Apache Spark

In pioneer days they used oxen for heavy pulling, and when one ox couldn’t budge a log, they didn’t try to grow a larger ox.

Presenter

M. Fanilo ANDRIANASOLO

- Business Center BI & Big Data @Worldline Lyon
- Technical advisor, Big Data engineer & scientist

Tech skills

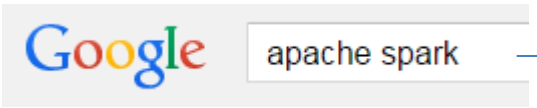
- Large scale Data Analytics – Hadoop ecosystem
- Appetite for linear algebra & Machine Learning

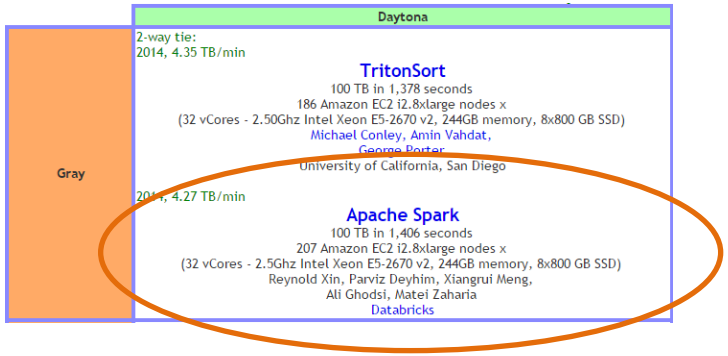
Passions

- Combine works from different data people
- Internships in Big Data workflows
- Apache Spark workshops
- Badminton player

Fanilo.Andrianasolo@univ-lyon2.fr @andfanilo

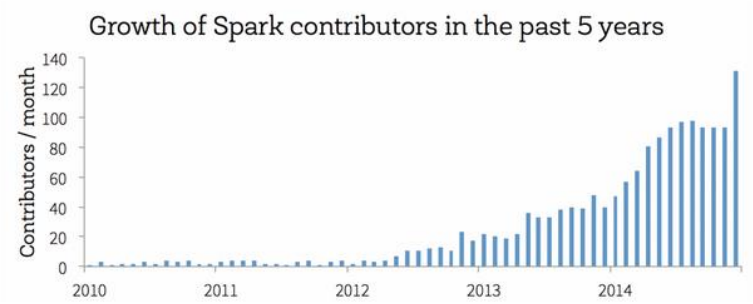
Is Spark that famous anyway ? (2015)





Daytona GraySort record

100 TB in 1,406s



Most active Big Data open source project

Based on Survey of More Than 2,100 Respondents, 13% Already Using Spark in Production, 20% Planning to Use in 2015, and 31% "Evaluating"

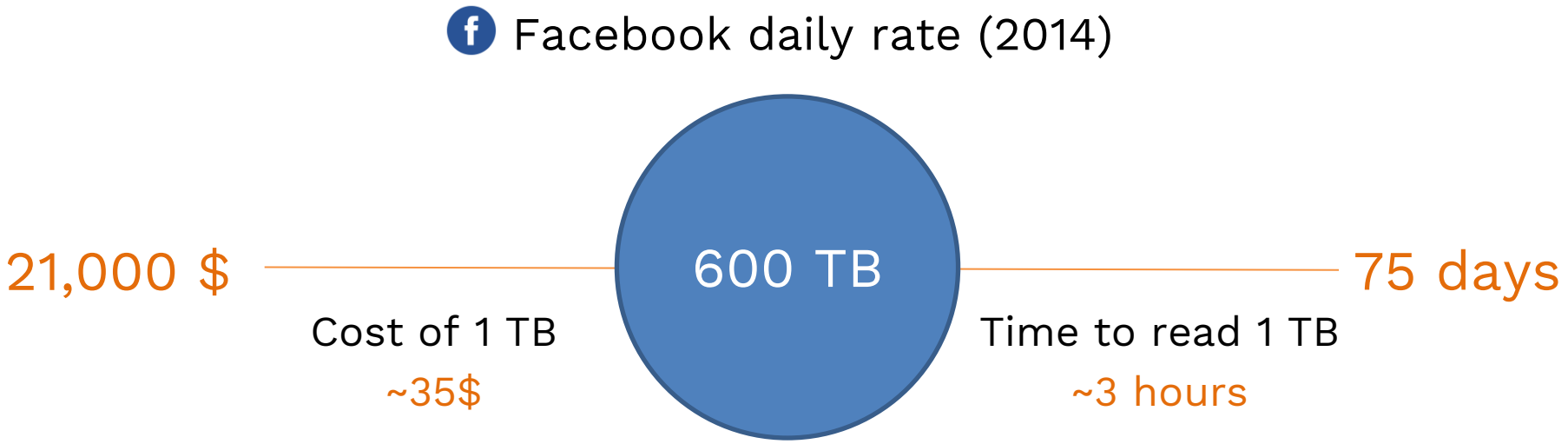
Production-ready and widely adopted

Agenda

- Prerequisites
- What is Spark designed for ?
- Through each Spark component
- Hands-on

