Unified Big Data Analytics Pipeline

连城 lian@databricks.com

What is Spark

- A fast and general engine for large-scale data processing
- An open source implementation of Resilient Distributed Datasets (RDD)
- Has an advanced DAG execution engine that supports cyclic data flow and *in-memory* computing

Why Spark

- Fast
  - Run machine learning like iterative programs up to 100x faster than Hadoop MapReduce in memory or 10x faster on disk
  - Run HiveQL compatible queries 100x faster than Hive (with Shark / Spark SQL)

Why Spark

- Compatibility
  - Compatible with most popular storage systems on top of HDFS
  - New users needn’t suffer ETL to deploy Spark

Why Spark

- Easy to use
  - Fluent Scala / Java / Python API
  - Interactive shell
  - 2-5x less code (than Hadoop MapReduce)

Why Spark

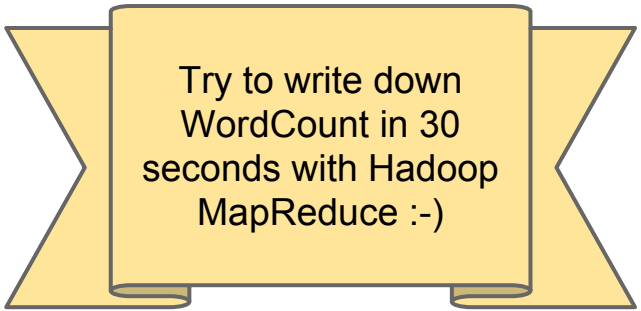
- Easy to use
  - Fluent Scala / Java / Python API
  - Interactive shell
  - 2-5x less code (than Hadoop MapReduce)

```
sc.textFile("hdfs://...")
  .flatMap(_ split " ")
  .map(_ -> 1)
  .reduceByKey(_ + _)
  .collectAsMap()
```

Why Spark

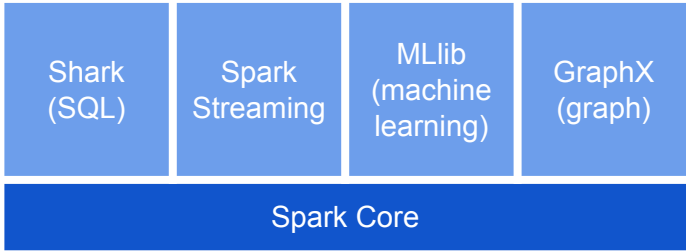
- Easy to use
  - Fluent Scala / Java / Python API
  - Interactive shell
  - 2-5x less code (than Hadoop MapReduce)

```
sc.textFile("hdfs://...")
  .flatMap(_ split " ")
  .map(_ -> 1)
  .reduceByKey(_ + _)
  .collectAsMap()
```

