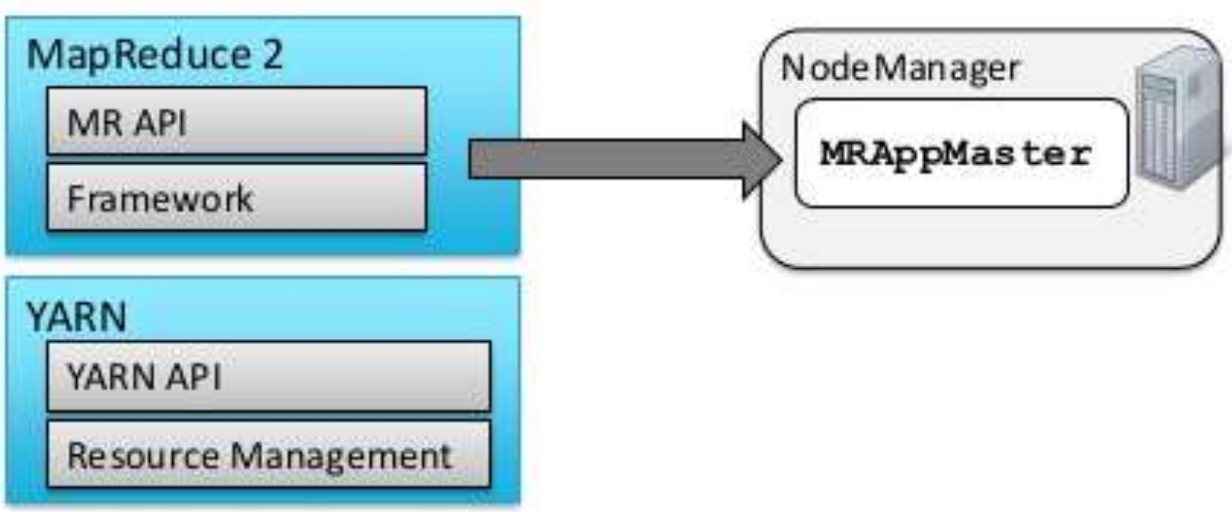
Estimated a 60% – 150% improvement on node usage per day using YARN

Eliminated Colo (~10K nodes) due to increased utilization

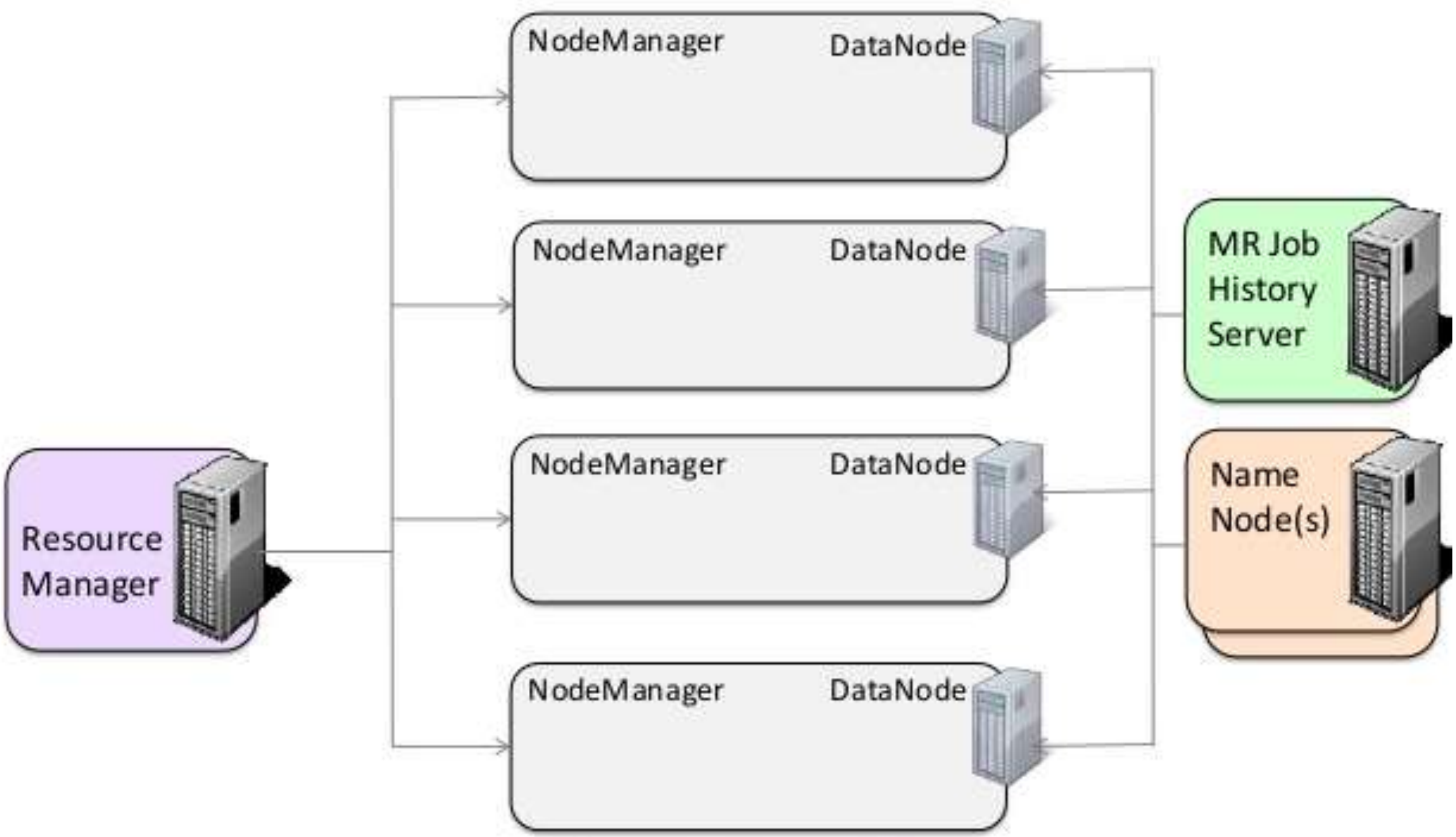
For more details check out the YARN SOCC 2013 paper

YARN and MapReduce

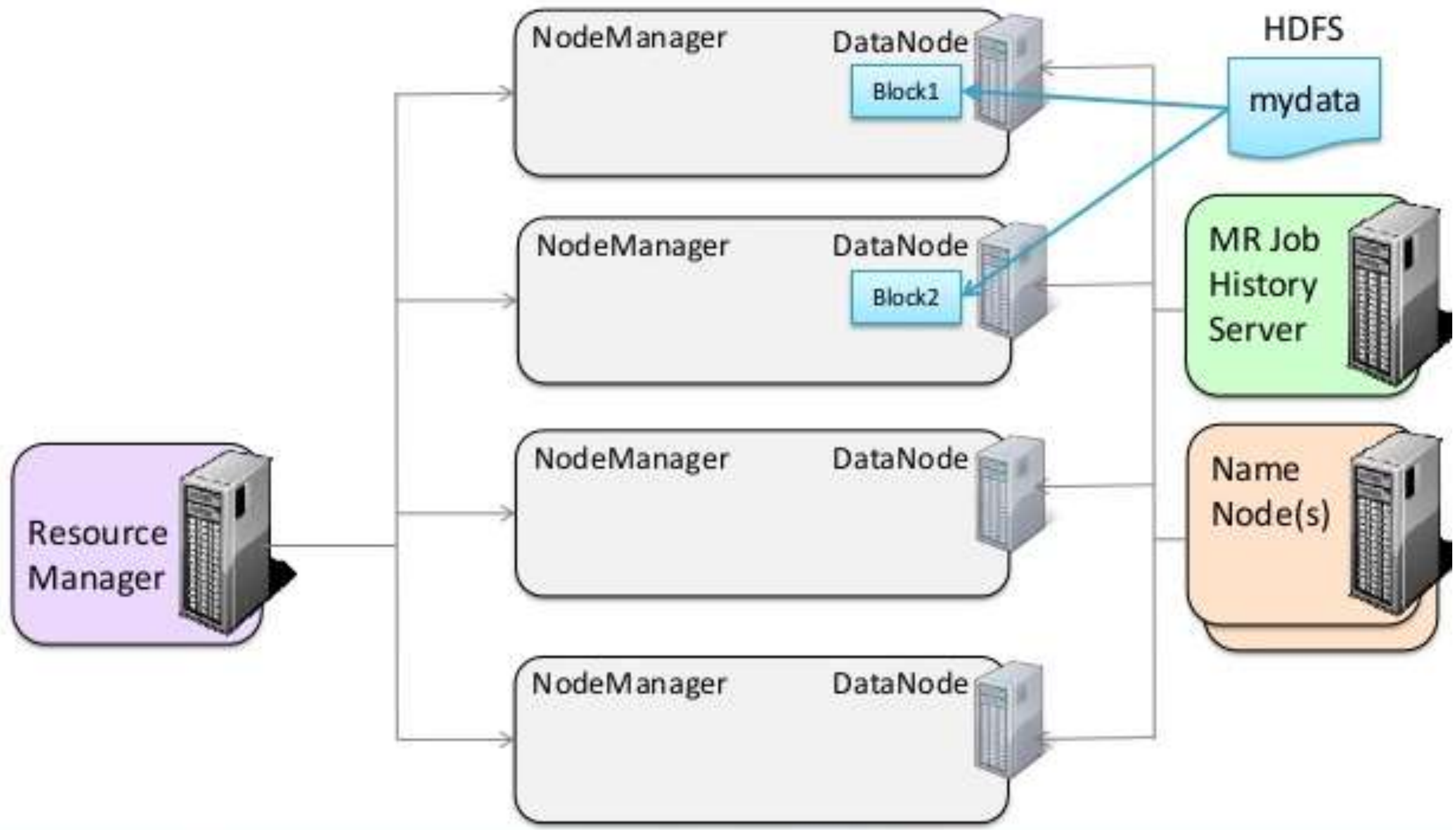
- YARN does not know or care what kind of application is running
  - Could be MR or something else (e.g., Impala)
- MR2 uses YARN
  - Hadoop includes a MapReduce ApplicationMaster (AM) to manage MR jobs
  - Each MapReduce job is a new instance of an application



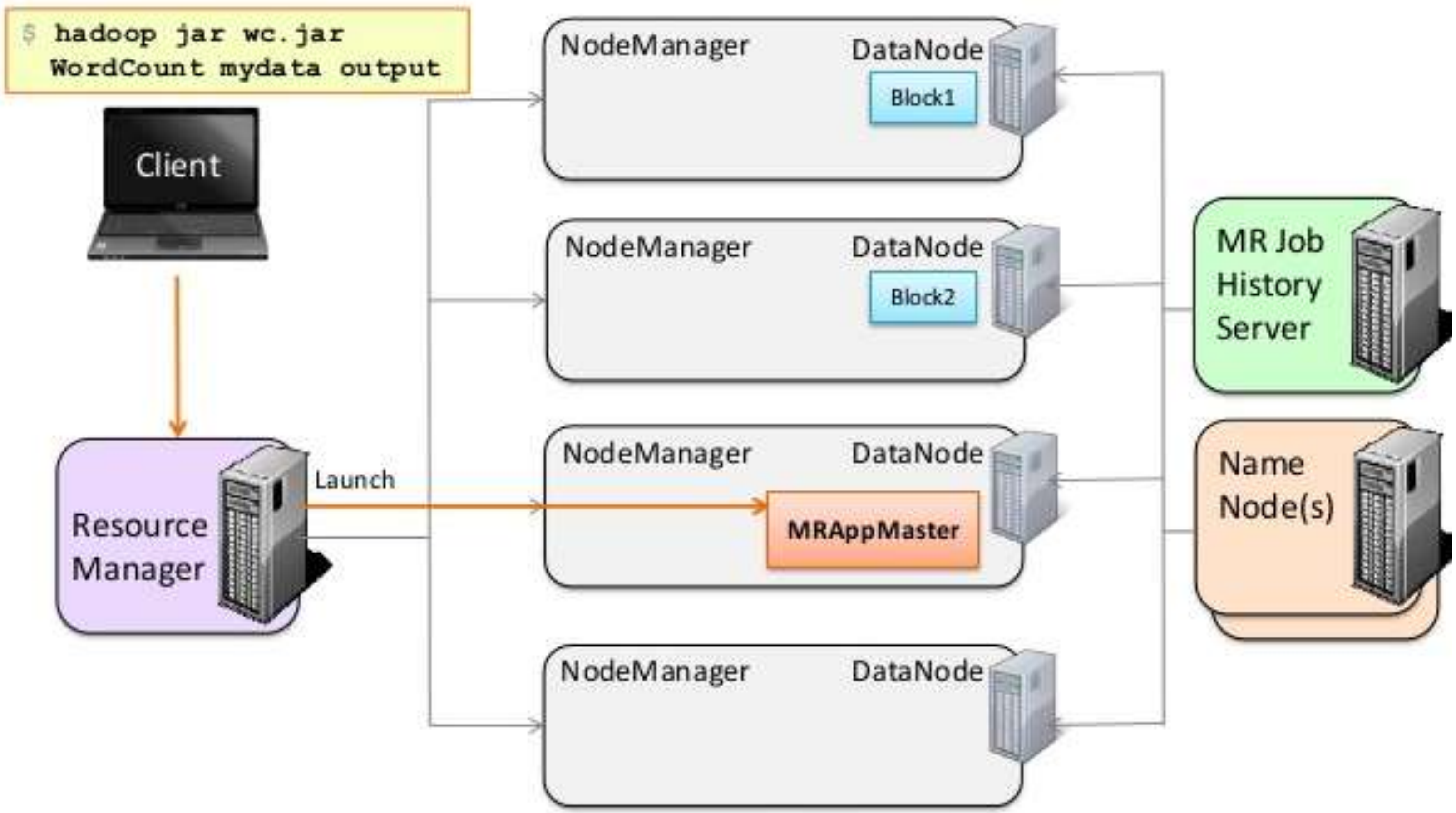
Running a MapReduce Application in MRv2 (1)



