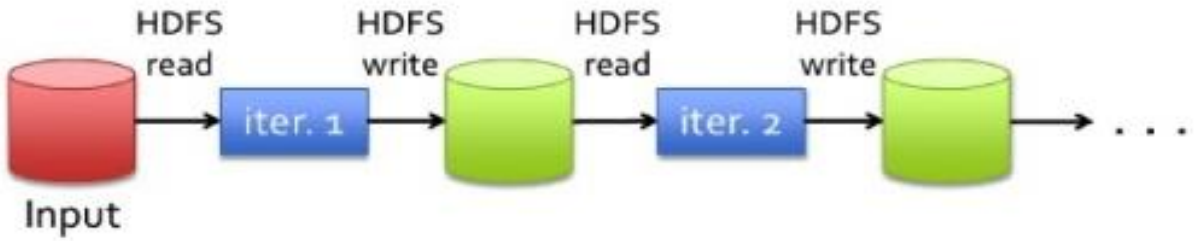
Chương 6
Các kĩ thuật xử lý dữ liệu lớn theo khối - phần 2

Apache Spark

Một nền tảng xử lý dữ liệu hợp nhất cho dữ liệu lớn

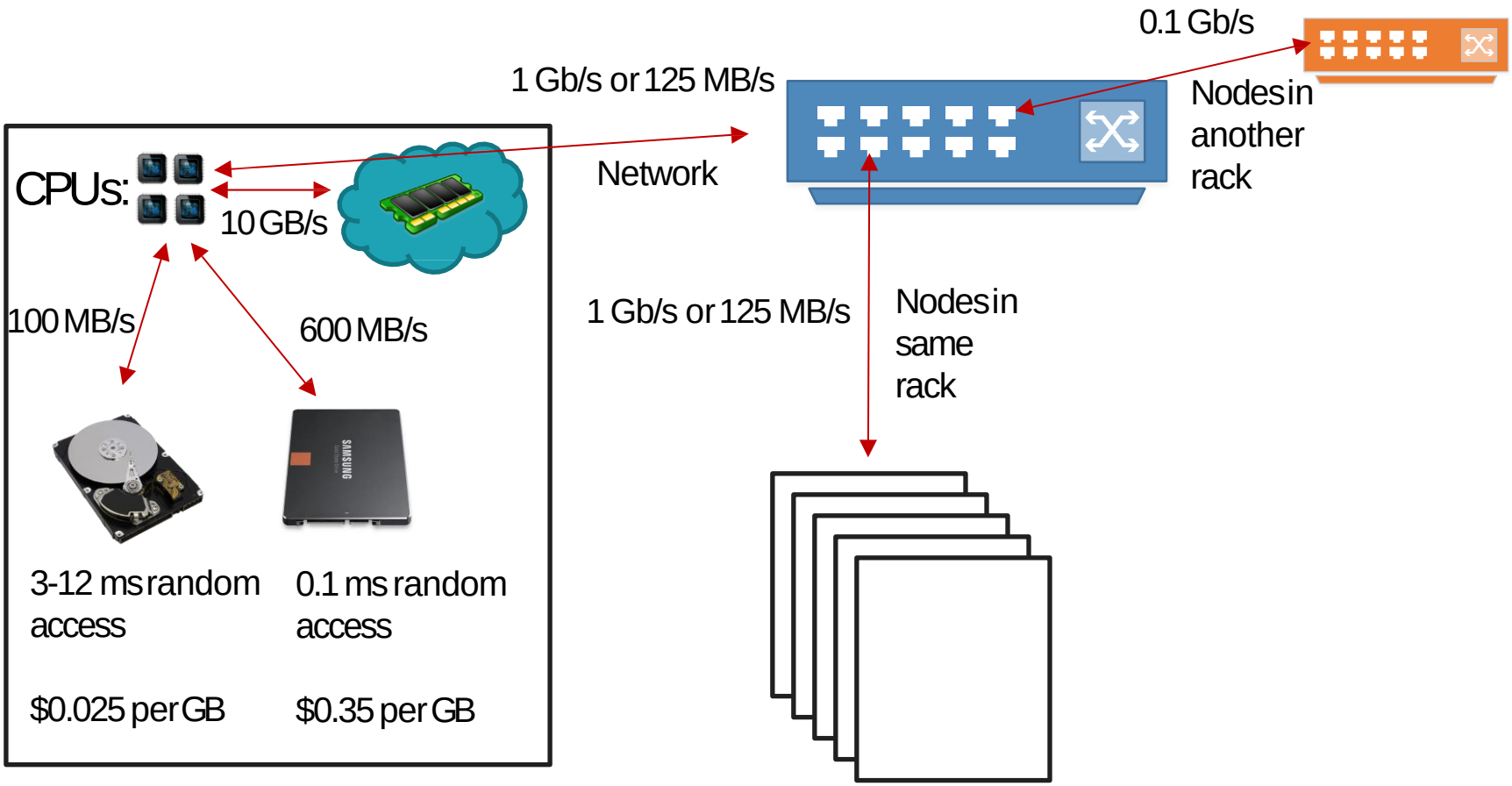
MapReduce với chuỗi các jobs

- Iterative jobs với MapReduce đòi hỏi thao tác I/O với dữ liệu trên HDFS



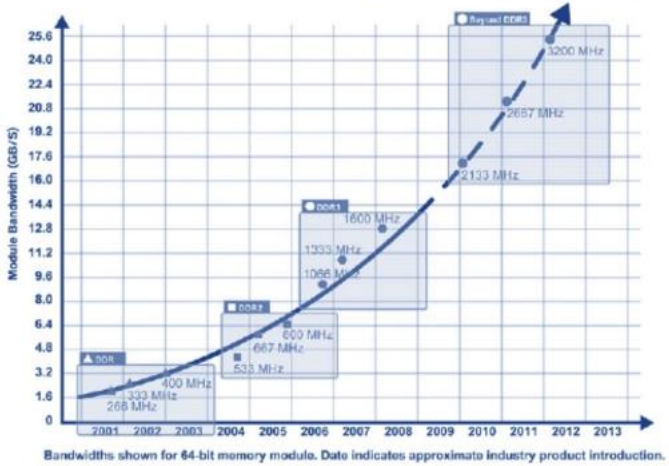
- Thực tế I/O trên ổ đĩa cứng rất chậm!

