ĐẠI HỌC
BÁCH KHOA

25 YEARS ANNIVERSARY
SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

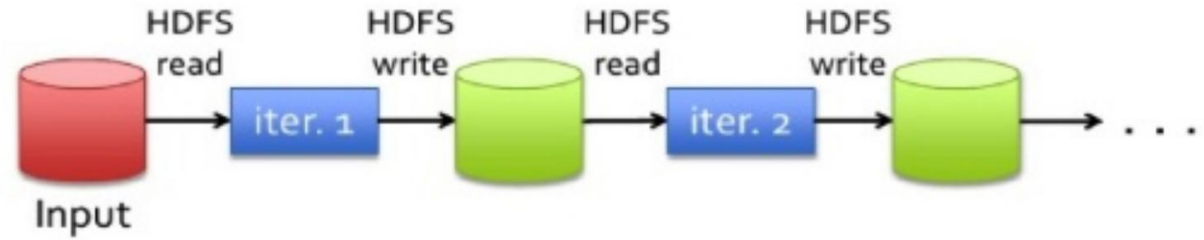
HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chapter 6
Batch processing - part 2
Apache Spark

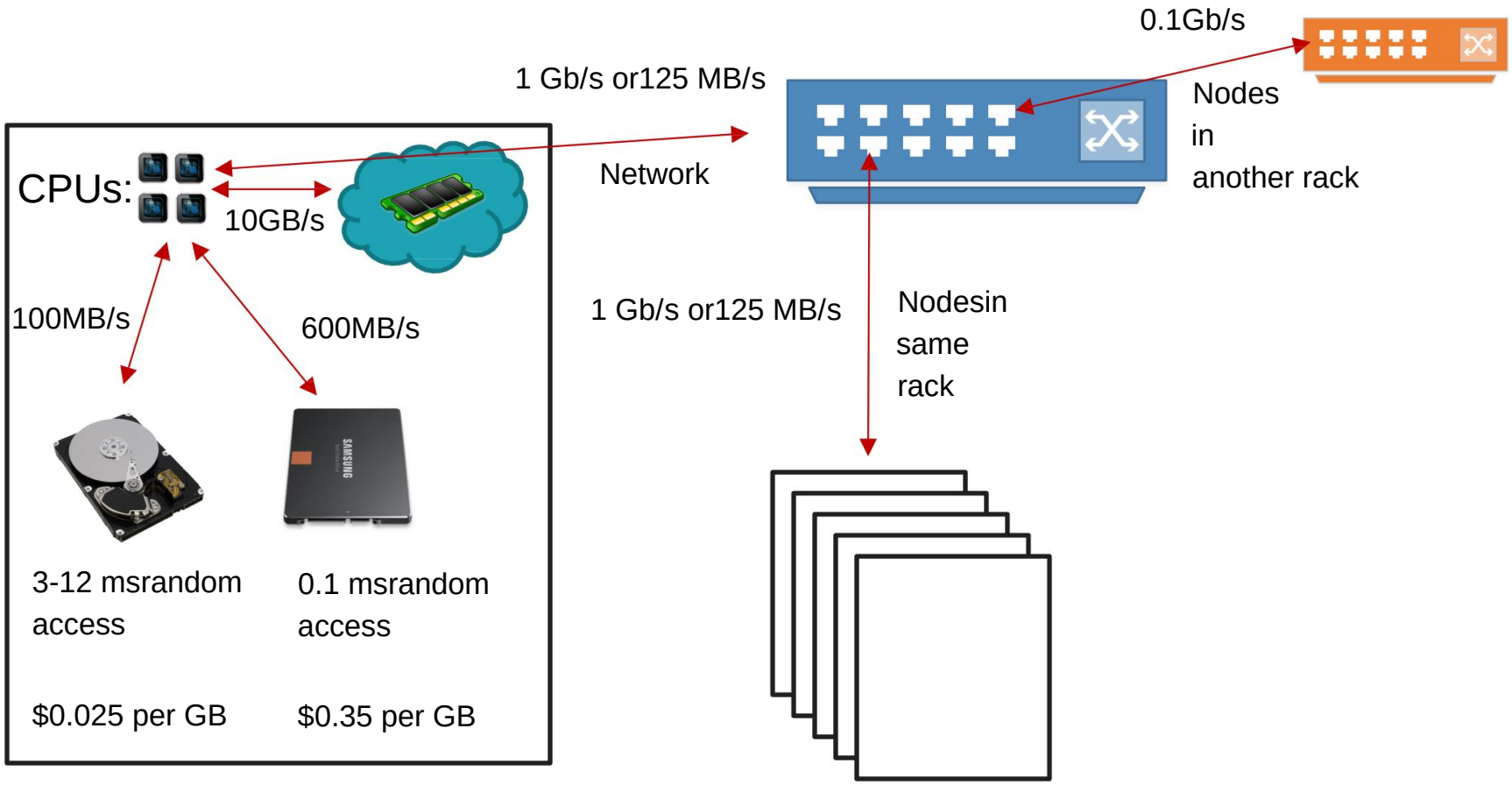
An unified analytics engine for large-scale data processing

Map Reduce: Iterative jobs

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

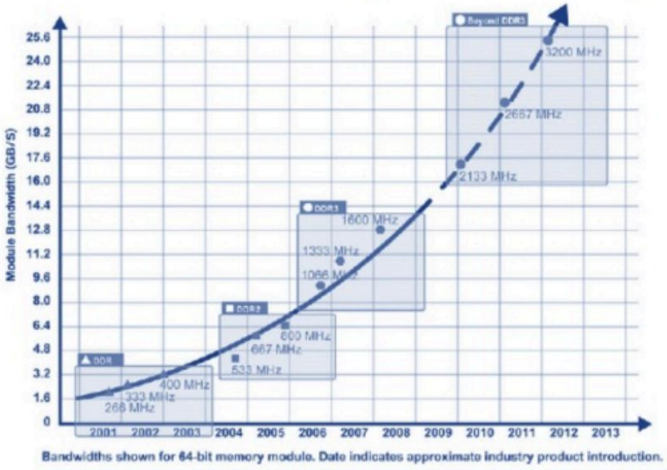


- è Disk I/O is very slow!



