Big Data Analytics with Apache Spark

An Mai
credit to Quang Vu (Aduro)

What to Expect from this talk

- Introduction to Big Data
- Introduction to Apache Spark
- How to process Big Data using PySpark

Motivation

Why Spark?

- Big Data is becoming bigger and bigger.
- Store, process and use big amount of data for machine learning.

Prerequisites

- Originally, it’s in Scala
- Basic programming skills and some analytical experiences
- Some experiences with Python

Big Data

Why all these matter?

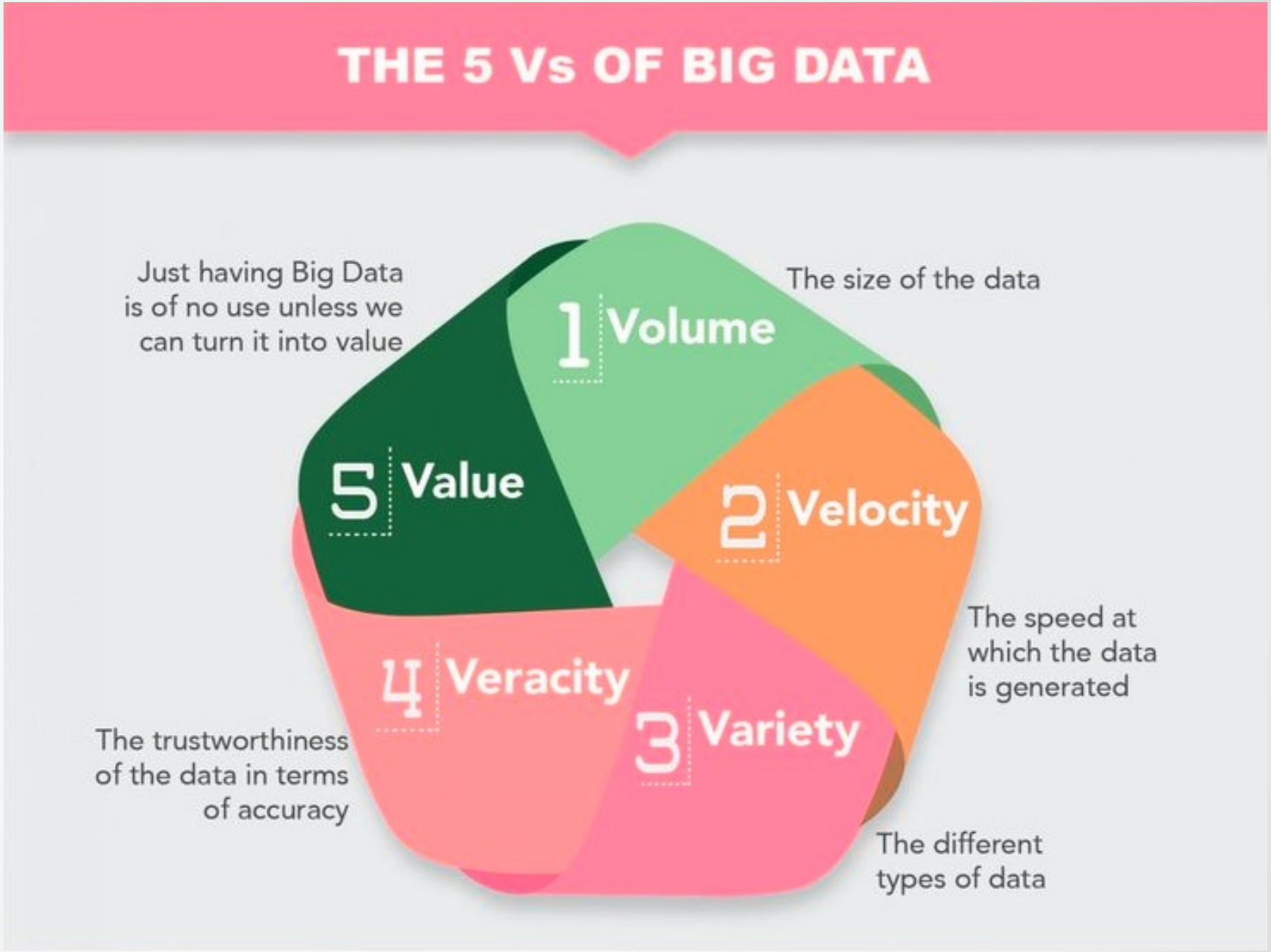
- In the past:
  - 1918, Spanish flu: 0.5B people infected, 10M people killed
  - No vaccine → slow its spread
- CDC (traditional way)
- 2009, a new flu virus was discovered
- Google Flu Trends
  - Detected outbreak 2 weeks ahead
  - Initially 97% accurate
- Nowcasting -> forecasting
- Humm, think about Covid19 today ...

Where does Big Data come from?

- Online activity
  - Click, Server request, Transaction, Network message, …
- User generated content (web and mobile)
  - Facebook, Instagram, Twitter, YouTube, …
- Health and scientific computing:
  - EMR, Genomics, medical images.
- Graph data
  - Social networks, computer networks, road networks
- Web server and machine system log files, source codes.
- IOT: GPS, sensors, ...
- Financial data.

Is it extremely complex?

- 4 Vs of big data is outupdated
- The other missing V comes from practical viewpoint



Types of data

Structured	Semi-structured	Unstructured
• relational database • formatted messages	• XML docum • tagged texts	• plain text edia

Big Data problem

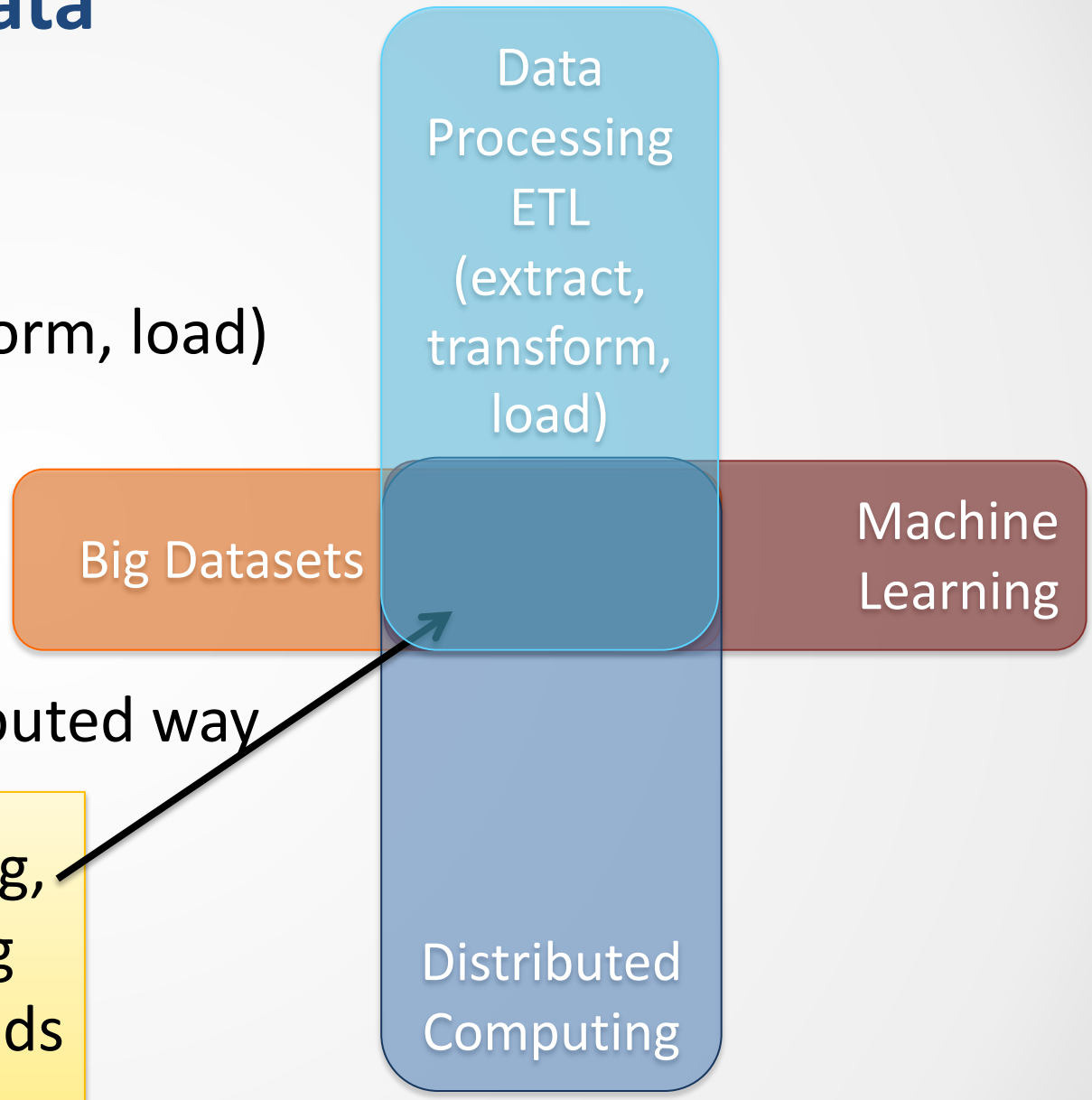
- Traditional analysis: Unix shell commands, R, Python
  - All run in a single machine!
- Data growing faster than computational speeds
- Growing data sources
  - A single machine cannot process or even store all the data!
  - Need a scalable, reliable, standard environment

➔Solution is to distribute data over cluster of machines

➔Based on clouds: storage and computational resources.

