TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Apache Spark overview

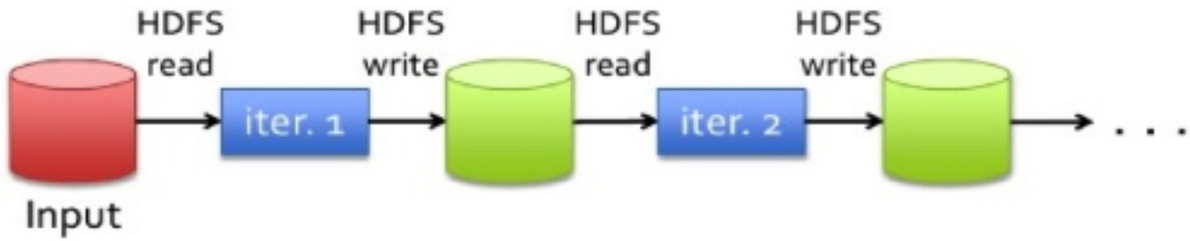
a unified analytics engine for large-scale data processing

Viet-Trung Tran

School of Information and Communication Technology

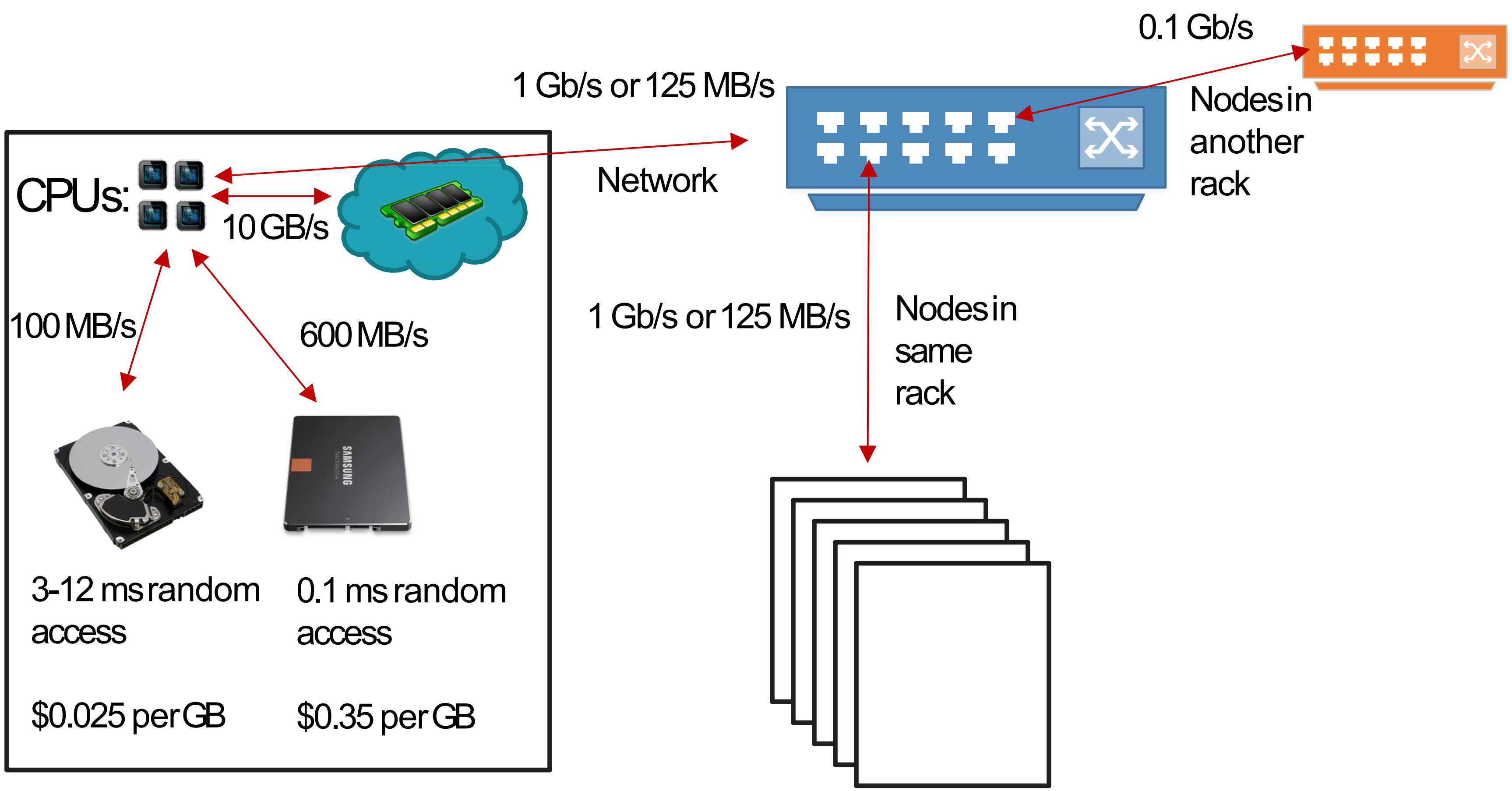
Map Reduce: Iterative jobs

- Iterative jobs involve a lot of disk I/O for each repetition



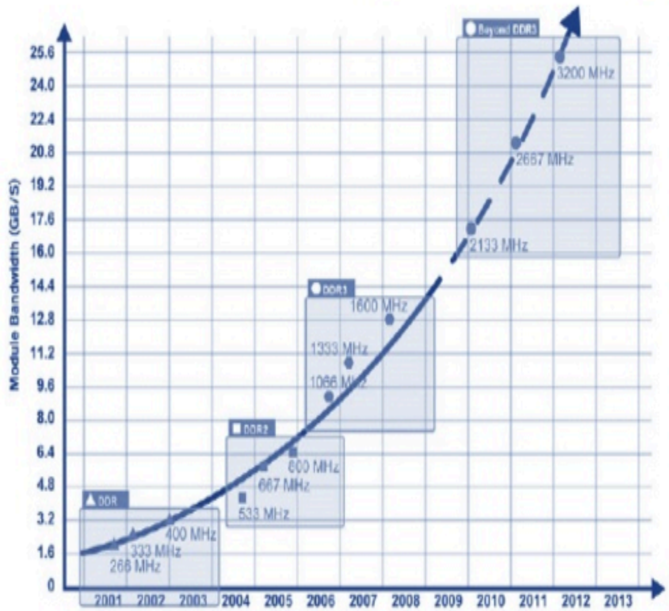
- ➔ Disk I/O is very slow!

I/O landscape



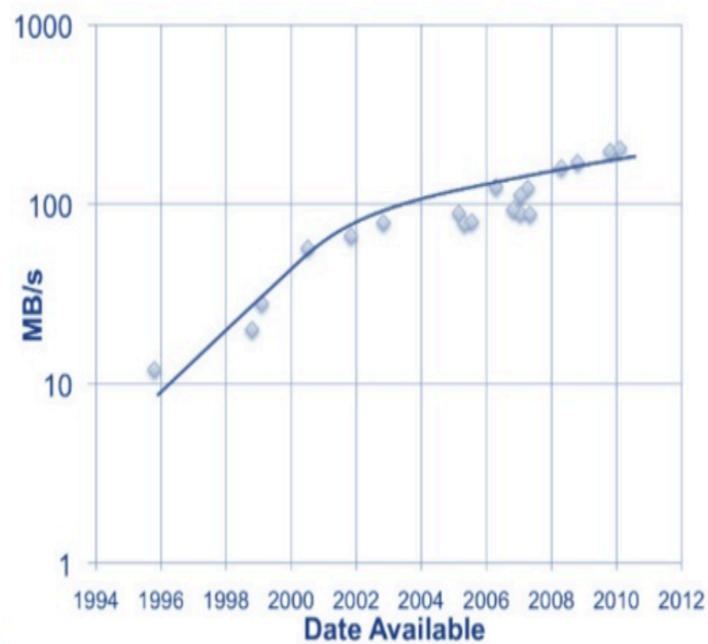
RAM is the new disk

- RAM throughput increasing **exponentially**



Bandwidths shown for 64-bit memory module. Date indicates approximate industry product introduction.