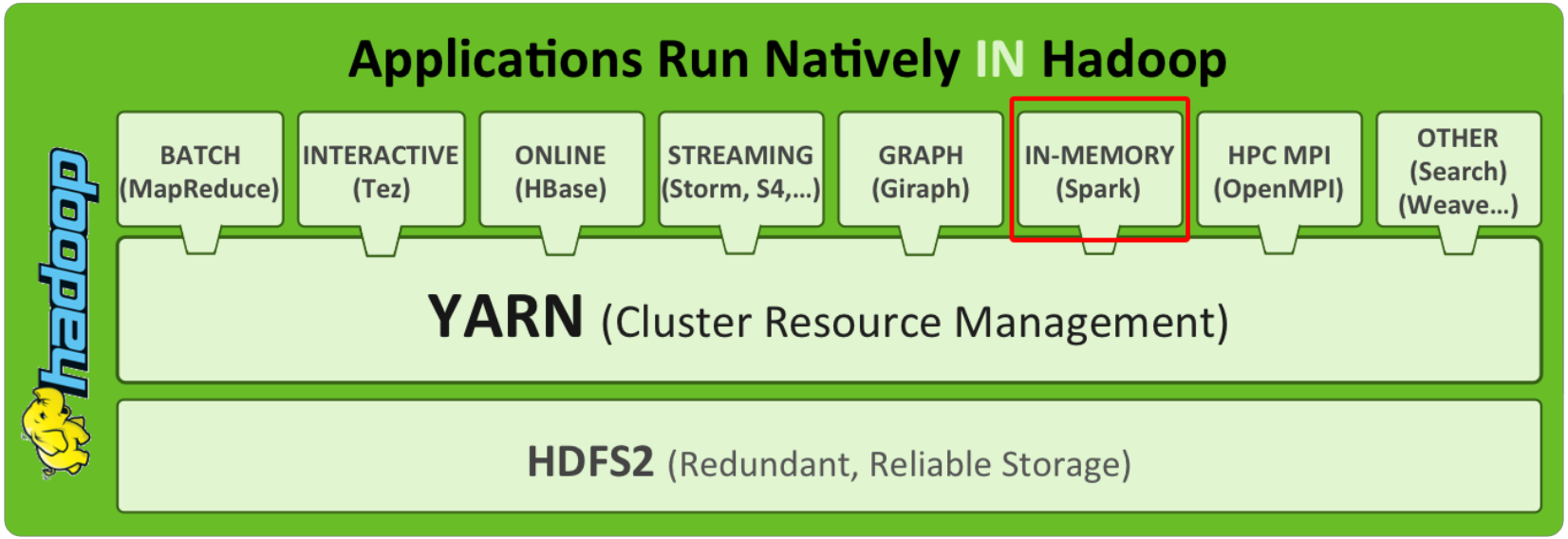


Why Spark

- Unified big data analytics pipeline for
  - Batch / interactive (Spark Core vs MR / Tez)
  - SQL (Shark / Spark SQL vs Hive)
  - Streaming (Spark Streaming vs Storm)
  - Machine learning (MLlib vs Mahout)
  - Graph (GraphX vs Giraph)



The Hadoop Family

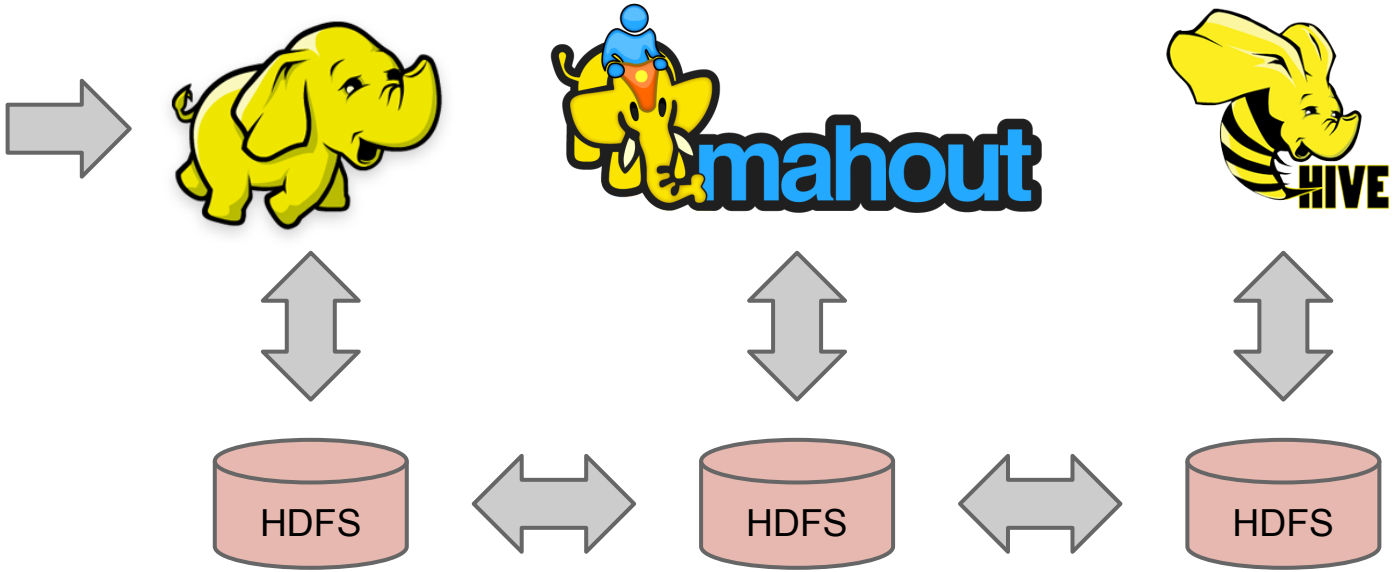


The Hadoop Family

General Batching	Specialized systems			
	Streaming	Iterative	Ad-hoc / SQL	Graph
MapReduce	Storm	Mahout	Pig	Giraph
	S4		Hive	
	Samza		Drill	
			Impala	