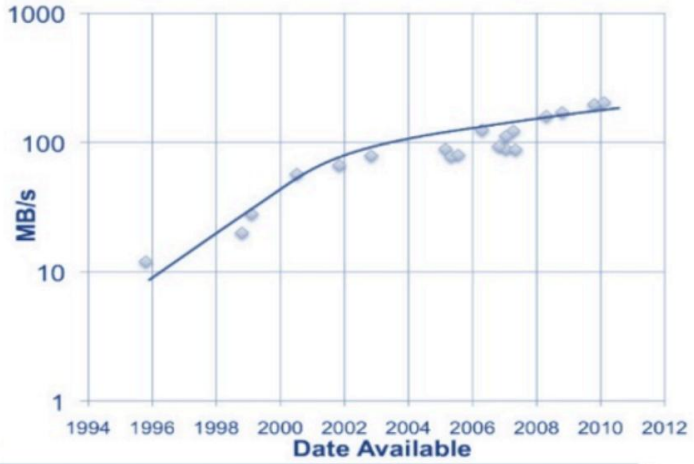
RAM is the new disk

- RAM throughput increasing **exponentially**



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

- Disk throughput increasing **slowly**

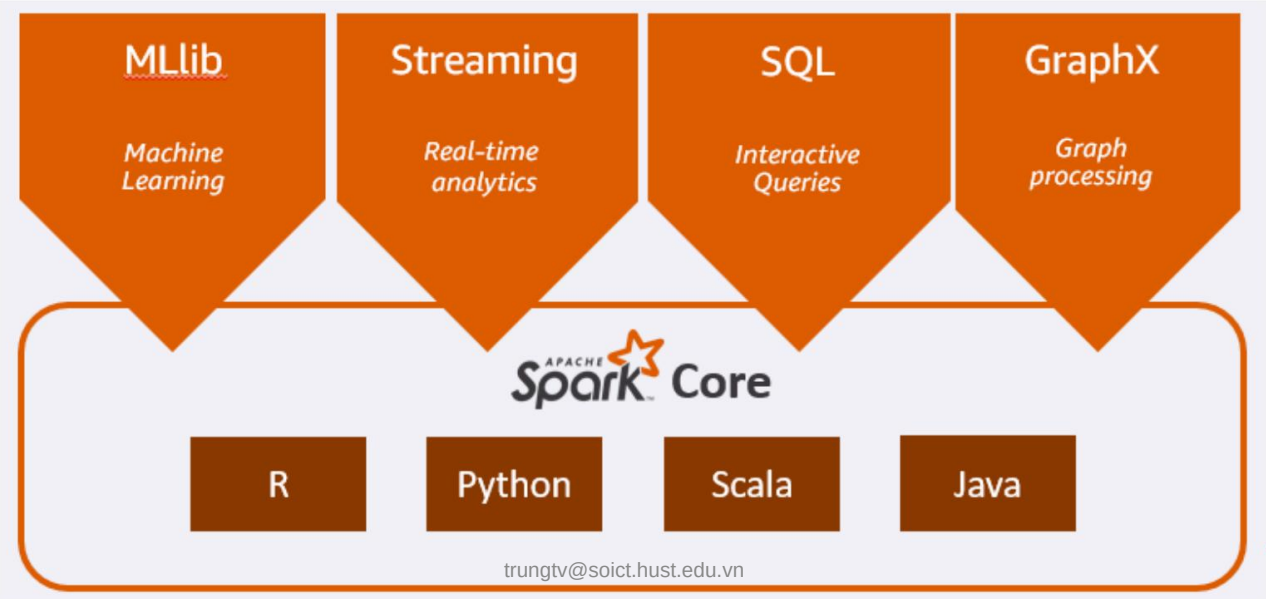


Memory-locality key to interactive response times

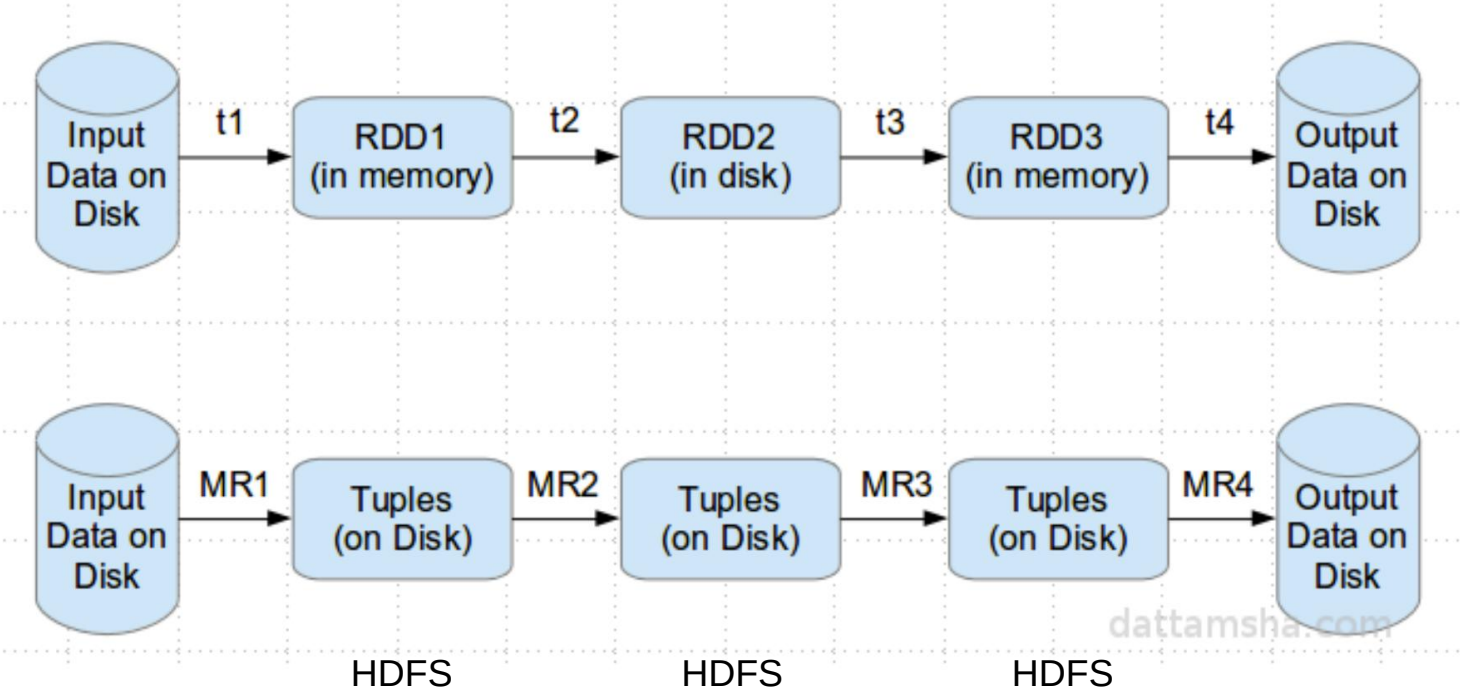
A unified analytics engine for large-scale data processing

- 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.



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)

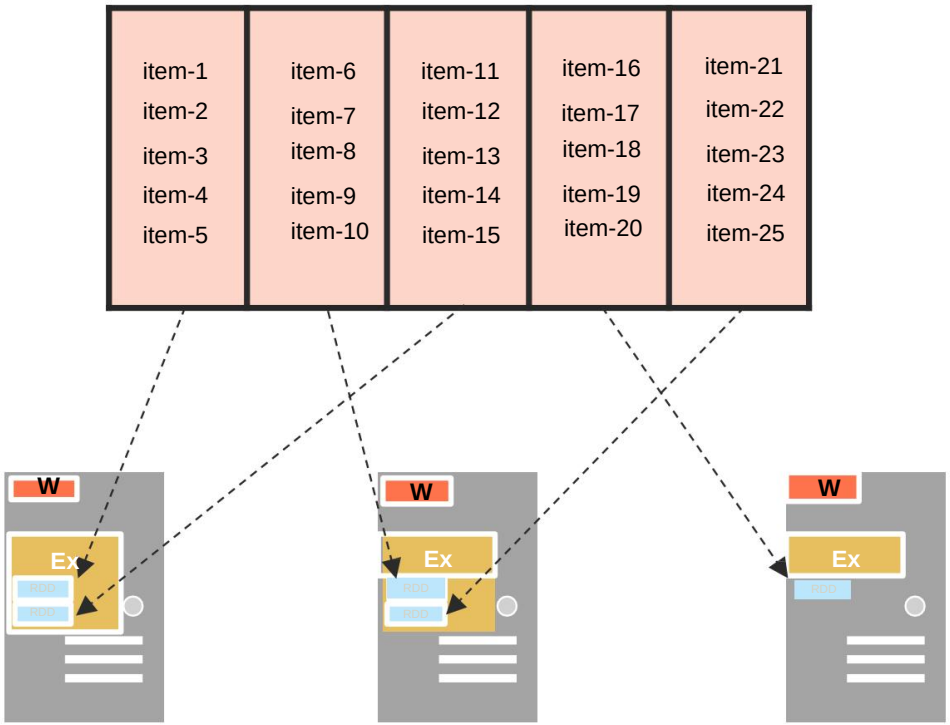
https://databricks.com/blog/2014/10/10/spark-petabyte-sort.html