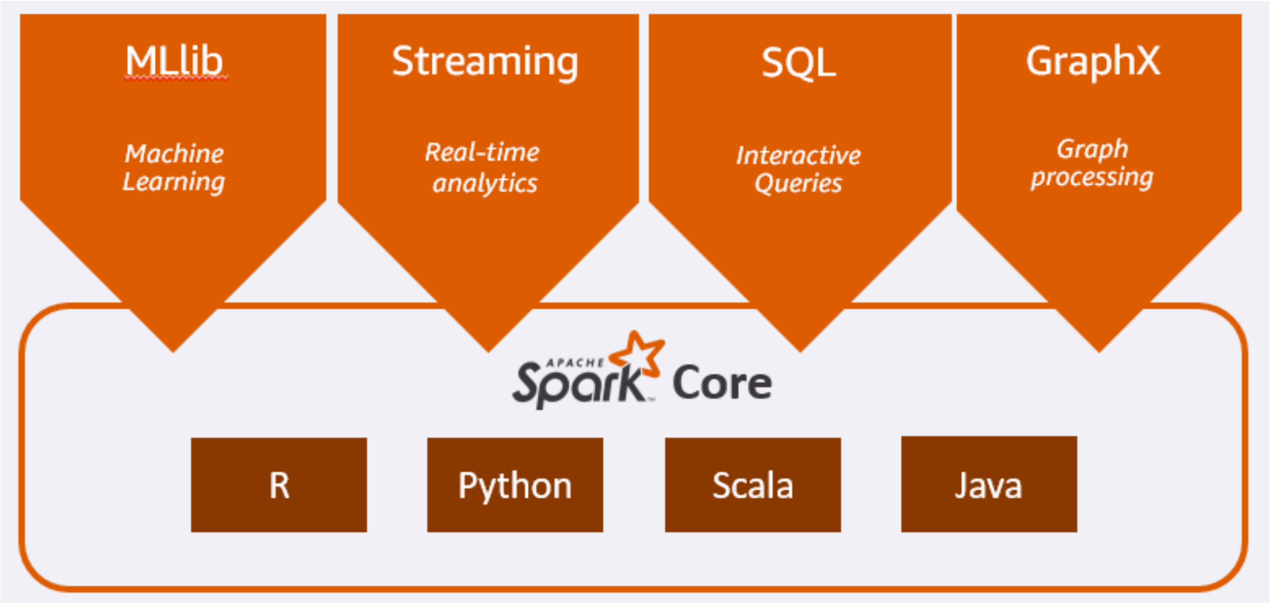
- Disk throughput increasing **slowly**



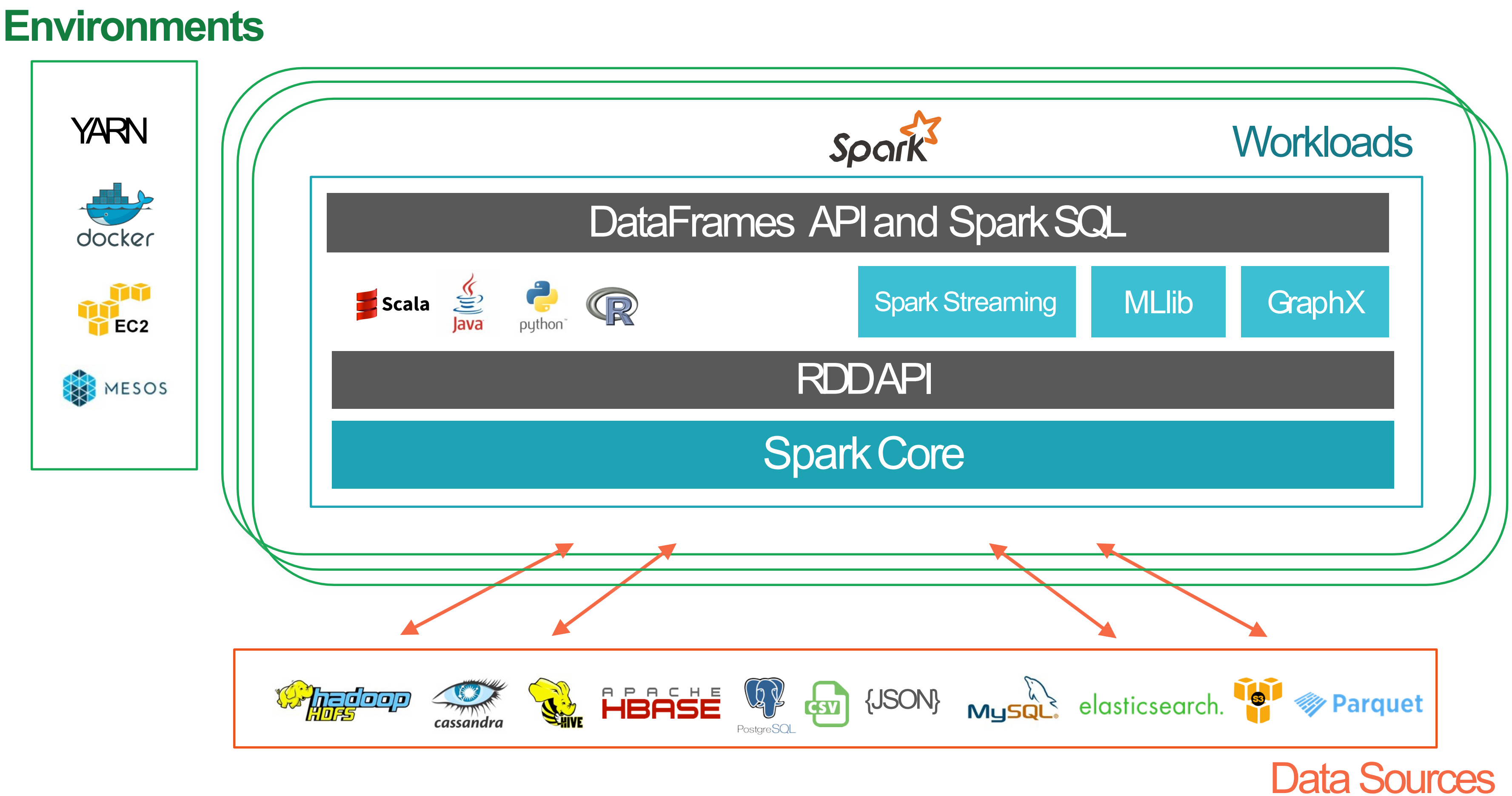
Memory-locality key to interactive response times

A unified analytics engine for large-scale data processing

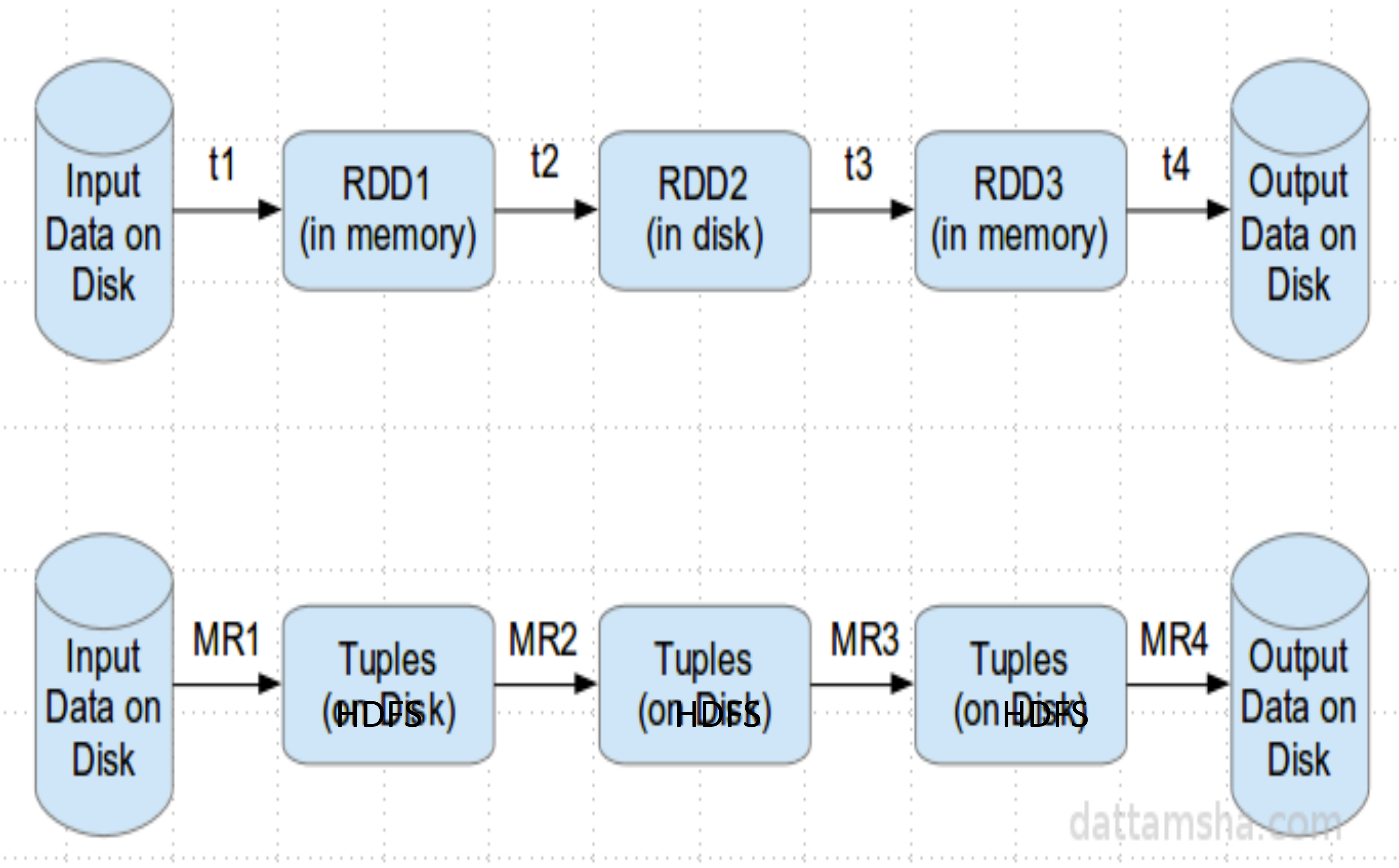
- Better support for
  - Iterative algorithms
  - Interactive data mining
- Fault tolerance, data locality, scalability
- Hide complexites: help users avoid the coding for structure the distributed mechanism.