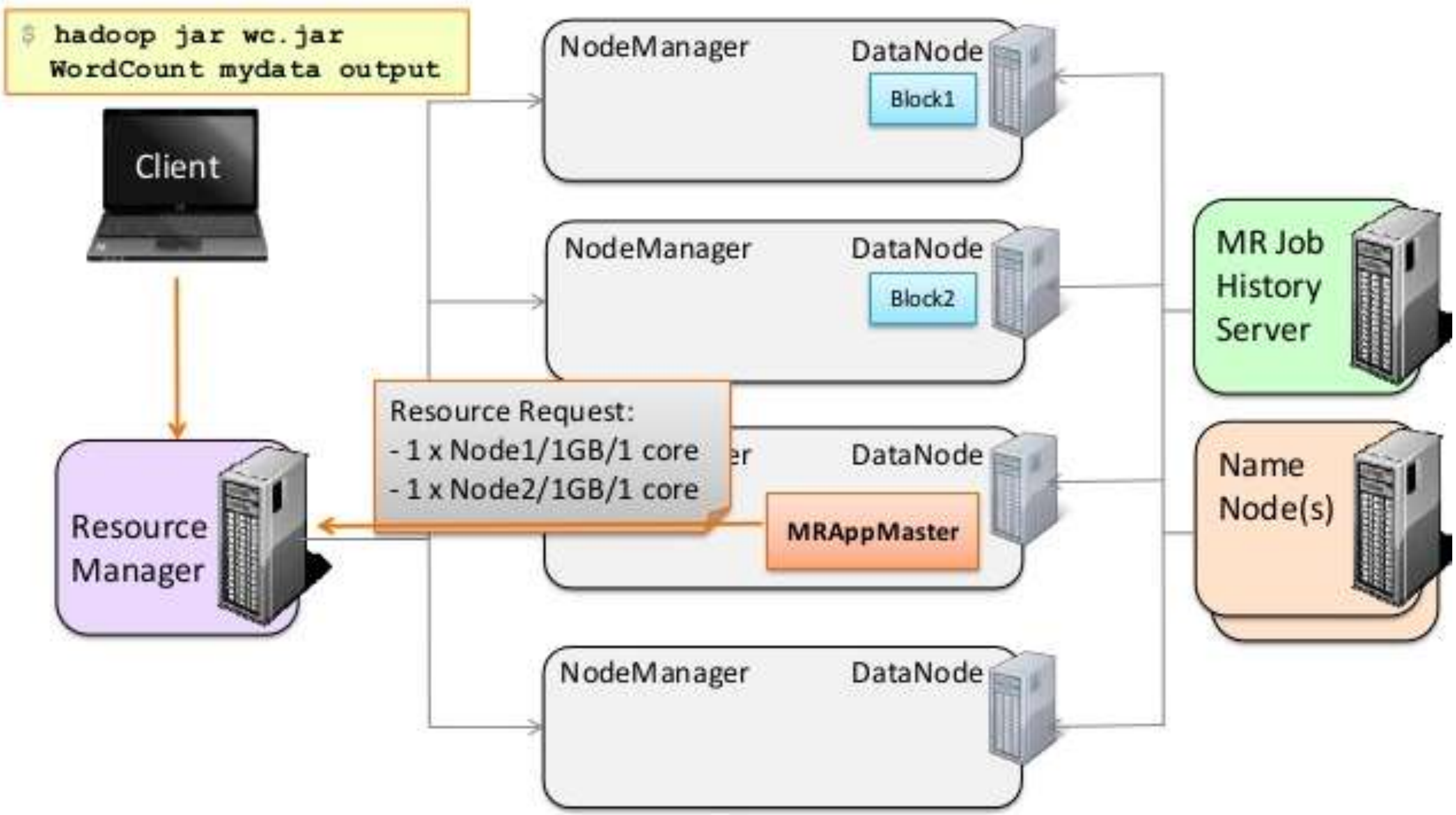
Running a MapReduce Application in MRv2 (2)



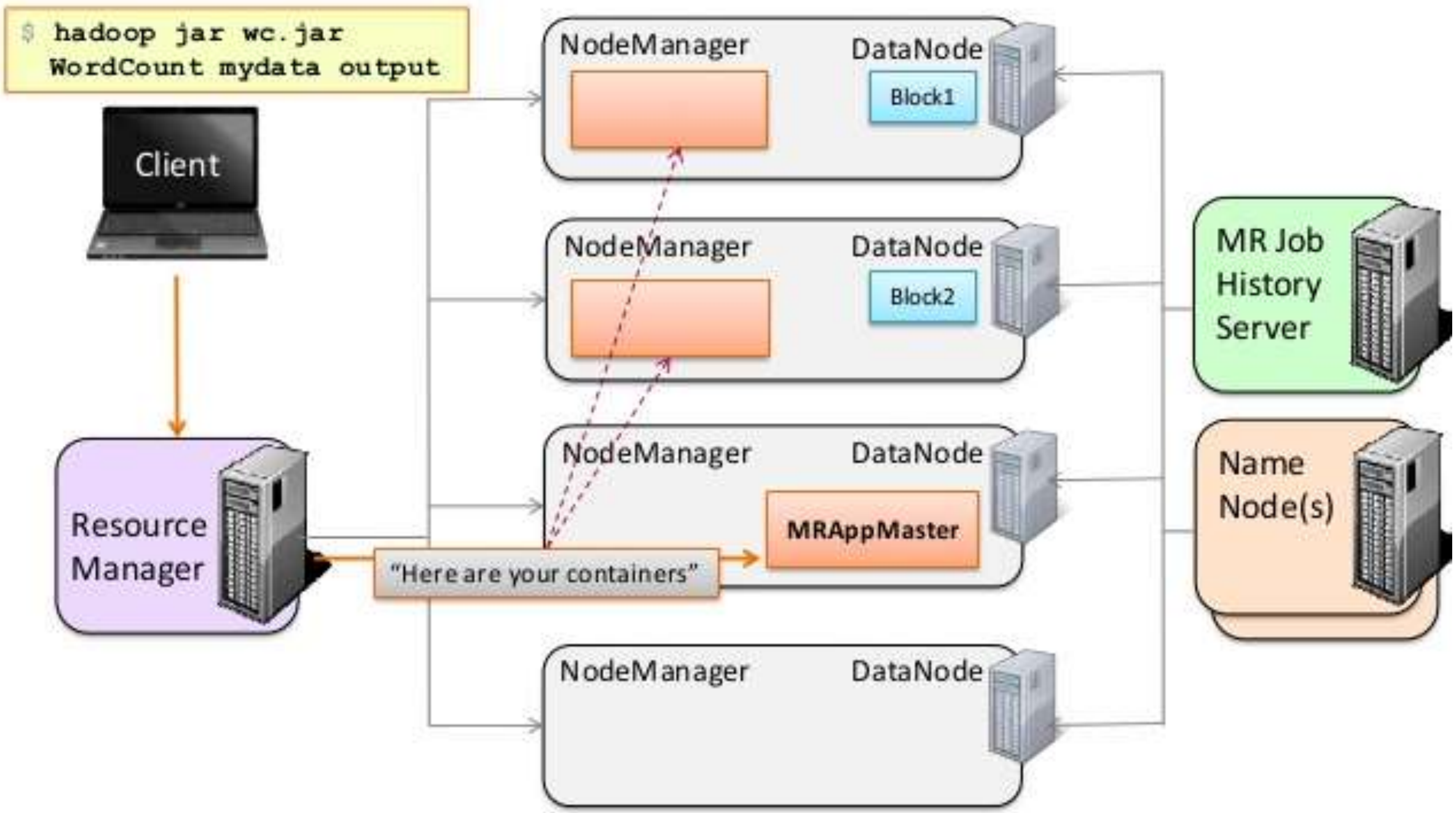
Running a MapReduce Application in MRv2 (3)



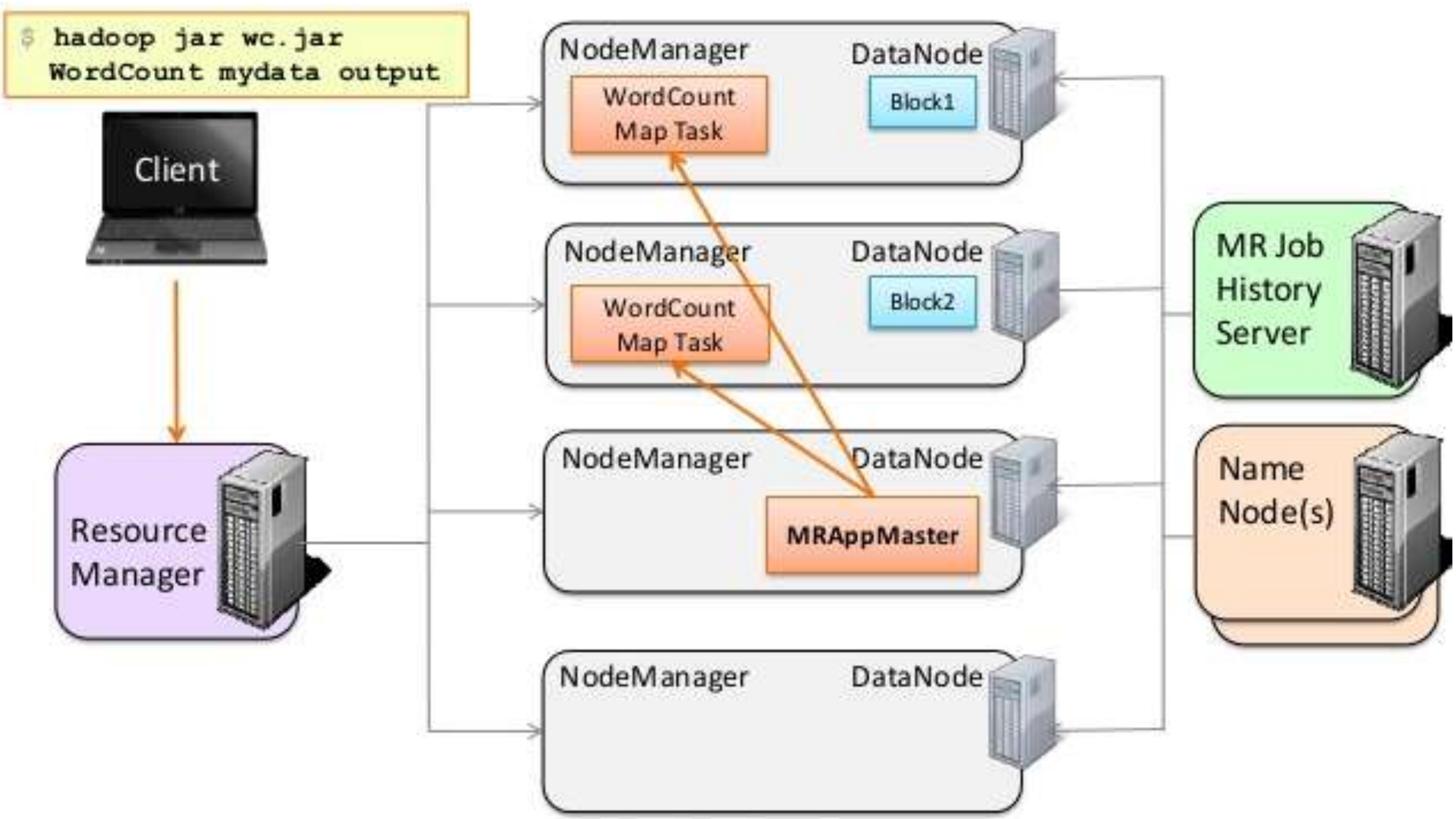
Running a MapReduce Application in MRv2 (4)