Processing & ML with big data

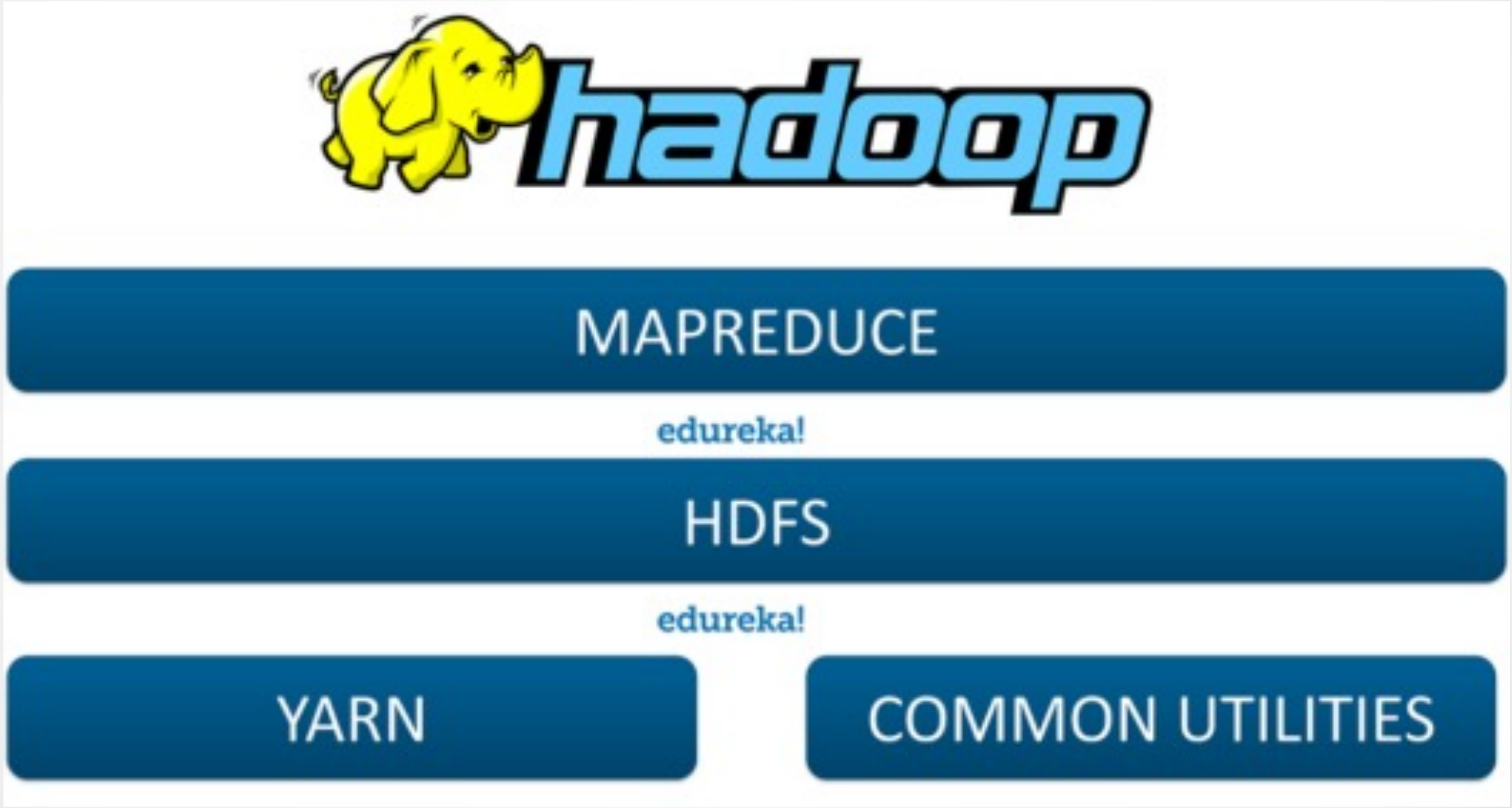
- Data processing
  - Traditional ETL (extract, transform, load)
  - Data Stores (HBase, ……..)
  - Tools for processing
- Machine Learning
- Leverage many machines in distributed way



Working in this are is really challenging, which is the main goal of **modern** big data analytics, and require new methods to store and process **Big Data**

Pioneer Approach

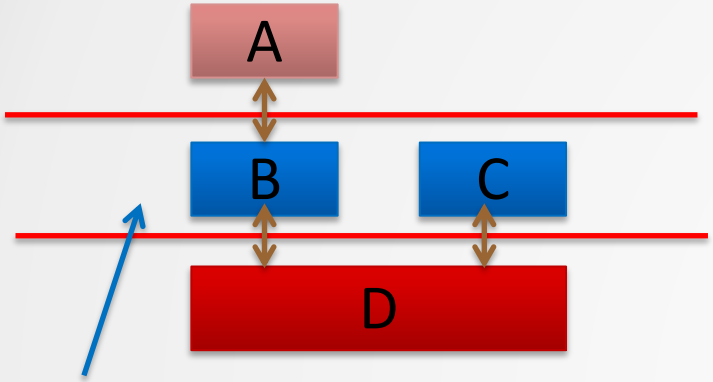
It's Hadoop


hadoop
MAPREDUCE
edureka!
HDFS
edureka!
YARN
COMMON UTILITIES

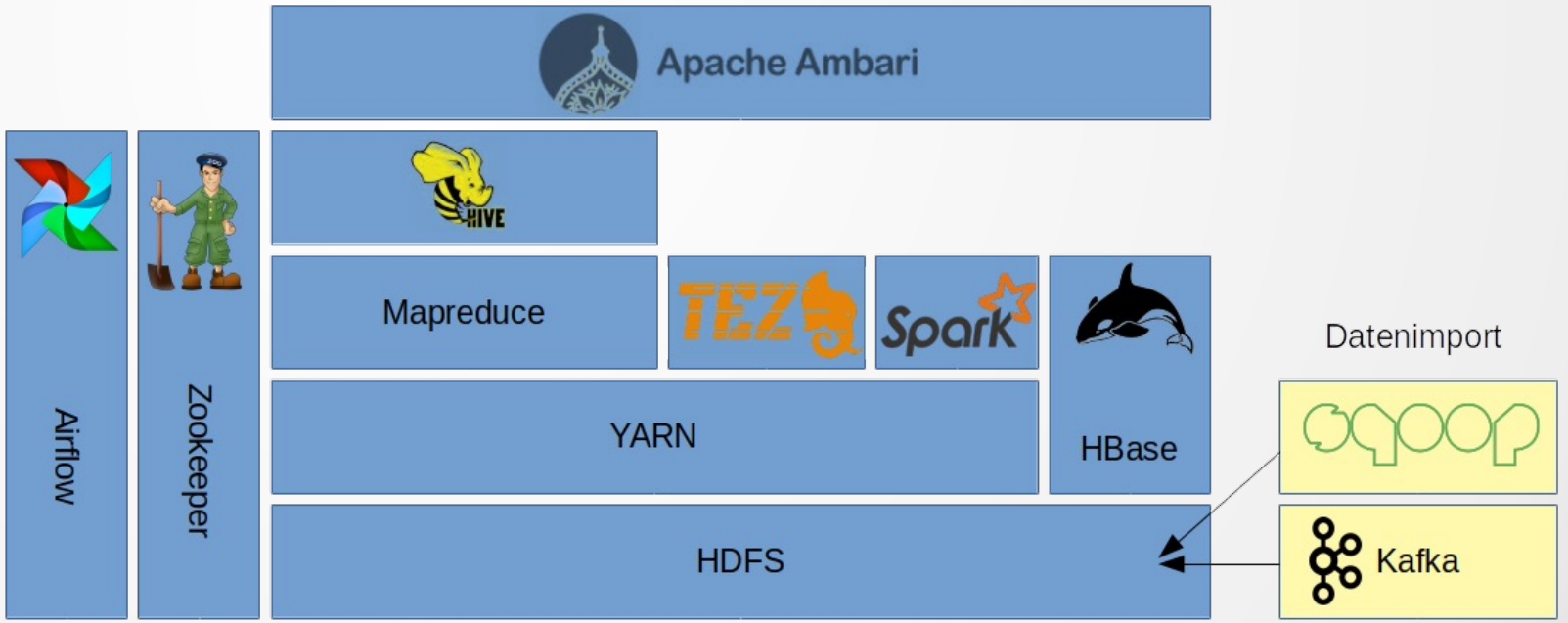
Hadoop Ecosystem

- Provide Scalability
  - on commercial hardware
- Support Fault Tolerance
- Can handle large and diverse datasets
  - Text, Graph, IoT, Streaming Data, Images, Videos…
- Enable Shared Environment
- Provides Value
  - Cost
  - Big community