A unified analytics engine for large-scale data processing



Memory instead of disk



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 World Record	Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

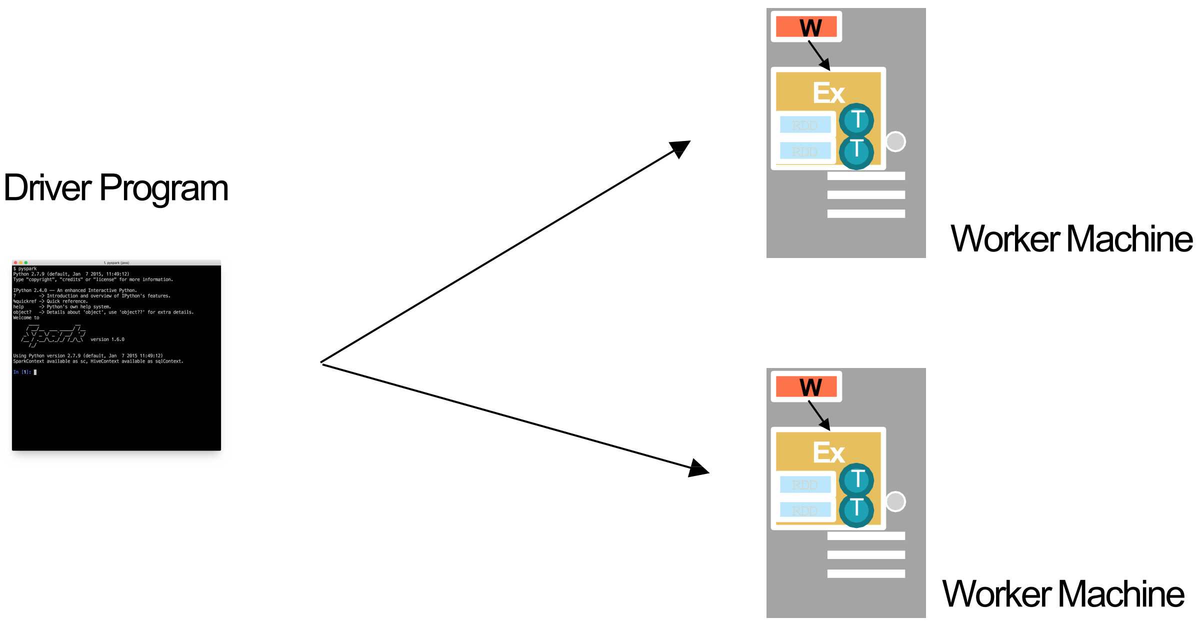
https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html

Interactive shell

```
1. pyspark (java)
$ pyspark
Python 2.7.9 (default, Jan  7 2015, 11:49:12)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.9 (default, Jan  7 2015 11:49:12)
SparkContext available as sc, HiveContext available as sqlContext.

In [1]:
```

(Scala, Python and Ronly)

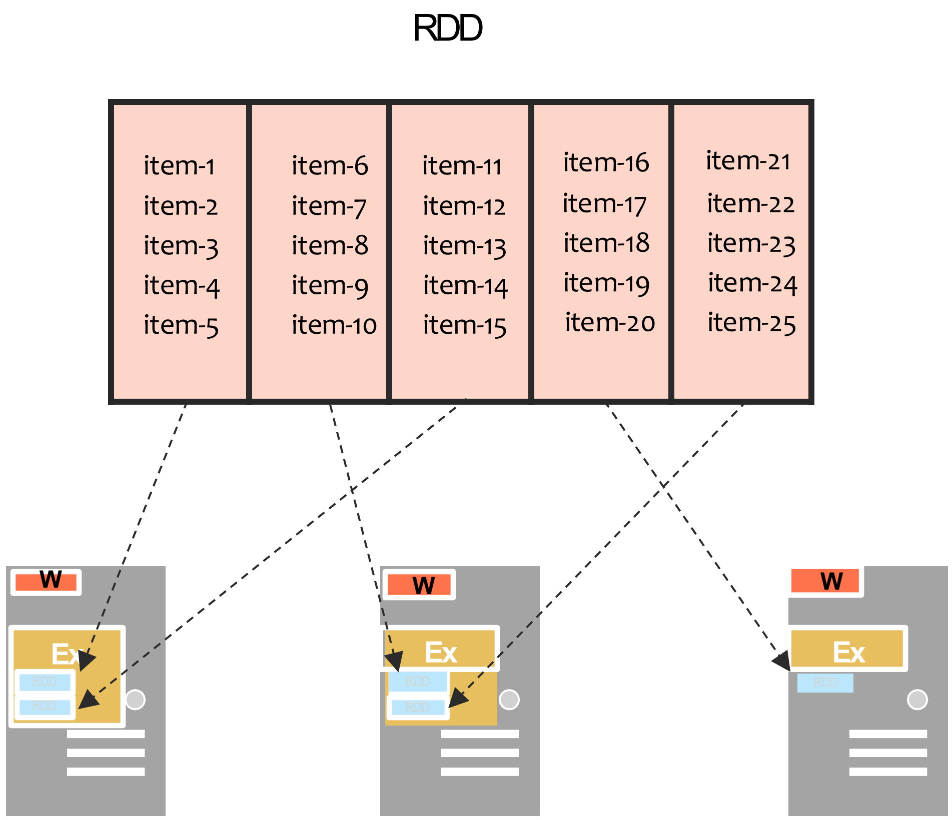
Spark execution overview



Resilient Distributed Dataset (RDD)

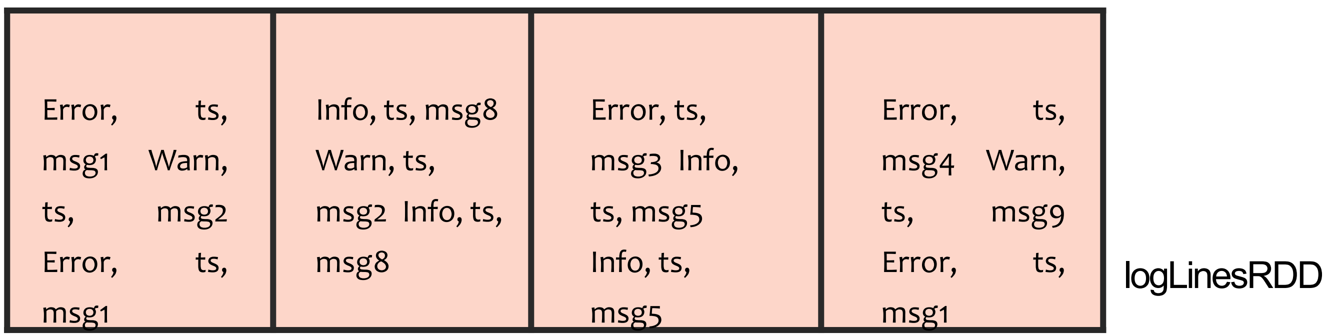
- RDDs are ***fault-tolerant, parallel data structures*** that let users explicitly persist ***intermediate results in memory***, control their partitioning to optimize data placement, and manipulate them using ***a rich set of operators***.
  - RDDs automatically rebuilt on machine failure
- coarse-grained transformations vs. fine-grained updates
  - e.g., map, filter and join) that apply the same operation to many data items at once.

More partitions = more parallelism



RDD creation

- A base RDD can be created 2 ways:
  - Parallelize a collection
  - Read data from an external source (S3, C*, HDFS, etc)



Parallelize

```scala
// Parallelize in Scala
val wordsRDD = sc.parallelize(List("fish", "cats", "dogs"))
```

```python
# Parallelize in Python
wordsRDD = sc.parallelize(["fish", "cats", "dogs"])
```

```java
// Parallelize in Java
JavaRDD<String> wordsRDD = sc.parallelize(Arrays.asList("fish", "cats", "dogs"));
```

- Take an existing in-memory collection and pass it to SparkContext's parallelize method
- Not generally used outside of prototyping and testing since it requires entire dataset in memory on one machine

Read from text file

```scala
// Read a local txt file in Scala
val linesRDD = sc.textFile("/path/to/README.md")
```