Toàn cảnh về I/O dữ liệu



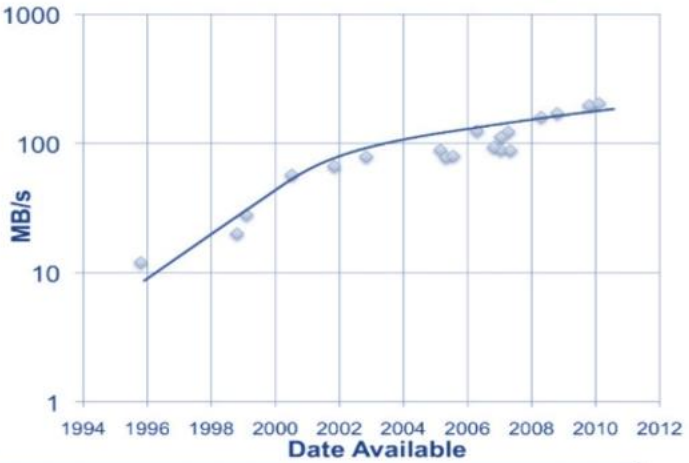
RAM có khả năng thay thế ổ đĩa cứng

- RAM throughput increasing **exponentially**



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

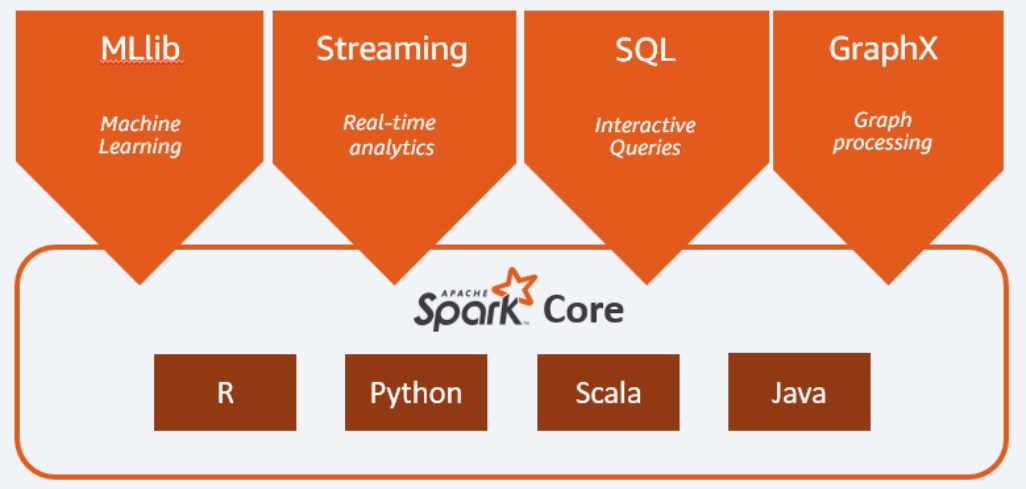
- Disk throughput increasing **slowly**



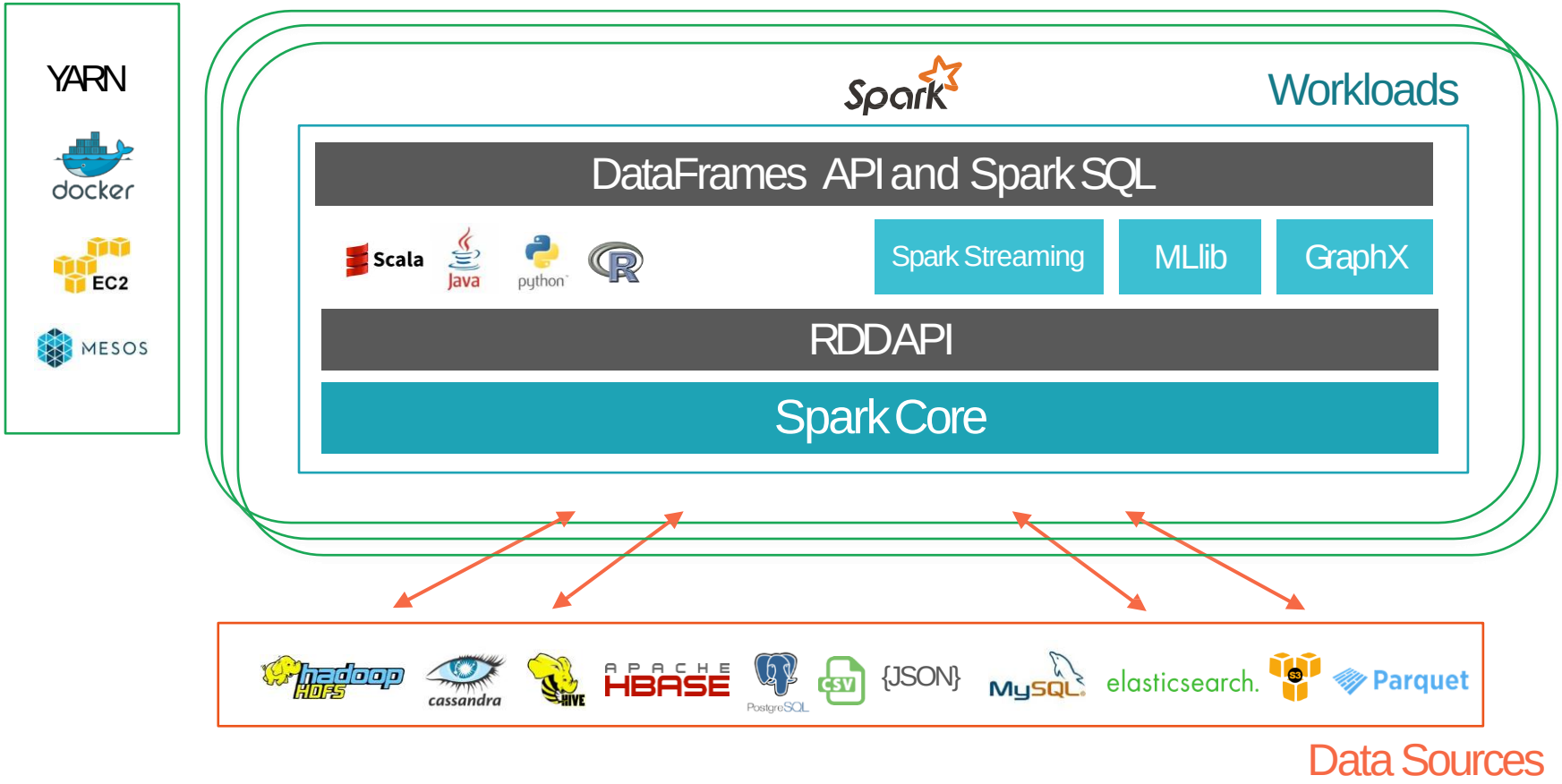
Memory-locality key to interactive response times

Một nền tảng xử lý dữ liệu hợp nhất cho dữ liệu lớn

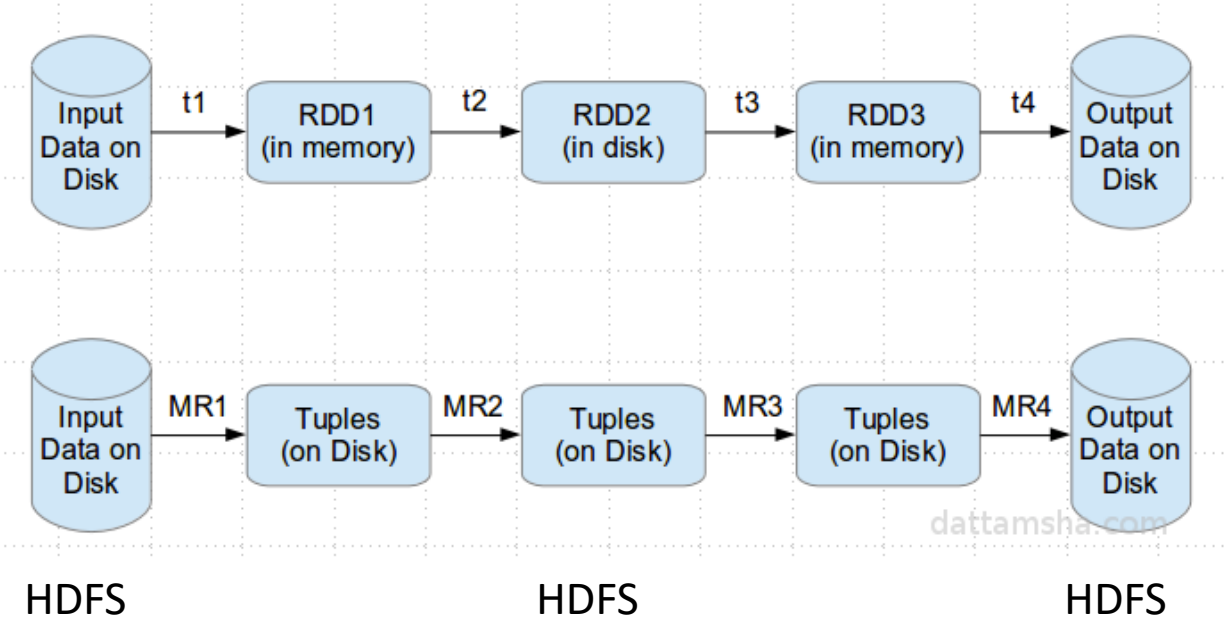
- Hỗ trợ tốt hơn MapReduce trong
  - Các giải thuật có tính lặp - Iterative algorithms
  - Khai phá dữ liệu trong môi trường tương tác - Interactive data mining
- Khả năng chịu lỗi, khai thác tính địa phương của dữ liệu, tính khả mở
- Ẩn đi sự phức tạp cua của môi trường phân tán khi lập trình



Environments



Khai thác bộ nhớ trong thay vì ổ đĩa HDD



Sự khác nhau giữa Spark và MapReduce

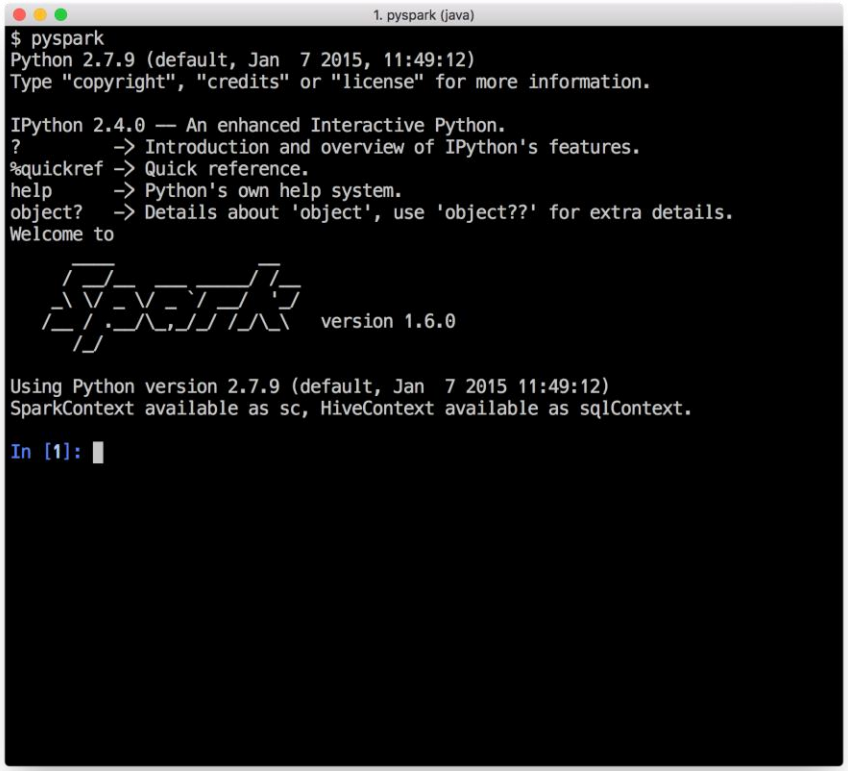
	Apache Hadoop MR	Apache Spark
Storage	Chỉ sử dụng HDD	Sử dụng cả bộ nhớ trong và HDD
Operations	Hai thao tác Map và Reduce	Bao gồm nhiều thao tác biến đổi (transformations) và hành động (actions) trên dữ liệu
Execution model	Xử lý theo khối – batch	Theo khối, tương tác , luồng
Languages	Java	Scala, Java, Python và R

So sánh hiệu năng Spark và MapReduce

	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)