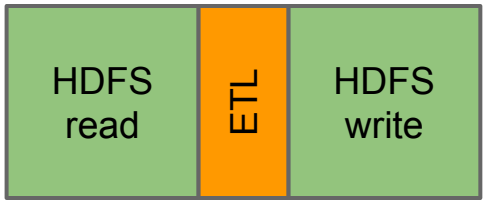
Big Data Analytics Pipeline Today

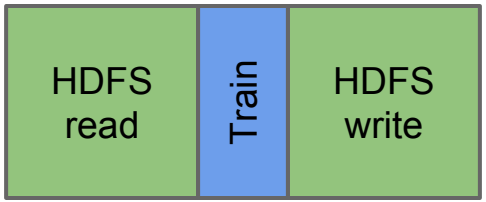
Using separate frameworks

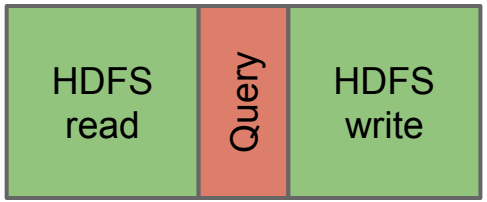


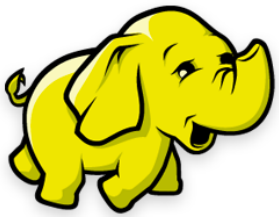
Big Data Analytics Pipeline Today

Using separate frameworks



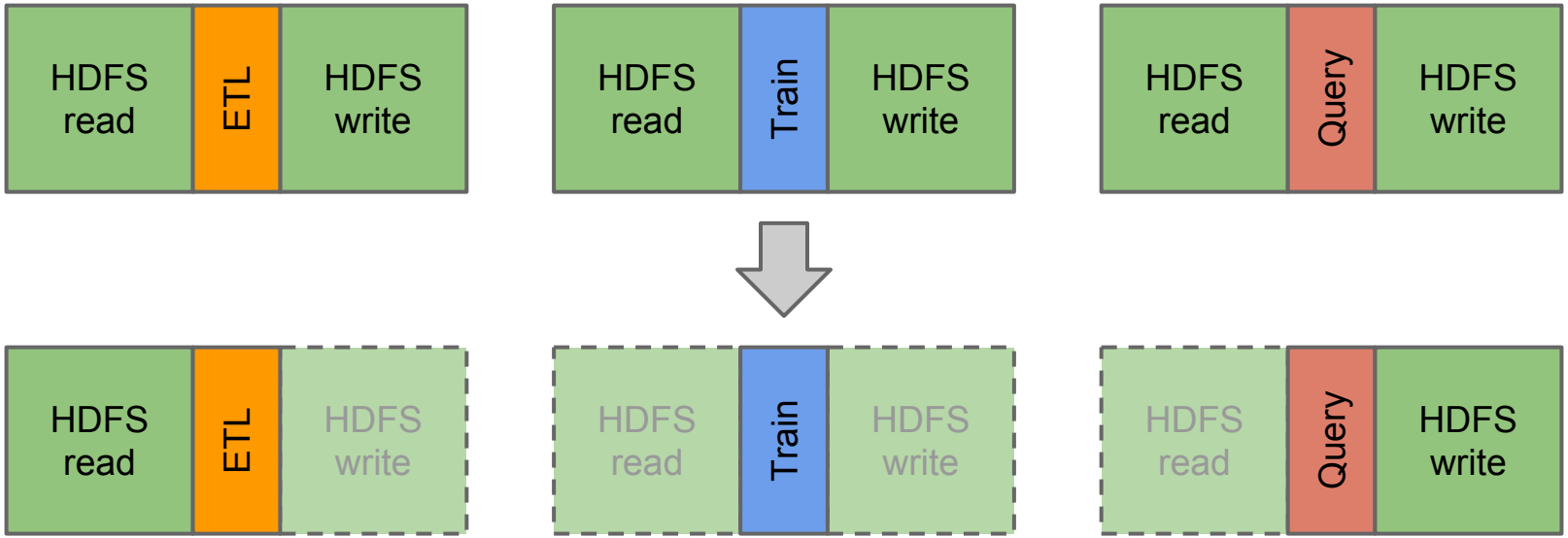






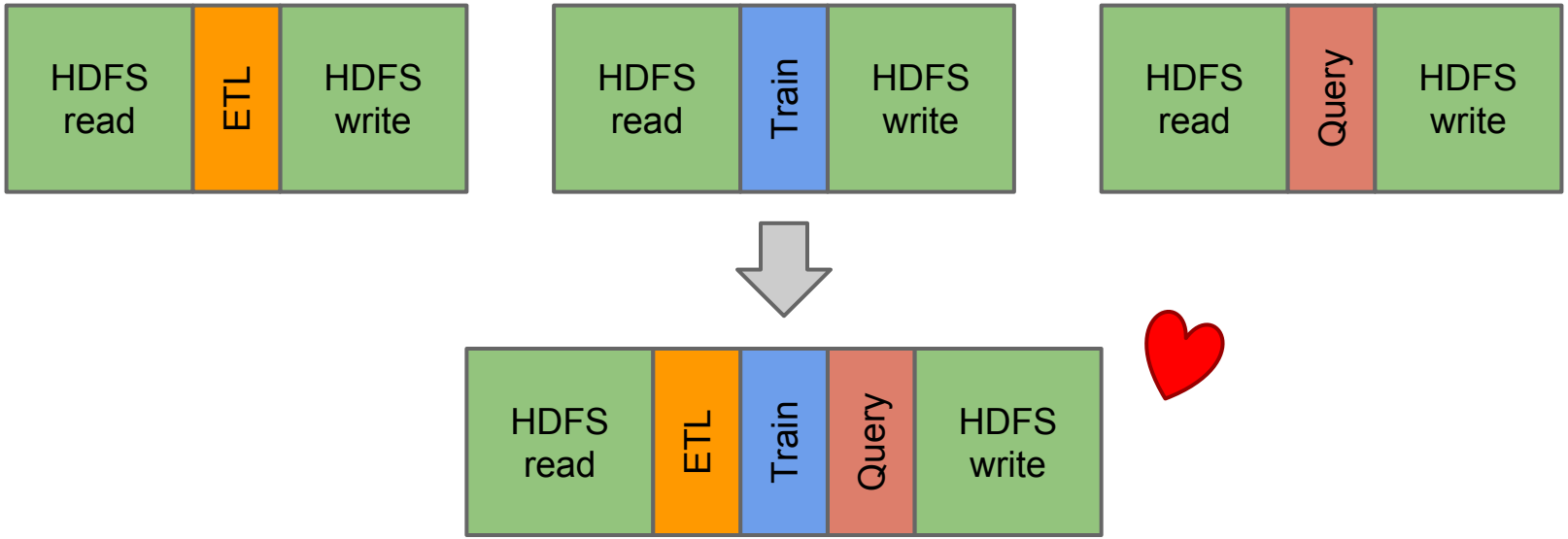