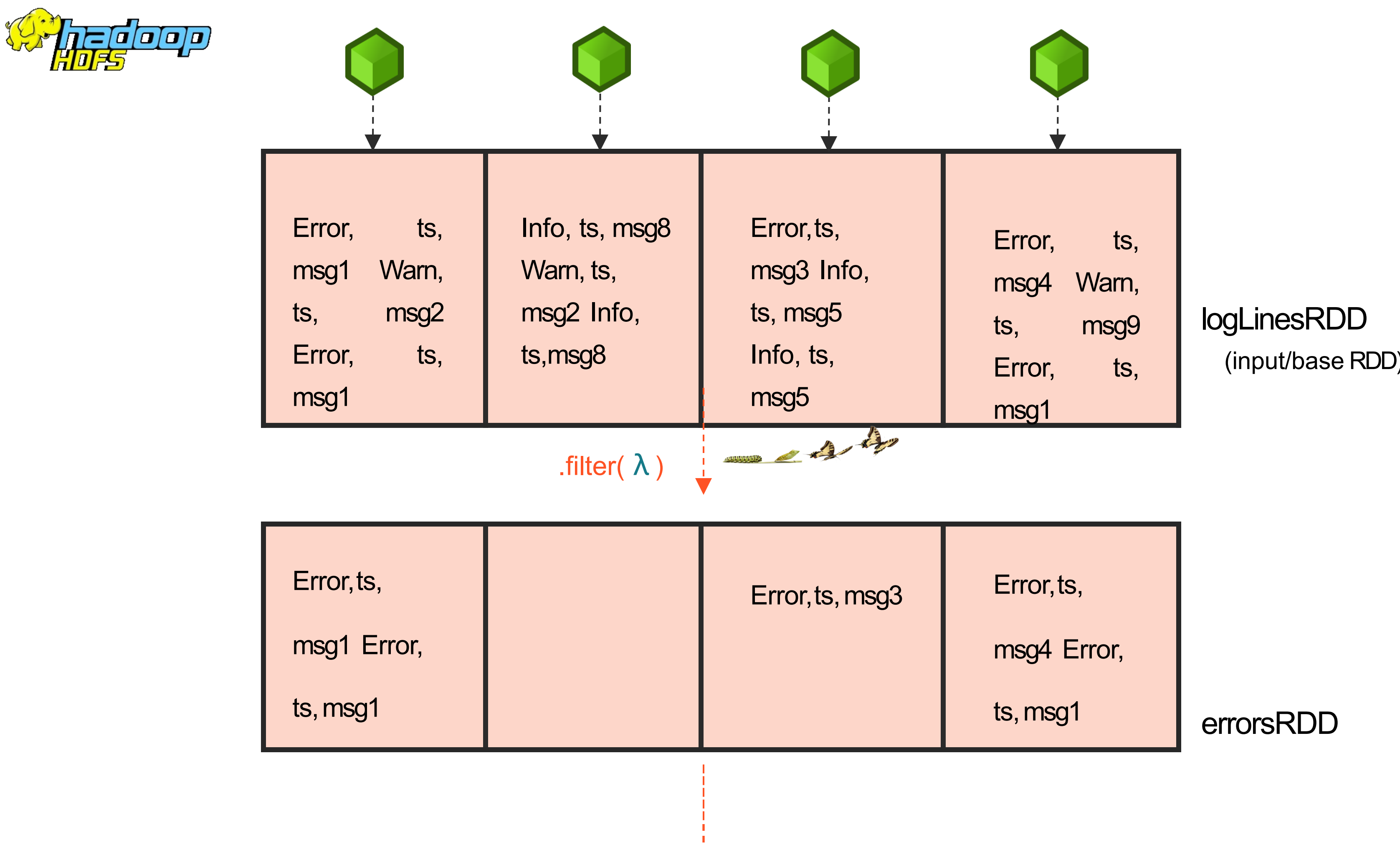
```python
# Read a local txt file in Python
linesRDD = sc.textFile("/path/to/README.md")
```

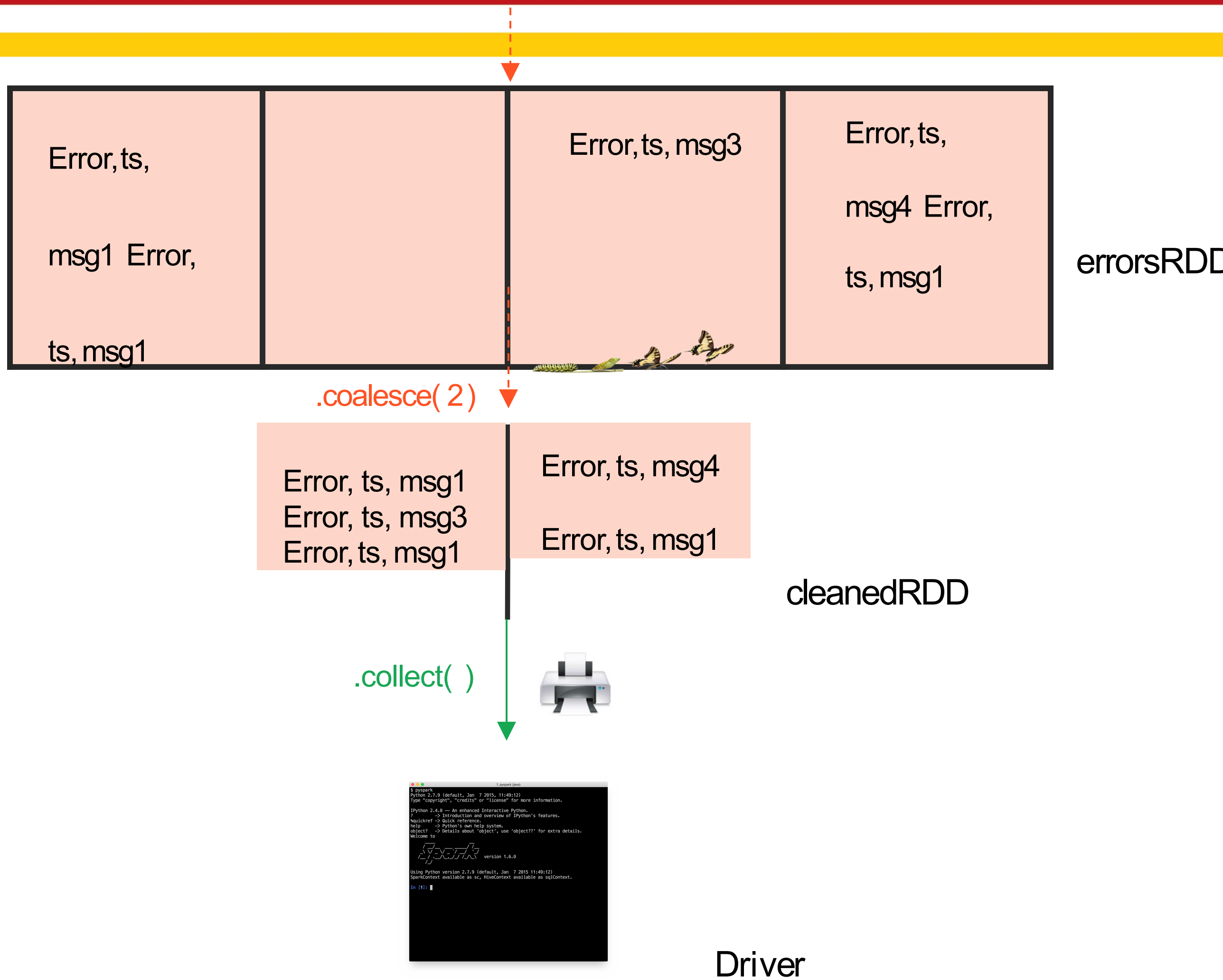
```java
// Read a local txt file in Java
JavaRDD<String> lines = sc.textFile("/path/to/README.md");
```

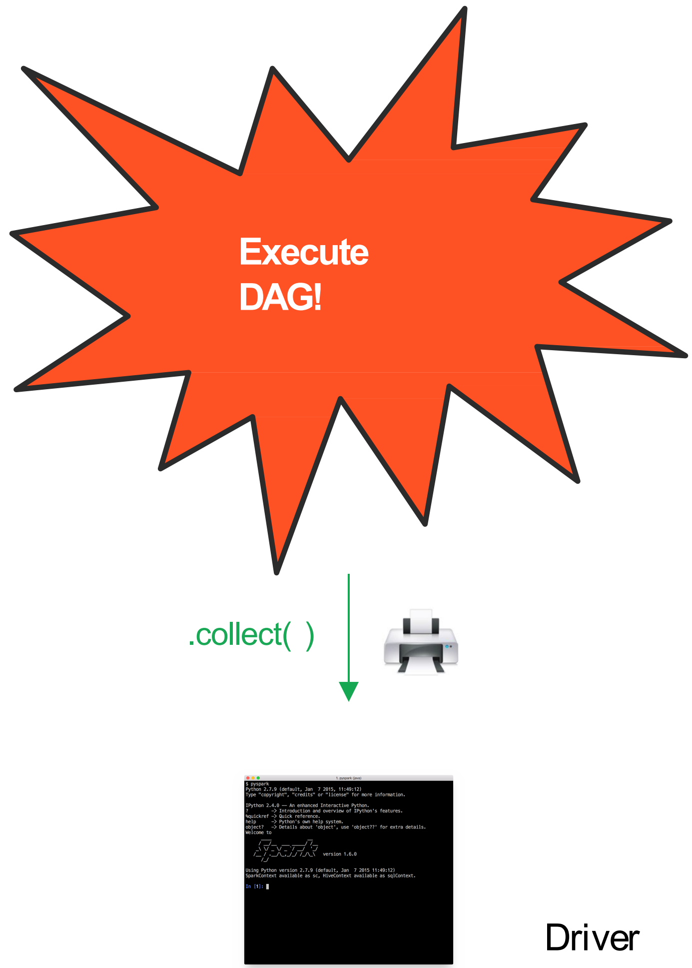
There are other methods to read data from HDFS, C*, S3, HBase, etc.