Prerequisites

Data storage is growing faster than computation speeds



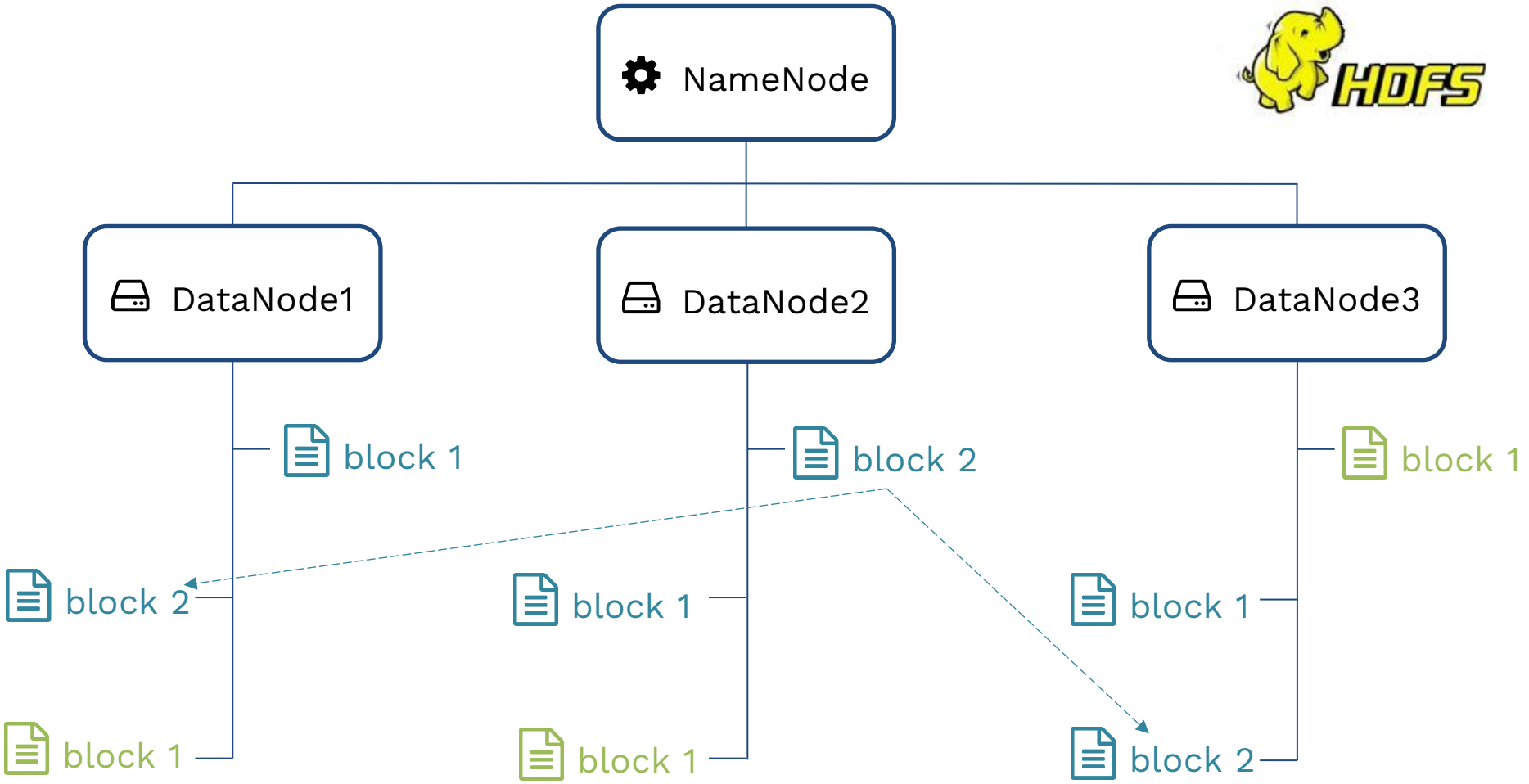
More and more organizations have to scale out their computations across clusters of machines to get “faster” results

Prerequisites

Cluster computing **is painful,** so we like when people design them for us like MapReduce

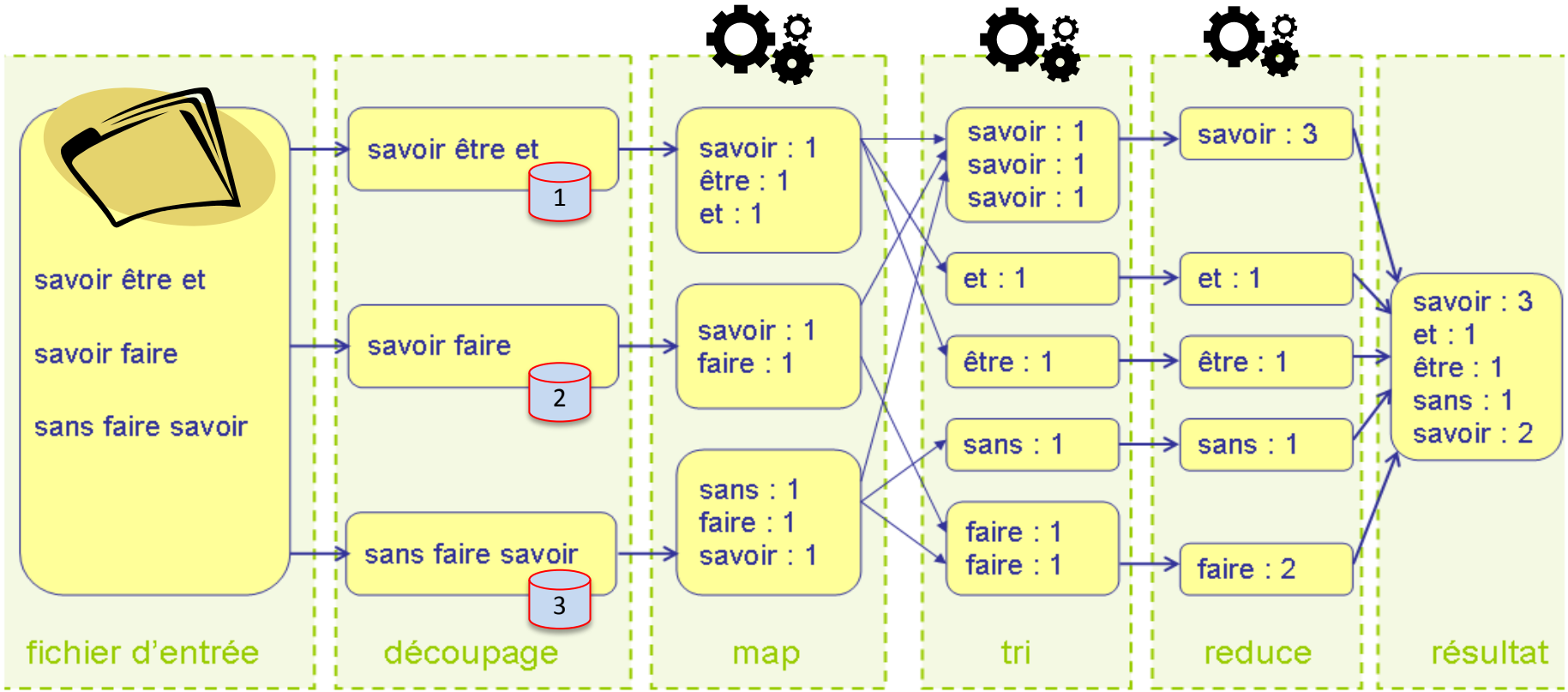
- Parallelism: is it possible to easily divide dog food equally into multiple bowls ?
- Fault-tolerance: so my bowl just broke and the food smells bad and I'm still hungry, but I have no idea where to go
- Multitenancy: HEY ! STOP EATING IN MY BOWL !

Prerequisites : Hadoop Distributed FS



Hadoop Distributed File System (HDFS) is a scalable distributed file system for large, distributed, data-intensive applications.