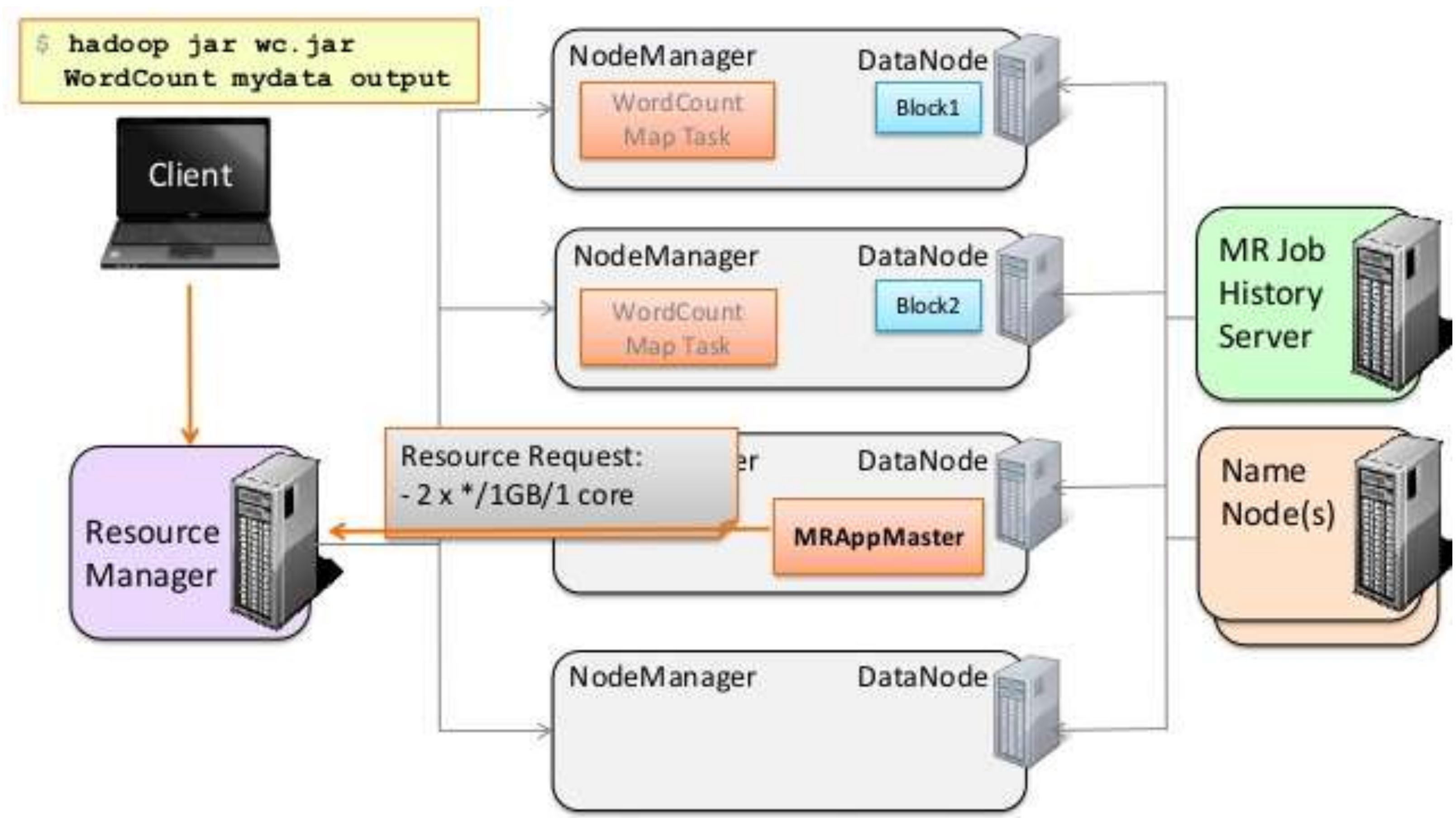
Running a MapReduce Application in MRv2 (5)



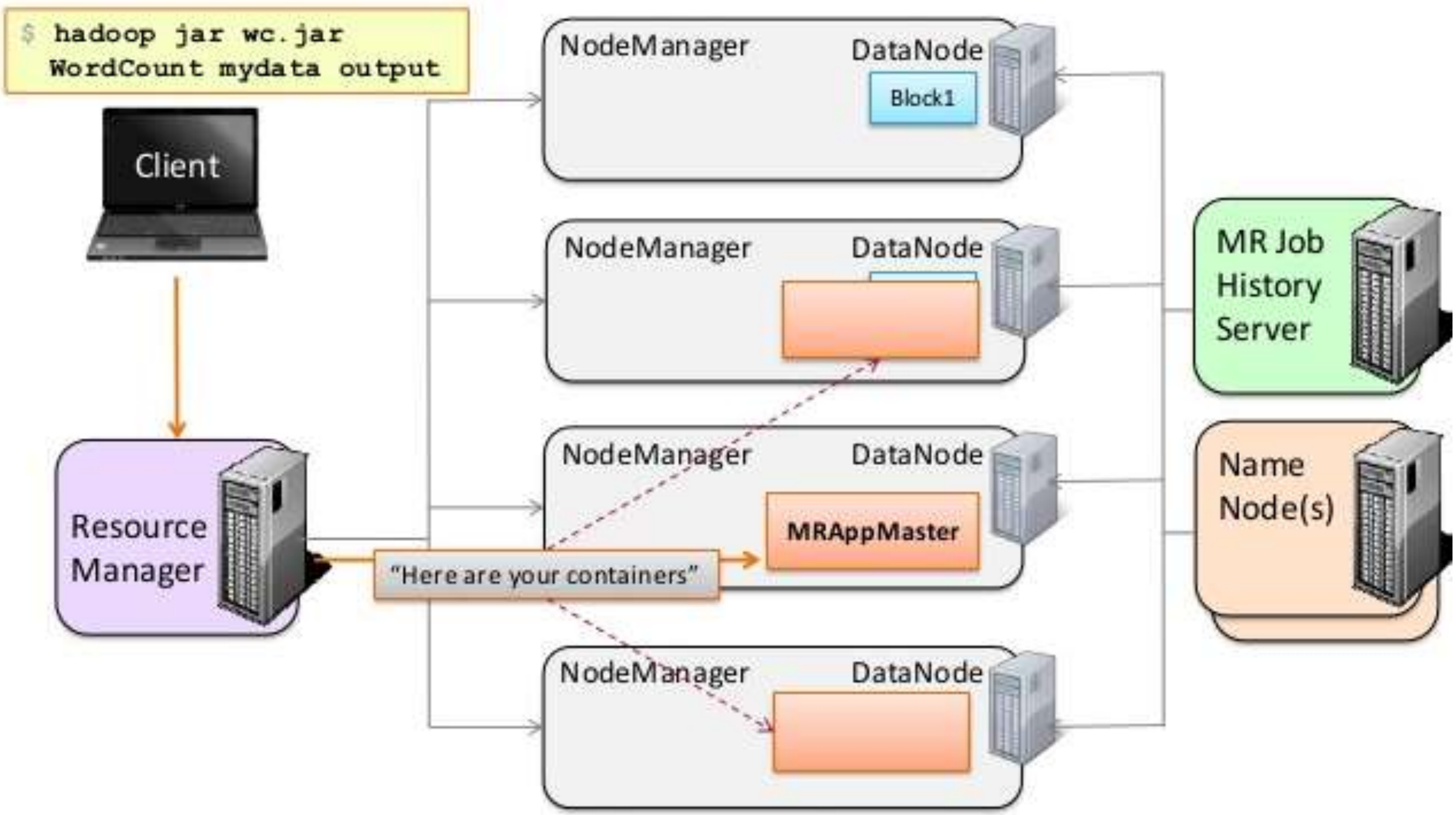
Running a MapReduce Application in MRv2 (6)



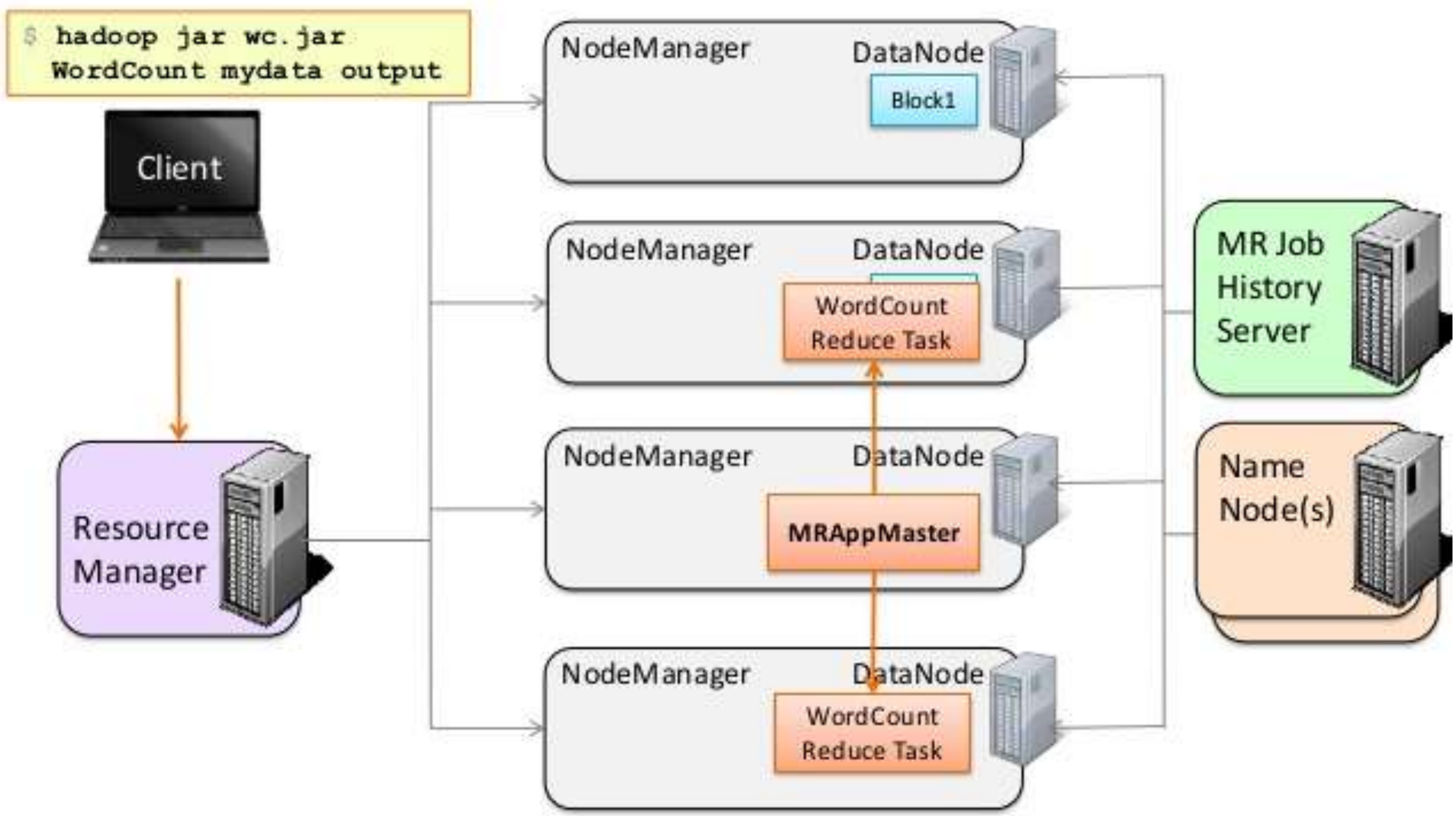
Running a MapReduce Application in MRv2 (7)