Operations on RDD

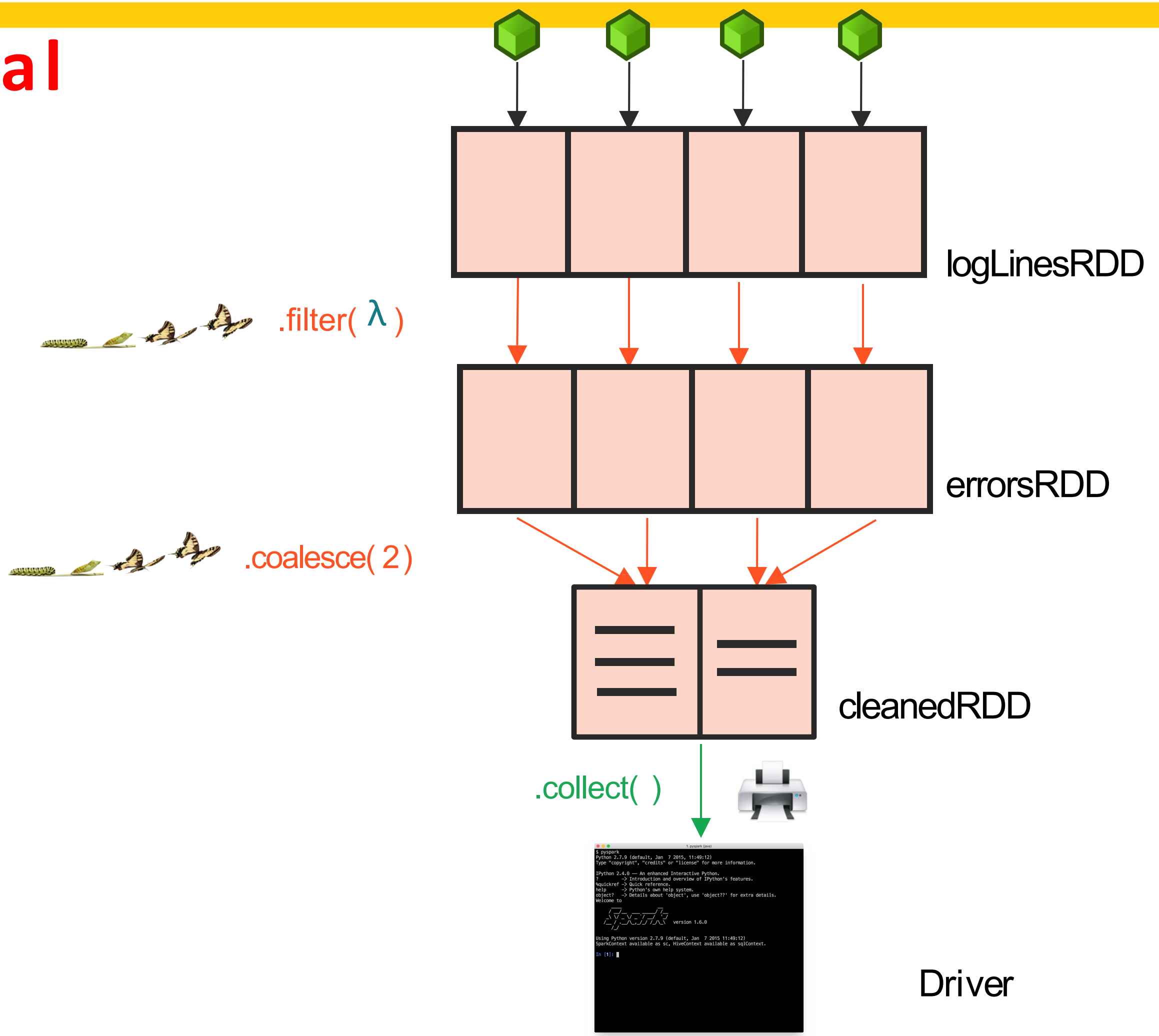
- Two types of operations: **transformations and actions**
- Transformations are lazy (not computed immediately)
- Transformations are executed when an action is run
- Persist (cache) distributed data in memory or disk


hadoop HDFS
Error, ts, msg1 Warn, ts, msg2 Error, ts, msg1
Info, ts, msg8 Warn, ts, msg2 Info, ts,msg8
Error,ts, msg3 Info, ts, msg5 Info, ts, msg5
Error, ts, msg4 Warn, ts, msg9 Error, ts, msg1
logLinesRDD
(input/base RDD)
.filter(λ)
Error,ts, msg1 Error, ts, msg1
Error,ts, msg3
Error,ts, msg4 Error, ts, msg1
errorsRDD

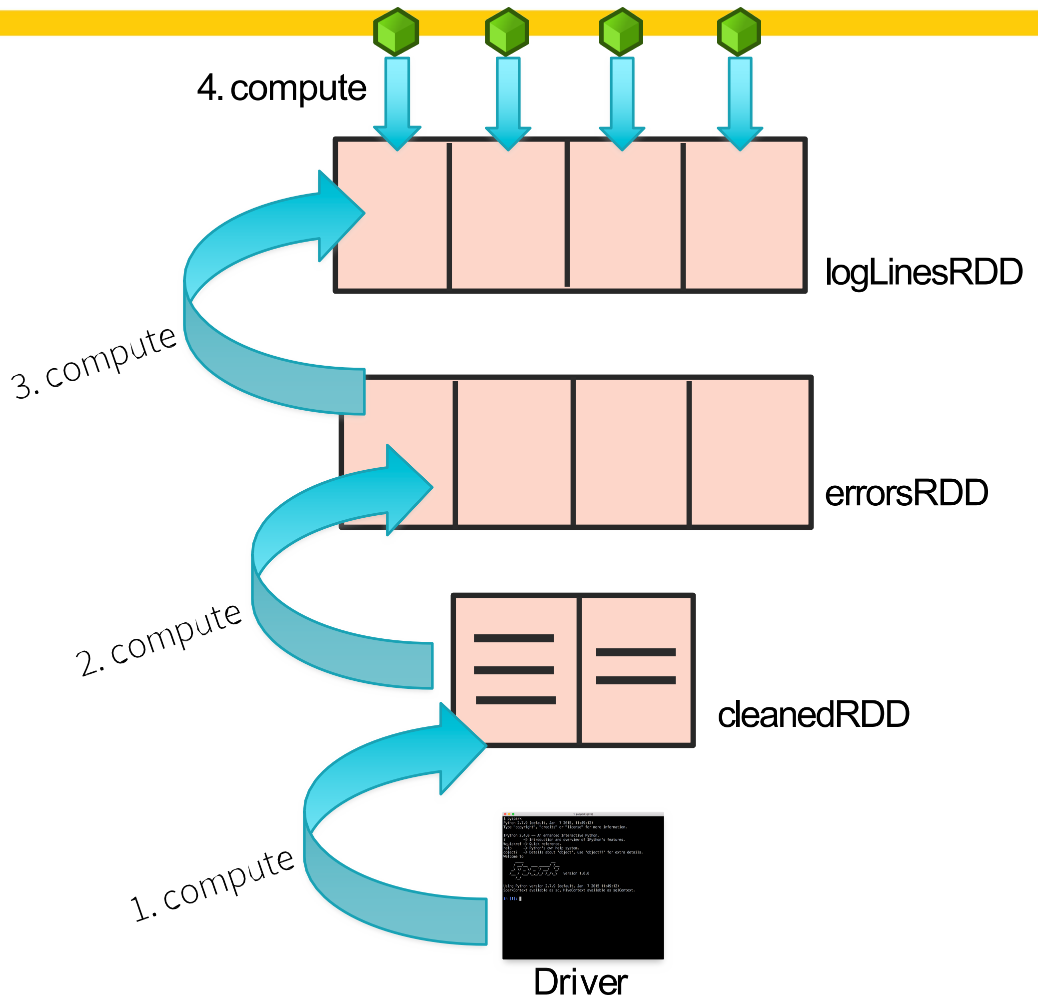

Error, ts,
msg1 Error,
ts, msg1
Error, ts, msg3
Error, ts,
msg4 Error,
ts, msg1
errorsRDD
.coalesce(2)
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.collect( )
Driver