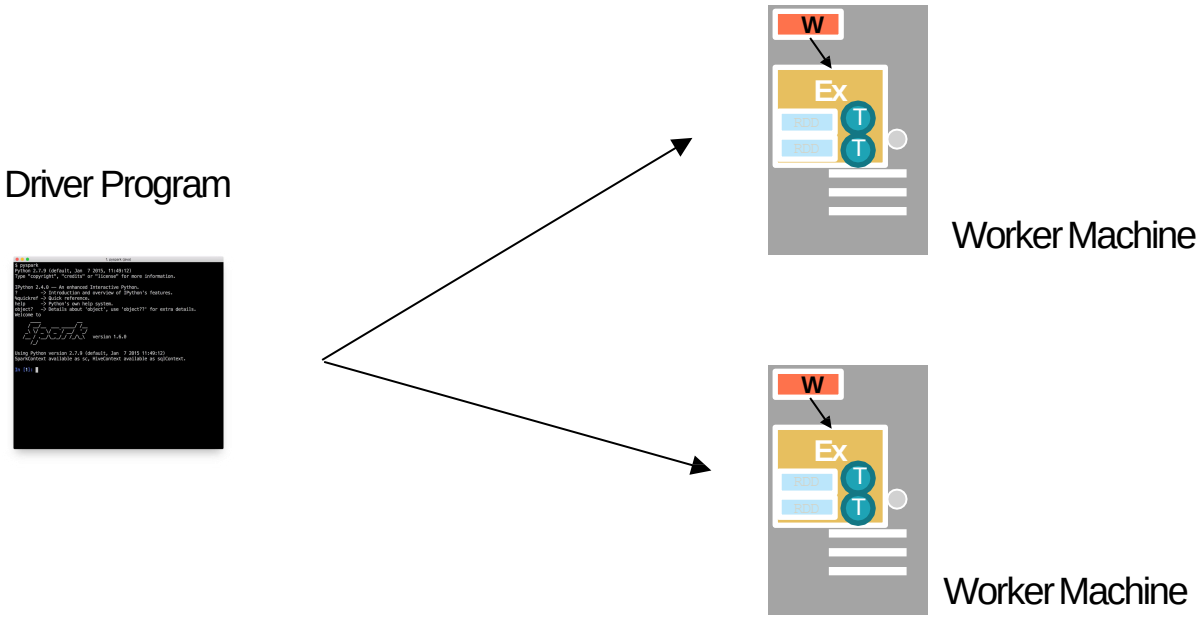
https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html

Giao diện dòng lệnh tương tác



(Scala, Python and Ronly)

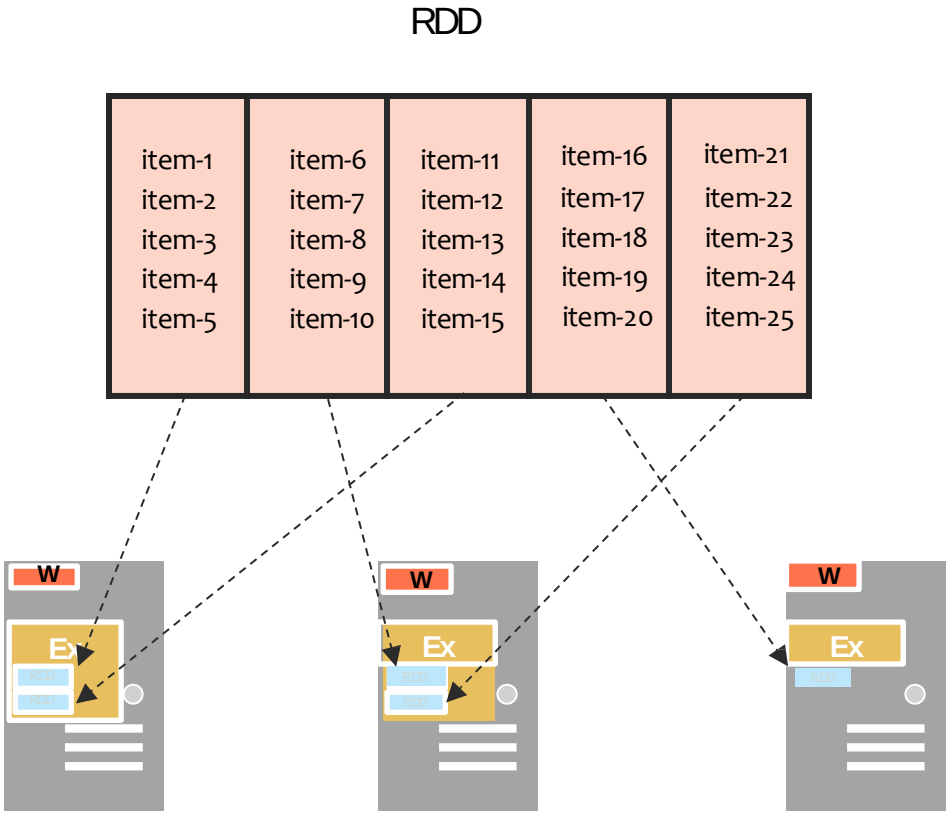
Thực thi chương trình Spark



Resilient Distributed Dataset (RDD)

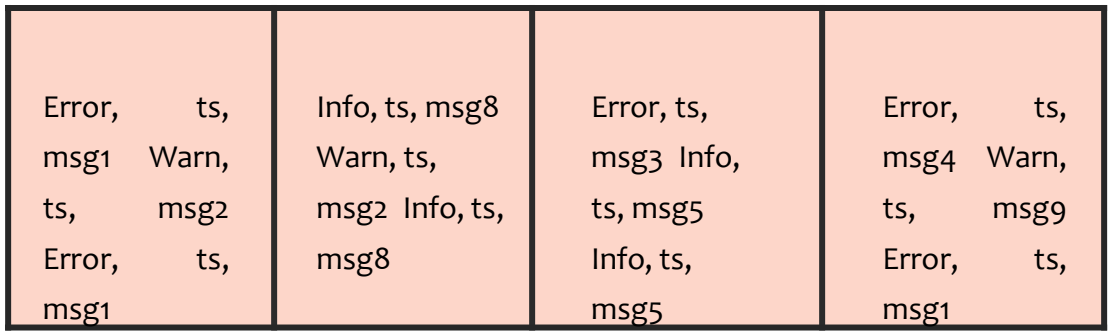
- RDDs là cấu trúc dữ liệu song song, có khả năng chịu lỗi (***fault-tolerant, parallel data structures)*** mà cho phép người dùng chỉ định lưu trữ dữ liệu trung gian trên bộ nhớ (***intermediate results in memory)***, điều khiển sự phân chia để tối ưu hóa việc phân tán dữ liệu, và cũng có thể thay đổi, chỉnh sửa những dữ liệu này sử dụng một tập các thao tác rất đa dạng (***a rich set of operators)***.
  - RDDs có khả năng tự động tái tạo lại khi bị lỗi
- RDD được thiết kế tối ưu cho các biến đổi thô, theo lô (coarse-grained transformations) thay vì hỗ trợ các thao tác cập nhật quá chi tiết (fine-grained updates)
  - Vd., map, filter và join mà tác động tới nhiều bản ghi dữ liệu đồng thời thay vì là các thao tác chỉ cập nhật lên một đối tượng dữ liệu riêng lẻ

Sự phân vùng của RDD và khả năng song song hóa



Khởi tạo RDD

- Một RDD cơ sở có thể được tạo theo 2 cách
  - Song song hóa một collection (ví dụ mảng trong Python)
  - Đọc dữ liệu từ một nguồn bên ngoài (S3, C*, HDFS, etc)



Phương thức Parallelize

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

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

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

- Một mảng làm thm số đầu vào cho phương thức parallelize của SparkContext
- Thường được sử dụng để thử nghiệm vì đòi hỏi toàn bộ dữ liệu phải có sẵn trên bộ nhớ trong.

RDD khởi tạo từ tệp tin văn bản

```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")
```

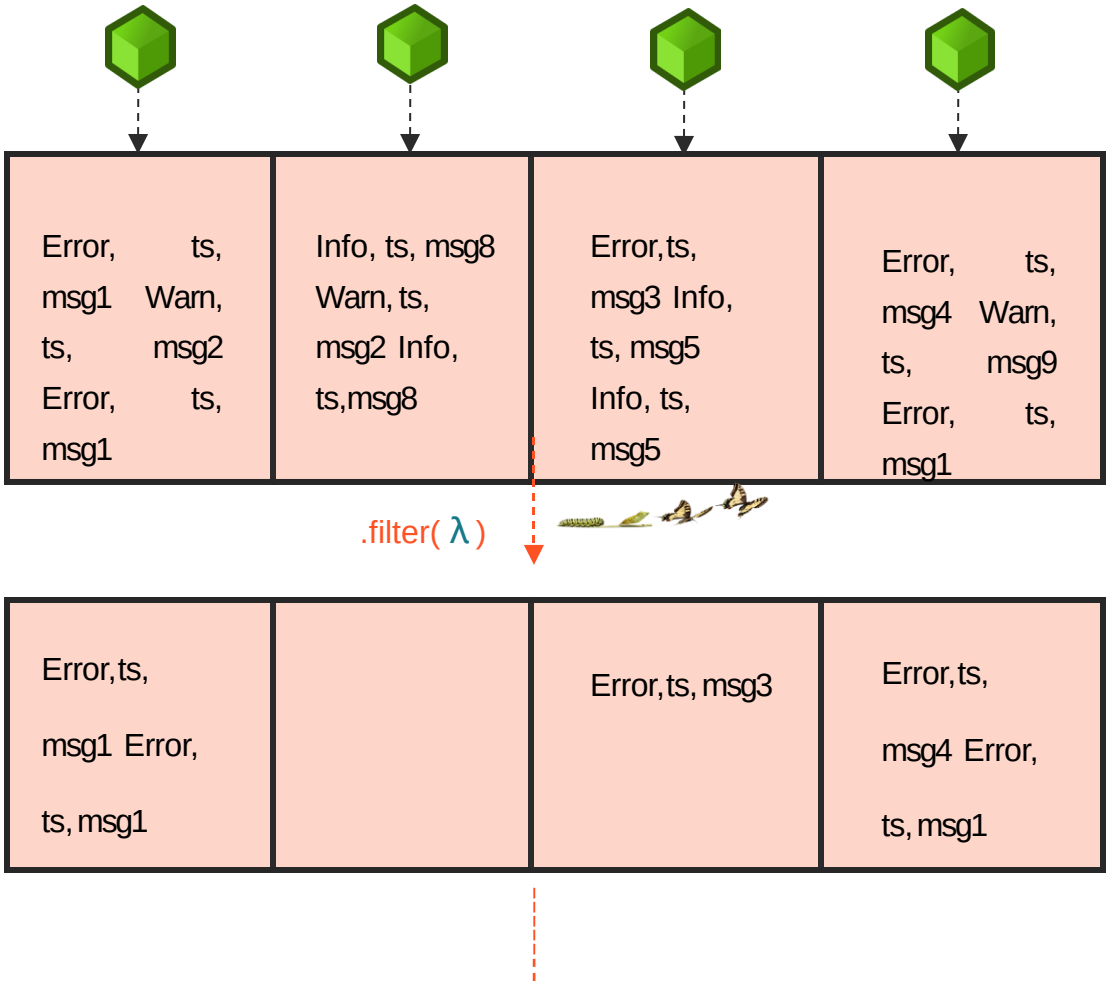
```java
// Read a local txt file in Java
JavaRDD<String> lines = sc.textFile("/path/to/README.md");
```