Big Data Analytics Pipeline Today

Using separate frameworks



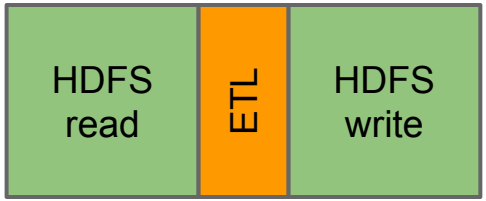
Big Data Analytics Pipeline Today

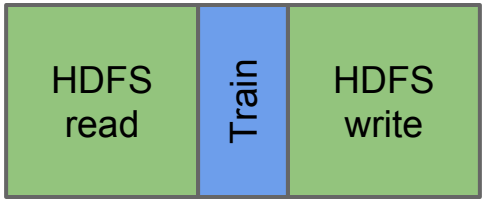
Using separate frameworks

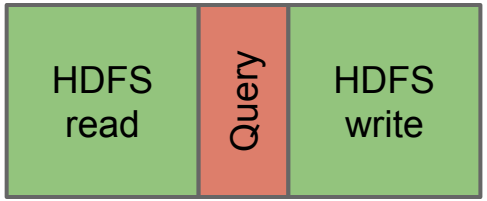


Big Data Analytics Pipeline Today

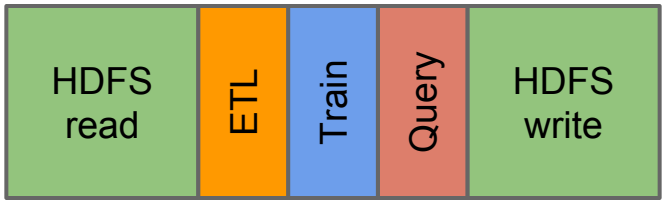
Using separate frameworks







How?