A picture of Hadoop Ecosystem

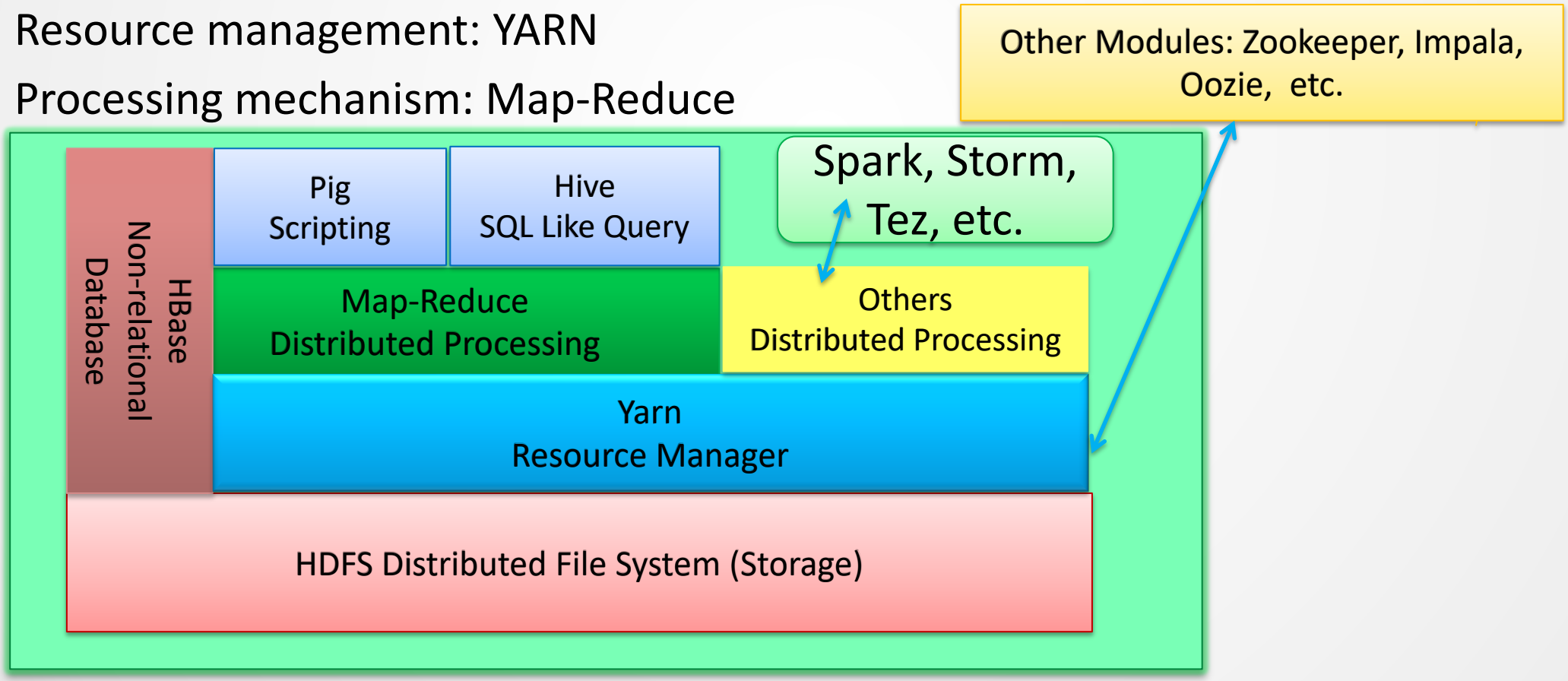


Layer Diagram



Basic Modules of Hadoop

- Hadoop Common: libraries and utilities needed for Hadoop cluster
- Storage: Distributed File System (HDFS)
- Resource management: YARN
- Processing mechanism: Map-Reduce

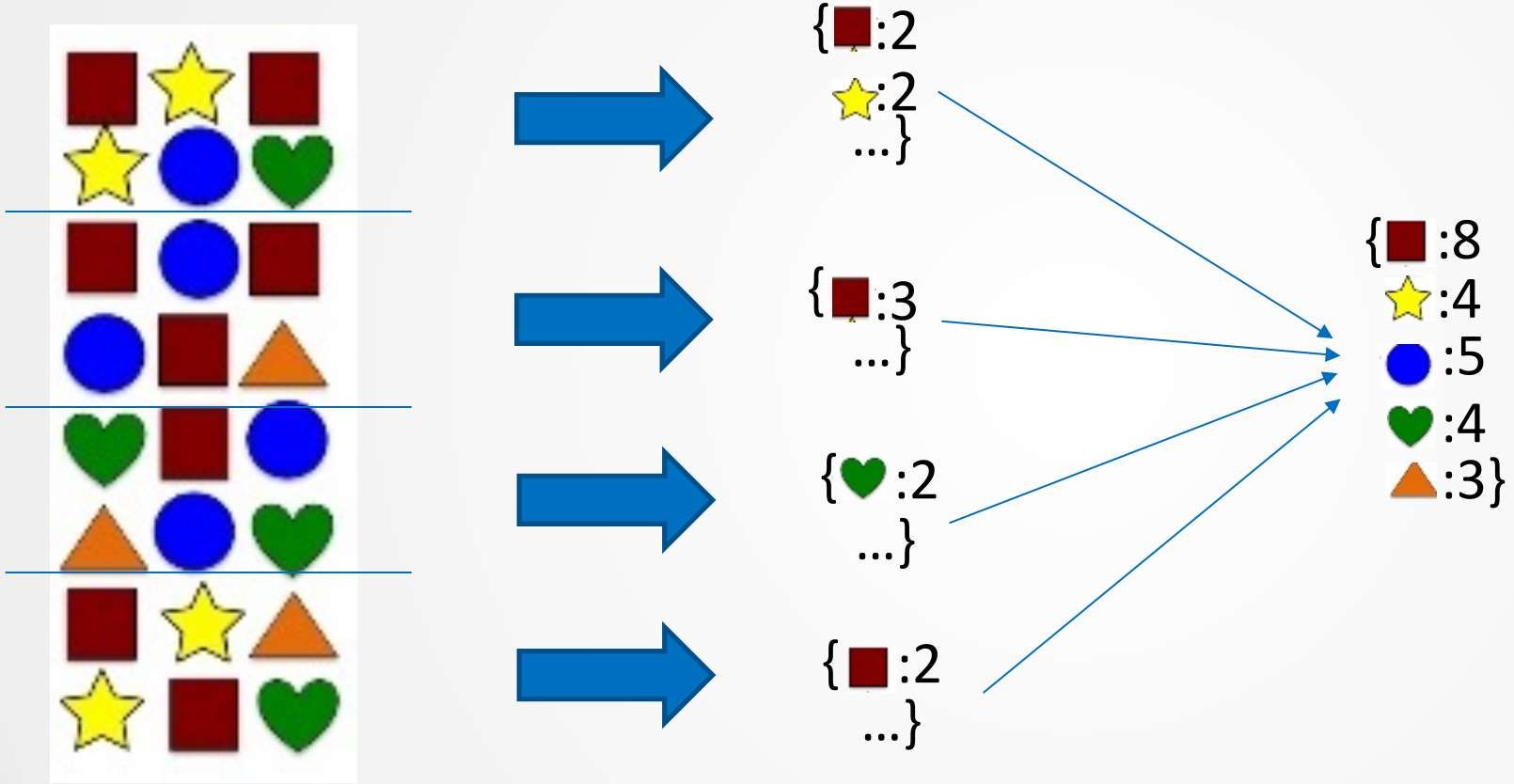


Computational Challenging
On the road to Map - Reduce

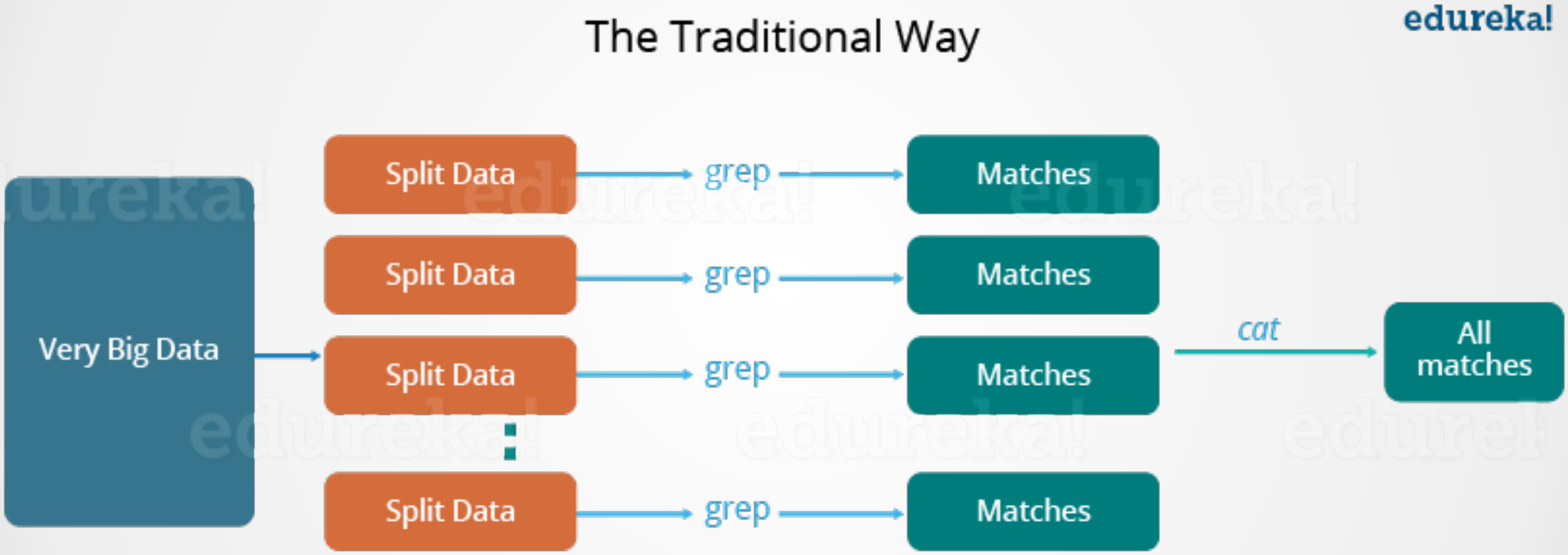
Cluster computing challenges

- How to split work across machines?
- How to deal with failures?