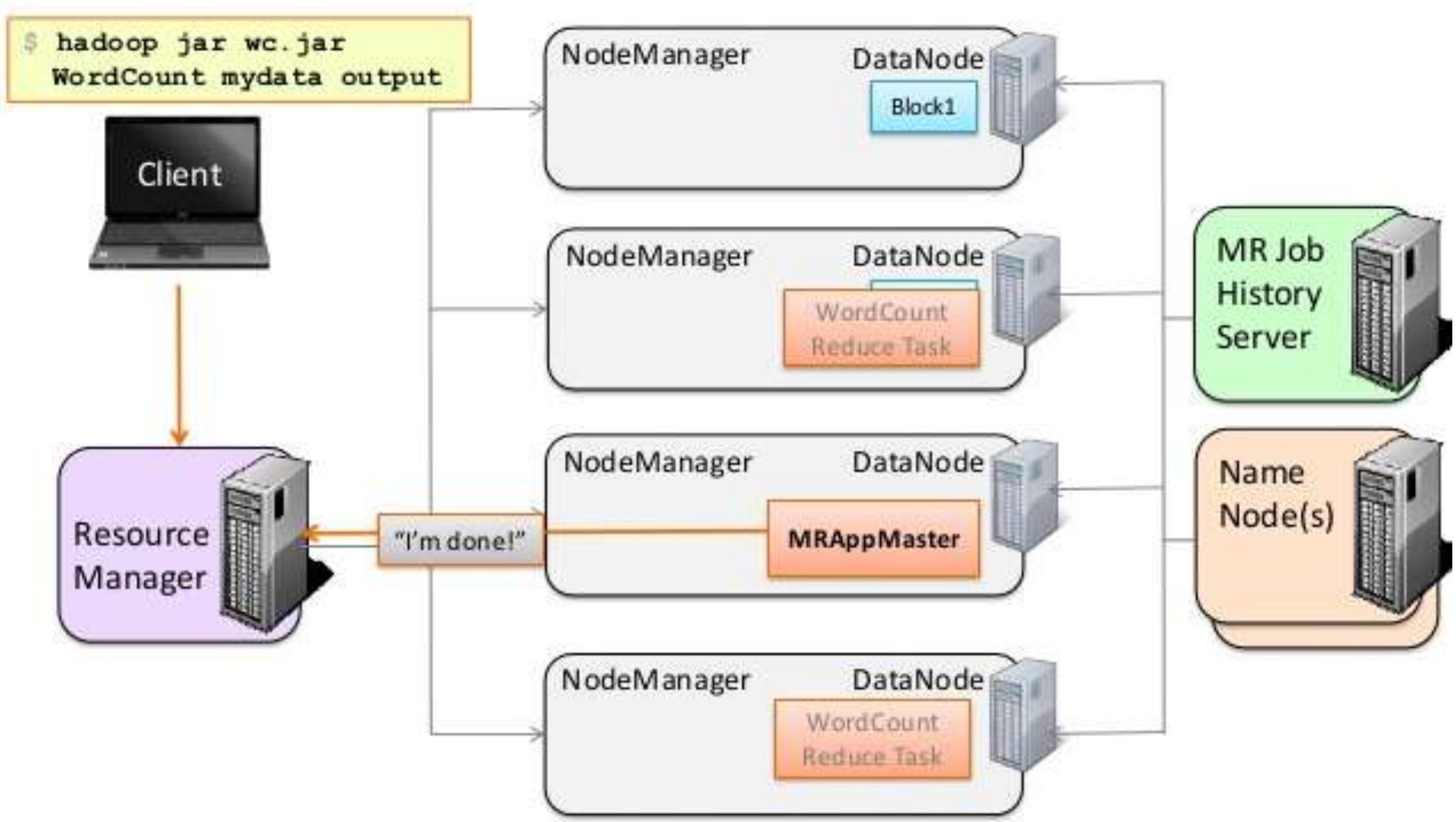
Running a MapReduce Application in MRv2 (8)



Running a MapReduce Application in MRv2 (9)

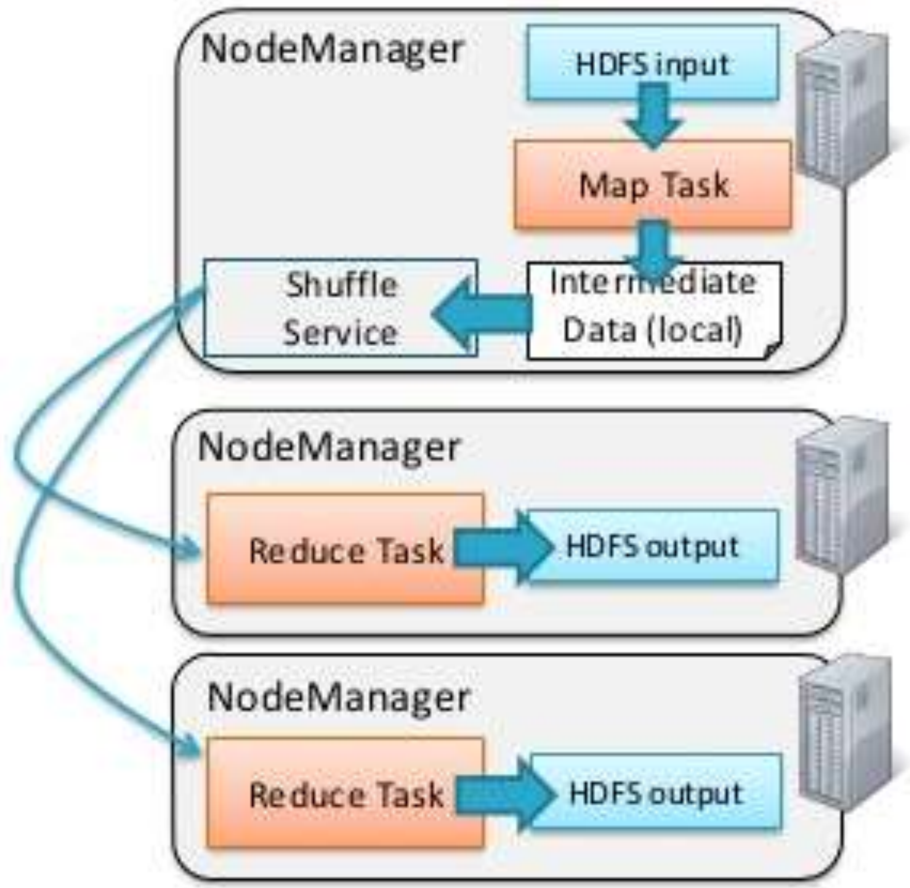


Running a MapReduce Application in MRv2 (10)



The MapReduce Framework on YARN

- In YARN, Shuffle is run as an auxiliary service
  - Runs in the NodeManager JVM as a persistent service



2 Introduction to Spark

What is Spark?

- Fast, expressive cluster computing system compatible with Apache Hadoop
  - Works with any Hadoop-supported storage system (HDFS, S3, Avro, …)
- Improves **efficiency** through:
  - In-memory computing primitives
  - General computation graphs
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell

→ Up to 100× faster

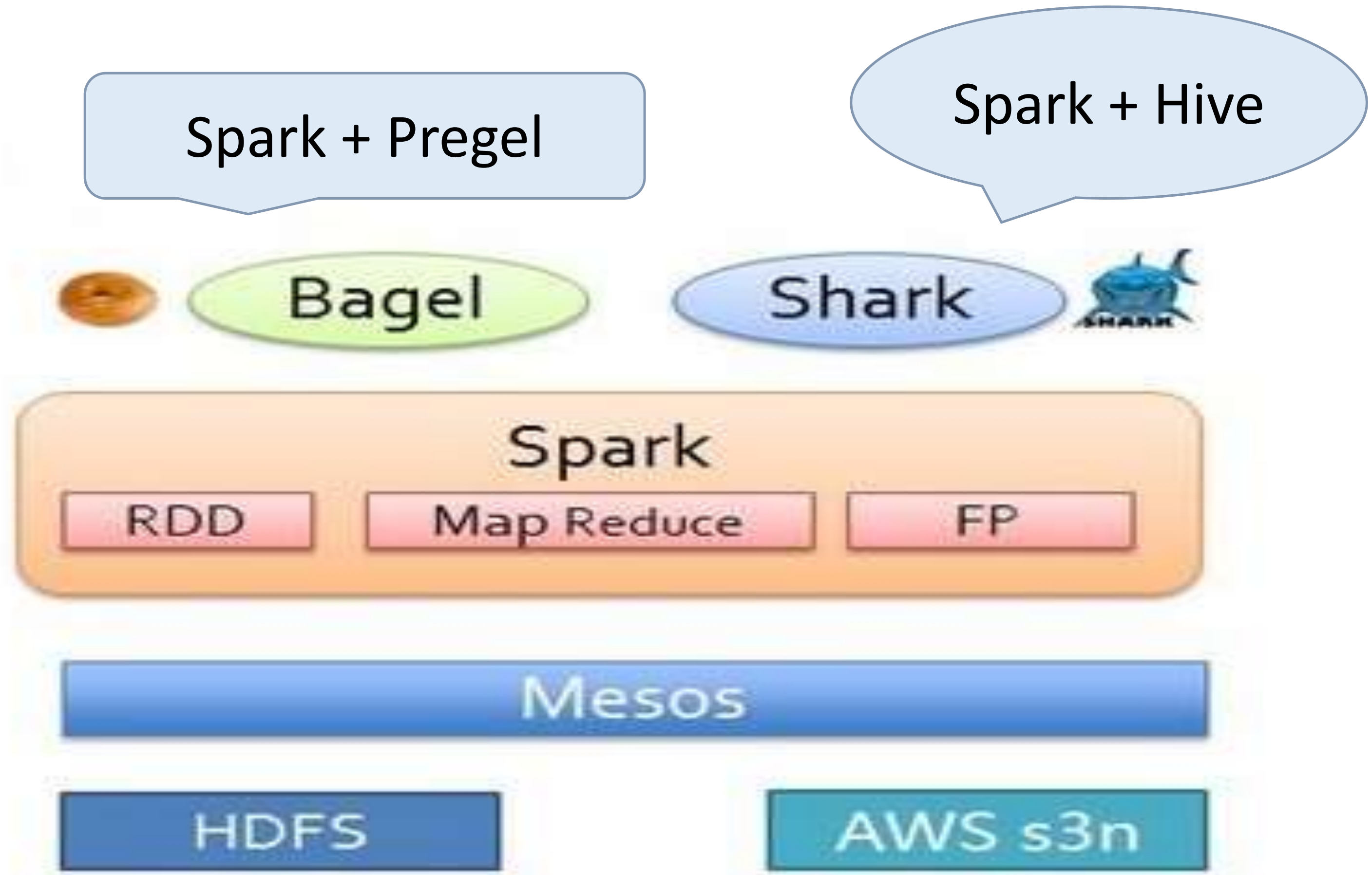
→ Often 2-10× less code

How to Run It & Languages

- Local multicore: just a library in your program
- EC2: scripts for launching a Spark cluster
- Private cluster: Mesos, YARN, Standalone Mode