Execute
DAG!
.collect( )
Driver

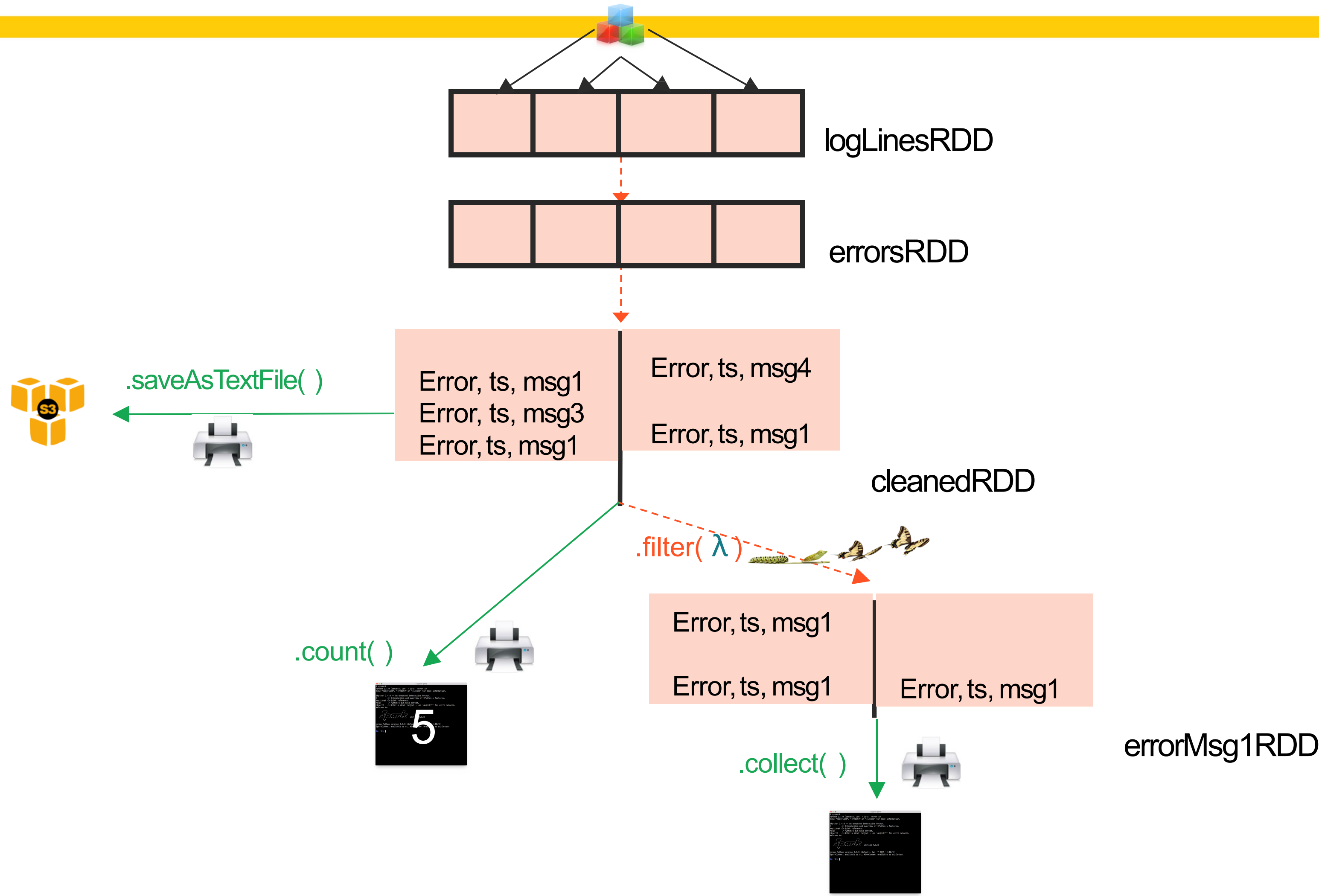
Logical


logLinesRDD
.filter(λ)
errorsRDD
.coalesce(2)
cleanedRDD
.collect( )
Driver

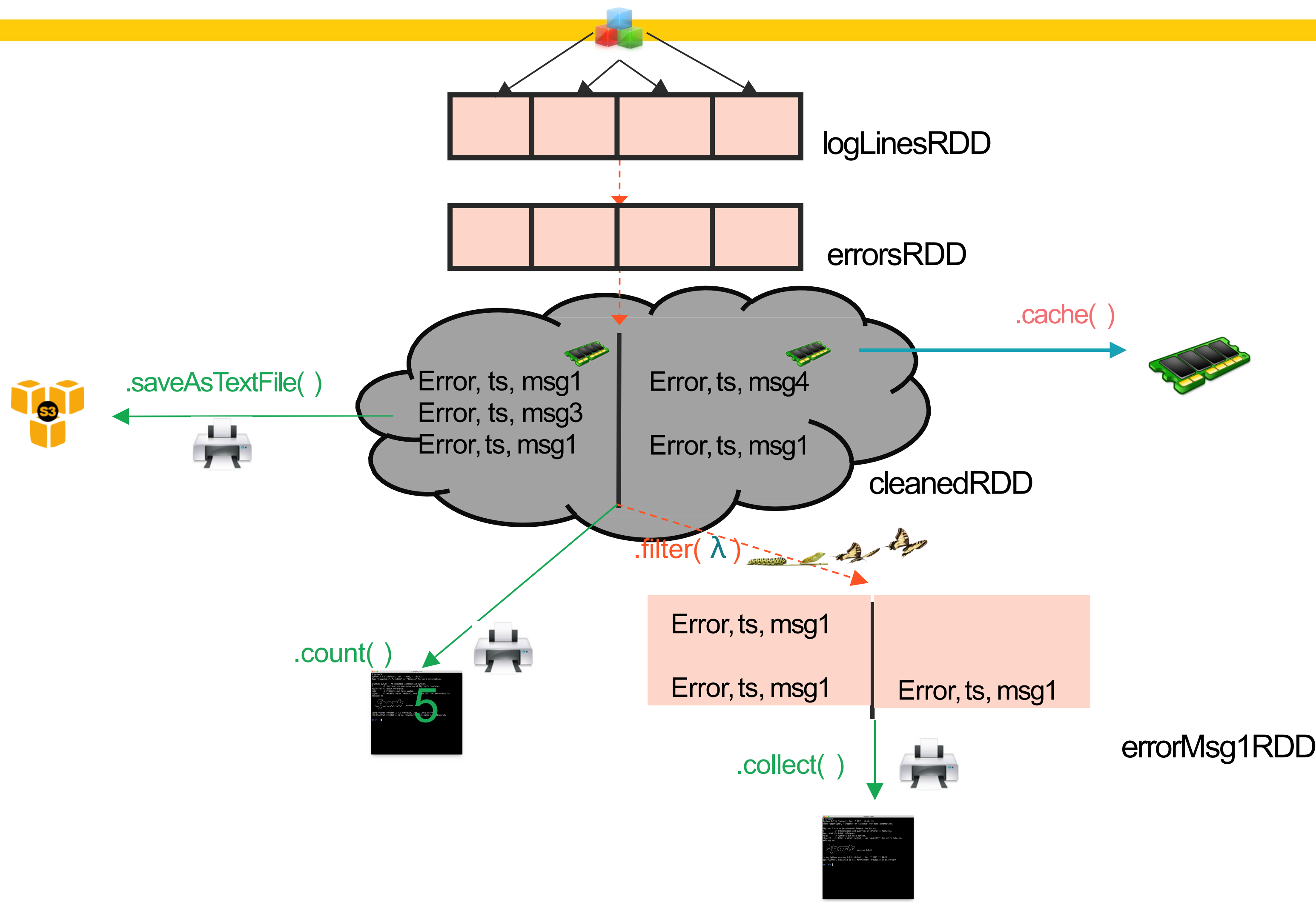
Physical



4. compute
logLinesRDD
3. compute
errorsRDD
2. compute
cleanedRDD
1. compute
Driver

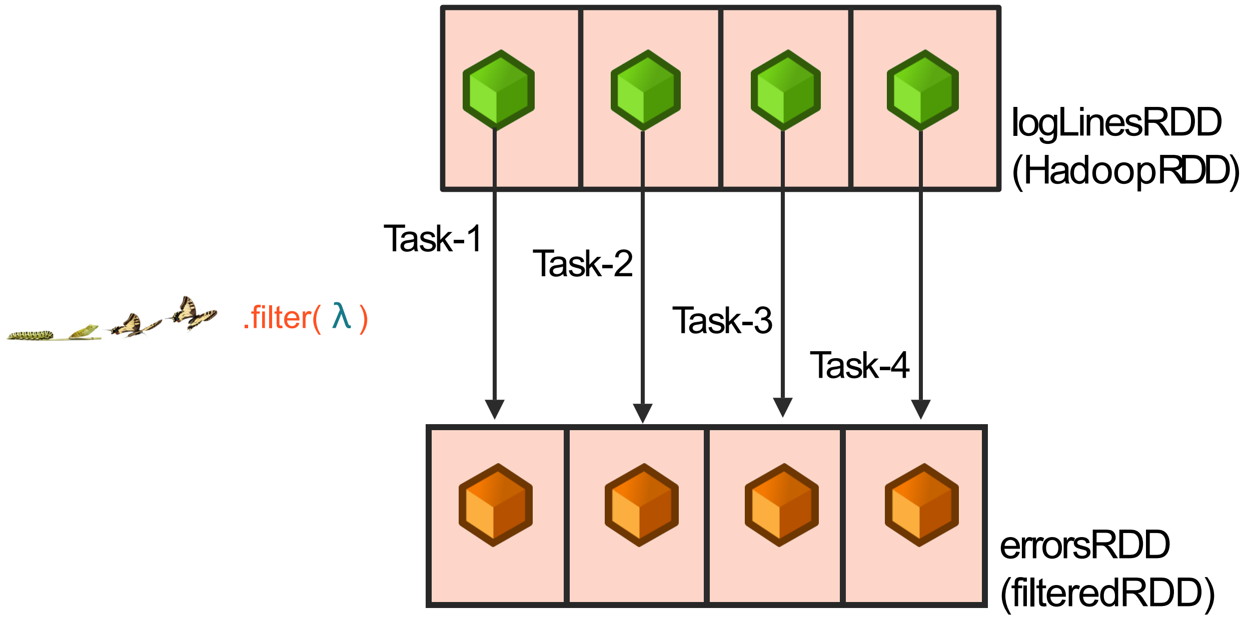


logLinesRDD
errorsRDD
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.saveAsTextFile( )
.filter(λ)
.count( )
5
Error, ts, msg1
Error, ts, msg1
Error, ts, msg1
errorMsg1RDD
.collect( )