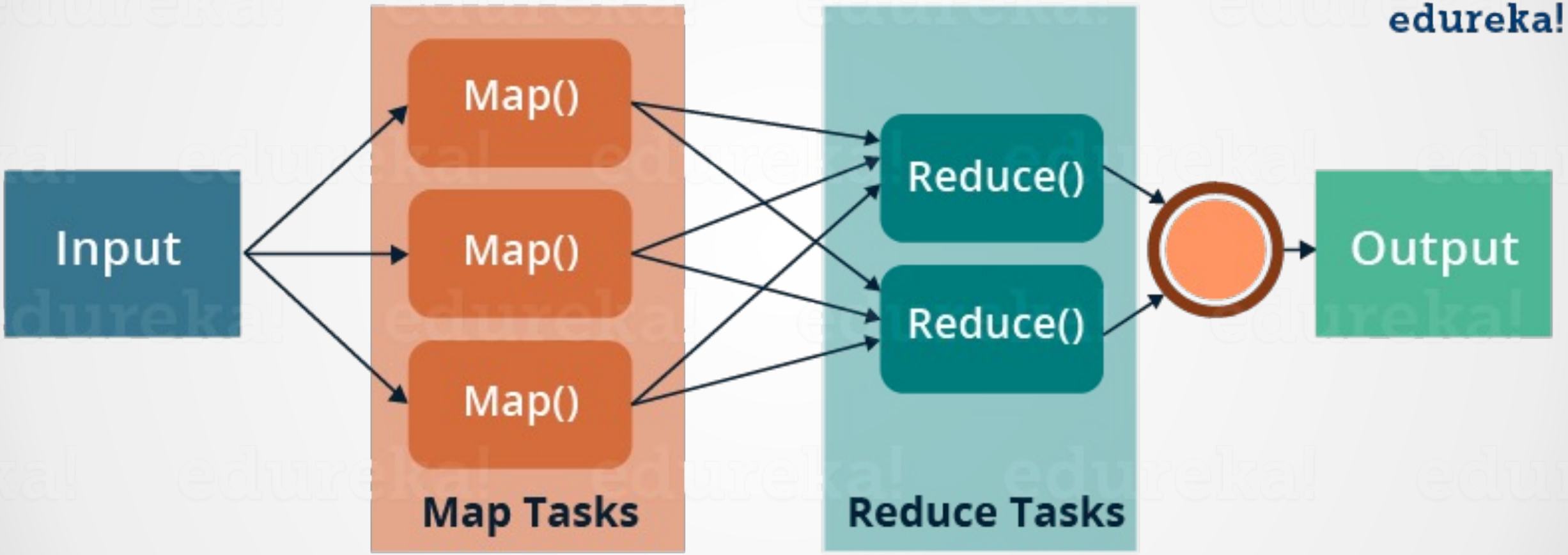
Counting example



Traditional approach (photo taken from Edureka)



MapReduce mechanism (photo taken from Edureka)



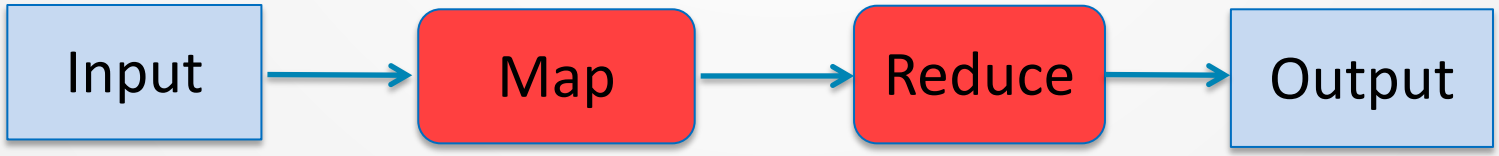
MapReduce: Simple Programming for Big Data

Based on Google's MR paper (2004)

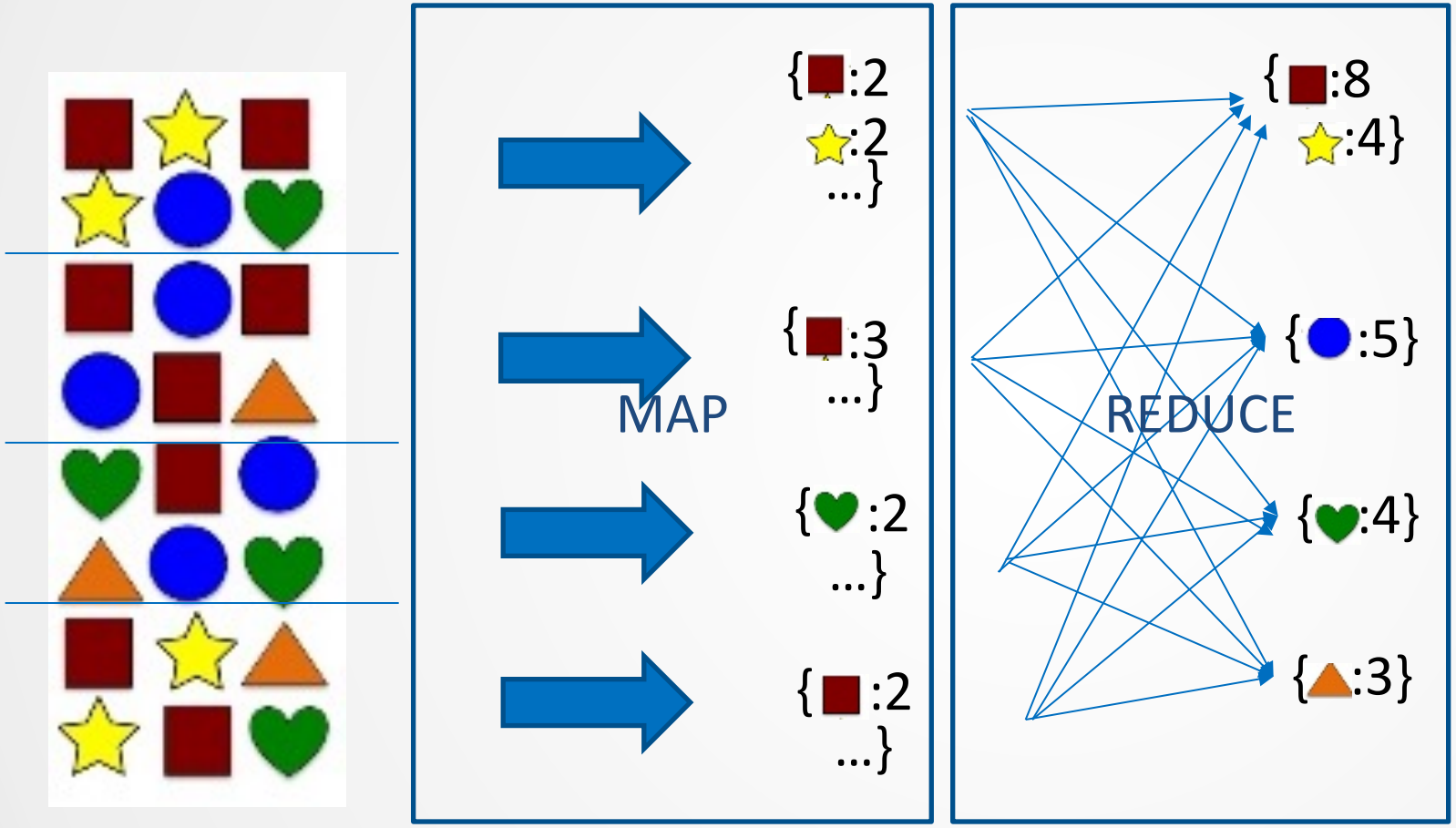
- MapReduce: simple programming paradigm for the Hadoop ecosystem
- Traditional parallel programming requires expertise of different computing/systems concepts
  - examples: multithreads, synchronization mechanisms (locks, semaphores, and monitors)
  - incorrect use: can crash your program, get incorrect results, or severely impact performance
  - **NOT** fault tolerant to hardware failure
- MapReduce Proramming: greatly simplifies running code in parallel
  - don't have to deal with any of above issues
  - only need to create, map and reduce functions