- APIs in Java, Scala and Python
- Interactive shells in Scala and Python

Spark Framework



Spark

Key Idea

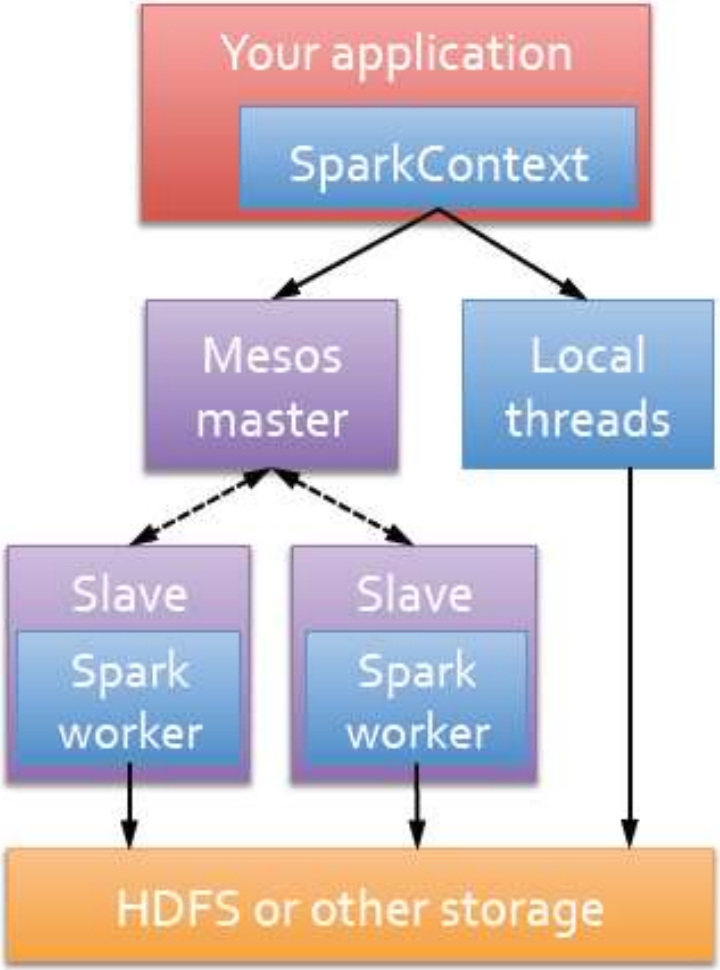
- **Work with distributed collections as you would with local ones**

- Concept: resilient distributed datasets (RDDs)
  - Immutable collections of objects spread across a cluster
  - Built through parallel transformations (map, filter, etc)
  - Automatically rebuilt on failure
  - Controllable persistence (e.g. caching in RAM)

Spark Runtime

- Spark runs as a library in your program
- (1 instance per app)

- Runs tasks locally or on Mesos
  - **new SparkContext** (masterUrl, jobname, [sparkhome], [jars])
  - MASTER=local[n] ./spark-shell
  - MASTER=HOST:PORT ./spark-shell



Example: Mining Console Logs

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

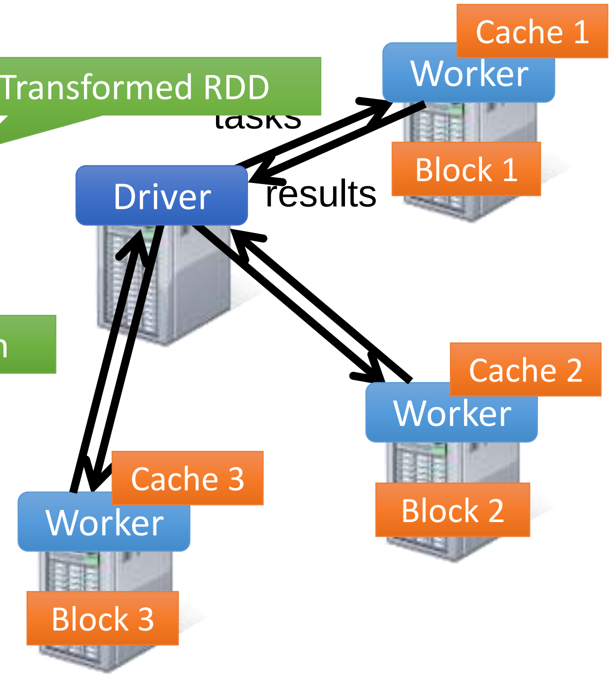
Base RDD

Transformed RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split('\t')[2])
messages.cache()
```

Action

```
messages.filter(lambda s: "foo" in s).count()
messages.filter(lambda s: "bar" in s).count()
. . .
```



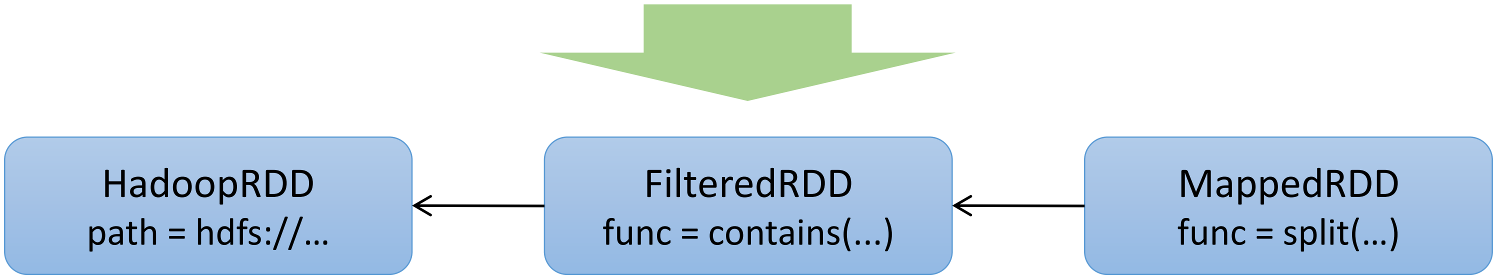
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

RDD Fault Tolerance

RDDs track the transformations used to build them (their *lineage*) to recompute lost data

E.g:

```
messages = textFile(...).filter(lambda s: s.contains(“ERROR”))
                        .map(lambda s: s.split(‘\t’)[2])
```

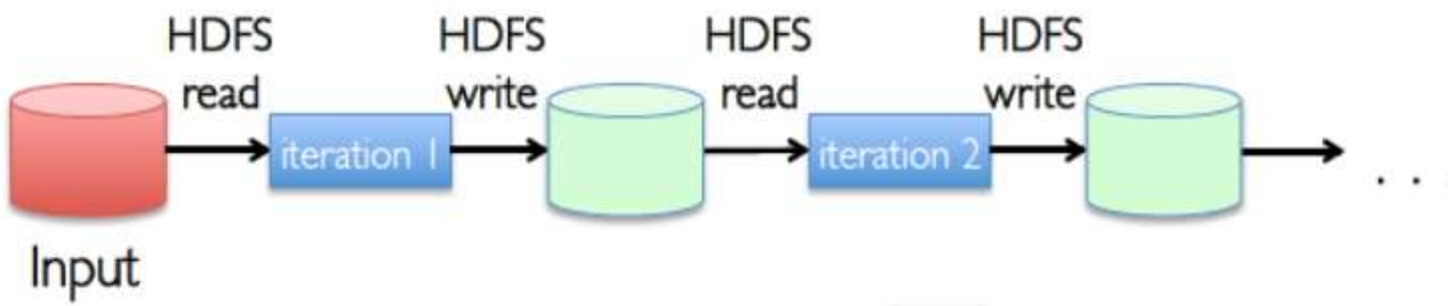


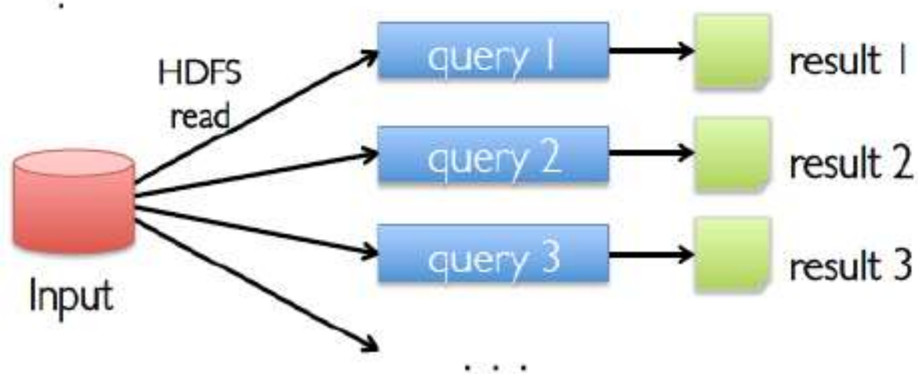
Which Language Should I Use?

- Standalone programs can be written in any, but console is only Python & Scala
- **Python developers:** can stay with Python for both
- **Java developers:** consider using Scala for console (to learn the API)

- Performance: Java / Scala will be faster (statically typed), but Python can do well for numerical work with NumPy

Iterative Processing in Hadoop



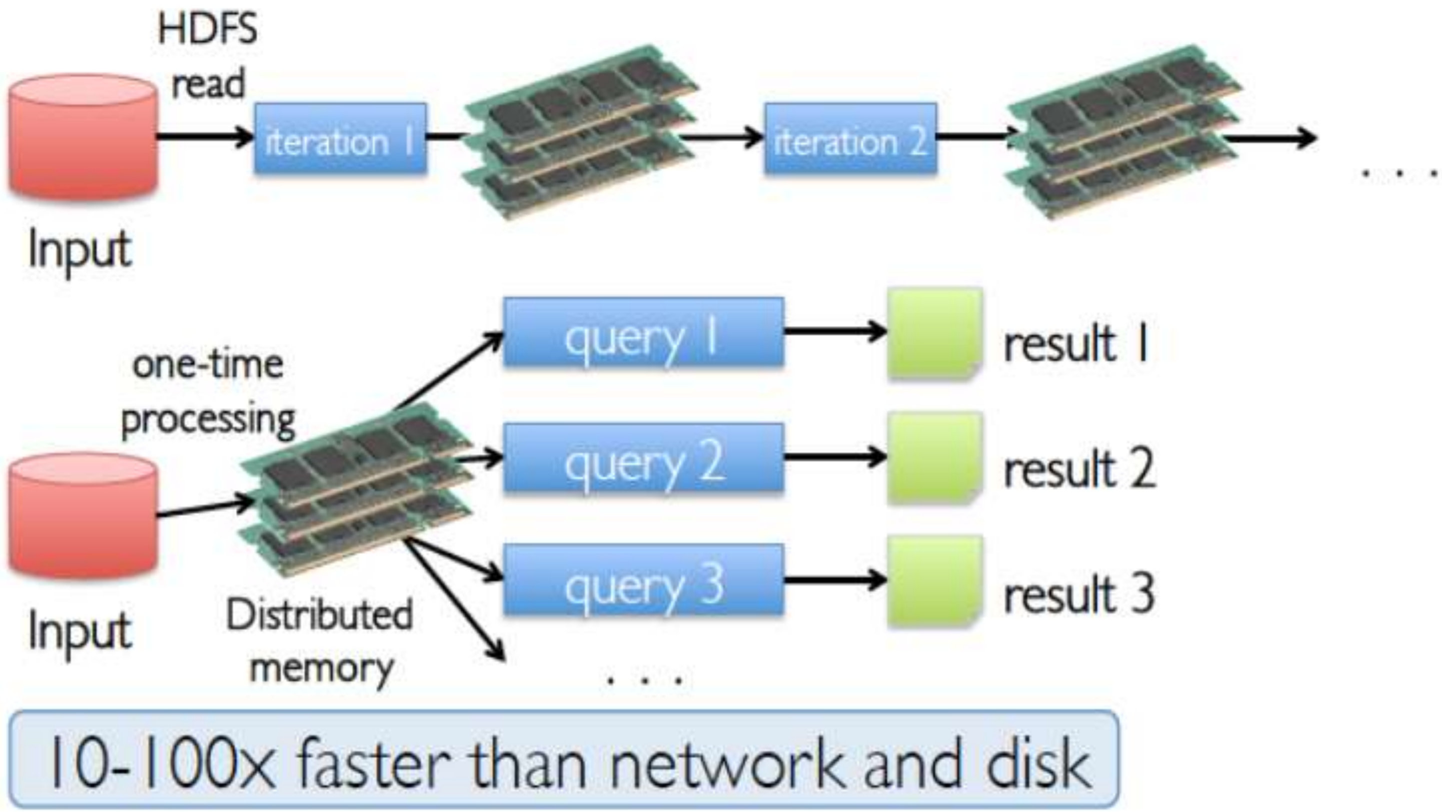


What is wrong with this approach?

Throughput Mem vs. Disk

- Typical throughput of disk: ~ 100 MB/sec
- Typical throughput of main memory: 50 GB/sec
- => Main memory is ~ 500 times faster than disk

Spark→ In Memory Data Sharing

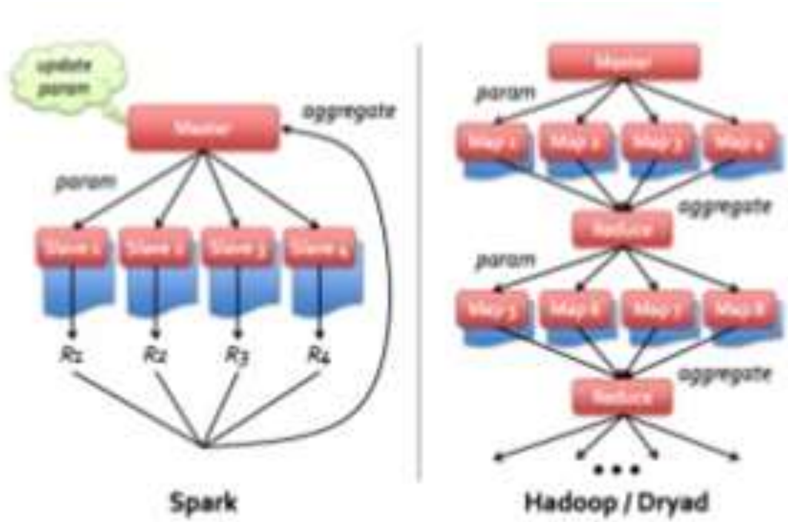


(from Matei Zaharia 2012, UC Berkeley)

Spark

Spark vs. Hadoop MapReduce (3)

- In-memory data flow model optimized for multi-stage jobs
- Novel approach to fault tolerance
- Similar programming style to Scalding/Cascading



Spark vs. Hadoop MapReduce (4)

	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

(from Ameet Talwalkar, UCLA, 2015)

On-Disk Sort Record
Time to sort 100TB

2013 Record: Hadoop

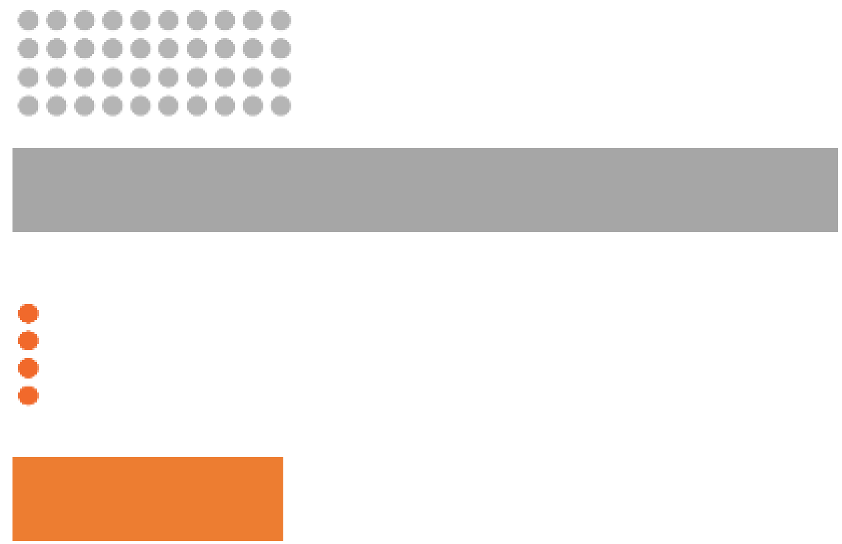
2100 machines

72 minutes

2014 Record: Spark

207 machines

23 minutes

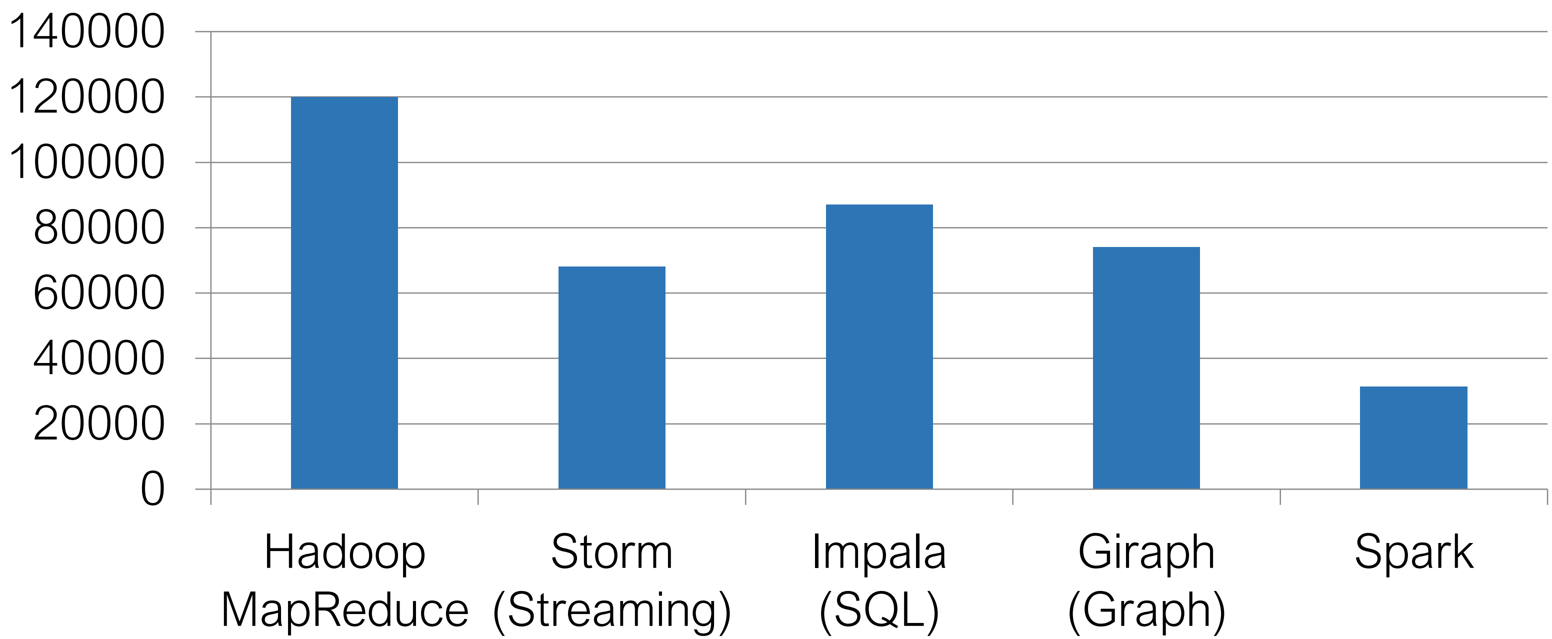


Also sorted 1PB in 4 hours

Source: Daytona GraySort benchmark, sortbenchmark.org

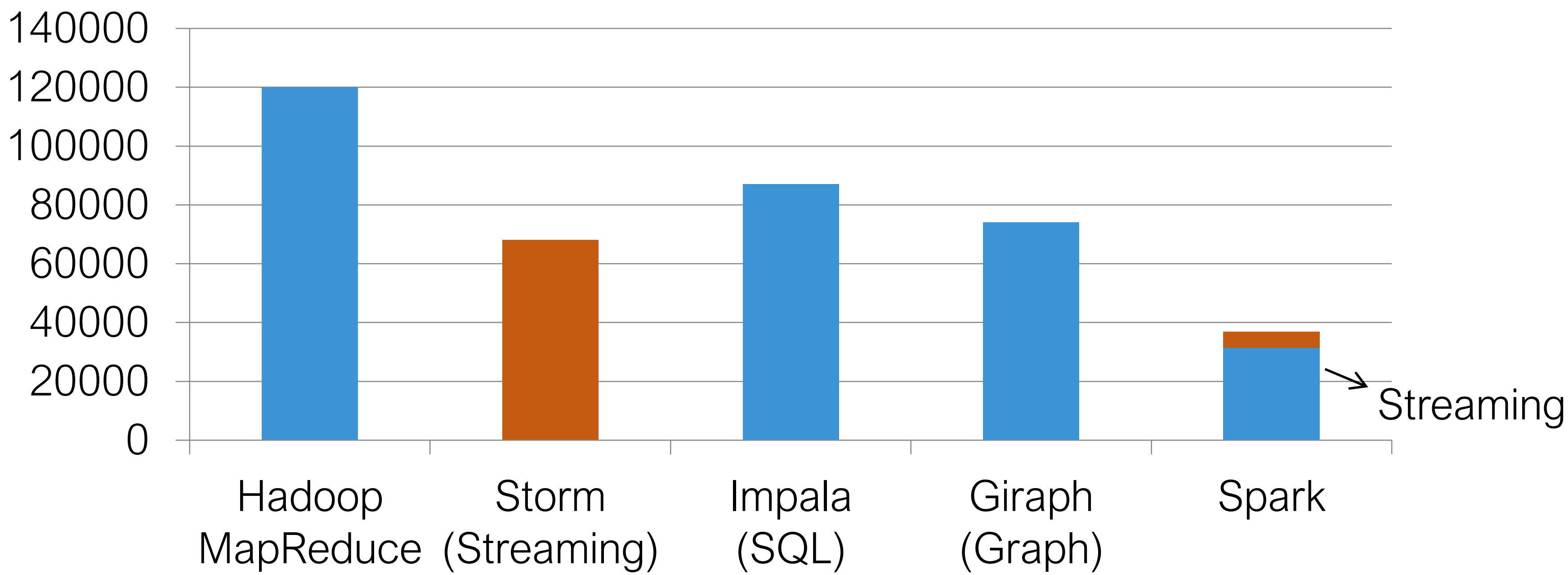
Spark

Powerful Stack – Agile Development (1)



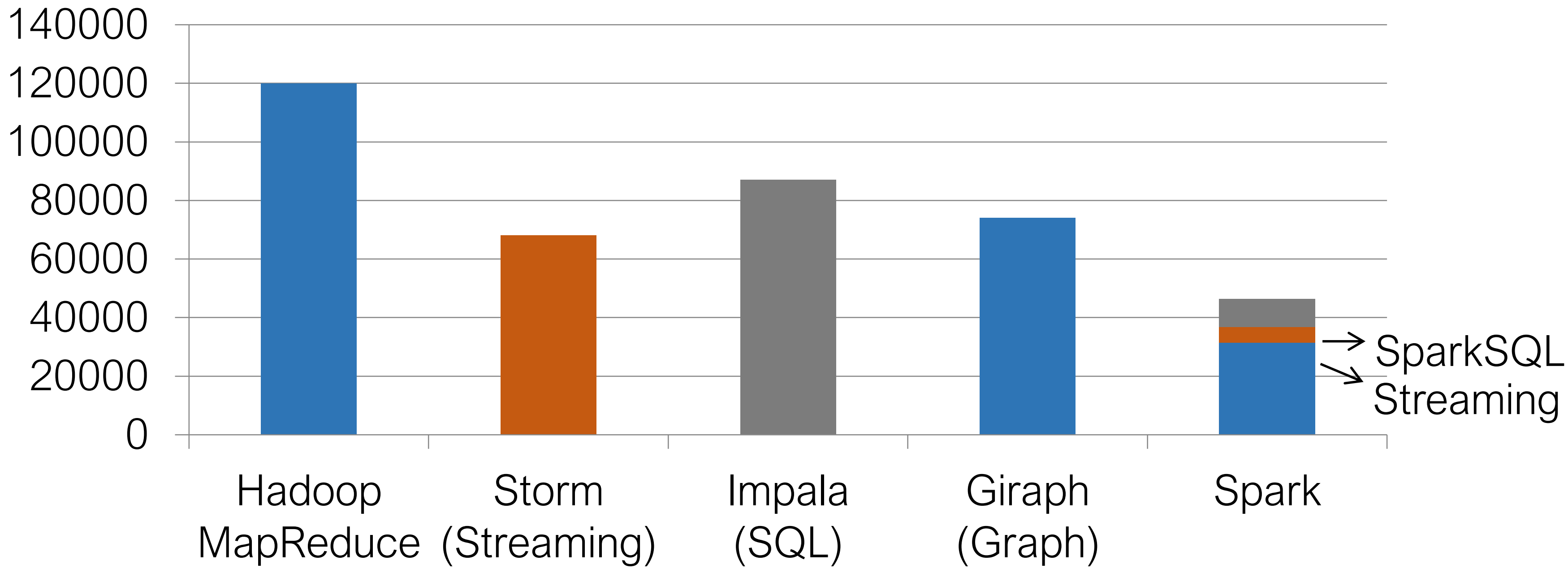
non-test, non-example source lines

Powerful Stack – Agile Development (2)



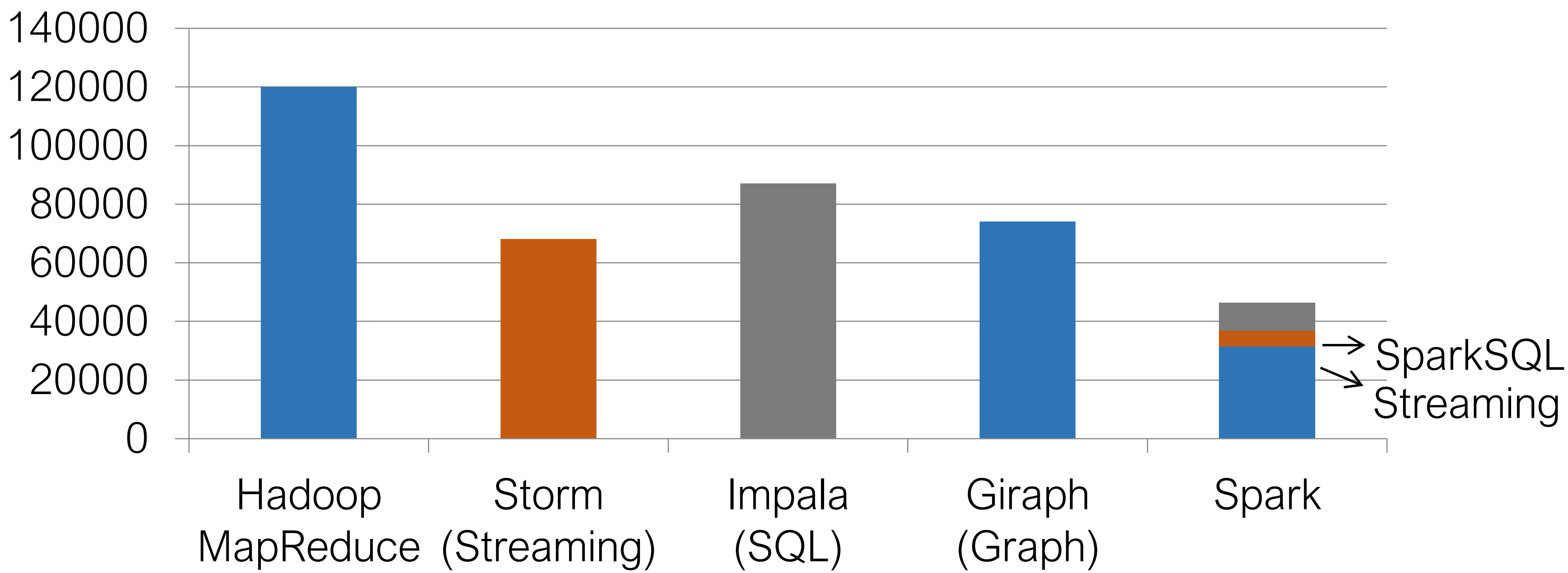
non-test, non-example source lines

Powerful Stack – Agile Development (3)



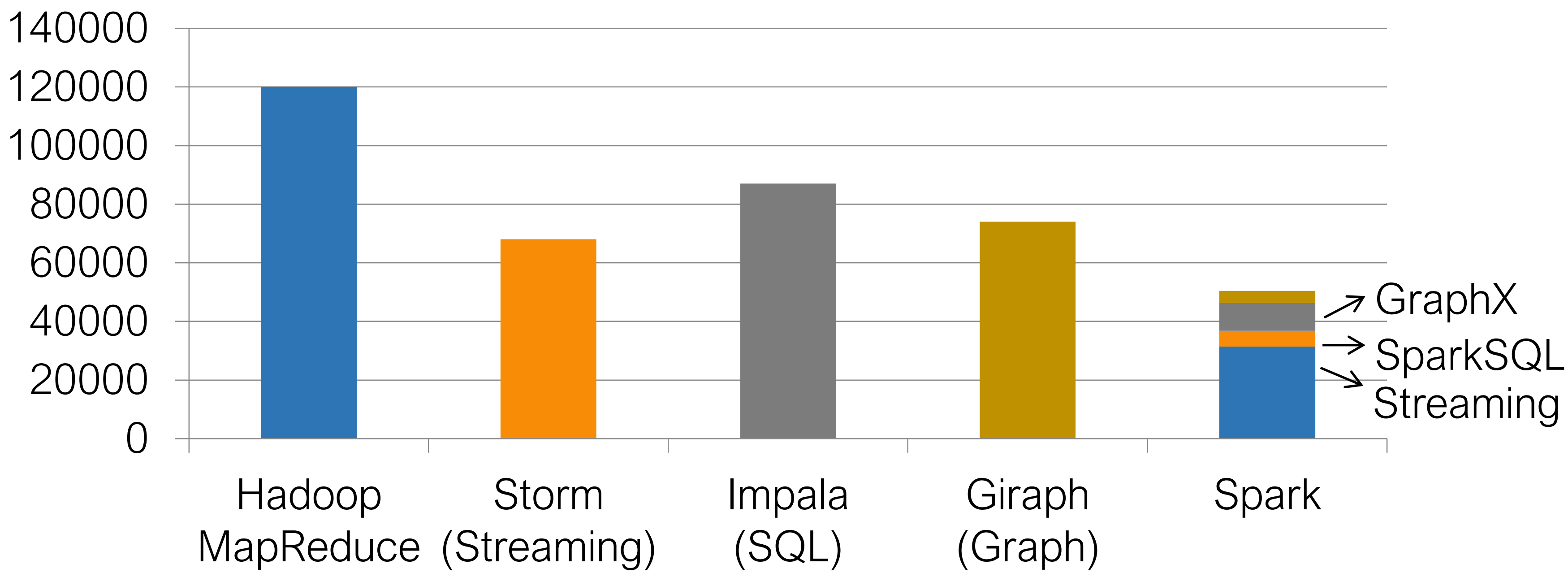
non-test, non-example source lines

Powerful Stack – Agile Development (4)



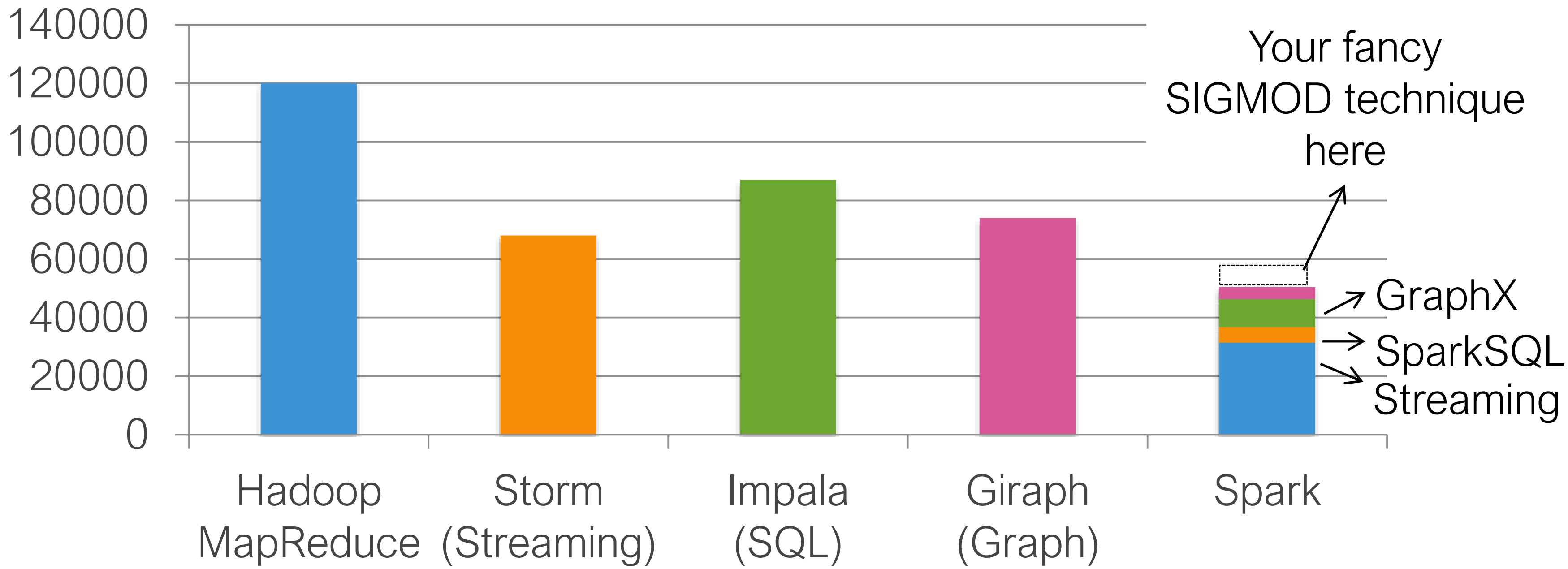
non-test, non-example source lines

Powerful Stack – Agile Development (5)



non-test, non-example source lines

Powerful Stack – Agile Development (6)



non-test, non-example source lines

3

Spark Programming

Learning Spark

- Easiest way: Spark interpreter (`spark-shell` or `pyspark`)
  - Special Scala and Python consoles for cluster use
- Runs in local mode on 1 thread by default, but can control with MASTER environment var:

```
MASTER=local      ./spark-shell          # local, 1 thread
MASTER=local[2]   ./spark-shell          # local, 2 threads
MASTER=spark://host:port ./spark-shell  # Spark standalone cluster
```

First Step: SparkContext

- Main entry point to Spark functionality
- Created for you in Spark shells as variable `sc`
- In standalone programs, you'd make your own (see later for details)

Creating RDDs

```
# Turn a local collection into an RDD
sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
sc.textFile("file.txt")
sc.textFile("directory/*.txt")
sc.textFile("hdfs://namenode:9000/path/file")

# Use any existing Hadoop InputFormat
sc.hadoopFile(keyClass, valClass, inputFmt,
conf)
```

Basic Transformations

```
nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
squares = nums.map(lambda x: x*x)   # => {1, 4, 9}

# Keep elements passing a predicate
even = squares.filter(lambda x: x % 2 == 0) # => {4}

# Map each element to zero or more others
nums.flatMap(lambda x: range(0, x))  # => {0, 0, 1, 0, 1, 2}
```

Range object (sequence of numbers 0, 1, ..., x-1)

Basic Actions

```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]

# Return first K elements
nums.take(2)   # => [1, 2]

# Count number of elements
nums.count()   # => 3

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

Working with Key-Value Pairs

- Spark's "distributed reduce" transformations act on RDDs of *key-value pairs*
- Python:

```python
pair = (a, b)
       pair[0] # => a
       pair[1] # => b
```

- Scala:

```scala
val pair = (a, b)
       pair._1 // => a
       pair._2 // => b
```

- Java:

```java
Tuple2 pair = new Tuple2(a, b);  // class scala.Tuple2

       pair._1 // => a
       pair._2 // => b
```

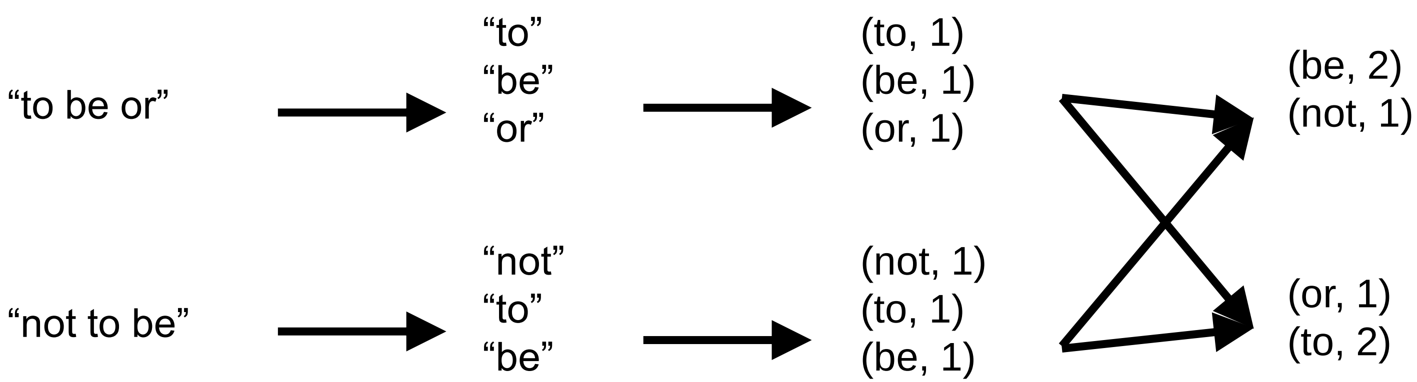
Some Key-Value Operations

```
pets = sc.parallelize([(“cat”, 1), (“dog”, 1), (“cat”, 2)])

pets.reduceByKey(lambda x, y: x + y)
# => {(cat, 3), (dog, 1)}

pets.groupByKey()
# => {(cat, Seq(1, 2)), (dog, Seq(1)}

pets.sortByKey()
# => {(cat, 1), (cat, 2), (dog, 1)}
```

reduceByKey also automatically implements combiners on the map side

Example: Word Count

```
lines = sc.textFile(“hamlet.txt”)

counts = lines.flatMap(lambda line:
line.split(“ ”)) \
               .map(lambda word: (word, 1)) \
               .reduceByKey(lambda x, y: x + y)
```