Ahhh!!!
DATA HAS
GRAVITY

Resilient Distributed Datasets

- RDDs
  - are *read-only*, *partitioned* collections of records
  - can be cached *in-memory* and are *locality aware*
  - can only be created through *deterministic operations* on either (1) *data in stable storage*, or (2) *other RDDs*

Resilient Distributed Datasets

- Core data sharing abstraction in Spark
- Computation can be represented by lazily evaluated RDD lineage DAG(s)

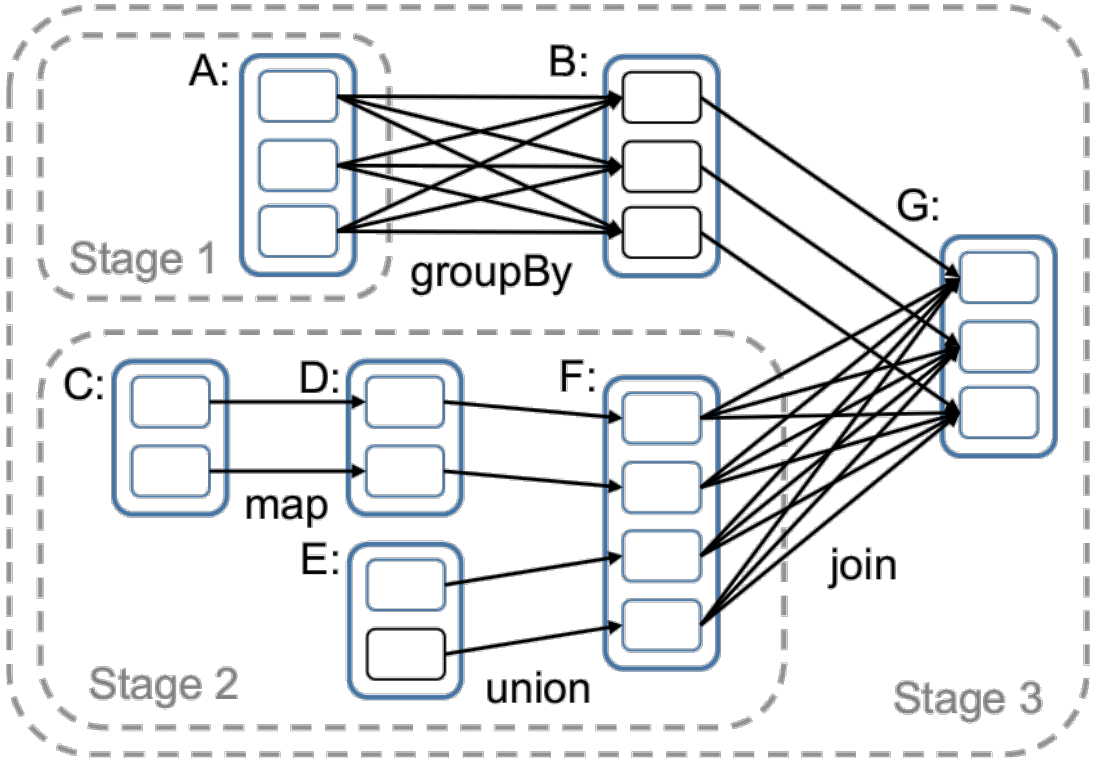
Resilient Distributed Datasets

- An RDD
  - is a read-only, partitioned, locality aware distributed collection of records
  - either points to a stable data source, or
  - is created through deterministic transformations on some other RDD(s)

Resilient Distributed Datasets

- Coarse grained
  - Applies the same operation on many records
- Fault tolerant
  - Failed tasks can be recovered in parallel with the help of the lineage DAG