Simple illustrative Paradigm

Map:	Reduce:
Apply a function to all the elements of List list1=[1,2,3,4,5]; square x = x * x list2=Map square(list1) print list2 -> [1,4,9,16,25]	Combine all the elements of list for a summary list1 = [1,2,3,4,5]; A = reduce (+) list1 Print A -> 15

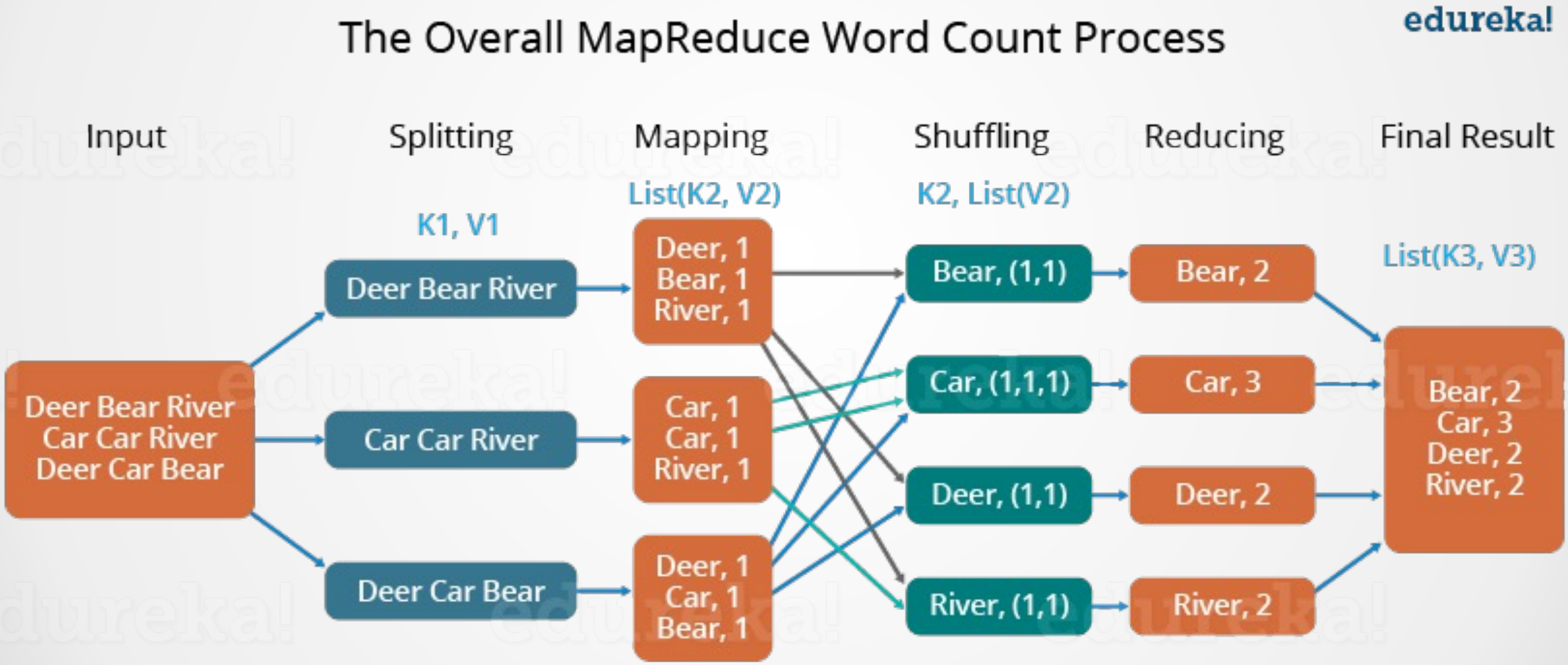


Back to the counting problem?



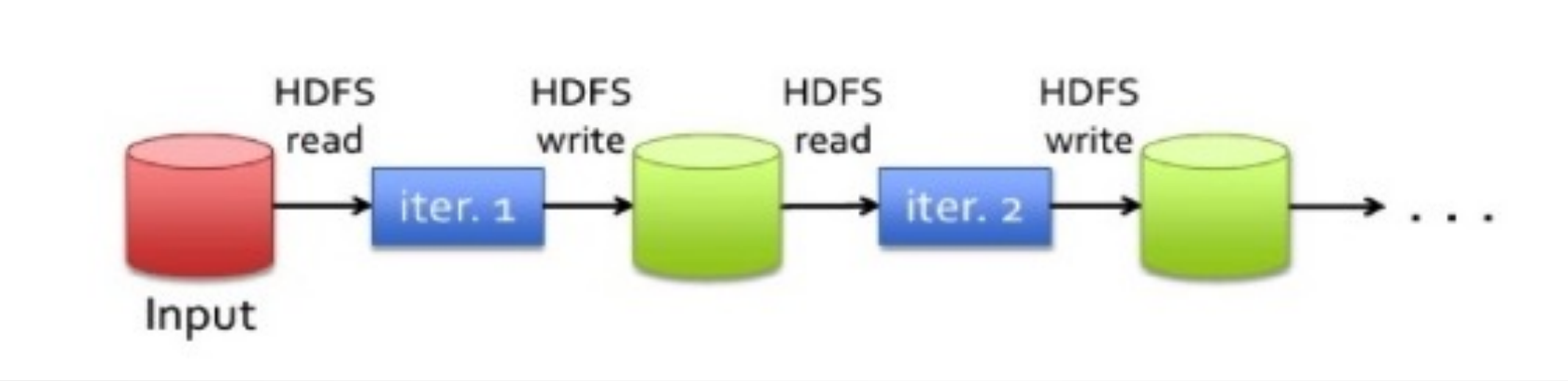
Another counting example:

(**Dear, Bear, River, Car, Car, River, Deer, Car and Bear**)



Map Reduce: it's still boring ...

- Iterative jobs involve a lot of disk I/O for each repetition
- In general, based on "Acyclic Data Flow" from Disk to Disk (HDFS)



➔ Disk I/O is very slow!

Other Shortcoming of MapReduce

- Forces data processing into Map and Reduce
  - Other workflows missing: join, filter, flatMap, groupByKey, union, intersection, …
- Read & write to Disk before & after Map Reduce (stateless machine)
  - **NOT** efficient for iterative tasks, i.e. Machine Learning
- Support only Java natively
  - Support for others languages needed
- Only for Batch processing
  - Interactivity, streaming data

Opportunity

- Tech Trend
  - Cost of memory is reducing → keep more data in memory
- Create new distributed execution engine
- The rise of data analytics.

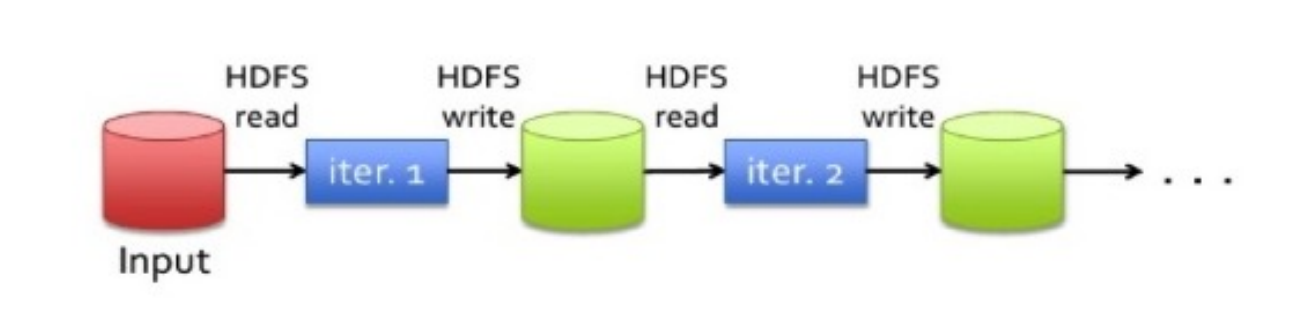
→ Apache Spark

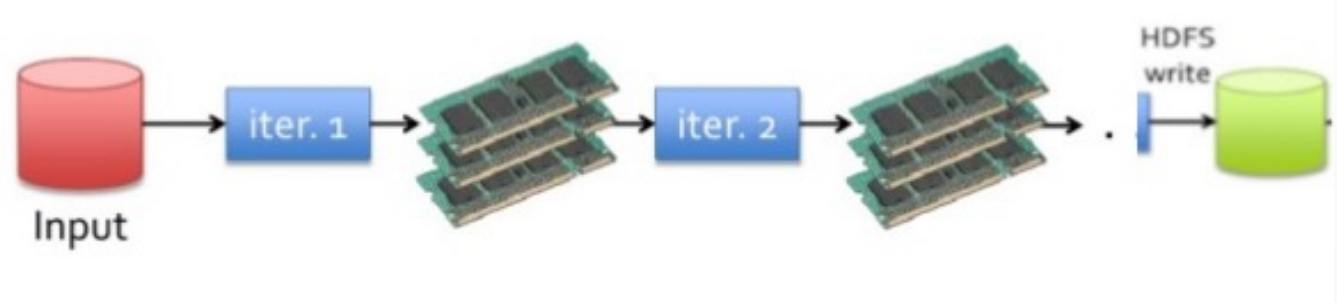
Spark comes into play

Apache Spark

- Built on top the idea of MapReduce.
- Modern, interactive and ease to use.
- Officially, Spark computing framework provides programming abstraction and parallel runtime to hide complexities (partition processes) of fault-tolerance and slow machines.