Resilient Distributed Dataset (RDD)

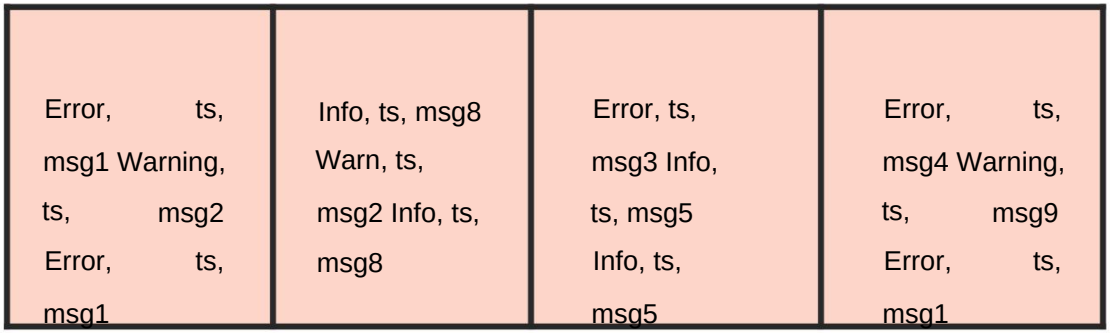
- RDDs are ***fault-tolerant, parallel data structures*** that let users explicitly persist ***intermediate results in memory,*** control their partitioning to optimize data placement, and manipulate them using ***a rich set of operators.***
- coarse-grained transformations vs. fine-grained updates
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

more partitions= more parallelism

RDD



RDD with 4 partitions



logLinesRDD

A base RDD can be created 2ways:

-Parallelize a collection

- Read data from an external source (S3, C*, HDFS, etc)

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"));
```

- Take an existing in-memory collection and pass it toSparkContext's parallelizemethod
- Not generally used outside of prototyping and testing since it requiresentiredataset in memory ononemachine

Read from Text File

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

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

```
// 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.