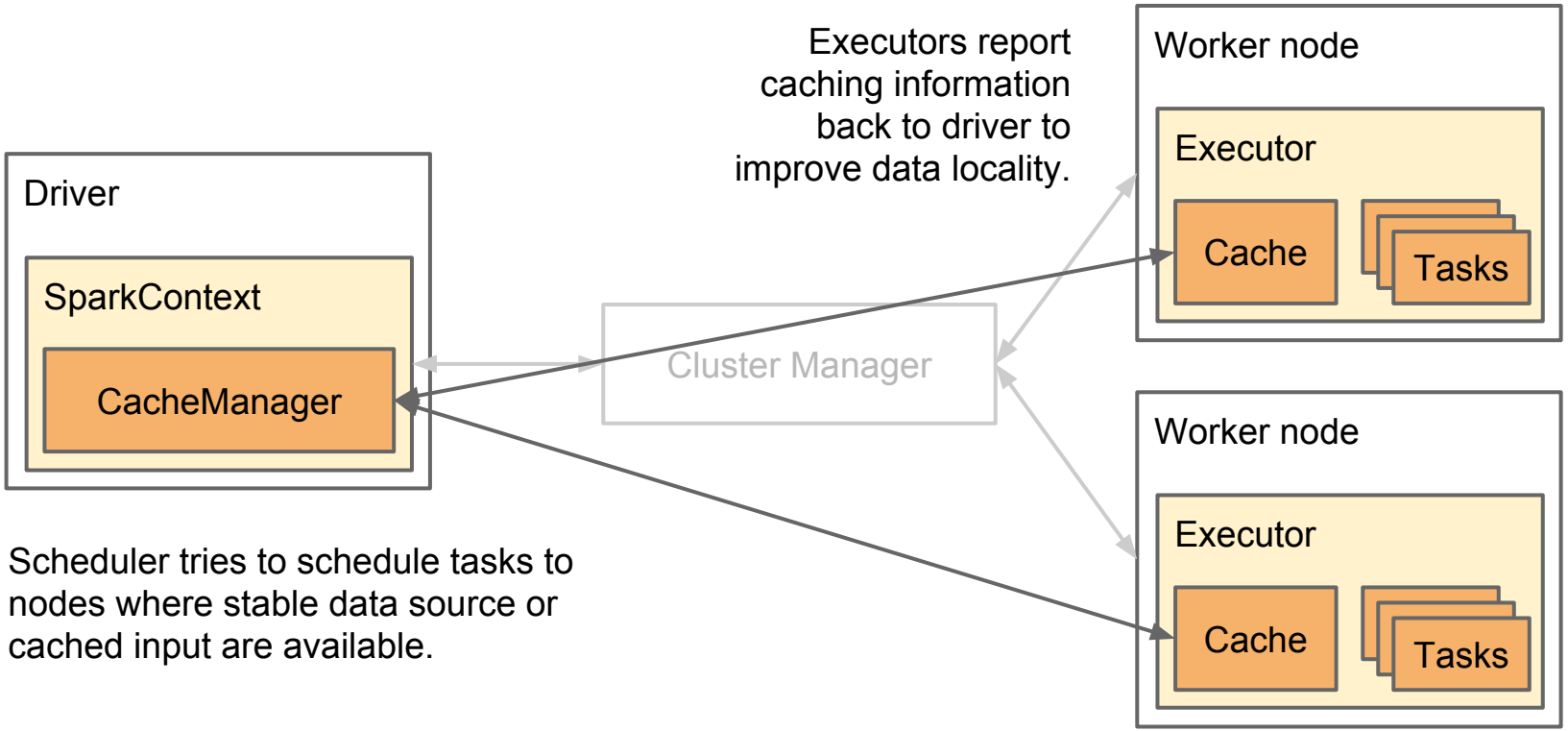
Resilient Distributed Datasets



Data Sharing in Spark

- Frequently accessed RDDs can be materialized and cached in memory
  - Cached RDD can also be replicated for fault tolerance
- Spark scheduler takes cached data locality into account

Data Sharing in Spark



Data Sharing in Spark

- As a consequence, intermediate results are
  - either cached *in memory*, or
  - written to local file system as shuffle map output and waiting to be fetched by reducer tasks (similar to MapReduce)
- Enables in-memory data sharing, avoids unnecessary HDFS I/O

Generality of RDDs

- From an expressiveness point of view
  - The coarse grained nature is not a big limitation since many parallel programs naturally *apply the same operation to many records*, making them easy to express
  - In fact, RDDs can emulate *any* distributed system, and will do so *efficiently* in many cases unless the system is sensitive to network latency

Generality of RDDs

- From system perspective
  - RDDs gave applications control over the most common bottleneck resources in cluster applications (i.e., network and storage I/O)
  - Makes it possible to express the same optimizations that specialized systems have and achieve similar performance

Limitations of RDDs

- Not suitable for
  - Latency (millisecond scale)
  - Communication patterns beyond all-to-all
  - Asynchrony
  - Fine-grained updates

Analytics Models Built over RDDs

MR / Dataflow

```
spark
  .textFile("hdfs://...")
  .flatMap(_ split " ")
  .map(_ -> 1)
  .reduceByKey(_ + _)
  .collectAsMap()
```

Streaming

```
val streaming = new StreamingContext(
  spark, Seconds(1))
streaming.socketTextStream(host, port)
  .flatMap(_ split " ")
  .map(_ -> 1)
  .reduceByKey(_ + _)
  .print()
streaming.start()
```