Use Memory instead of Disk





➔ 10-100x faster than Hadoop MapReduce

Spark and Map Reduce differences

	Apache Hadoop MR	Apache Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Many transformations and actions, including Map and Reduce
Execution model	Batch	Batch, iterative, streaming
Languages	Java	Scala, Java, Python and R

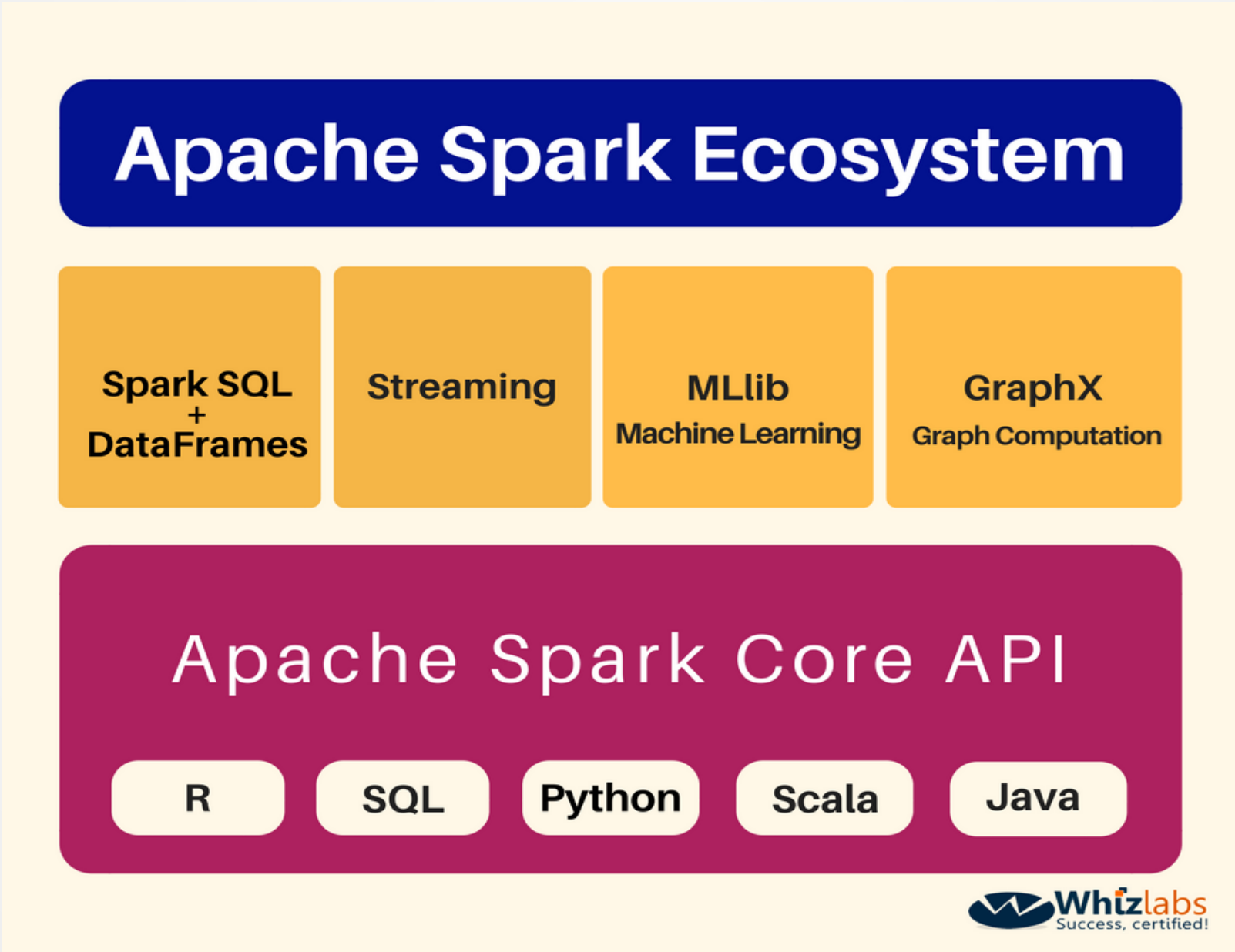
Apache Spark vs Apache Hadoop

	Hadoop MR Record (2013)	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Spark, 3x faster with 1/10 the nodes

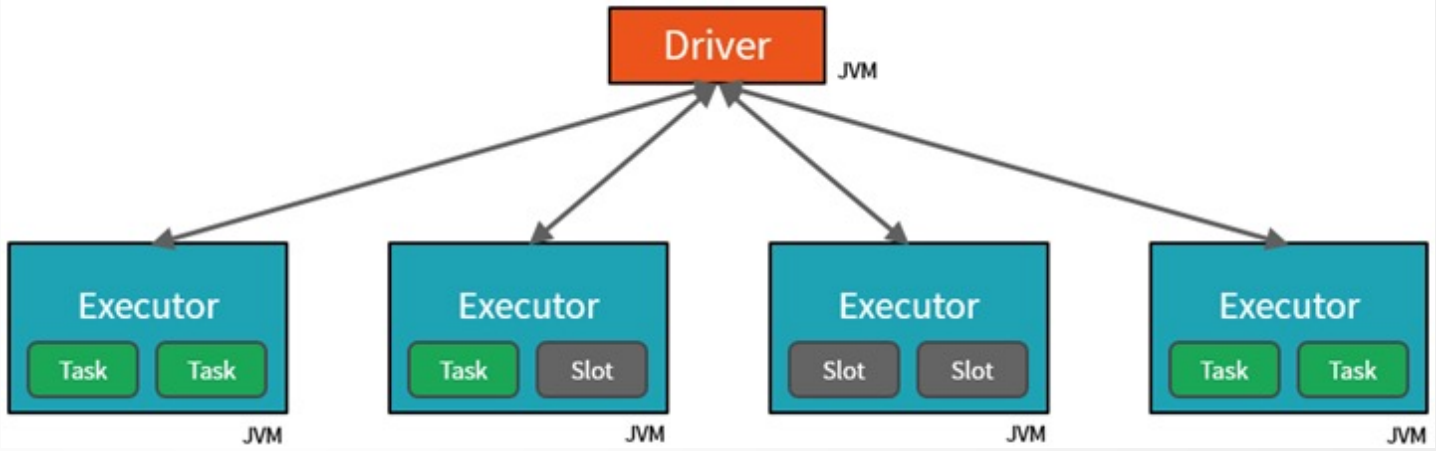
Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)
http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Components


Apache Spark Ecosystem
Spark SQL + DataFrames
Streaming
MLlib Machine Learning
GraphX Graph Computation
Apache Spark Core API
R
SQL
Python
Scala
Java
Whizlabs Success, certified!

Architecture

- A master-worker type architecture
  - A driver or master node
  - Worker nodes



- The master send works to the workers and either instructs them to pull data from memory or from hard disk (or from another source like S3 or HDSF)

Architecture(2)

- A Spark program first creates a *SparkContext* object
  - *SparkContext* tells Spark how and where to access a cluster
  - The *master* parameter for a *SparkContext* determines which type and size of cluster to use

Master parameter	Description
local	Run Spark locally with one worker thread (no parallelism)
local[K]	Run Spark locally with K worker threads (ideal set to number of cores)
spark://HOST:PORT	Connect to a Spark standalone cluster
mesos://HOST:PORT	Connect to a Mesos cluster
yarn	Connect to a YARN cluster