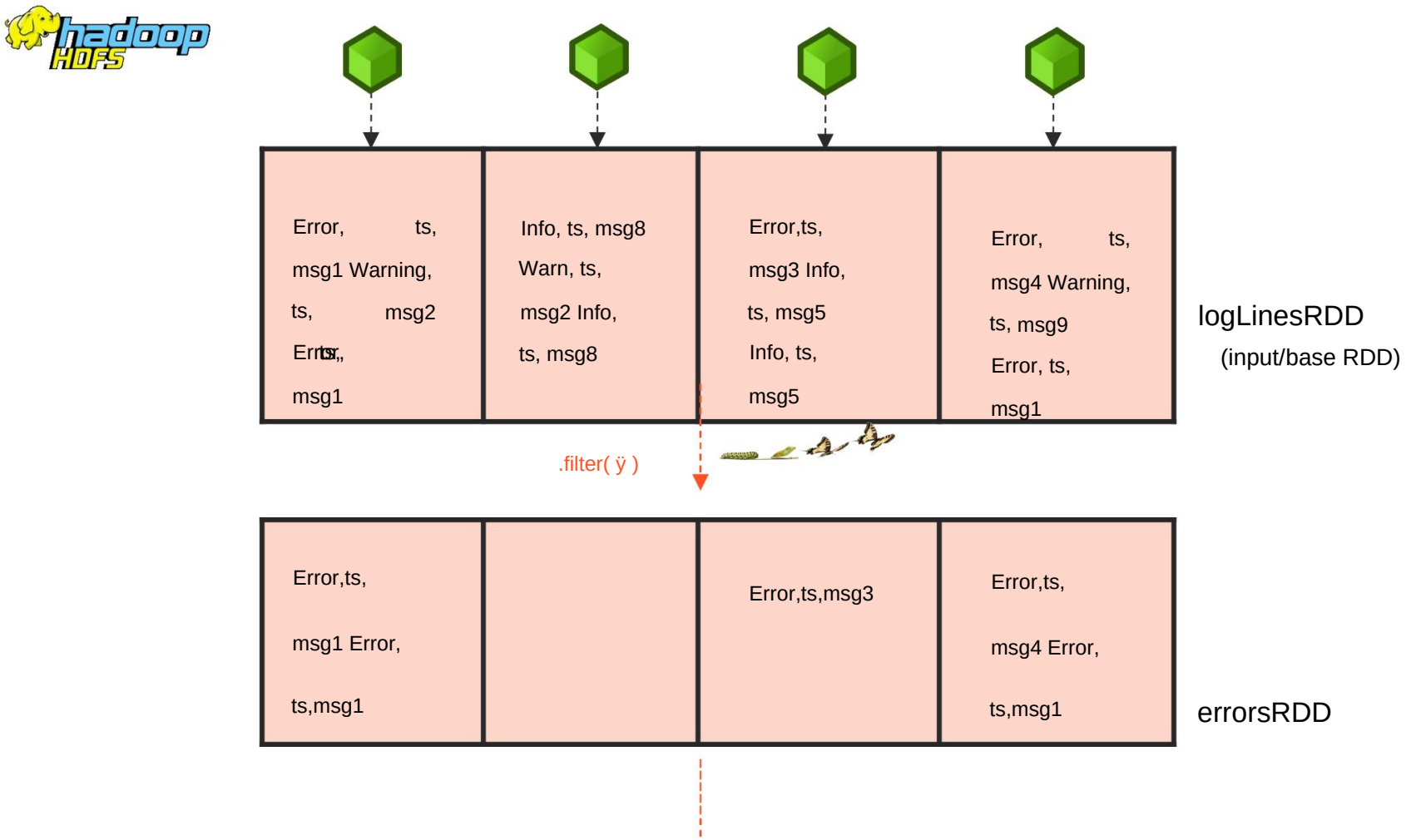
Operations on Distributed Data

- Two types of operations: transformations and actions •

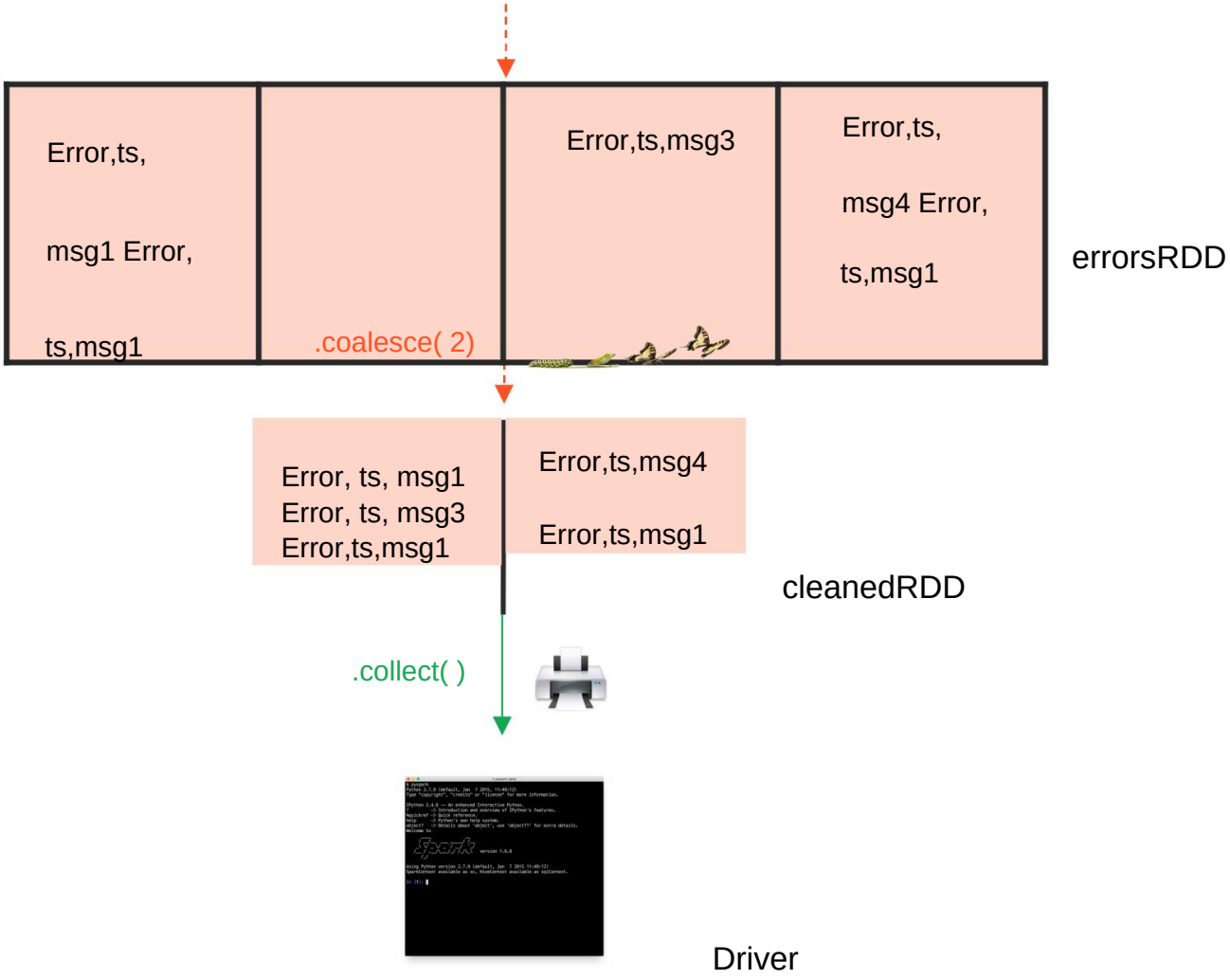
Transformations are lazy (not computed immediately) • Transformations are executed when an action is run