Ngoài ra có các phương thức đọc từ HDFS, C*, S3, HBase, etc.

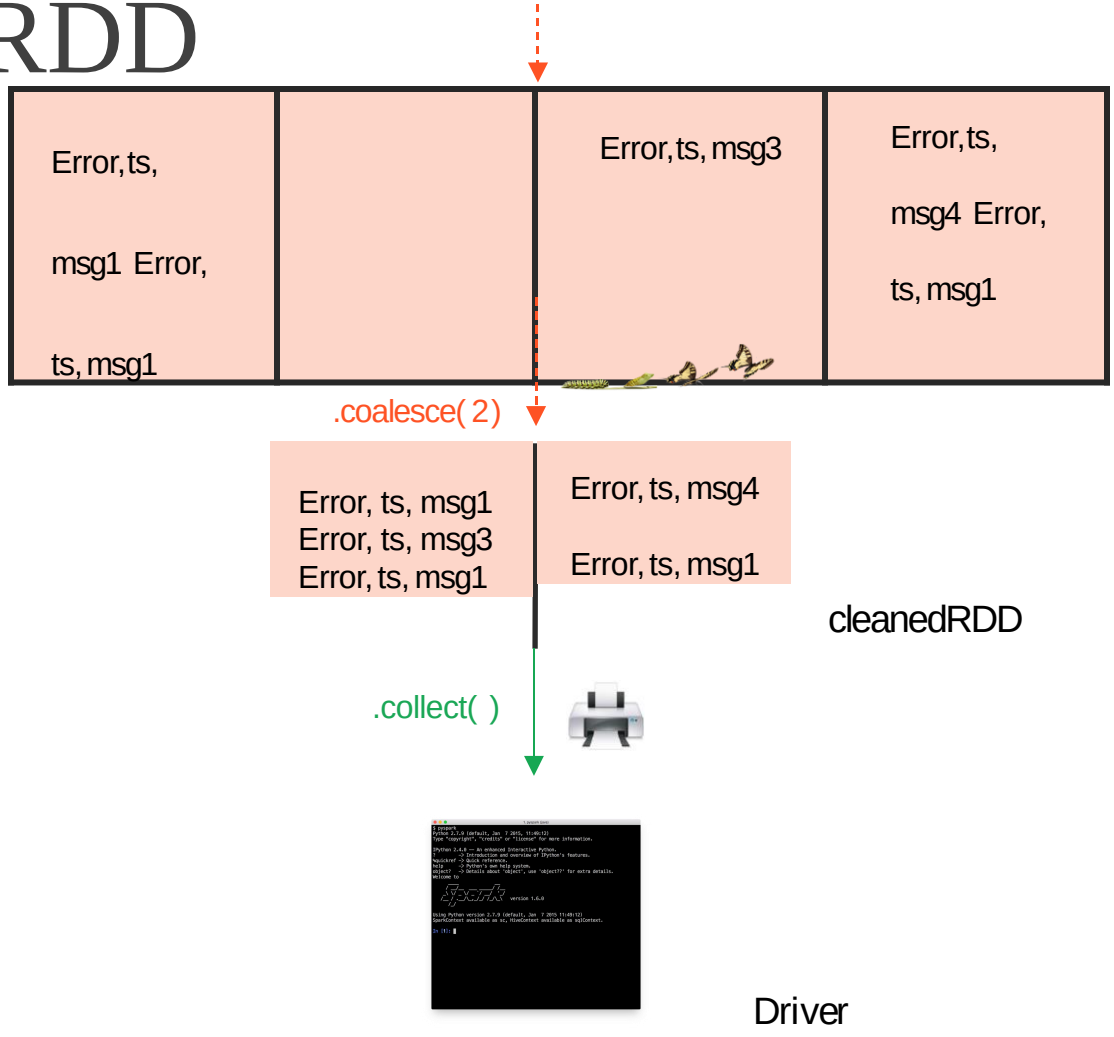
Các thao tác trên RDD

- 2 dạng thao tác: **transformations and actions**
- Transformations là lazy (không được tính toán ngay lập tức)
- Transformations chỉ được thực thi khi một action được gọi
- Có thể lưu trữ lâu dài (hoặc tạm thời) dữ liệu của RDD trên RAM và cứng

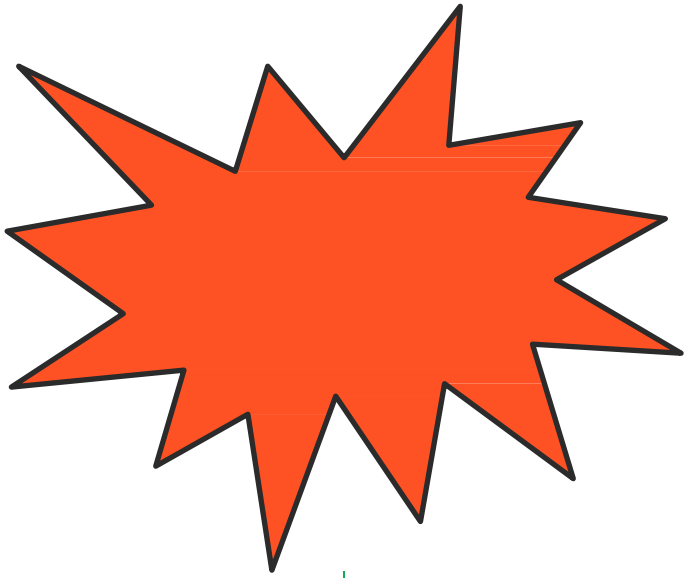
logLinesRDD



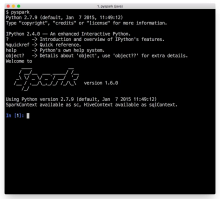
errorsRDD


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

Execute DAG!

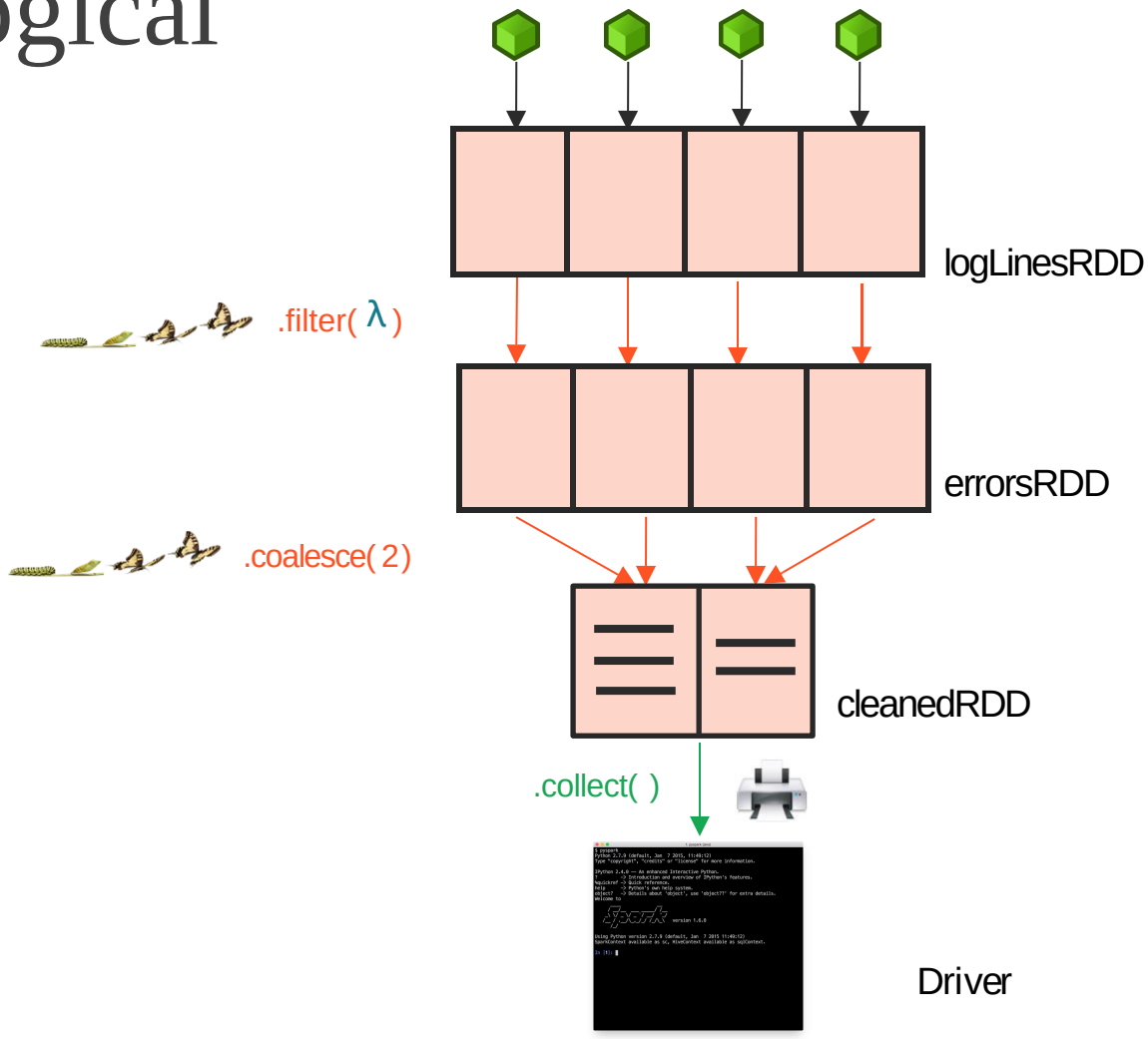


.collect( )