Prerequisites : MapReduce

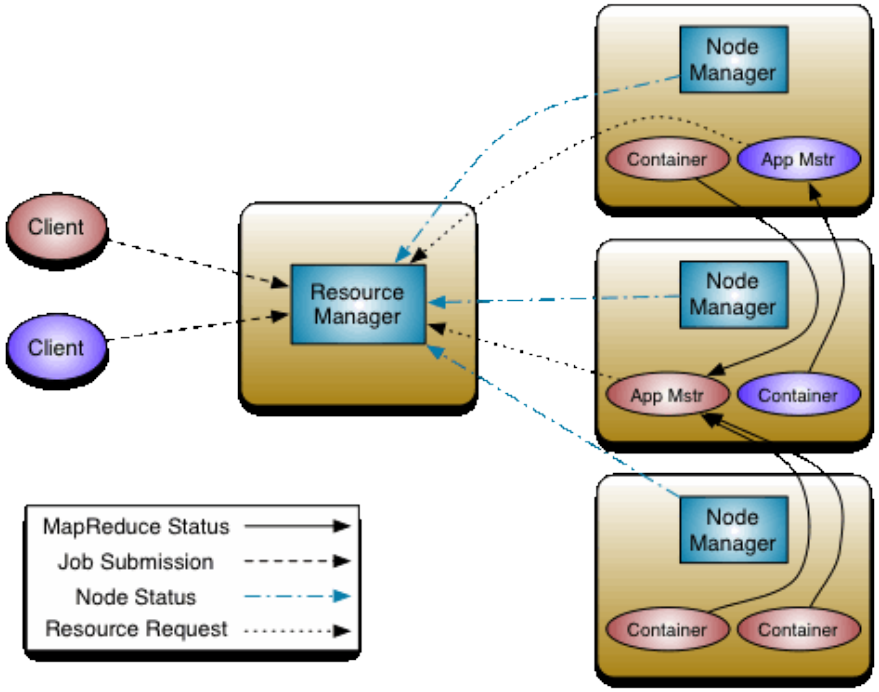


Data locality : Moving Computation is Cheaper than Moving Data

Source : http://blog.inovia-conseil.fr/?p=46

Prerequisites : YARN

YARN provides APIs for requesting and working with cluster resources



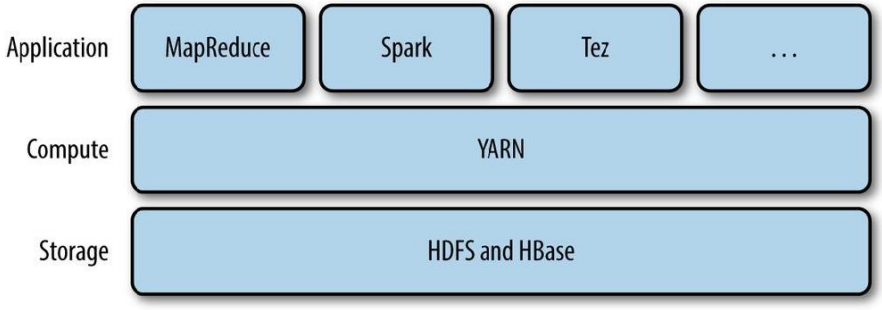


Figure 4-1. YARN applications

YARN hides the resource management details from the user to facilitate the management of parallel applications.

Prerequisites

MapReduce doesn't behave "well" for other parallel workloads than batch processing.

A wide range of specialized models were created, each solving a specific problem in a distributed fashion.

Batch

Iterative

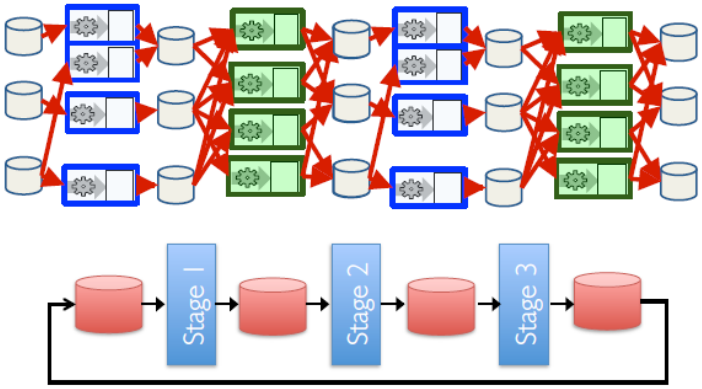
Streaming

Interactive

It is expensive and unwieldy to compose computations in different systems where intermediate datasets are "large".

What is Spark designed for ?

All those workflows require some form of data sharing between jobs.



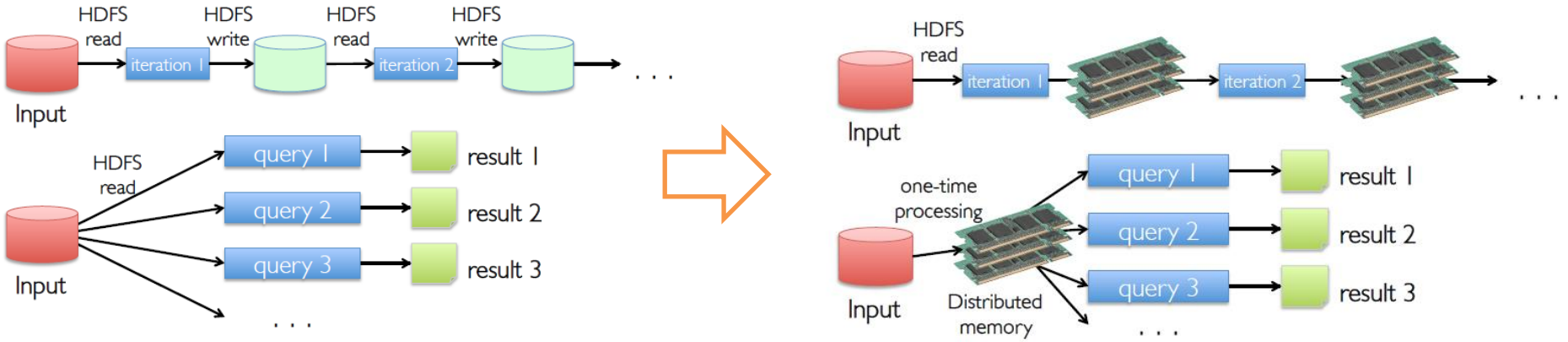
In MapReduce, each stage passes through the hard drives
→ lots of Disk I/O

Streaming, interactive and iterative require multiple passes on the same data
→ lots of Disk I/O

What if we share data across stages with Memory instead of Disk ?

What is Spark designed for ?

Let's manage an efficient data sharing abstraction across parallel computation stages with Memory instead of Disk



10-100x faster

...oh wait, also we can mix streaming, machine learning and batch processing all-in-one, which is the original goal

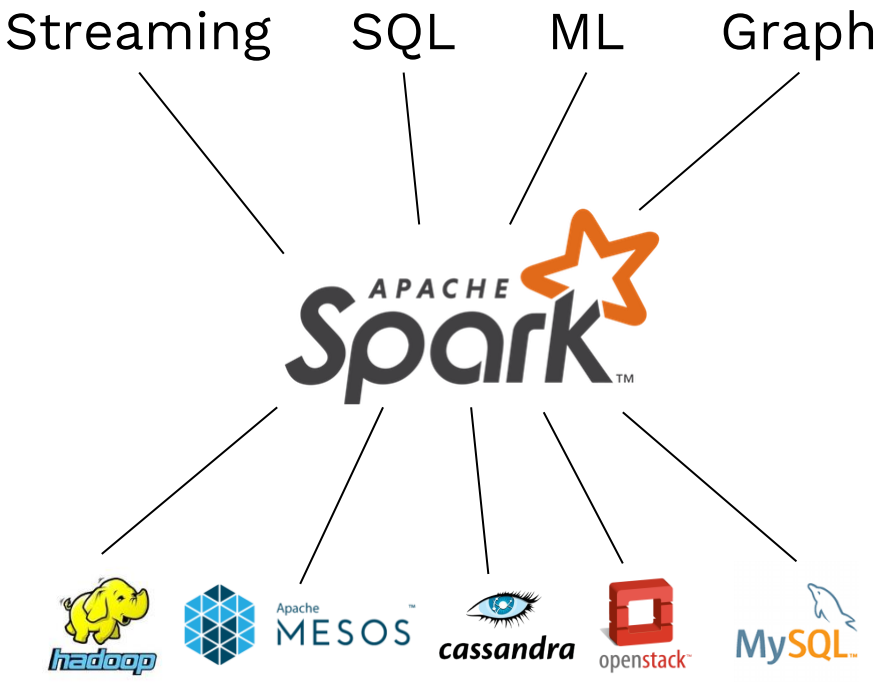
What is Spark designed for ?