- Persist (cache) distributes data in memory or disk

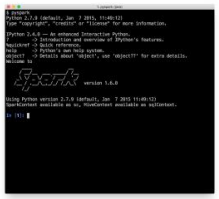
Transformation: Filter



Action: Collect

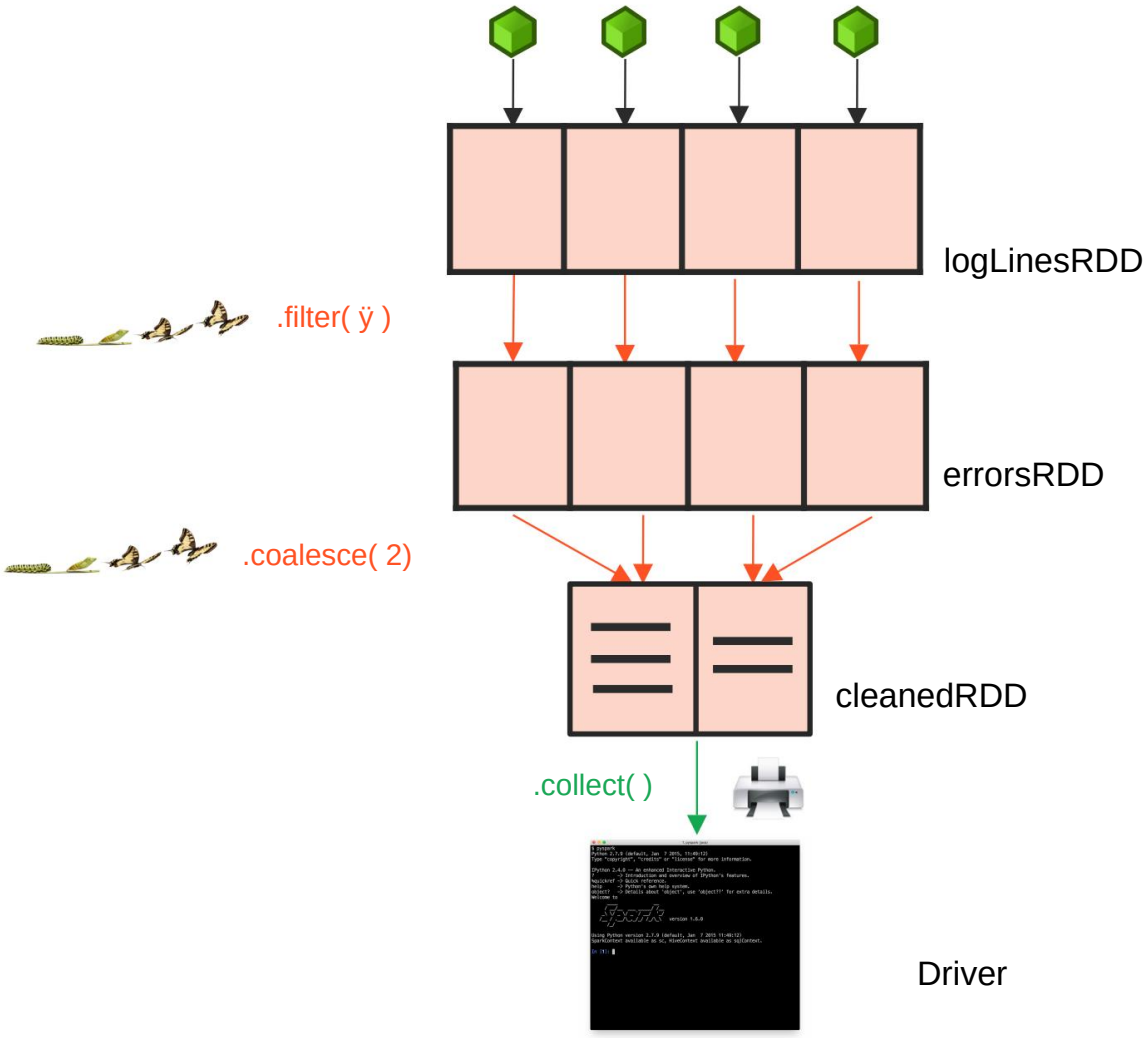


DAG execution

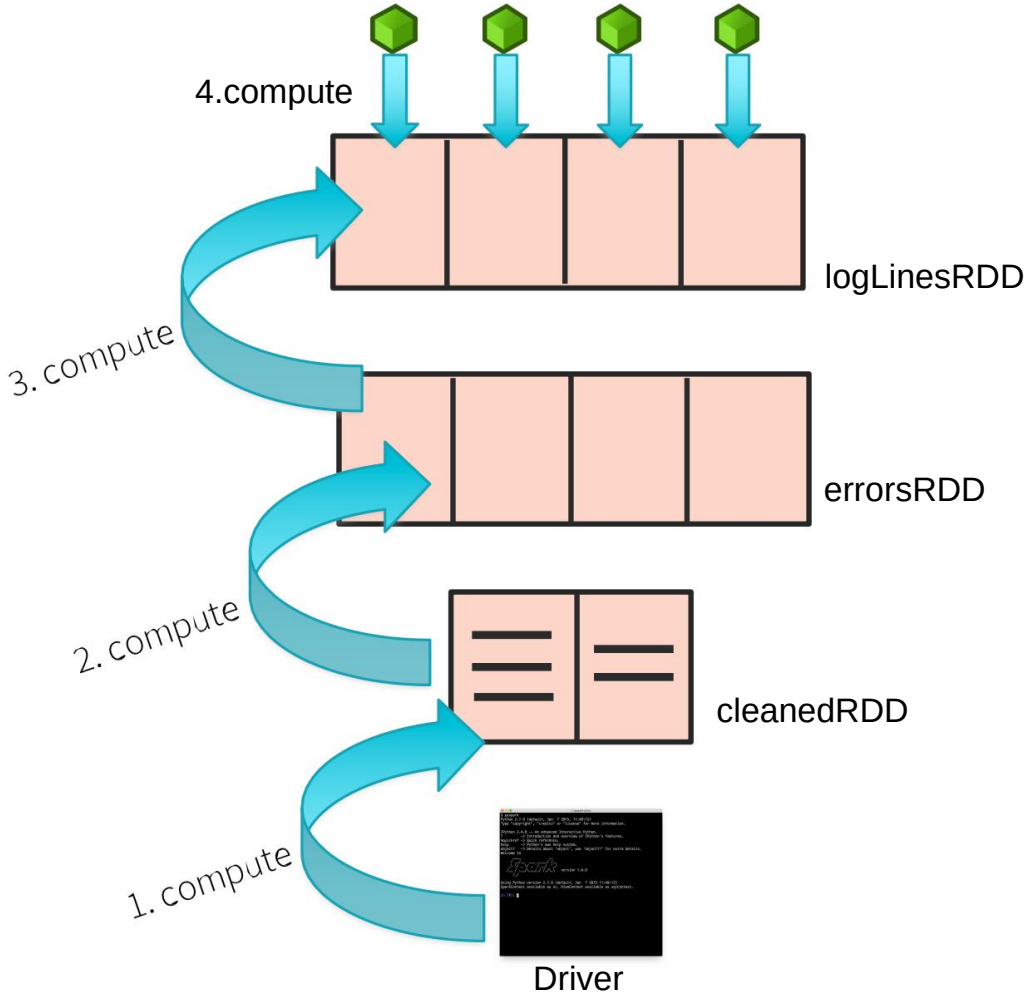


Driver

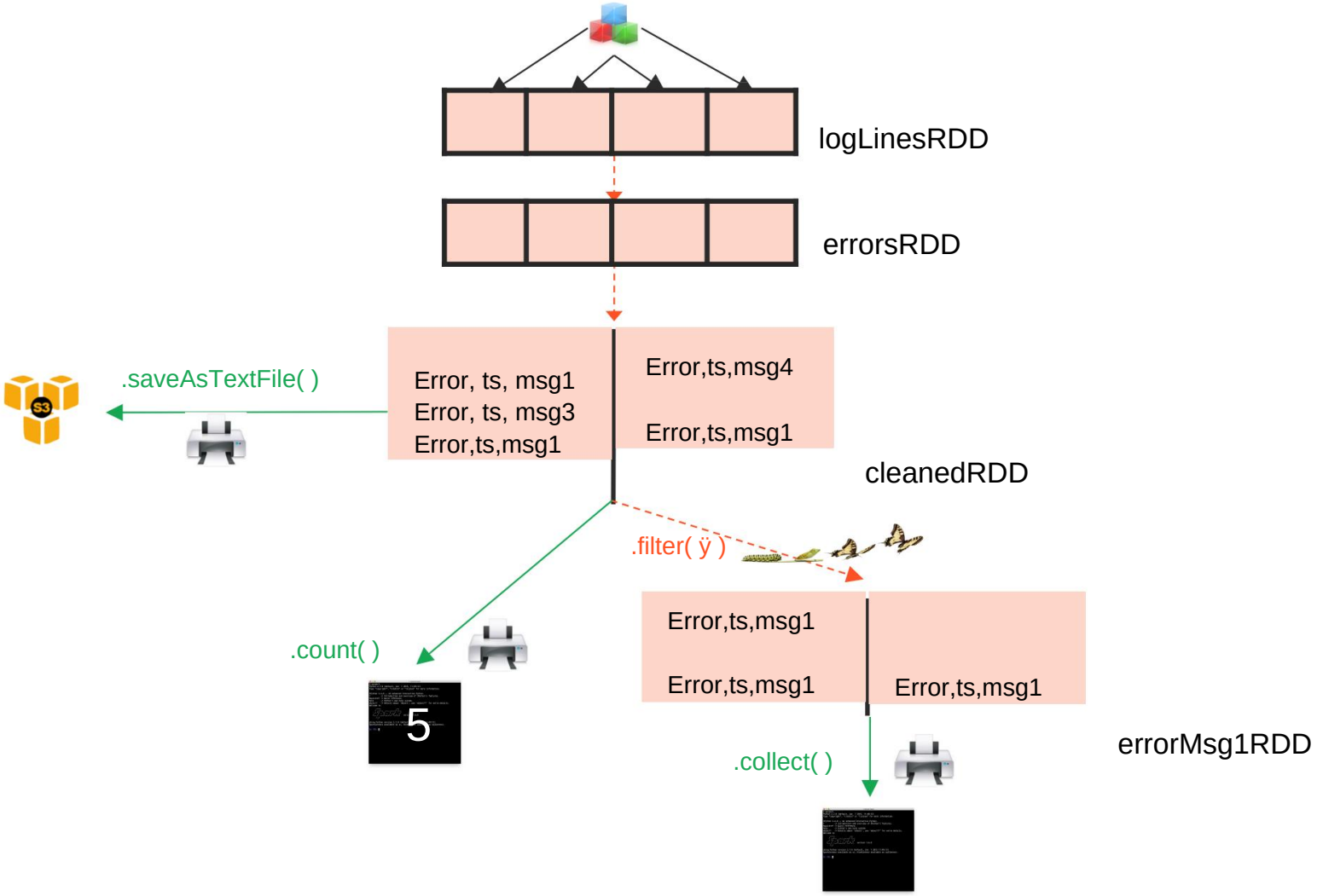
Logical



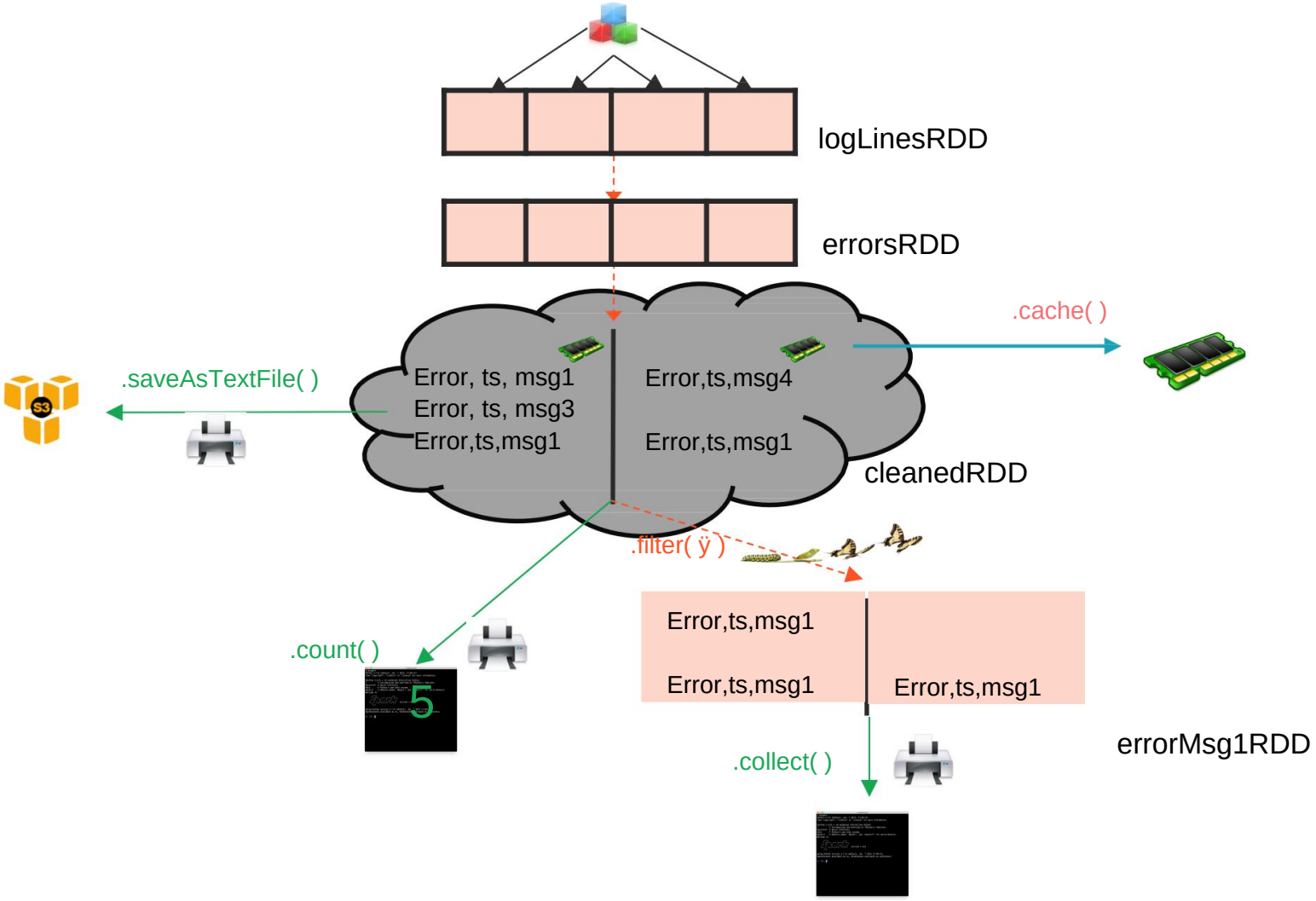
Physical



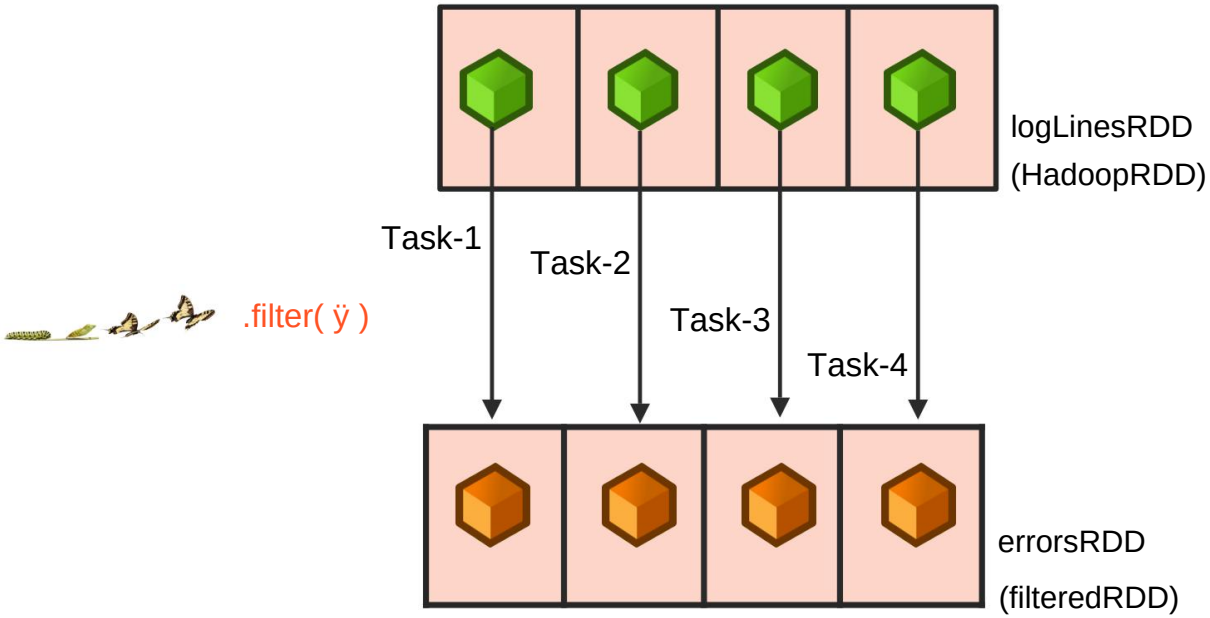
DAG



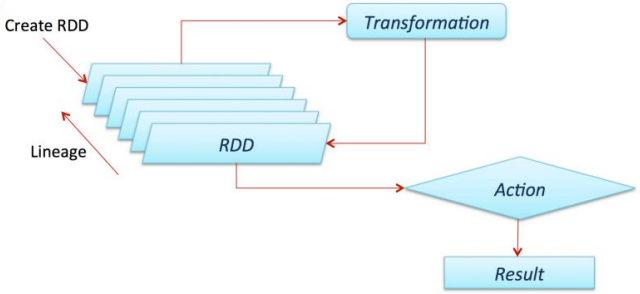
Cache

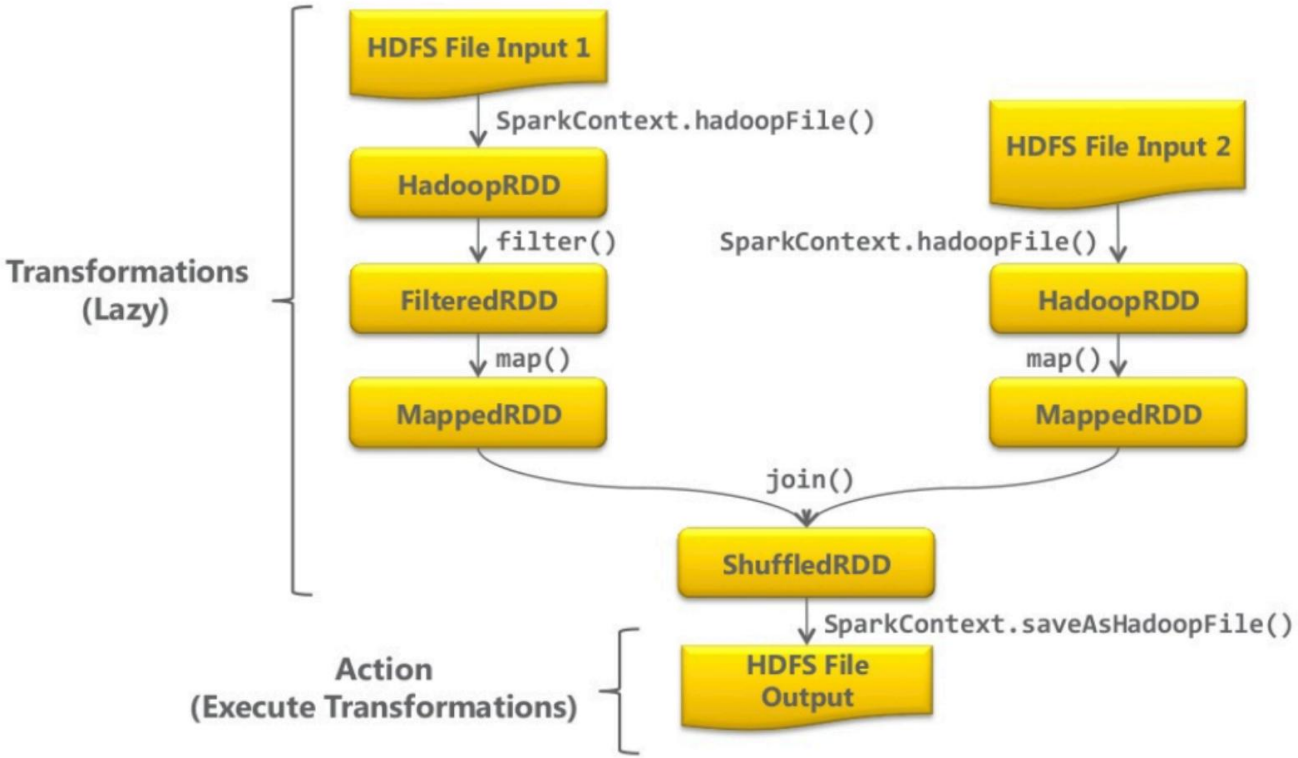


Partition >>> Task >>> Partition



RDD Lineage





Resilient Distributed Dataset (RDD)

• Initial RDD on **disks** (HDFS, etc) • Intermediate RDD on **RAM**

• Fault recovery is based on **lineage** • RDD operations are distributed