DataFrame

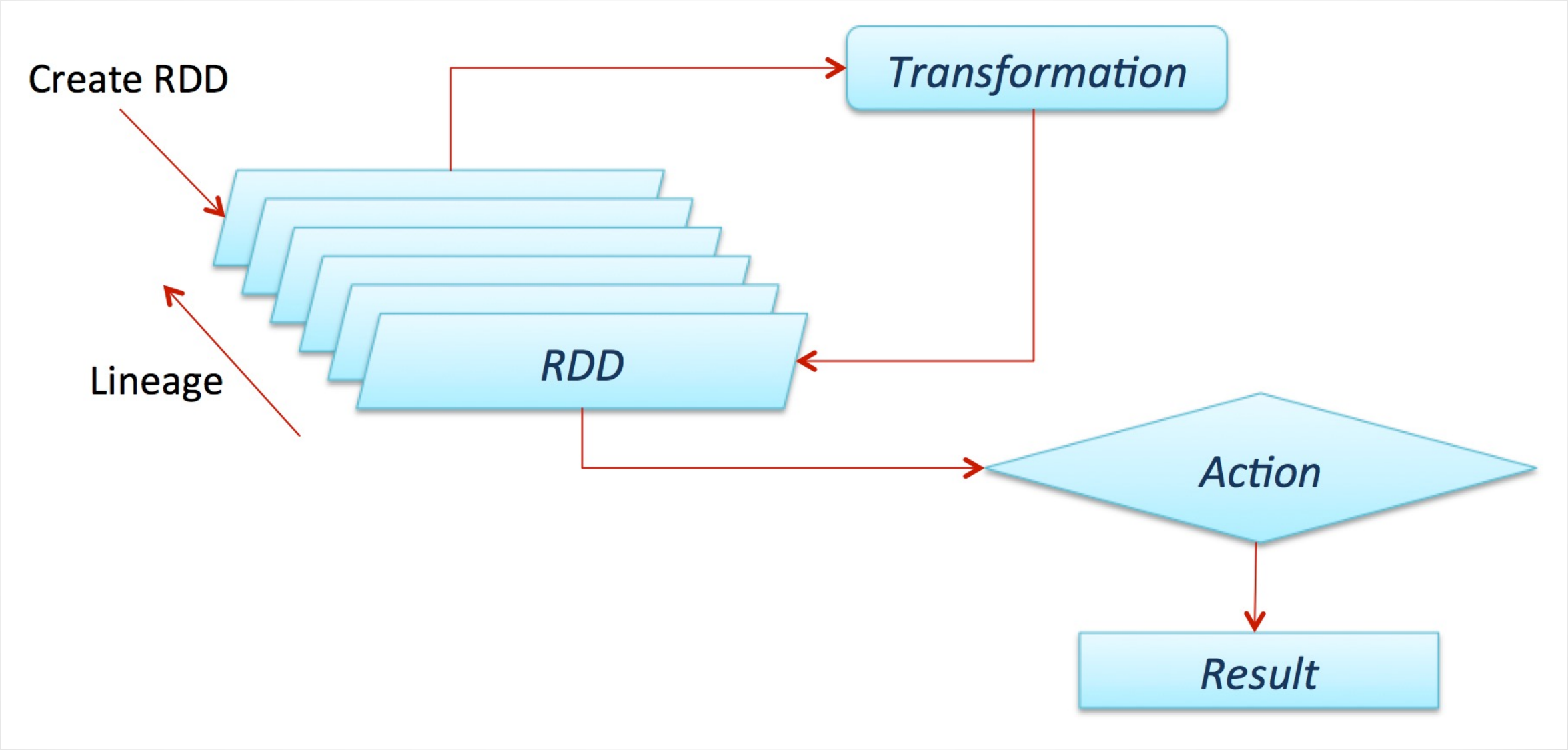
- A primary abstraction in Spark
  - Immutable once constructed
  - Track lineage information to efficiently re-compute lost data
  - Enable operations on collection of elements in parallel
- To construct DataFrame
  - By *parallelizing* existing Python collections (lists)
  - By transforming an existing Spark DF or pandas DF
  - From *files* in HDFS or other storage system

Using DataFrame: an example in pyspark

```
>>> data = [('Alice', 1), ('Bob', 2), ('Bob', 2)]
>>> df1 = sqlContext.createDataFrame(data, ['name', 'age'])
```

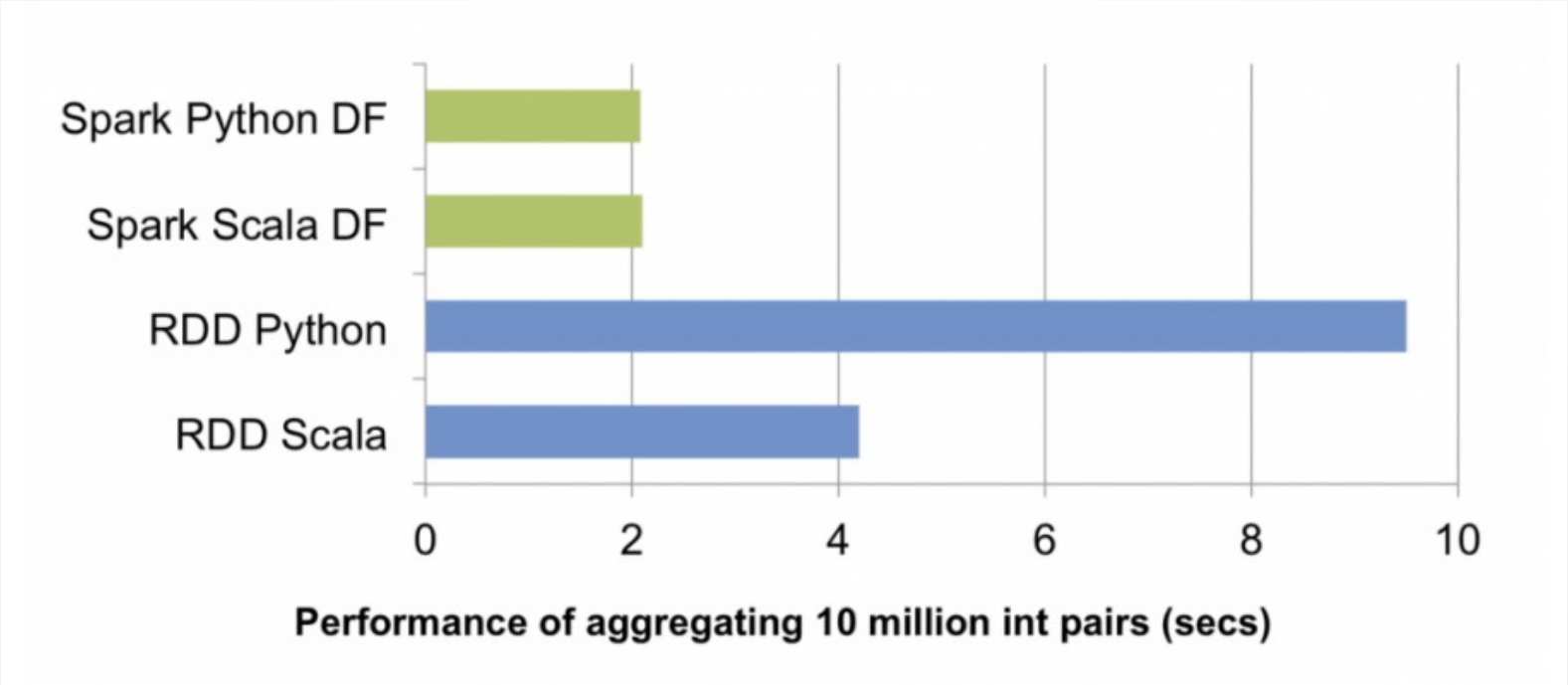
[Row(name=u'Alice', age=1), Row=(name=u'Bob', age=2), Row=(name=u'Bob', age=2)]

Original Mechanism



RDDs vs. DFs

- RDDs provide a low level interface into Spark
- DFs have a schema
- DFs are cached and optimized by Spark community
- DFs are built on top of the RDDs and the core Spark API



Transformations

- Create new *DF* from an existing one
- Use lazy evaluation
  - Just recipes
  - Nothing executes
  - Spark saves recipes for transformation source

Some useful Transformations

Transformation	Description
select(**cols*)	Selects columns from this DataFrame
drop(*col*)	Returns a new Dataframe that drops the specific column
filter(*func*)	Returns a new DataFrame formed by selecting those rows of the source on which *func* returns true
where(*func*)	Where is an alias for filter
distinct()	Returns a new DataFrame that contains the distinct rows of the source DataFrame
sort(**cols, **kw*)	Returns a new DataFrame sorted by the specified columns and in the sort order specified by *kw*

Using Transformations

```
>>> data = [('Alice', 1), ('Bob', 2), ('Bob', 2)]
>>> df1 = sqlContext.createDataFrame(data, ['name', 'age'])
>>> df2 = df1.distinct()
```

[Row(name=u'Alice', age=1), Row=(name=u'Bob', age=2)]

```
>>> df3 = df2.sort("age", asceding=False)
```

[Row=(name=u'Bob', age=2), Row(name=u'Alice', age=1)]

Actions

- Cause Spark to execute recipe to transform source
- Mechanisms for getting results out of Spark

Some useful Actions

Action	Description
show(*n, truncate*)	Prints the first n rows of this DataFrame
take(*n*)	Returns the first n rows as a list of Row
collect()	Returns all the records as a list of Row (*)
count()	Returns the number of rows in this DataFrame
describe(**cols*)	Exploratory Data Analysis function that computes statistics (count, mean, stddev, min, max) for numeric columns