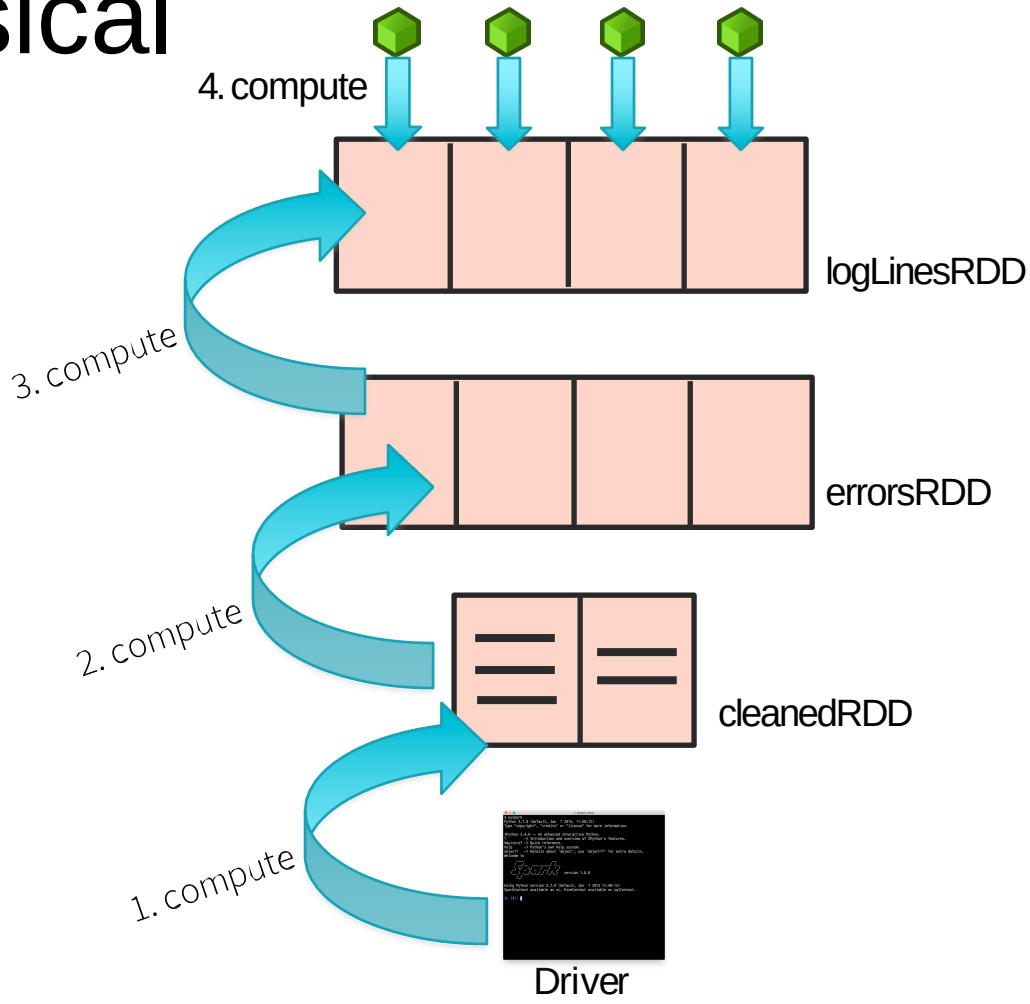


Driver

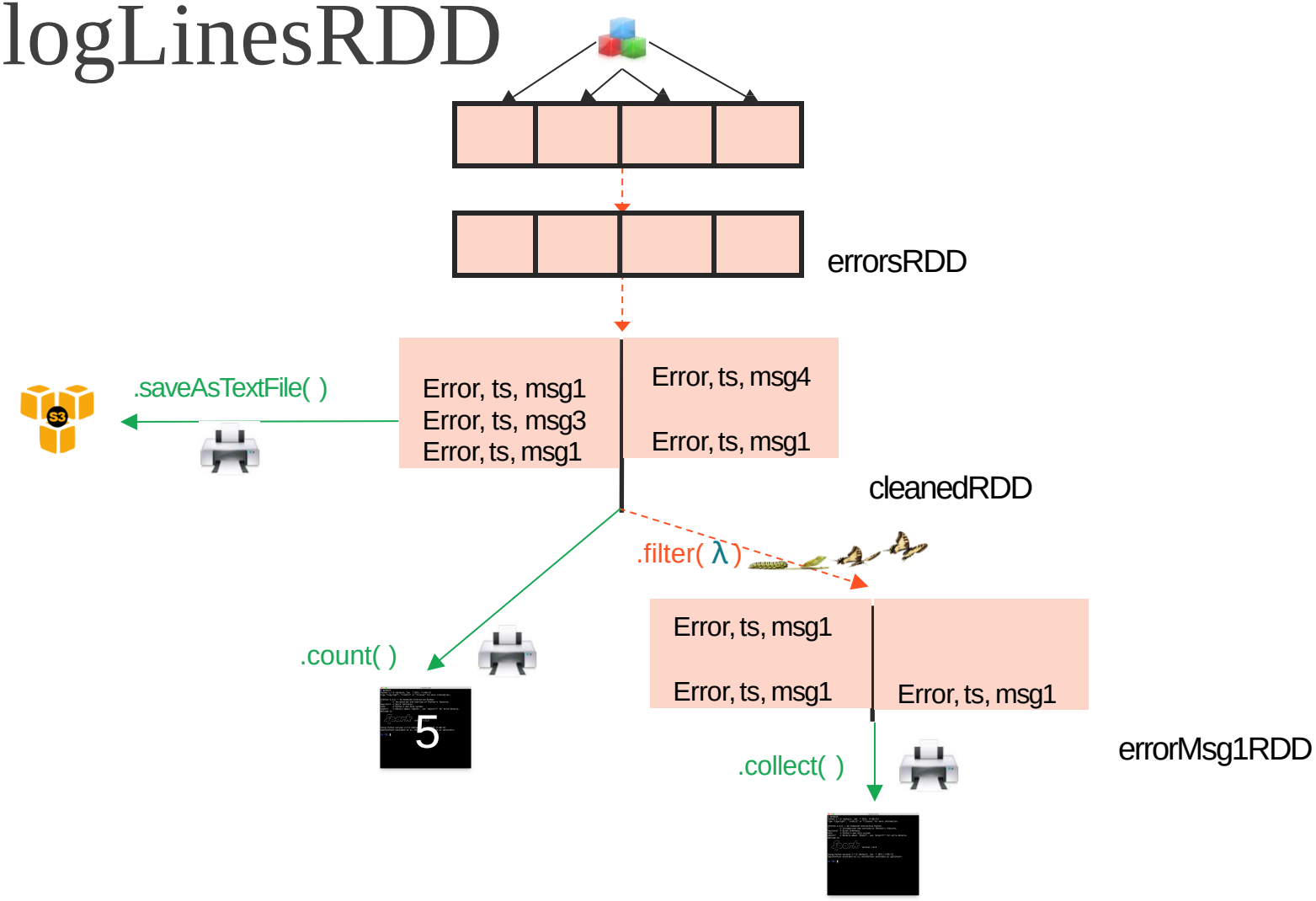
Logical


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

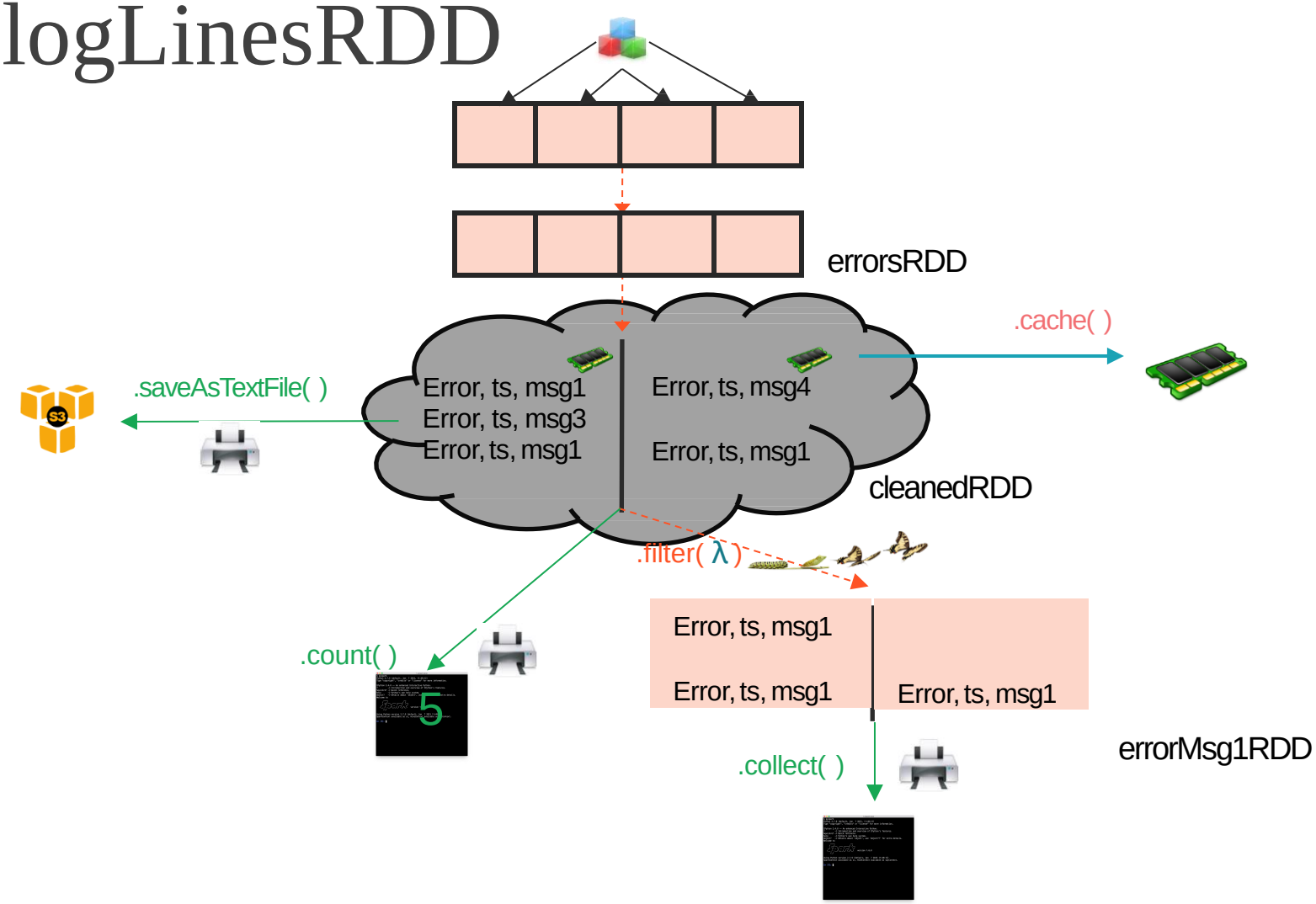
Physical


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

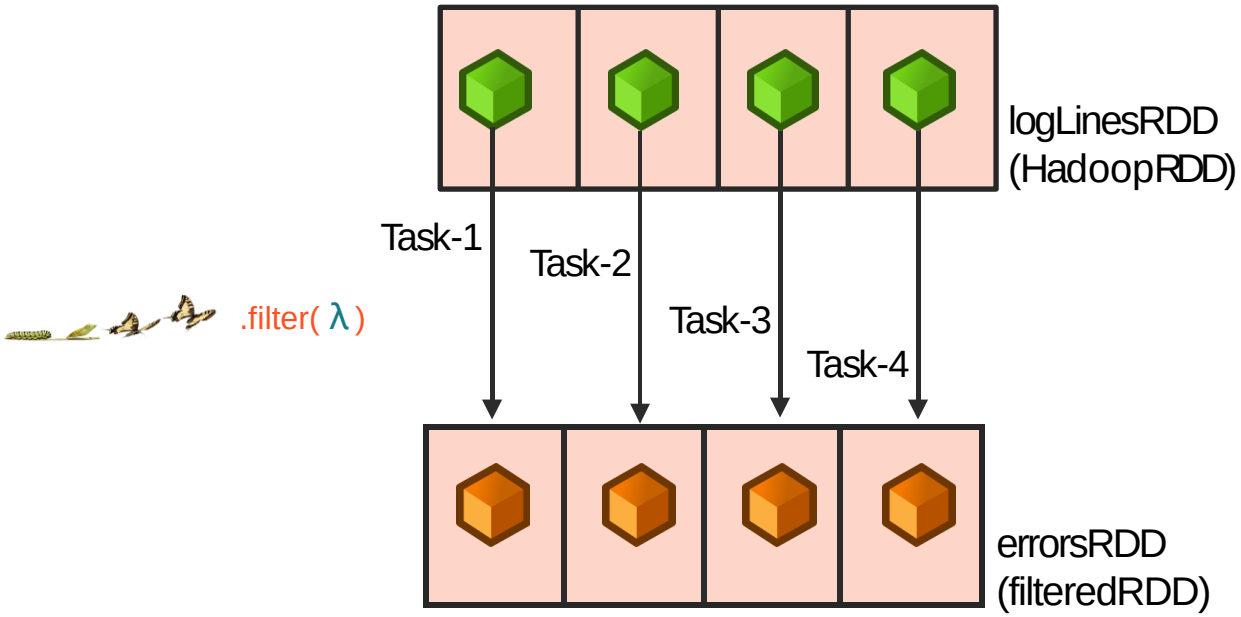
logLinesRDD



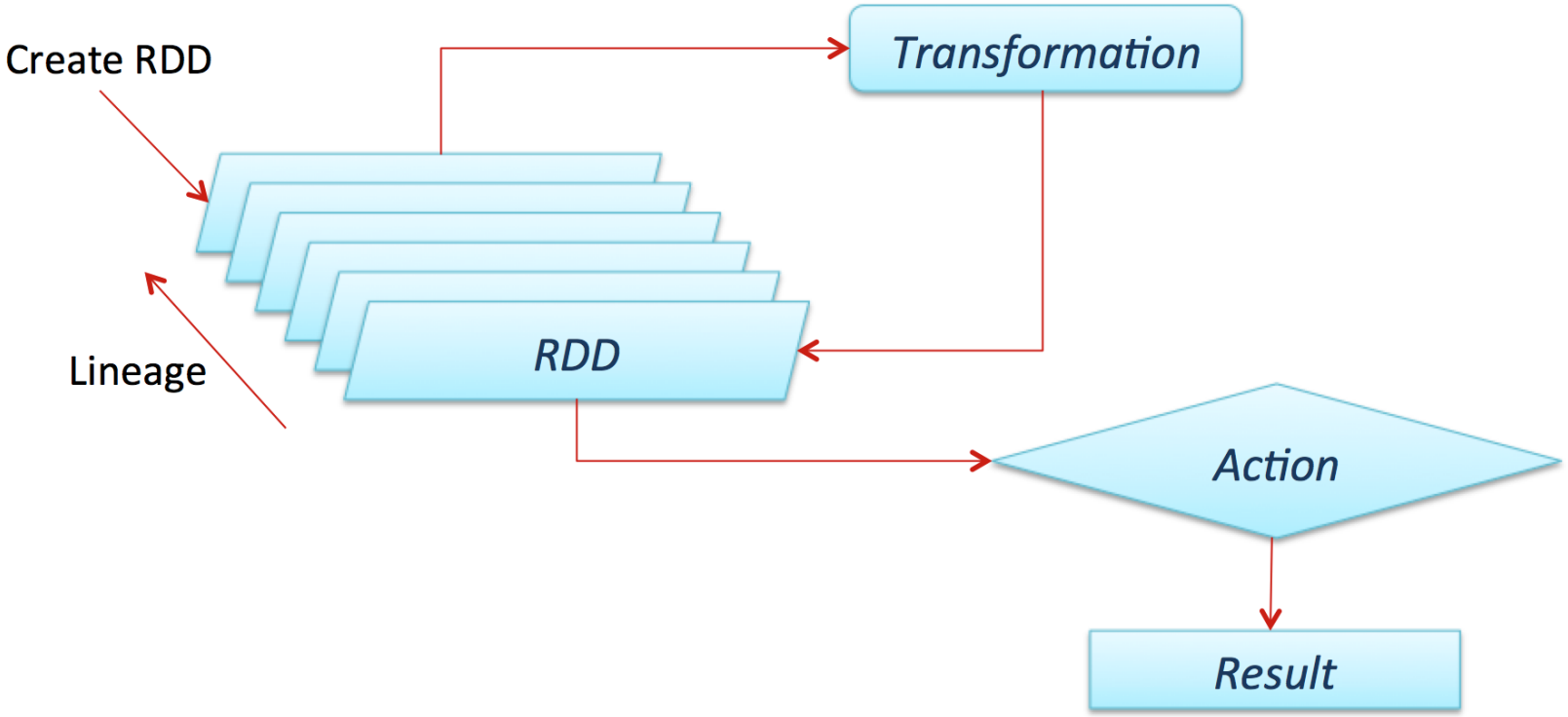
logLinesRDD



Partition >> Task >> Partition



Chịu lỗi sử dụng kỹ thuật Lineage



Một vài ghi nhớ về RDD

- Khởi tạo RDD từ disks (HDFS, etc)
- RDD trung gian nằm trên RAM
- Khôi phục lỗi sử dụng lineage
- Các thao tác trên RDD là phân tán

Quy trình lập trình trên Spark

- Tạo RDD từ nguồn dữ liệu bên ngoài hoặc từ các collections trong bộ nhớ
- Lazily transform thành các RDD trung gian
- cache() RDD để tái sử dụng
- Gọi các actions để thực thi tính toán song song và nhận về các kết quả

Transformations (lazy)