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

- 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

DataFrames versus RDDs

- For new users familiar with data frames in other programming languages, this API should make them feel at home
- For existing Spark users, the API will make Spark easier to program than using RDDs
- For both sets of users, DataFrames will improve performance through intelligent optimizations and code-generation

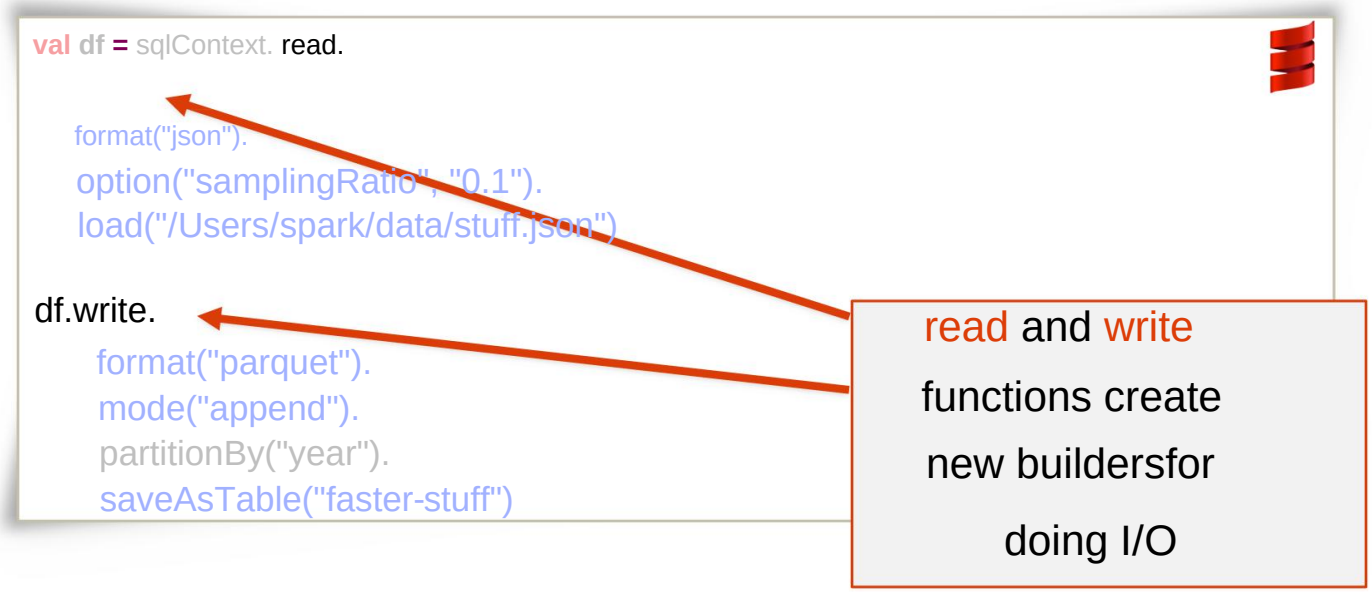
Write Less Code: Input & Output

Unified interface to reading/writing data ina variety of formats.

```
val df = sqlContext.
    read.
    format("json").
    option("samplingRatio", "0.1").
    load("/Users/spark/data/stuff.json")

df.write.
     format("parquet").