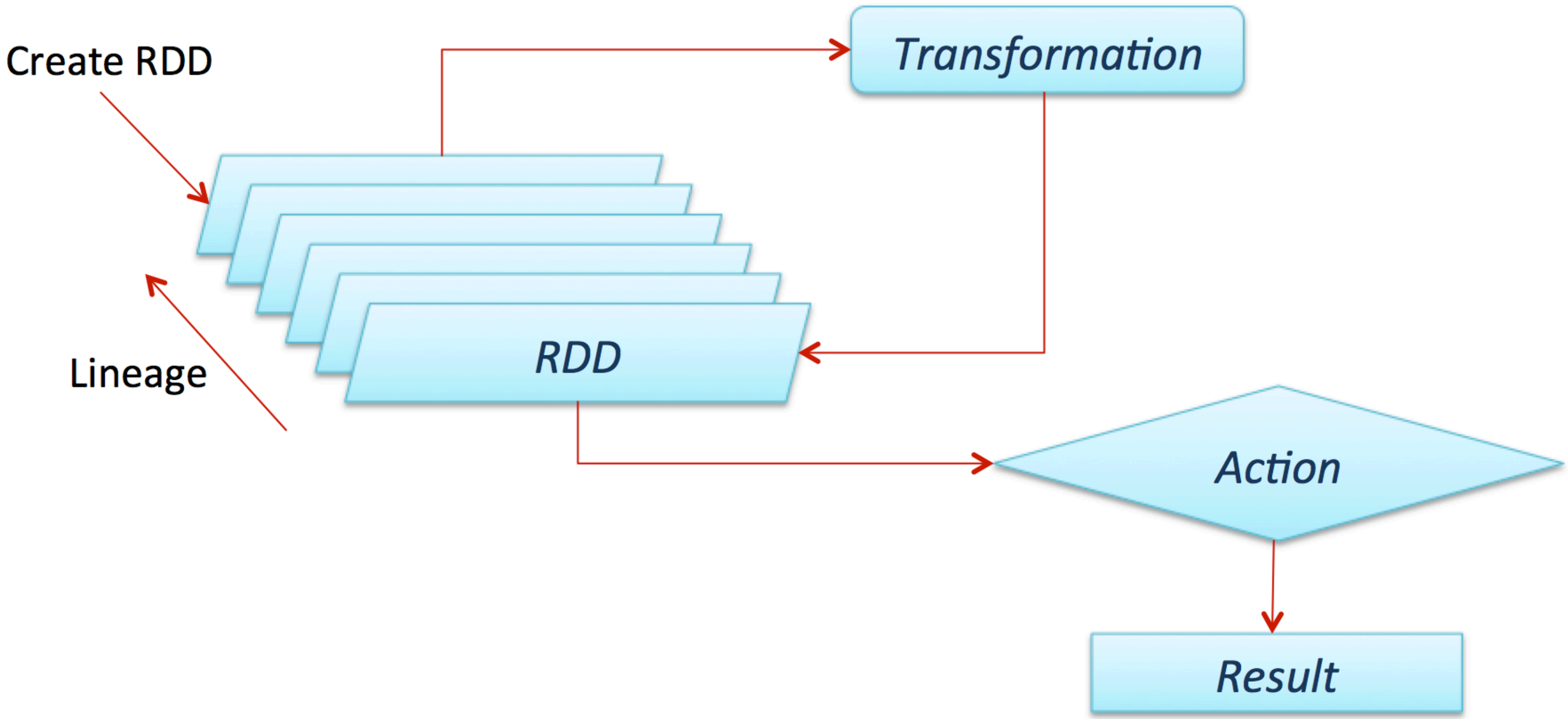


logLinesRDD
errorsRDD
.cache( )
.saveAsTextFile( )
S3
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.filter(λ)
.count( )
5
Error, ts, msg1
Error, ts, msg1
Error, ts, msg1
errorMsg1RDD
.collect( )

Partition >> Task >> Partition



Lineage mechanism



Notes on RDD

- Initial RDD on **disks** (HDFS, etc)
- Intermediate RDD on **RAM**
- Fault recovery based on **lineage**
- RDD operations is distributed

Spark program routine

- Create RDD from external data or create RDD from a collection in driver program
- Lazily transform them into new RDD
- cache() some RDD for reuse
- Perform actions to execute parallel computation and produce results

Transformations (lazy)

```
map()
flatMap()
filter()
mapPartitions()
mapPartitionsWithIndex()
sample()
union()
intersection()
distinct()
groupByKey()
reduceByKey()
sortByKey()
join()
cogroup()
cartesian()
pipe()
coalesce()
repartition()
partitionBy()
...
...
```

Actions

```
reduce()
collect()
count()
first()
take()
takeSample()
saveToCassandra()
takeOrdered()
saveAsTextFile()
saveAsSequenceFile()
saveAsObjectFile()
countByKey()
foreach()
…
```

Some Types of RDDs

- HadoopRDD
- FilteredRDD
- MappedRDD
- PairRDD
- ShuffledRDD
- UnionRDD
- PythonRDD

- DoubleRDD
- JdbcRDD
- JsonRDD
- VertexRDD
- EdgeRDD

- CassandraRDD *(DataStax)*
- GeoRDD *(ESRI)*
- EsSpark *(ElasticSearch)*

Hands-on with RDDs

DataFrame

- A primary abstraction in Spark 2.0
  - 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 or pandas DataFrame
  - From files in HDFS or other storage system

Using DataFrame

- >>> 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)]

Transformations

- Create new DataFrame from an existing one
- Use lazy evaluation
  - Nothing executes
  - Spark saves recipe for transformation source

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

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

- >>> 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 Programming Routine (Dataframe)

- Create DataFrames from external data or createDataFrame from a collection in driver program
- Lazily transform them into new DataFrames
- cache() some DataFrames for reuse
- Perform actions to execute parallel computation and produce results

Machine Learning Library (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

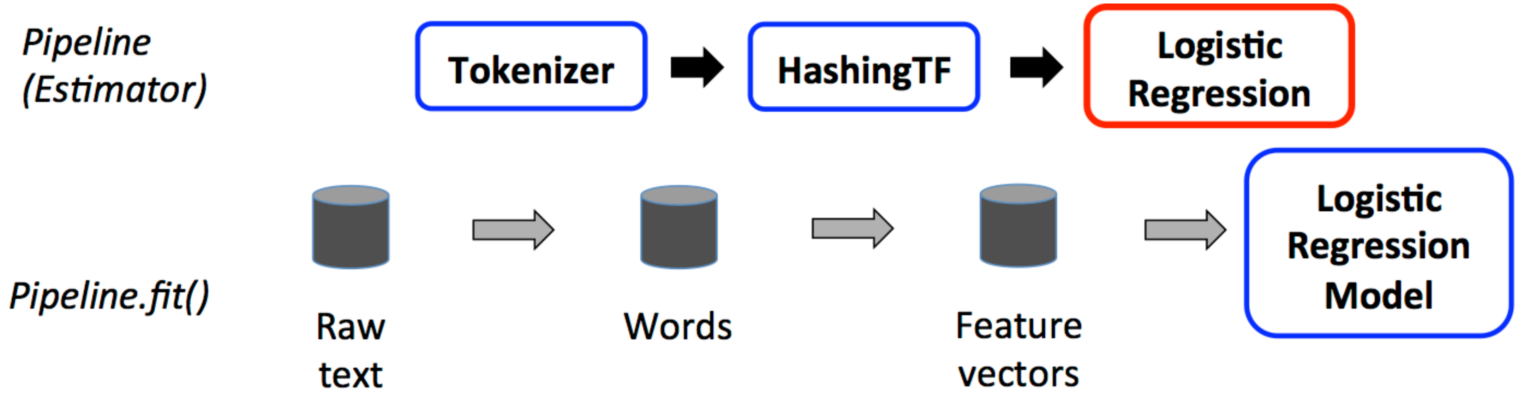
- A Transformer is a class which can transform a DataFrame into another DataFrame
- A Transformer implements transform()
- Examples
  - HashisngTF
  - LogisticRegressionModel
  - Binarizer