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

Actions

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

Một vài dạng RDD phổ biến

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

- DoubleRDD
- JdbcRDD
- JsonRDD
- VertexRDD
- EdgeRDD

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

RDD demo

- docker pull jupyter/pyspark-notebook
- docker run -p 8888:8888 jupyter/pyspark-notebook
- http://127.0.0.1:8888/tree hoặc
- http://192.168.99.100:8888/tree trên Windows
- Điền token key thích hợp
- Upload thư mục notebook vào container

Dataframe

- Là một lớp trừu tượng hóa dữ liệu chính của Spark 2.0
  - Có tính bất biến, không thay đổi được sau khi được khởi tạo
  - Lưu vết lịch sử các thay đổi theo cơ chế lineage để tái tạo lại khi bị lỗi
  - Cho phép thực thi các thao tác trên tập dữ liệu trên cơ chế song song hóa
- Để khởi tạo các Dataframe
  - Song song hóa các collections trong bộ nhớ (vd. List, set)
  - Biến đổi từ một dataframe đã có trên Spark hoặc từ pandas
  - Từ các tệp tin trên HDFS hoặc các hệ thống lưu trữ khác

Job	U-Tot	U-Avg	D-Tot	D-Avg	M-Tot	M-Avg	S-Tot	S-Avg
admin.	24.0	1.71	3398.0	2.65	13807.0	2.63	10113.0	2.61
blue-collar	54.0	3.86	1893.0	2.6	17051.0	2.55	4678.0	2.56
entrepreneur	11.0	3.67	485.0	2.71	2664.0	2.49	532.0	2.62
housemaid	17.0	5.67	396.0	2.46	2089.0	2.69	296.0	2.49
management	16.0	5.33	866.0	2.62	5192.0	2.49	1166.0	2.33
retired	17.0	3.4	865.0	2.49	3110.0	2.44	268.0	2.88
self-employed	11.0	2.2	341.0	2.56	2525.0	2.79	904.0	2.39
services	33.0	5.5	1345.0	2.53	5903.0	2.57	2990.0	2.63
student	1.0	1.0	19.0	2.11	112.0	2.73	1709.0	2.07
technician	21.0	1.75	2113.0	2.73	9348.0	2.55	5897.0	2.58
unemployed	11.0	2.2	303.0	2.44	1701.0	2.68	585.0	2.33
unknown	39.0	4.33	29.0	2.23	633.0	2.71	173.0	2.34

Sử dụng 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

- Tạo ra các dataframe mới từ các dataframe đã có
- Sử dụng cơ chế lazy evaluation
  - Thực thi muộn nhất có thể
  - Spark lưu lại các transformation để sử dụng khi gọi các action

Transformation	Description
select(*cols)	Lấy một vài cột trong dataframe
drop(col)	Trả về một dataframe mới bằng cách xóa đi một vài cột từ dataframe gốc
filter(func)	Trả về một dataframe mới bằng cách chỉ giữ lại các dòng thỏa mãn điều kiện lựa chọn của hàm func
where(func)	where là một tên gọi khác của filter
distinct()	Trả về dataframe mới không chứa các dòng trùng lặp
sort(*cols, **kw)	Trả về dataframe mới với các dòng được sắp xếp theo cols và theo trật tự quy định bởi kw

Ví dụ sử dụng 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

- Yêu cầu Spark thực thi các transformations
- Là kỹ thuật để lấy về dữ liệu kết quả trên 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

Ví dụ sử dụng 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()

Quy trình lập trình trên Spark với 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
- Các giải thuật học máy
  - Bao gồm các giải thuật phổ biến như phân lớp, hồi quy, phân nhóm, lọc cộng tác
- Xây đựng đặc trưng - Featurization
  - Trích rút, biến đổi, giảm chiều, lựa chọn các đặc trưng
- Các tiện ích
  - Đại số tuyến tính, thống kê, …

ML: Transformer

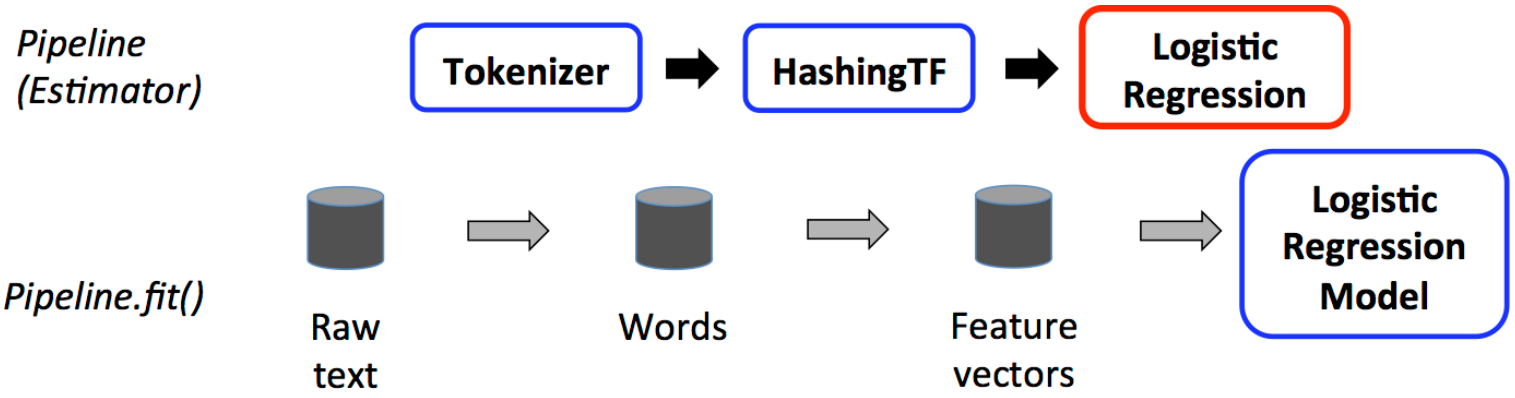
- Transformer là một lớp mà có thể tạo ra dataframe mới bằng cách biến đổi một dataframe ban đầu
- Transformer cài đặt phương thức transform()
- Ví dụ
  - HashisngTF
  - LogisticRegressionModel
  - Binarizer

ML: Estimator

- Estimator là lớp mà lấy đầu vào là 1 dataframe và trả về một transformer
- Estimator cài đặt phương thức fit()
- Ví dụ
  - LogisticRegression
  - StandardScaler
  - Pipeline

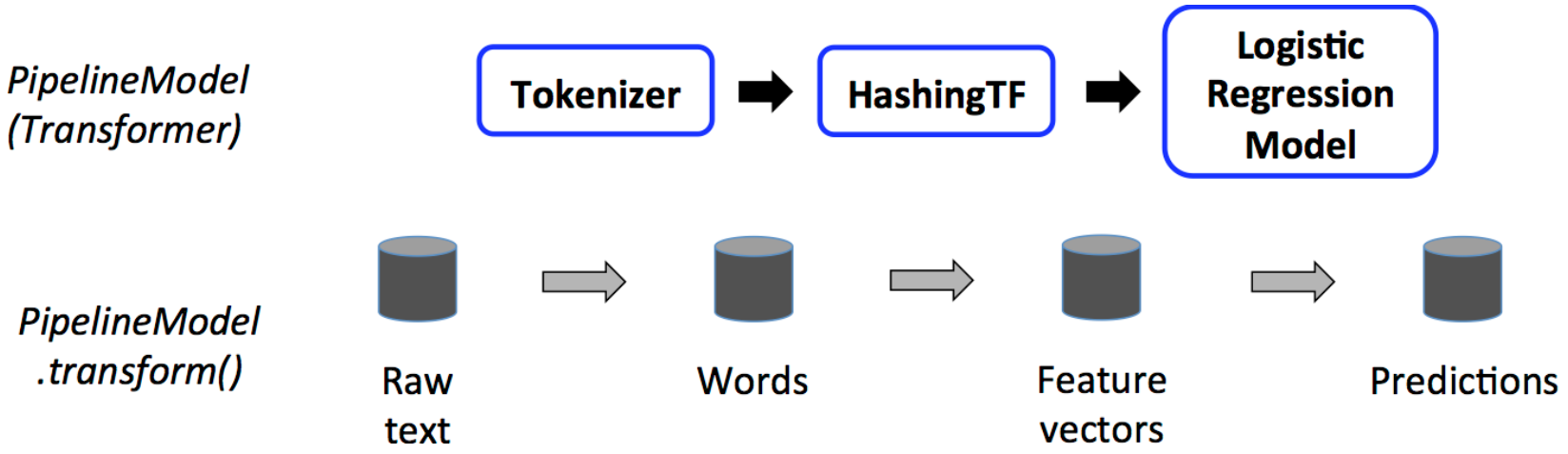
ML: Pipeline

- Pipeline là một estimator mà bao gồm 1 chuỗi các màn (stage) mà mỗi stage có thể là estimator hoặc là transformer



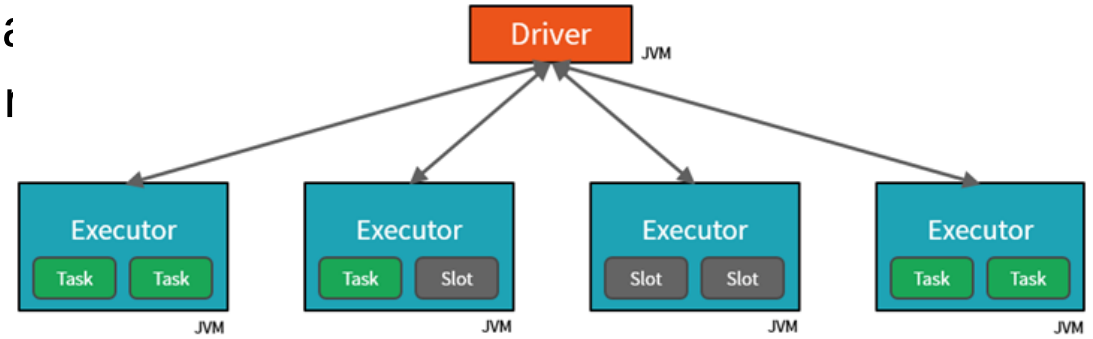
ML: PipelineModel

- Kết quả sau khi gọi Pipeline.fit() trả về một PipelineModel. PipelineModel được sử dụng khi kiểm thử.