Analytics Models Built over RDDs

SQL

```
val hiveContext = new HiveContext(spark)
import hiveContext._
val children = hiveql(
  """SELECT name, age FROM people
    |WHERE age < 13
  """.striMargin).map {
    case Row(name: String, age: Int) =>
      name -> age
  }.foreach(println)
```

Machine Learning

```
val points = spark.textFile("hdfs://...")
  .map(_.split.map(_.toDouble)).splitAt(1)
  .map { case (Array(label), features) =>
    LabeledPoint(label, features)
  }
val model = KMeans.train(points)
```

Mixing Different Analytics Models

MapReduce & SQL

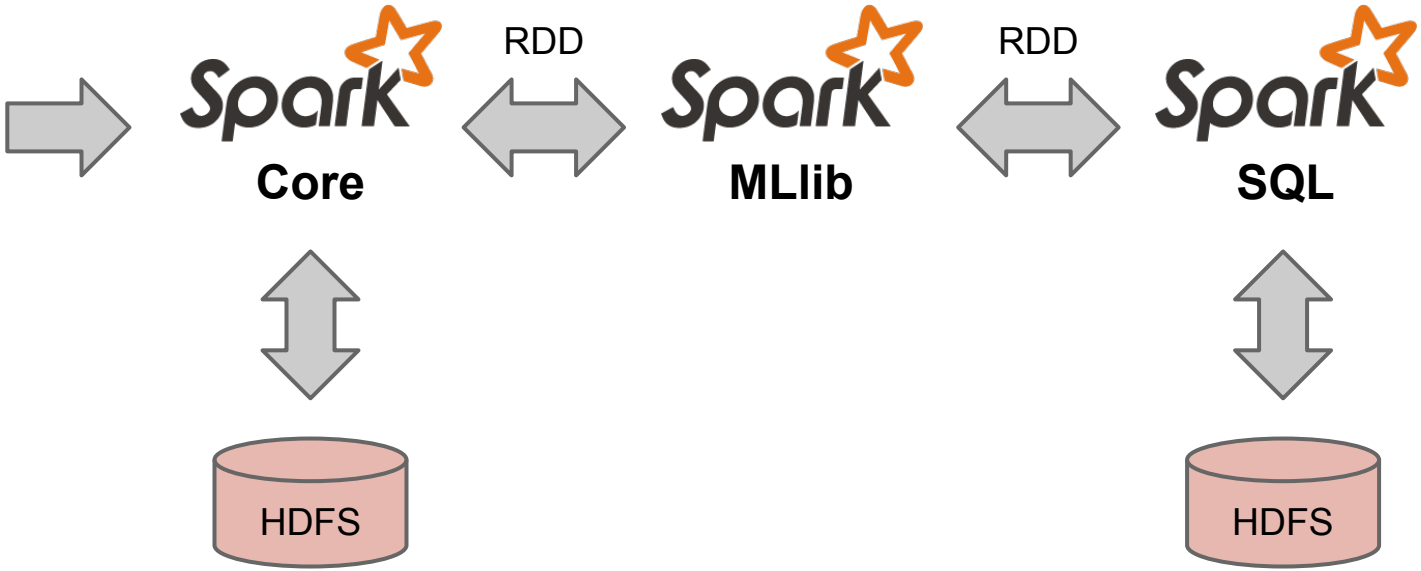
```
// ETL with Spark Core
case class Person(name: String, age: Int)
val people = spark.textFile("hdfs://people.txt").map { line =>
  val Array(name, age) = line.split(",")
  Person(name, age.trim.toInt)
}.registerAsTable("people")
// Query with Spark SQL
val teenagers = sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
teenagers.collect().foreach(println)
```

Mixing Different Analytics Models

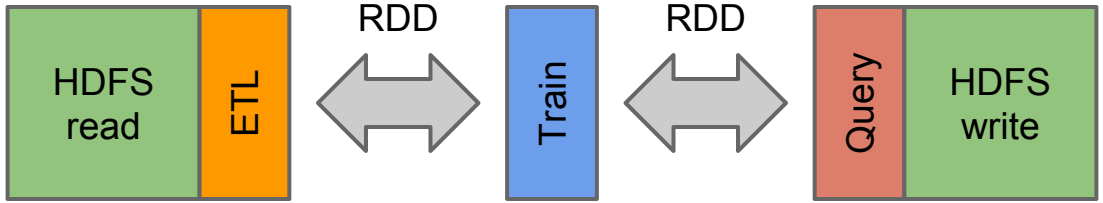
SQL & iterative computation (machine learning)

```
// ETL with Spark SQL & Spark Core
val trainingData = sql("""SELECT e.action, u.age, u.latitude, u.longitude
                          |FROM Users u JOIN Events e ON u.userId = e.userId
                        """.stripMargin).map {
  case Row(_, age: Double, latitude: Double, longitude: Double) =>
    val features = Array(age, latitude, longitude)
    LabeledPoint(age, features)
}
// Training with MLlib
val model = new LogisticRegressionWithSGD().run(trainingData)
```