     mode("append").
     partitionBy("year").
     saveAsTable("faster-stuff")
```

Write Less Code: Input & Output

Unified interface to reading/writing data ina variety of formats.



Write Less Code: Input & Output

Unified interface to reading/writing data ina variety of formats.

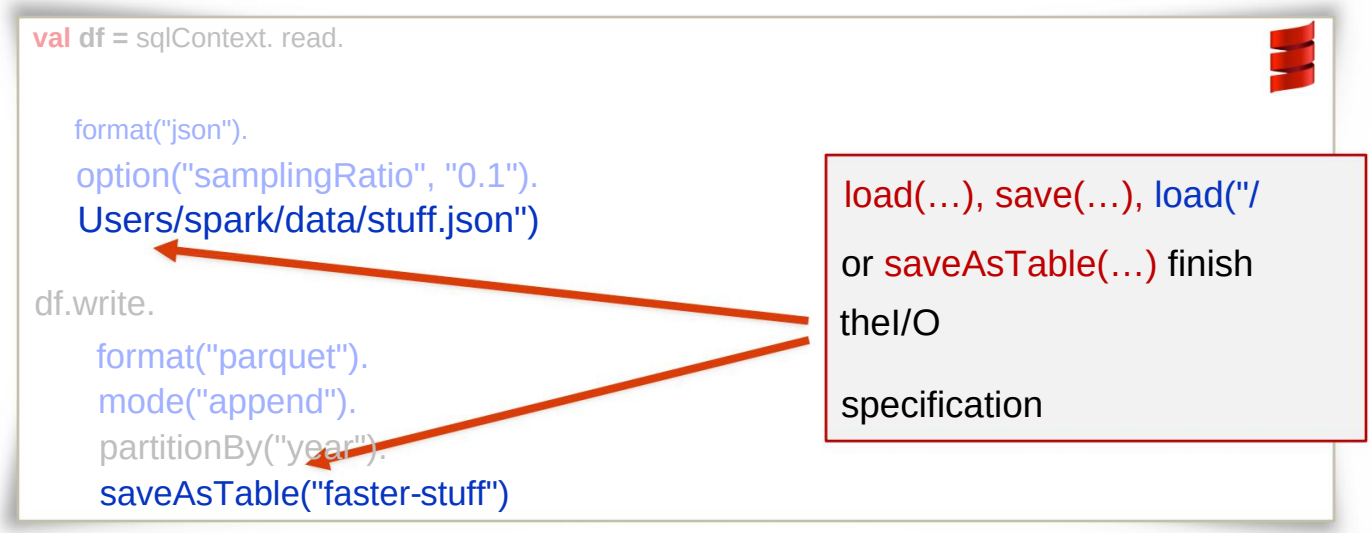
```
val df = sqlContext.read.
    format("json").
    option("samplingRatio", "0.1").
    load("/Users/spark/data/stuff.json")

df.write.
    mode("append").
    format("parquet").
    partitionBy("year").
    saveAsTable("faster-stuff")
```

Builder methods specify:

- partitioning
- handling of existing data

Write Less Code: Input & Output

Unified interface to reading/writing data ina variety of formats.