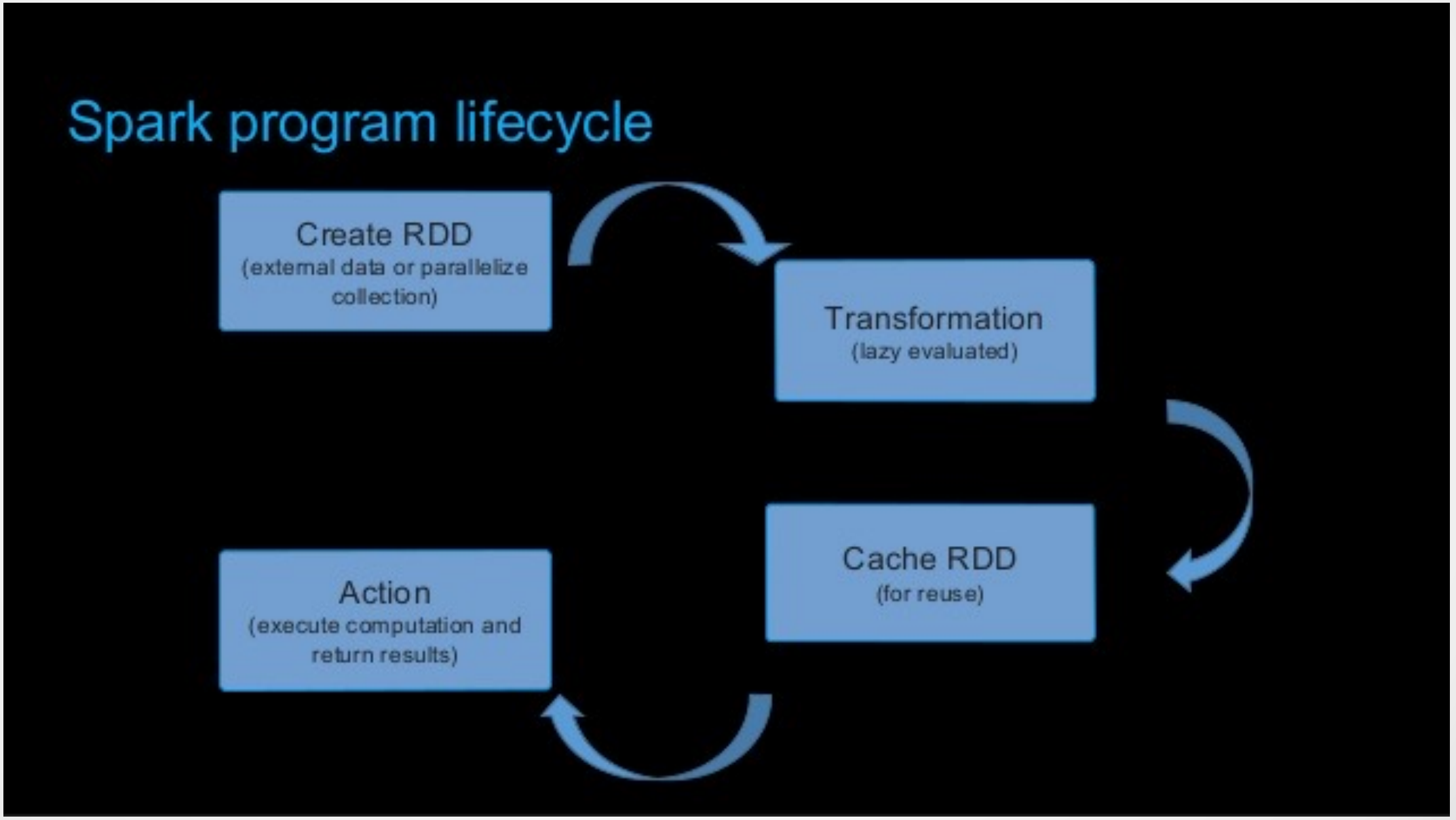
Using Actions: an example in pyspark

```
>>> data = [('Alice', 1), ('Bob', 2)]
>>> df = sqlContext.createDataFrame(data, ['name', 'age'])
>>> df.collect()
[Row(name=u'Alice', age=1), Row=(name=u'Bob', age=2)]
>>> df.count()
2
>>> df.show()
+-------+--------+
|name|  age |
+-------+-------+
|Alice|       1|
|Bob |       2|
+-----+-------+
```

Caching

```
>>> linesDF = sqlContext.read.text('…')
>>> linesDF.cache()
>>> commentsDF = linesDF.filter(isComment)
>>> print linesDF.count(), commentsDF.count()
>>> commentsDF.cache()
```

Spark Program Lifecycle



Machine Learning Library in Spark (MLlib)

- 2 packages
  - spark.mllib
  - spark.ml
- ML algorithms
  - Common learning algorithms such as classification, regression, clustering and collaborative filtering
- Featurization
  - Feature extraction, transformations, dimensionality reduction and selection
- Utilities
  - Linear algebra, statistics, data handling, …

ML: Transformer

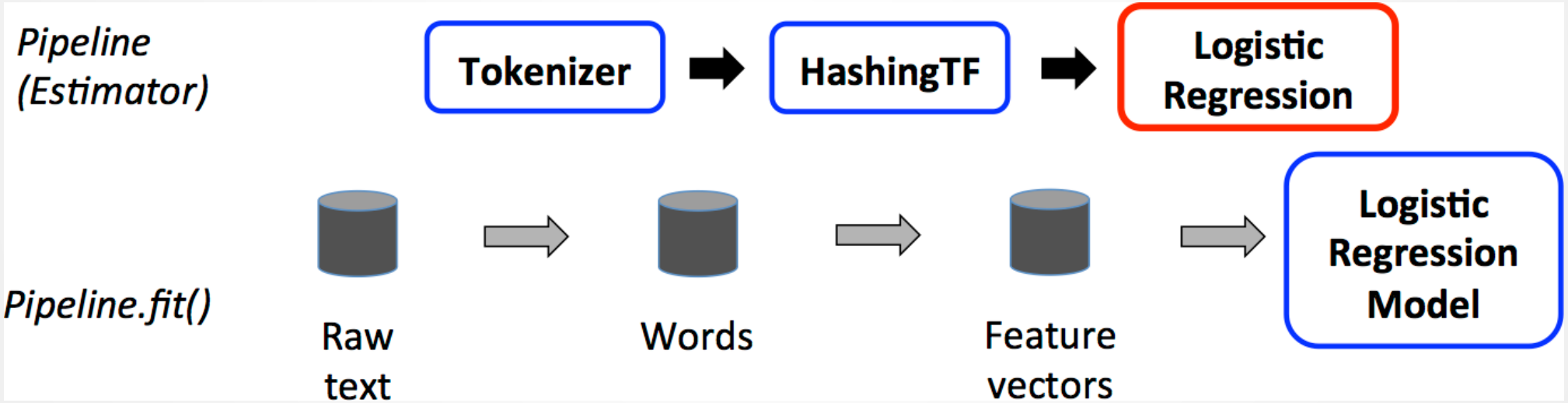
- A *Transformer* is a class which can transform a DF into another DF
- A *Transformer* implements *transform()*
- Examples
  - HashisngTF
  - LogisticRegressionModel
  - Binarizer

ML: Estimator

- An *Estimator* is a class which can take a DF and return a Transformer
- An *Estimator* implements *fit()*
- Examples
  - LogisticRegression
  - StandardScaler
  - Pipeline

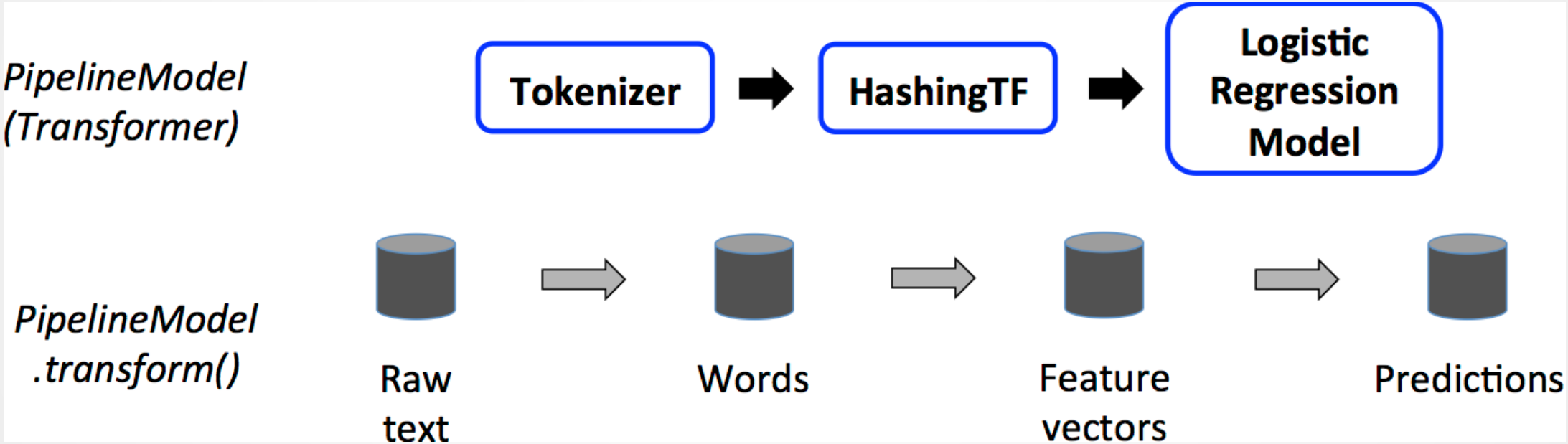
ML: Pipeline

- A *Pipeline* is an estimator that specified as a sequence of stages and each stage can be either estimators or transformers.



ML: PipelineModel

- After a Pipeline.fit() runs, it produces a *PipelineModel.* This *PipelineModel* is used at test time.



References

- Data Science and Engineering with Spark @EDX
- Spark Documentation
  - http://spark.apache.org/docs/latest/

Takeaways

- Hadoop (HDFS, MapReduce)
  - Provides an easy solution for processing of Big Data
  - Brings a paradigm shift in programming distributed system
- Spark
  - Has extended MapReduce for in-memory-based
  - for streaming, interactive, iterative and machine learning tasks
  - Ease to use with the rise of Machine Learning
- Changing the World
  - Made data processing cheaper and more efficient and scalable
  - Is the foundation of many other tools and softwares

Q&A

A demo

THANK YOU!!!