Design a unified programming abstraction that capture these disparate workloads and enables new applications.

1. **Unified engine**
   Support end-to-end applications

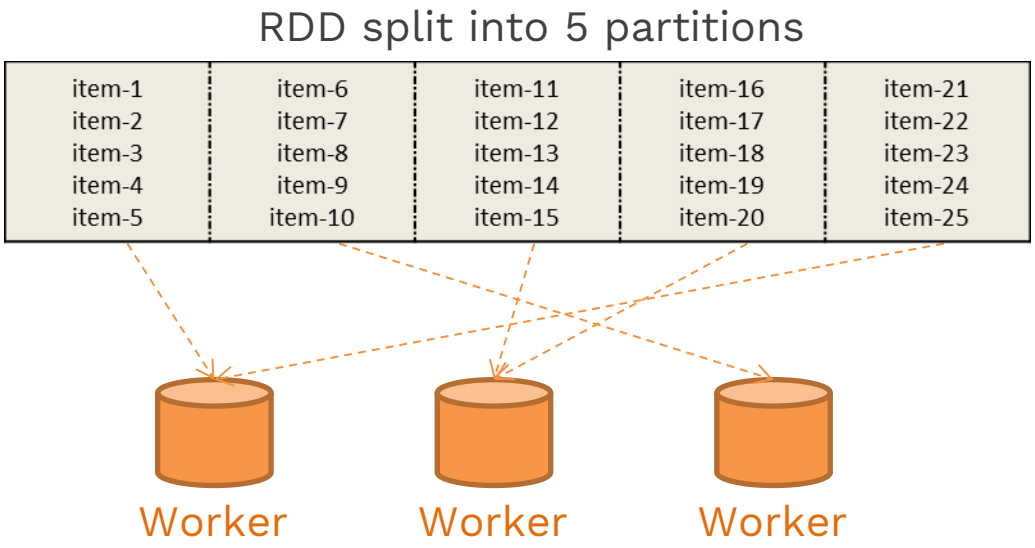
2. **High-level APIs**
   Easy to use, rich optimizations

3. **Integrate broadly**
   Storage systems, libraries, etc..



Primary abstraction - RDDs

A Resilient Distributed Dataset (RDD) is a partitioned collection of objects spread across a cluster, stored in memory or on disk.



Number of partitions is defined by the programmer.

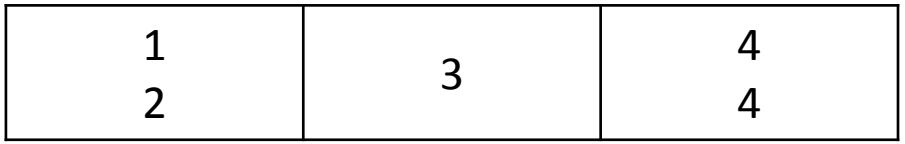
More partitions = more parallelism

Spark operations

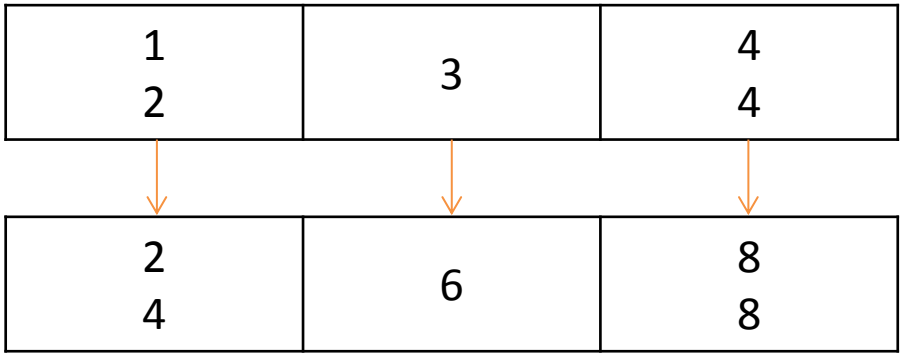
Transformations

Create a new dataset from an existing one

```
>> rdd = sc.parallelize([1, 2, 3, 4, 4])
```




```
>> rdd.map(lambda x: x * 2)
```