Unified Big Data Analytics Pipeline



Unified Big Data Analytics Pipeline

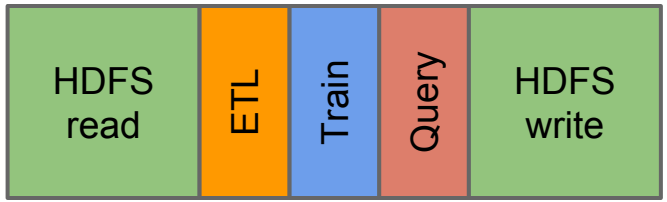


Spark Core

Spark MLlib

Spark SQL

Unified Big Data Analytics Pipeline

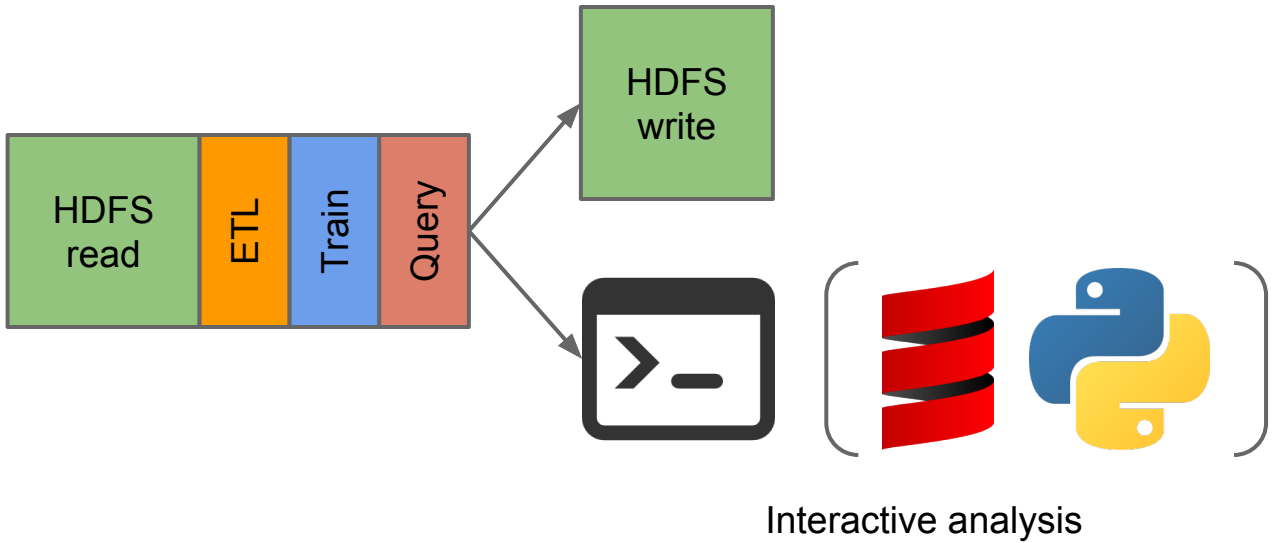


Spark Core

Spark MLlib

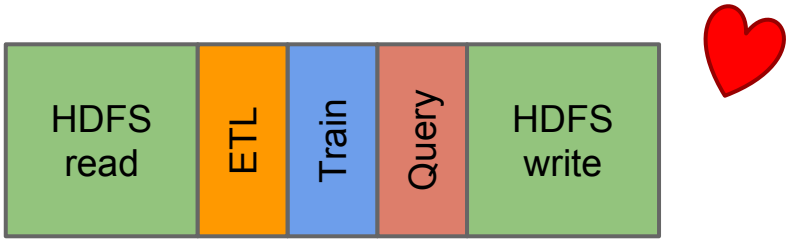
Spark SQL

Unified Big Data Analytics Pipeline



Unified Big Data Analytics Pipeline

- Unified in-memory data sharing abstraction
- Efficient runtime brings significant performance boost and enables interactive analysis
- “One stack to rule them all”



Thanks!

Q&A