Data Sources supported by DataFrames

built-in

external

and more…

Write Less Code: High-Level Operations

- Solve common problems concisely with DataFrame functions:
  - selecting columns and filtering
  - joining different data sources
  - aggregation (count, sum, average, etc.)
  - plotting results (eg, with Pandas)

Write Less Code: Compute an Average

```
private IntWritable one = new IntWritable(1); private IntWritable
output =new IntWritable(); protected void map(LongWritable
key,
                                Text value,
                                Context context) {
        String[] fields = value.split("\t");
        output.set(Integer.parseInt(fields[1])); context.write(one,
        output);
}

-------------------------


IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable();

protected void reduce(IntWritable key,
                                Iterable<IntWritable> values,
                                Context context) {
        int sum = 0; int
        count = 0; for
        (IntWritable value: values) { sum +=
                value.get(); count++;


        } average.set(sum / (double) count);
        context.write(key, average);
}
```

```
rdd = sc.textFile(...).map(_.split(" ")) rdd.map { x => (x(0),
(x(1).toFloat, 1)) }. reduceByKey { case ((num1, count1),
        (num2, count2)) =>
            (num1 + num2, count1 + count2)
        }.
        map { case (key, (num, count)) => (key, num / count) }. collect()
```

```
rdd = sc.textFile(...).map(lambda s: s.split()) rdd.map(lambda x:
(x[0], (float(x[1]), 1))).\ reduceByKey(lambda t1, t2: (t1[0] + t2[0],
        t1[1] + t2[1])).\ map(lambda t: (t[0], t[1][0] / t[1][1])).\ collect()
```

Write Less Code: Compute safely Average

UsingRDDs

```
rdd = sc.textFile(...).map(_.split(" ")) rdd.map { x =>
(x(0), (x(1).toFloat, 1)) }. reduceByKey { case ((num1,
      count1), (num2, count2)) =>
          (num1 + num2, count1 + count2) }.

      map { case (key, (num, count)) => (key, num / count) }. collect()
```

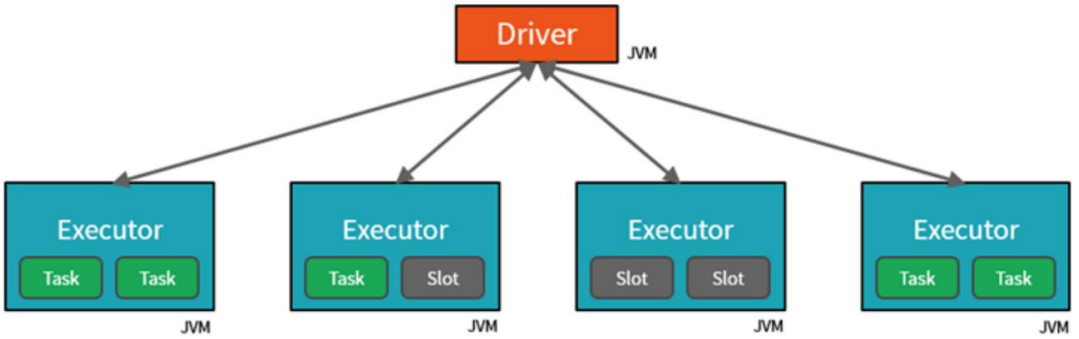
Full APIDocs • Scala • Java • Python • R

Using DataFrames

```
import org.apache.spark.sql.functions._

val df = rdd.map(a => (a(0), a(1))).toDF("key", "value") df.groupBy("key")

    .agg(avg("value")) .collect()
```

Architecture

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



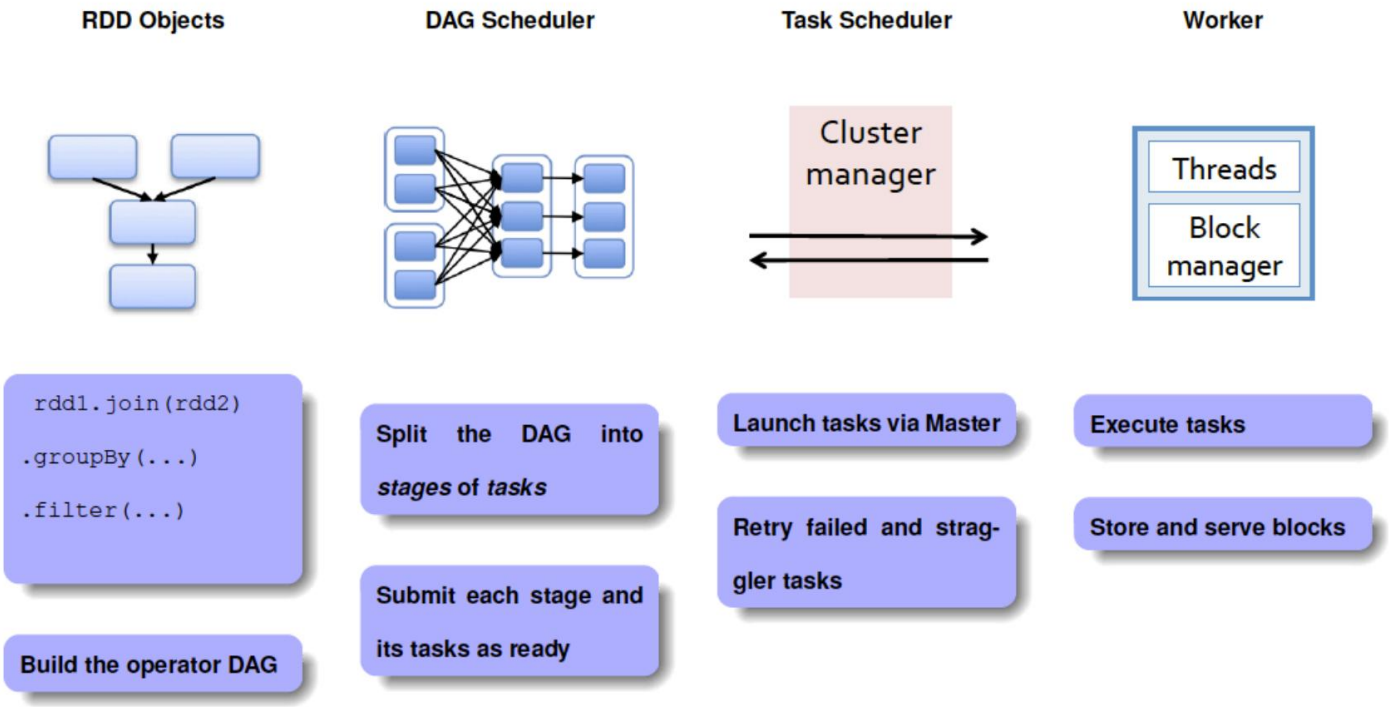
- The master sends works to the workers and either instructs them to pull data from memory or from hard disk (or from another source like S3 or HDSF)

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



Demo

References

- Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12).* 2012.
- Armbrust, Michael, et al. "Spark sql: Relational data processing in spark." *Proceedings of the 2015 ACM SIGMOD international conference on management of data.* 2015.
- Zaharia, Matei, et al. "Discretized streams: Fault-tolerant streaming computation at scale." *Proceedings of the twenty-fourth ACM symposium on operating systems principles.* 2013.
- Chambers, Bill, and Matei Zaharia. *Spark: The definitive guide: Big data processing made simple.* "O'Reilly Media, Inc.", 2018.

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attention!!!

soict.hust.edu.vn/ fb.com/groups/soict