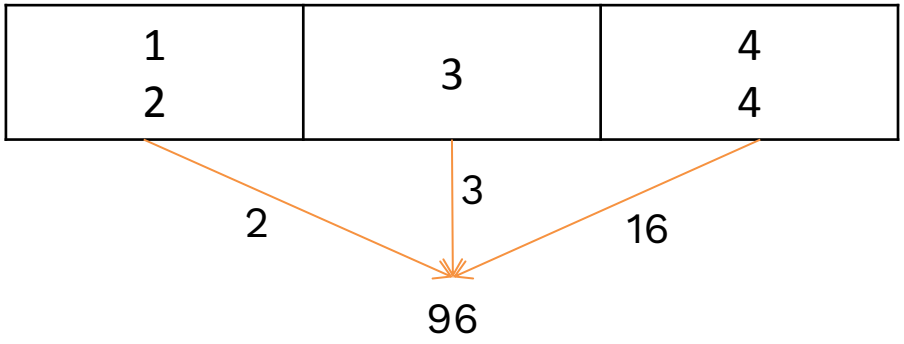


Actions

Return a value after running a computation

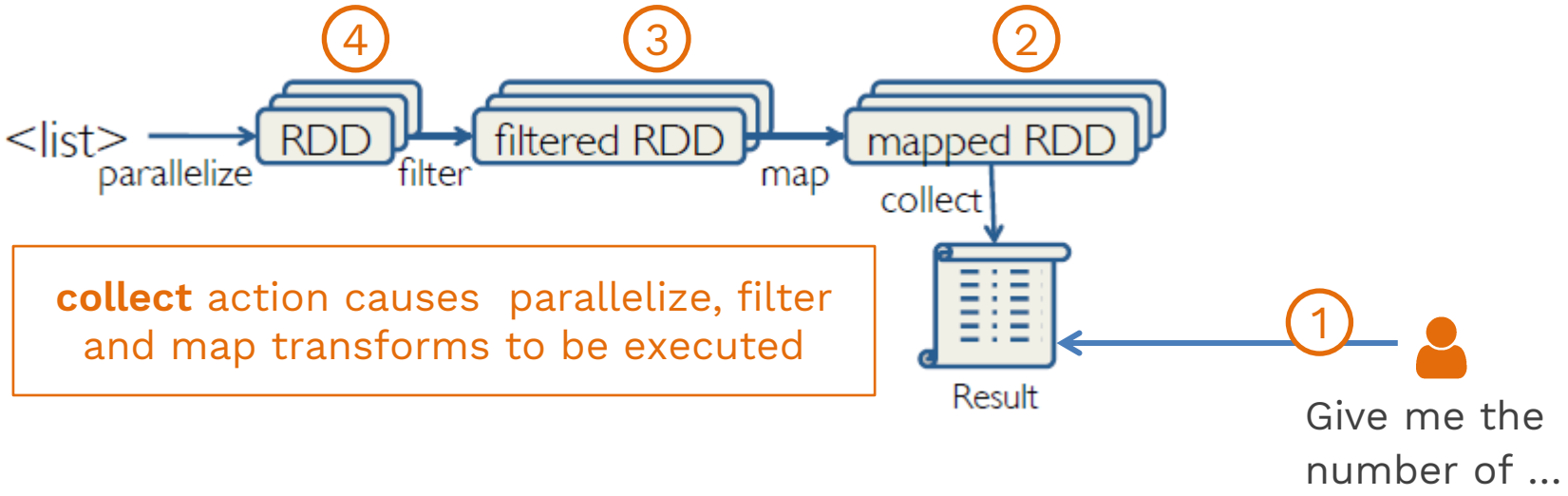
```
>> rdd = sc.parallelize([1, 2, 3, 4, 4])
```


```
>> rdd.reduce(lambda a,b : a * b)
```



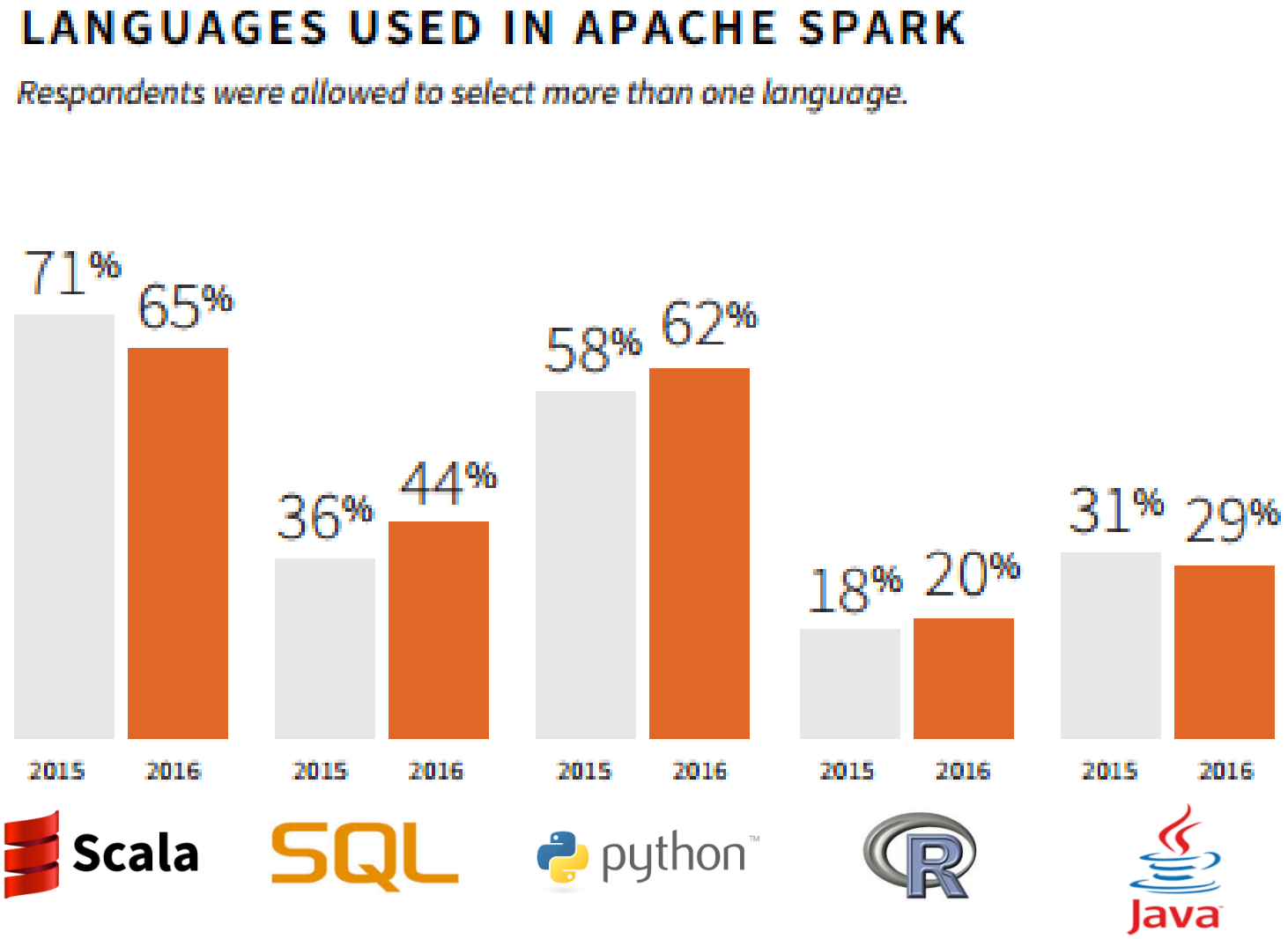
Spark operations

Lazy evaluation: Spark remembers the set of transformations applied to the base dataset, they are not computed right away.



- Optimization : better placement of tasks on the cluster, caching between processing stages for data sharing
- Fault-tolerance : Spark recovers more easily from failures and slow workers

Multi language API



Back to Spark major components

SQL : Structured programming

Mllib : Machine Learning

Streaming : near real-time processing

GraphX : graph processing

Structured programming

Spark processing results in the transformation of previous RDDs by an opaque compute function on opaque data, which prevents any optimization by the engine.

Data in RDDs and functions in transformations are not known of the Spark engine

- Memory management is hard
- Order of transformations can't be optimized

Structured programming

We limit the space of what can be expressed by **structuring** the data and transformations to enable optimization.

```
df.printSchema()
```

```
root
 |-- policyID: integer (nullable = true)
 |-- statecode: string (nullable = true)
 |-- county: string (nullable = true)
```

```
df.select("policyID", "statecode", "county").show(3)
```

```
+--------+---------+-----------+
|policyID|statecode|     county|
+--------+---------+-----------+
|  119736|       FL|CLAY COUNTY|
|  448094|       FL|CLAY COUNTY|
|  206893|       FL|CLAY COUNTY|
+--------+---------+-----------+
```

Datasets are SparkSQL’s strongly-typed, immutable collection of objects mapped to a relational schema.