Multiple Datasets

```
visits = sc.parallelize([(“index.html”, “1.2.3.4”),
                         (“about.html”, “3.4.5.6”),
                         (“index.html”, “1.3.3.1”)])

pageNames = sc.parallelize([(“index.html”, “Home”),
(“about.html”, “About”)])

visits.join(pageNames)
# (“index.html”, (“1.2.3.4”, “Home”))
# (“index.html”, (“1.3.3.1”, “Home”))
# (“about.html”, (“3.4.5.6”, “About”))

visits.cogroup(pageNames)
# (“index.html”, (Seq(“1.2.3.4”, “1.3.3.1”), Seq(“Home”)))
# (“about.html”, (Seq(“3.4.5.6”), Seq(“About”)))
```

Controlling the level of parallelism

- All the pair RDD operations take an optional second parameter for number of tasks

```
words.reduceByKey(lambda x, y: x + y, 5)
words.groupByKey(5)
visits.join(pageViews, 5)
```

Using Local Variables

- External variables you use in a closure will automatically be shipped to the cluster:

```
query = raw_input("Enter a query:")
pages.filter(lambda x: x.startswith(query)).count()
```

- Some caveats:
  - Each task gets a new copy (updates aren't sent back)
  - Variable must be Serializable (Java/Scala) or Pickle-able (Python)
  - Don't use fields of an outer object (ships all of it!)

Closure Mishap Example

```
class MyCoolRddApp {
  val param = 3.14
  val log = new Log(...)
  ...

  def work(rdd: RDD[Int]) {
    rdd.map(x => x + param)
       .reduce(...)
  }
}
```

NotSerializableException: MyCoolRddApp (or Log)

How to get around it:

```
class MyCoolRddApp {
  ...

  def work(rdd: RDD[Int]) {
    val param_ = param
    rdd.map(x => x + param_)
       .reduce(...)
  }
}
```

References only local variable instead of `this.param`

Spark

Build Spark

- Requires Java 6+, Scala 2.9.2

```
git clone git://github.com/mesos/spark
cd spark
sbt/sbt package

# Optional: publish to local Maven cache
sbt/sbt publish-local
```

Add Spark into Your Project

- Scala and Java: add a Maven dependency on

  groupId: `org.spark-project`
  artifactId: `spark-core_2.9.1`
  version: `0.7.0-SNAPSHOT`

- Python: run program with our `pyspark` script

Create a SparkContext

Scala

```
import spark.SparkContext
import spark.SparkContext._

val sc = new SparkContext(“masterUrl”, “name”, “sparkHome”,
Seq(“app.jar”))
```

Cluster URL, or local / local[N]

App name

Spark install path on cluster

List of JARs with app code (to ship)

Java

```
import spark.api.java.JavaSparkContext;

JavaSparkContext sc = new JavaSparkContext(
    “masterUrl”, “name”, “sparkHome”, new String[]
{“app.jar”}));
```

Python

```
from pyspark import SparkContext

sc = SparkContext(“masterUrl”, “name”, “sparkHome”,
[“library.py”]))
```

Complete App: Scala

```
import spark.SparkContext
import spark.SparkContext._

object WordCount {
  def main(args: Array[String]) {
    val sc = new SparkContext(“local”,
“WordCount”, args(0), Seq(args(1)))
    val lines = sc.textFile(args(2))
    lines.flatMap(_.split(“ ”))
         .map(word => (word, 1))
         .reduceByKey(_ + _)
         .saveAsTextFile(args(3))
  }
}
```

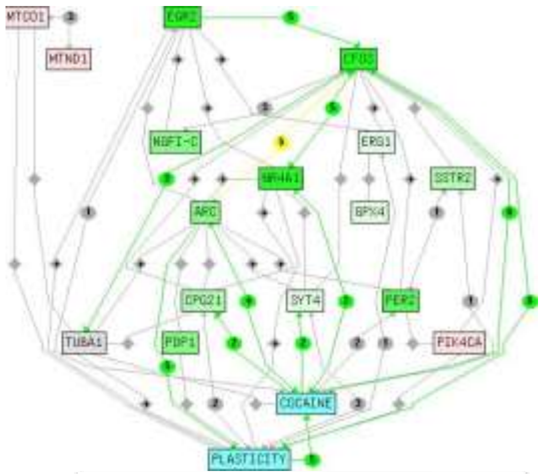
Spark

Complete App: Python

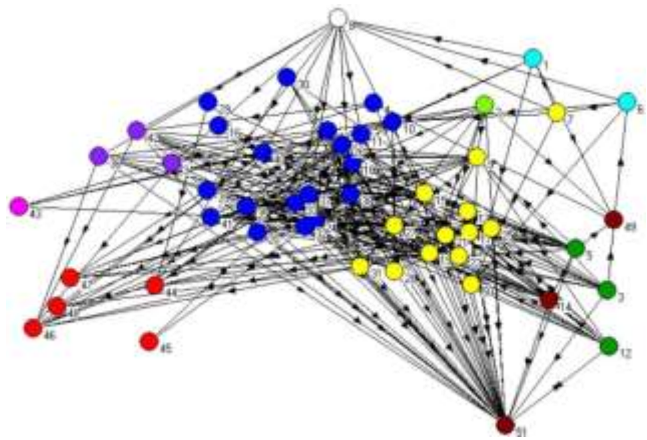
```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( “local”, “WordCount”,
sys.argv[0], None)
    lines = sc.textFile(sys.argv[1])

    lines.flatMap(lambda s: s.split(“ ”)) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda x, y: x + y) \
         .saveAsTextFile(sys.argv[2])
```

4 Graph Computing

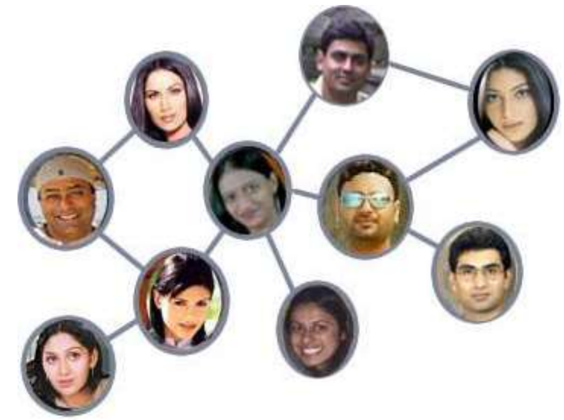
Graphs are very where



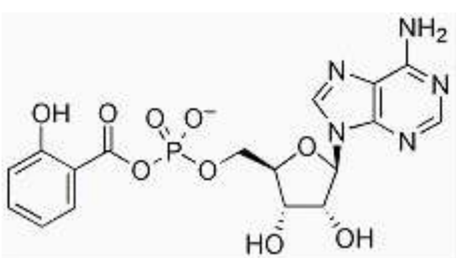
Biological Network



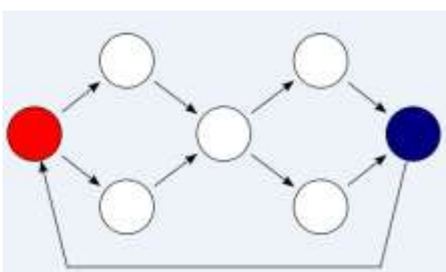
Ecological Network



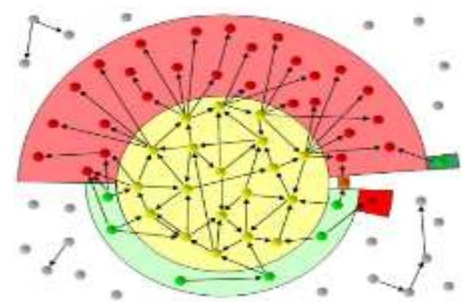
Social Network



Chemical Network



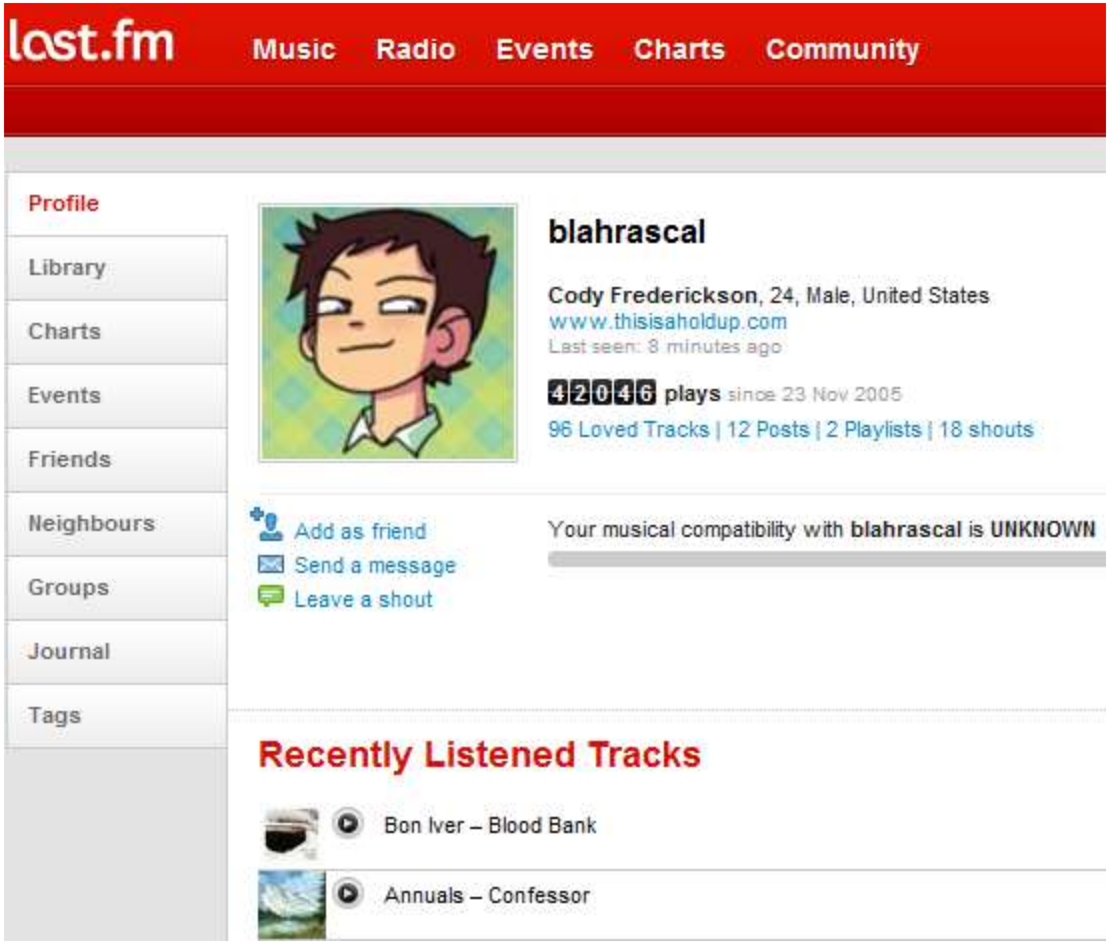
Program Flow



Web Graph

Complex Graphs

- Real-life graph contains complex contents – labels associated with nodes, edges and graphs.



Node Labels:

Location, Gender, Charts, Library, Events, Groups, Journal, Tags, Age, Tracks.

Large Graphs

	# of Users	# of Links
Facebook	***400 Million***	52K Million
Twitter	105 Million	10K Million
LinkedIn	60 Million	0.9K Million
Last.FM	40 Million	2K Million
LiveJournal	25 Million	2K Million
del.icio.us	5.3 Million	0.7K Million
DBLP	0.7 Million	8 Million

Thank you!