Kiến trúc chung

- Kiến trúc master worker
  - 1 driver hay ma
  - Nhiều Worker r



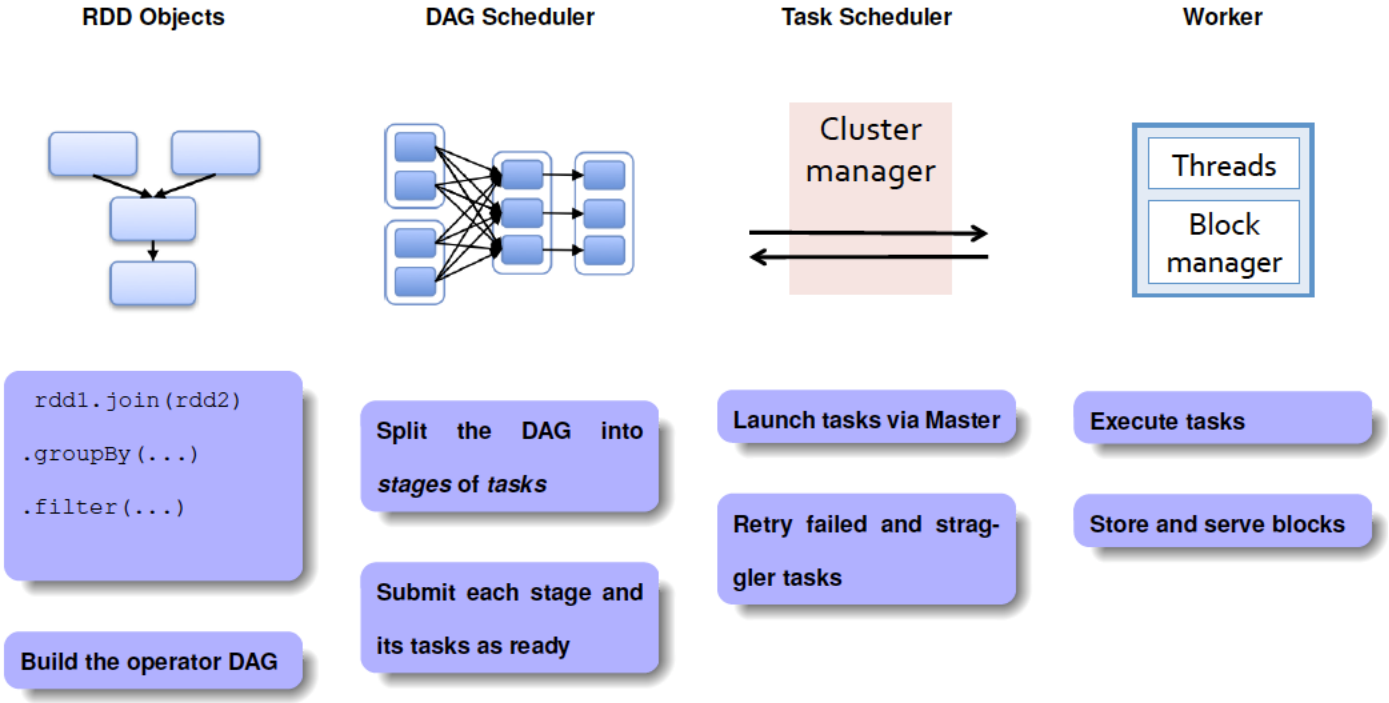
- Master gửi các công việc tới các workers và chỉ định lấy dữ liệu từ bộ nhớ trong hoặc bên ngoài (vd. S3 hoặc HDFS)

Kiến trúc chung (2)

- Một chương trình Spark program đầu tiên cần tạo một SparkContext object
  - SparkContext chỉ định cách Spark sử dụng các tài nguyên tính toán được cấp phát
  - Tham số cho SparkContext là kiểu và kích thước của cụm tính toán được sử dụng

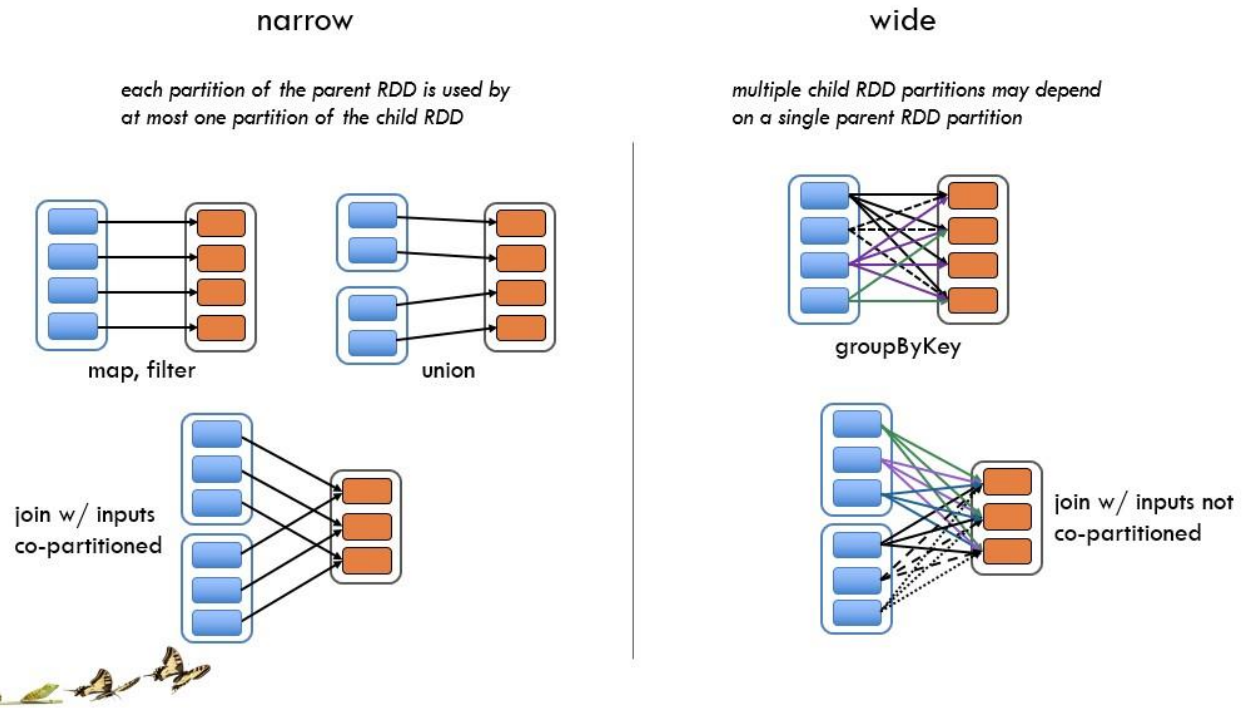
Master parameter	Description
local	Chạy cục bộ, sử dụng 1 luồng tính toán
local[K]	Chạy cục bộ sử dụng song song K luồng tính toán
spark://HOST:PORT	Kết nối tới một Spark standalone cluster
mesos://HOST:PORT	Kết nối tới một Mesos cluster
yarn	Kết nối tới một YARN cluster

Chu trình một công việc trên Spark



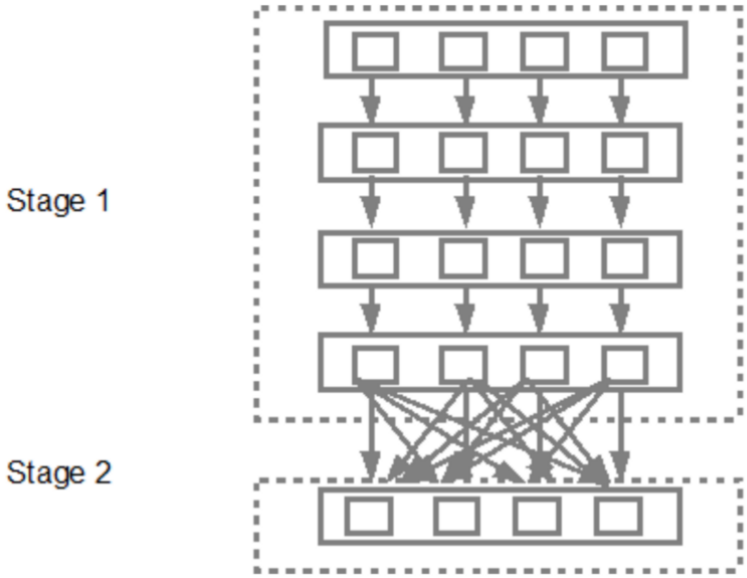
Phụ thuộc dạng Narrow hoặc Wide

- Có 2 dạng biến đổi
  - Narrow transformation—Không đòi hỏi dữ liệu phải được phân tán lại giữa các phân vùng partitions. Vd. Map, filter etc..
  - Wide transformation—đòi hỏi dữ liệu phải buộc phân tán lại.



Mối quan hệ giữa Spark transformations và stages

- Các narrow transformations sẽ được nhóm lại trong cùng một stage



Tổng kết

- Hadoop: Hệ sinh thái lưu trữ và xử lý dữ liệu lớn kinh tế và khả mở
- Spark: Nền tảng phân tích dữ liệu hợp nhất cho dữ liệu lớn

Thank you for your attention!

Q&A