Structured programming

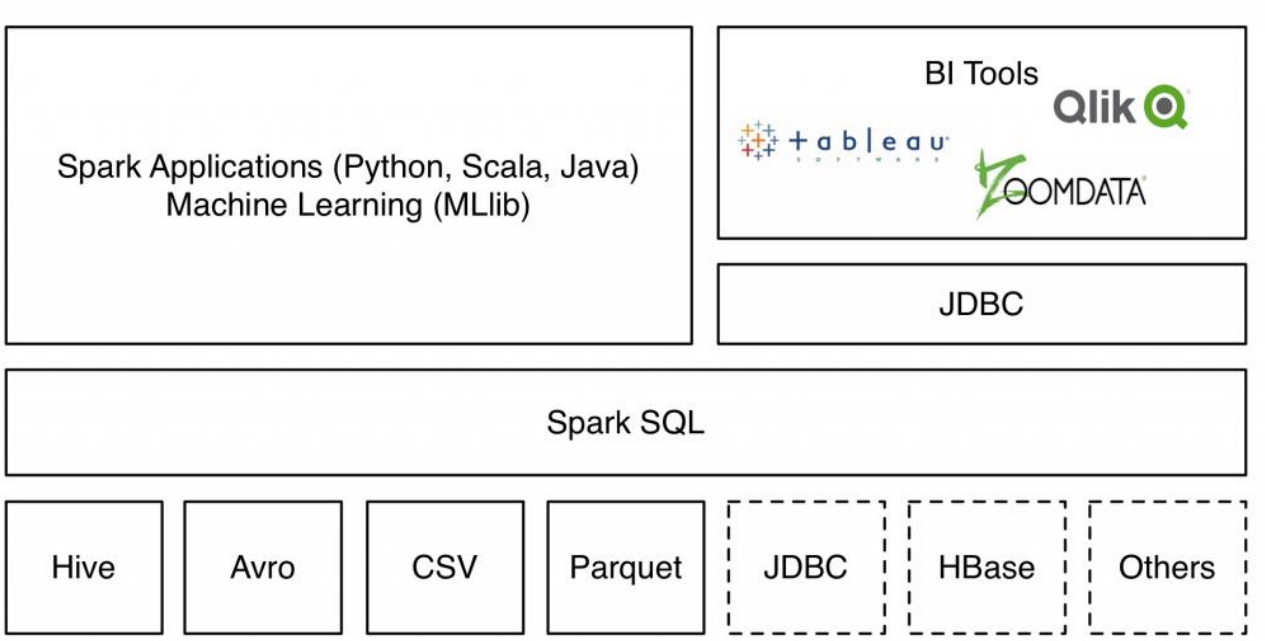
You can interact with Datasets through SQL language, or using a functional API close to the RDD API (with map, filter functions…)

```
// Register the DataFrame as a global temporary view
df.createGlobalTempView("people")

// Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.people").show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

```
// Select everybody, but increment the age by 1
df.select($"name", $"age" + 1).show()
// +-------+---------+
// |   name|(age + 1)|
// +-------+---------+
// |Michael|     null|
// |   Andy|       31|
// | Justin|       20|
// +-------+---------+
```

Structured programming

The Data Sources API provides a pluggable mechanism for accessing structured data though Spark SQL.



Data sources API provide a tight optimization integration so filtering and column pruning can be pushed all the way down to the data source in many cases.

Structured programming

Datasets offer convenience of RDDs with the performance optimizations of Spark SQL and the strong static type-safety of Scala.

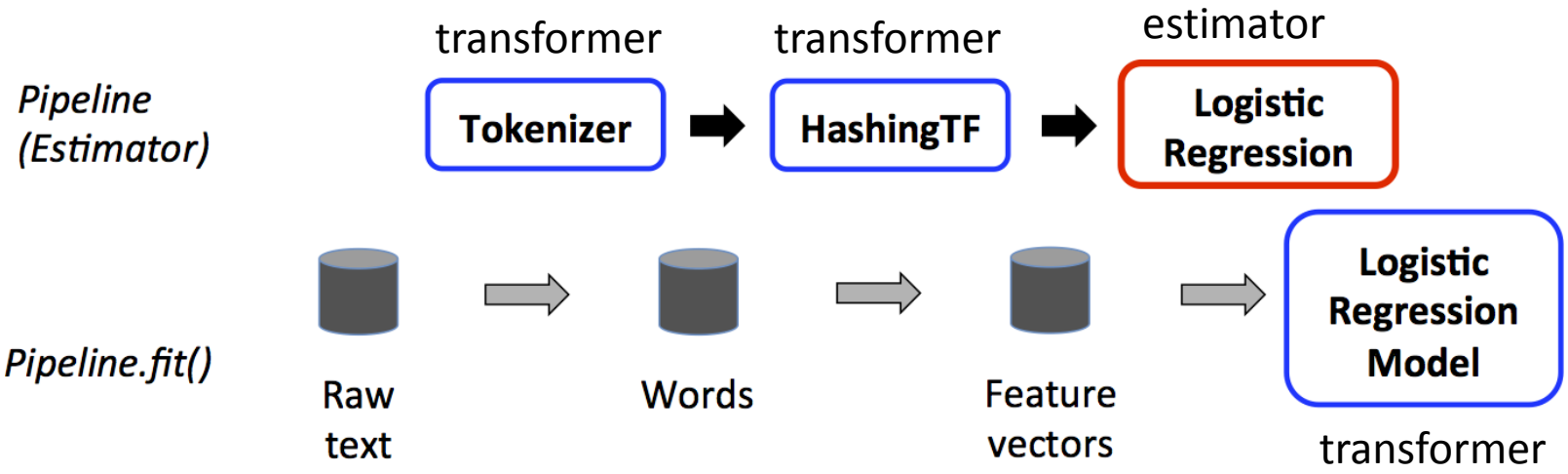
```scala
val df = ctx.read.json("people.json")

// Convert data to domain objects.
case class Person(name: String, age: Int)
val ds: Dataset[Person] = df.as[Person]
ds.filter(_.age > 30)
```