ML: Estimator

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

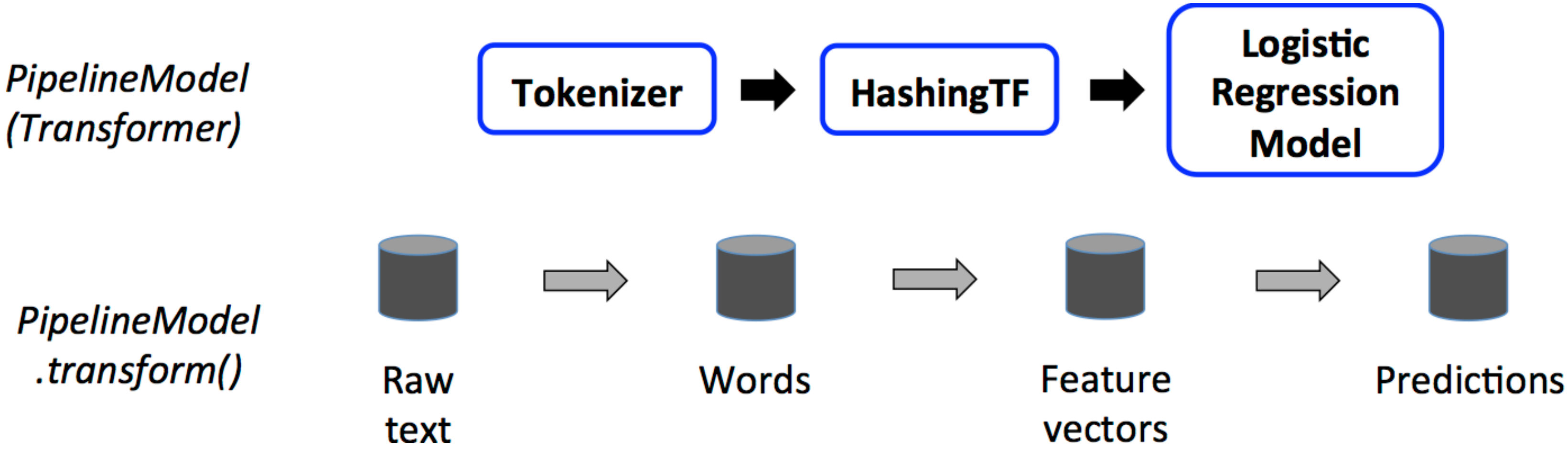
ML: Pipeline

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



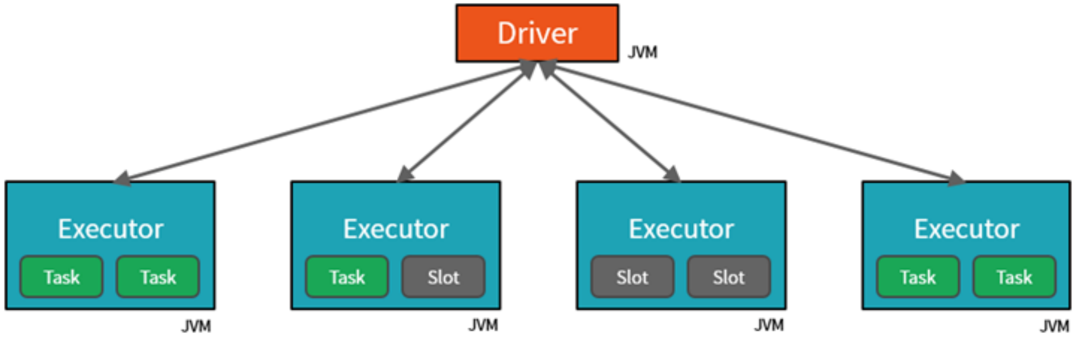
ML: PipelineModel

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



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)

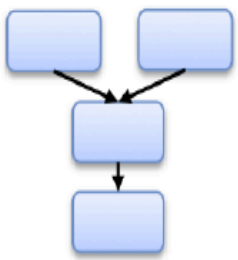
Architecture(2)

- 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

Lifetime of a job in Spark

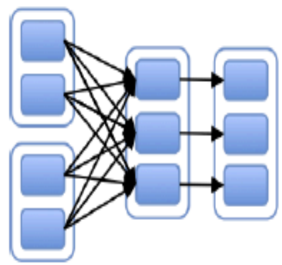
RDD Objects



```
rdd1.join(rdd2)
.groupBy(...)
.filter(...)
```

Build the operator DAG

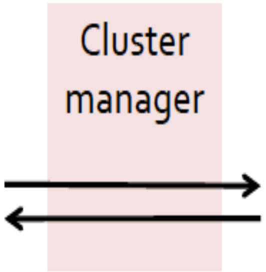
DAG Scheduler



Split the DAG into *stages* of *tasks*

Submit each stage and its tasks as ready

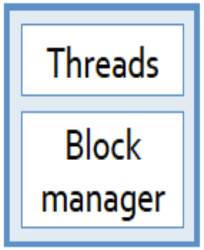
Task Scheduler



Launch tasks via Master

Retry failed and strag-gler tasks

Worker

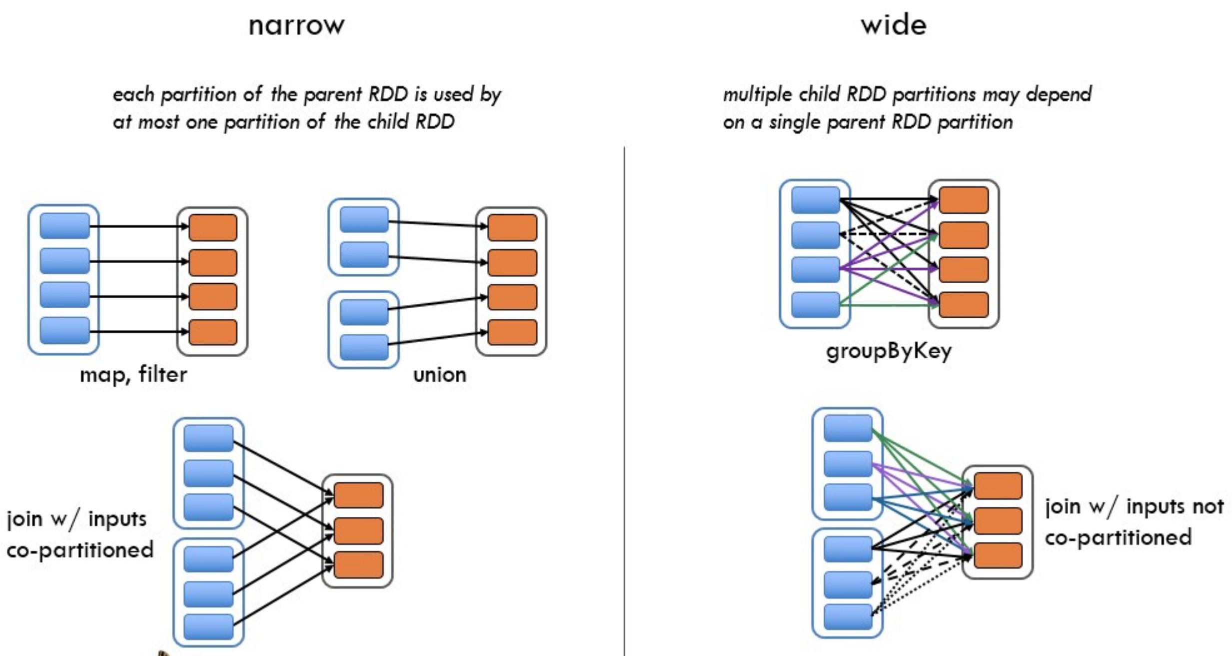


Execute tasks

Store and serve blocks

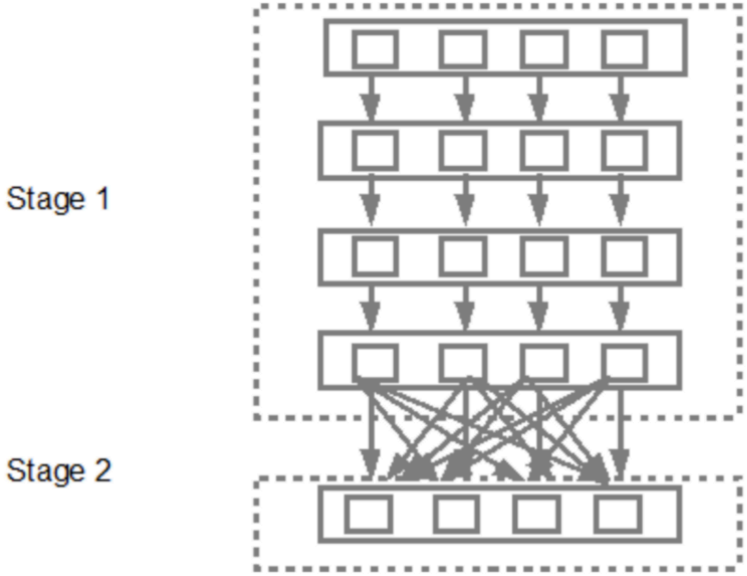
Narrow vs. Wide Dependencies

- Two types of transformations
  - Narrow transformation—doesn’t require the data to be shuffled across the partitions. for example, Map, filter etc..
  - wide transformation—requires the data to be shuffled for example, reduceByKey etc..



Spark transformations & stages

- The narrow transformations will be grouped (pipe-lined) together into a single stage



Summary

- Hadoop: Scalable and economical data storage and processing
- Spark: a unified analytics engine for large-scale data processing

Thank you for your attention!

Q&A