```scala
// Compute histogram of age by name.
val hist = ds.groupBy(_.name).mapGroups {
  case (name, people: Iter[Person]) =>
    val buckets = new Array[Int](10)
    people.map(_.age).foreach { a =>
      buckets(a / 10) += 1
    }
    (name, buckets)
}
```

With Spark 2.0, SparkSQL is the **preferred** primary and feature rich interface for Spark users.

Machine Learning with Spark

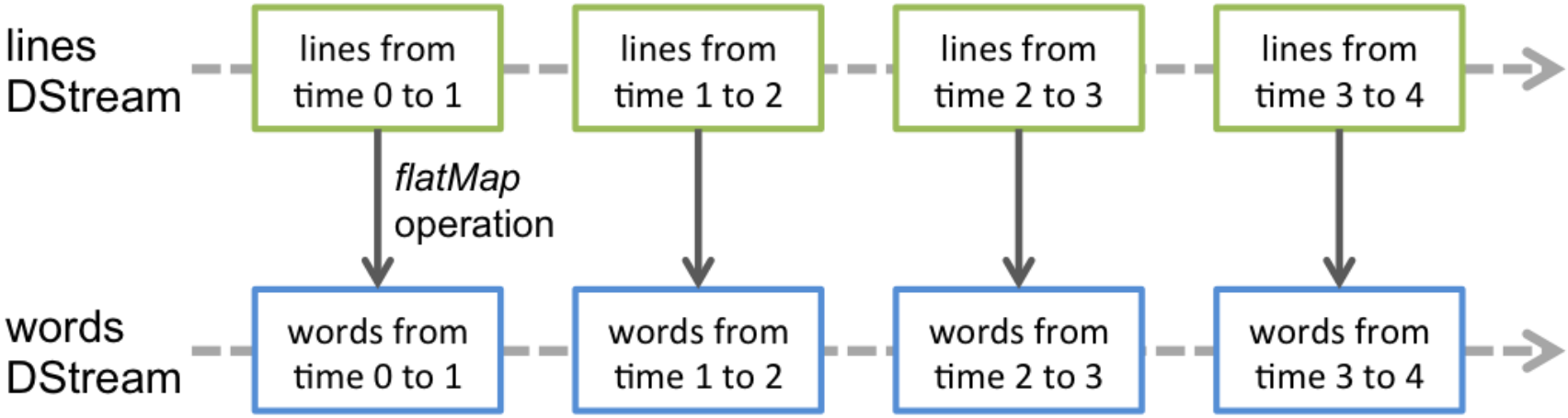
Mllib is Apache Spark's extension to provide distributed machine learning algorithms on top of Spark's abstraction.



1. **Pipelines** : tools for constructing, evaluating and tuning
2. **Featurization** : feature extraction, transformation, dimensionality reduction, selection
3. **ML algorithms** : common classification, regression, clustering and collaborative filtering
4. **Persistence** : saving and loading algorithms, models and Pipelines

Streaming

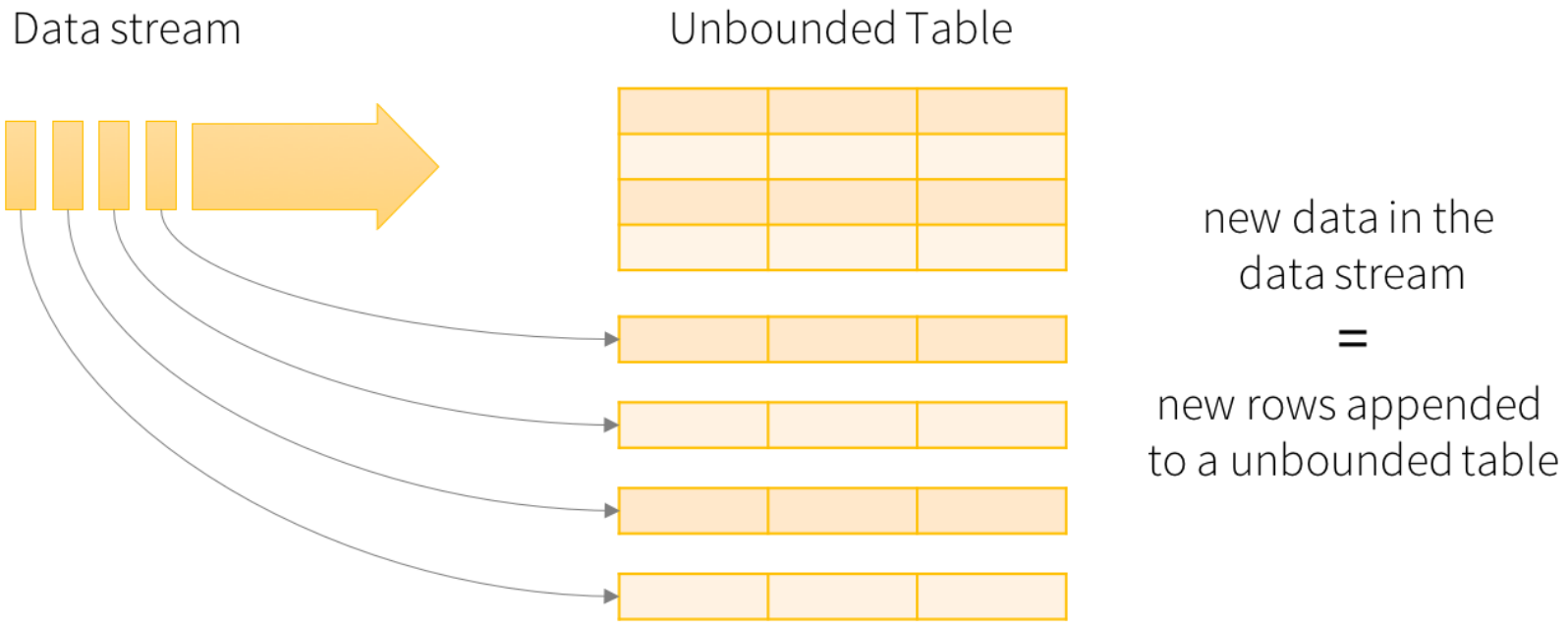
A discretized stream (DStream) represents series of stateless, deterministic batch computations on small time intervals, and is modelled as a continuous stream of RDDs.



DStreams support many of the transformations and properties available on normal Spark RDD's.

Structured Streaming

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.

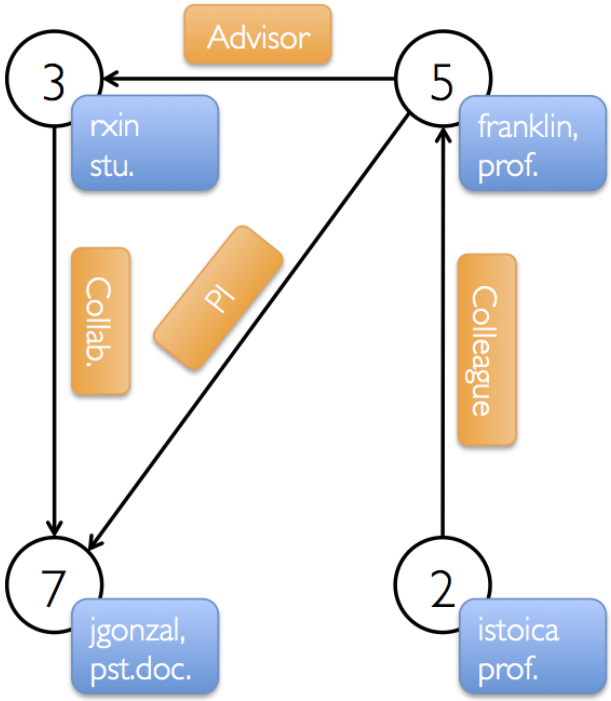


Data stream as an unbounded table

GraphX: graph computing

GraphX represents a Graph abstraction with RDDs, as a directed multigraph with properties attached to each vertex and edge.

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Unified engine

Spark's main contribution is to enable previously disparate cluster workloads to be composed.

As cluster applications grow in complexity, they will need to combine different types of computation (e.g., machine learning and SQL) and processing modes (e.g., interactive and streaming).

MLlib

Core

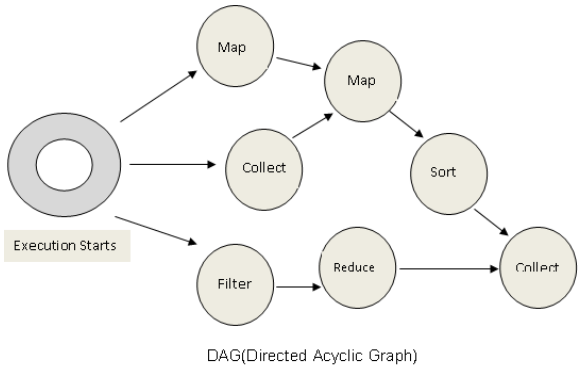
Spark Streaming

```
val predictions = model.predictOn(trainingData)

predictions.foreachRDD { rdd =>
  val modelString = model.latestModel().clusterCenters
    .map(c => c.toString.slice(1, c.toString.length-1)).mkString("\n")
  val predictString = rdd.map(p => p.toString).collect().mkString("\n")
  val dateString = Calendar.getInstance().getTime.toString.replace(" ", "-")
  Utils.printToFile(outputDir, dateString + "-model", modelString)
  Utils.printToFile(outputDir, dateString + "-predictions", predictString)
}

ssc.start()
ssc.awaitTermination()
```

Conclusion

Spark maintains MapReduce's linear scalability and fault tolerance, but extends it in a few important ways:

Can execute a more general computation graph

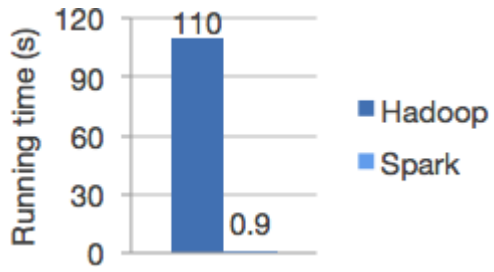


DAG(Directed Acyclic Graph)

Provides a rich set of transformations

Map	Take
Filter	GroupByKey
Join	Sample
Intersect	Cartesian
Union	First
Reduce	CountByKey
Count	Distinct
SaveAs	Cogroup
SortByKey	Repartition

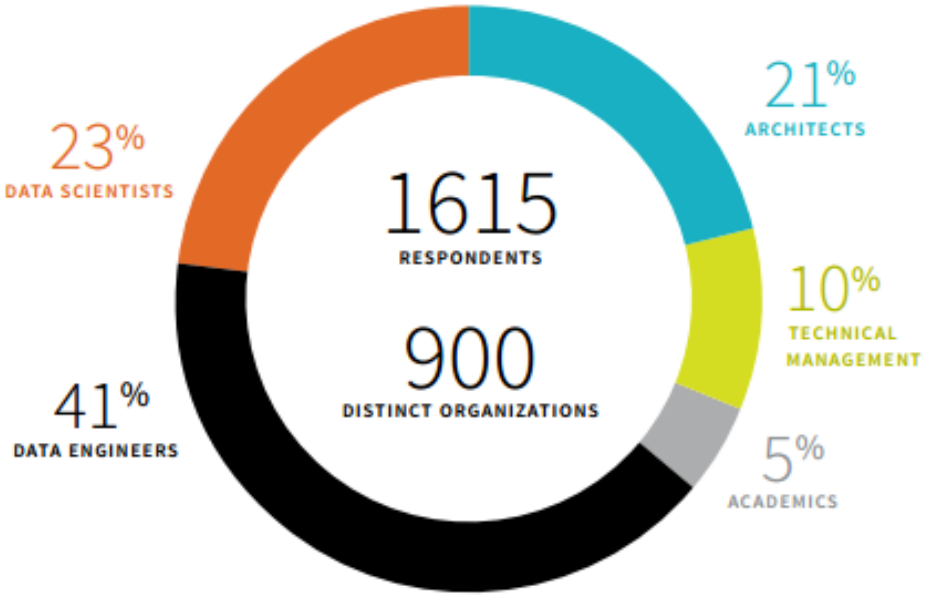
Data can be materialized in memory at any point

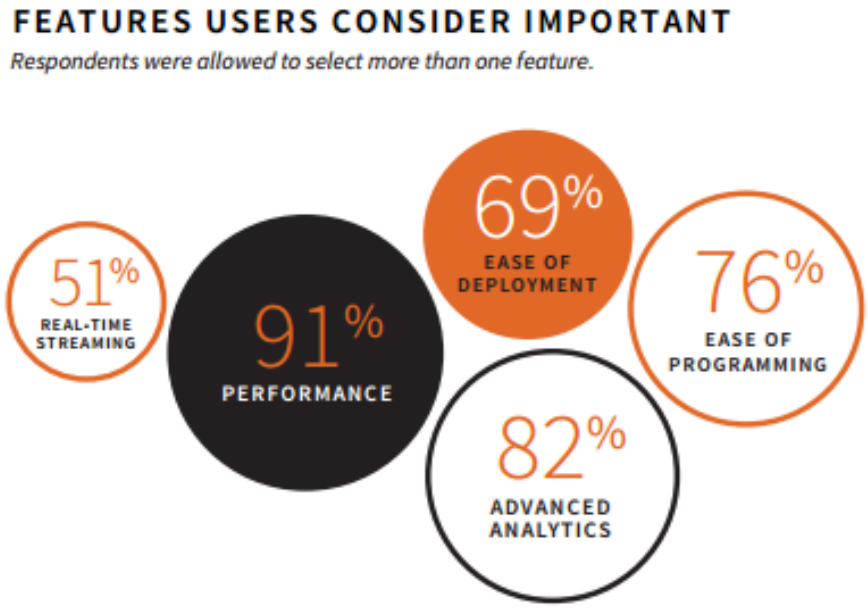


Logistic regression in Hadoop and Spark

Conclusion

Spark acknowledges that the biggest bottleneck in building distributed data applications is not CPU, disk, or network, but **analyst productivity**





References

Matei Zaharia, *Apache Spark 2.0*, https://spark-summit.org/2016/events/keynote-1-day-2/ , 2016

Matei Zaharia, *An Architecture for Fast and General Data Processing on Large Clusters*, 2014

Holden Karau, Andy Konwinski and Patrick Wendell, *Learning Spark. Lightning-fast data analysis*, 2015

Ameet Talwalkar & co, *Scalable Machine Learning*, https://courses.edx.org/courses/BerkeleyX/CS190.1x/1T2015/info, 2016