1. Spark 이해하기

강경인

1. Spark 이해하기

1.1 Spark 개요

1.2 Scala 개요

1.3 Spark 설치 및 실행

1. Spark 이해하기

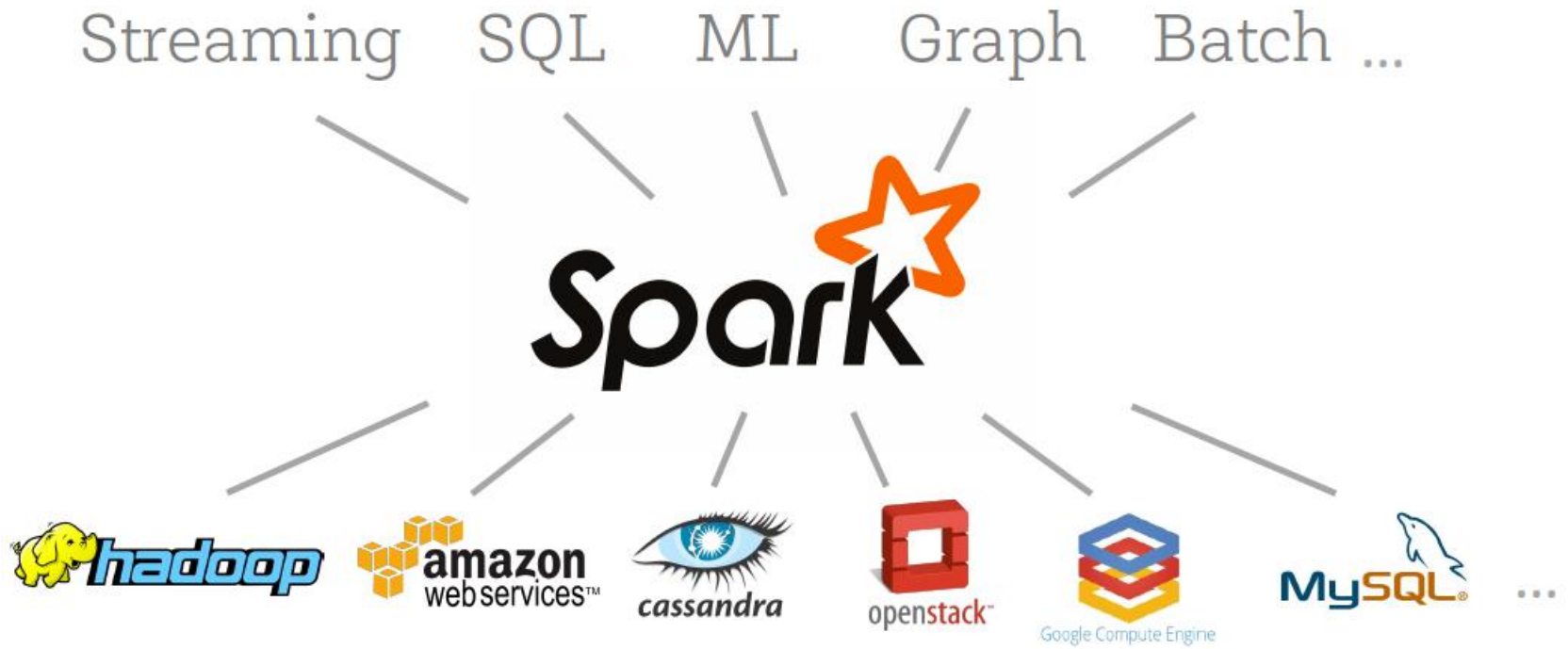
1.1 Spark 개요

1.2 Scala 개요

1.3 Spark 설치 및 실행

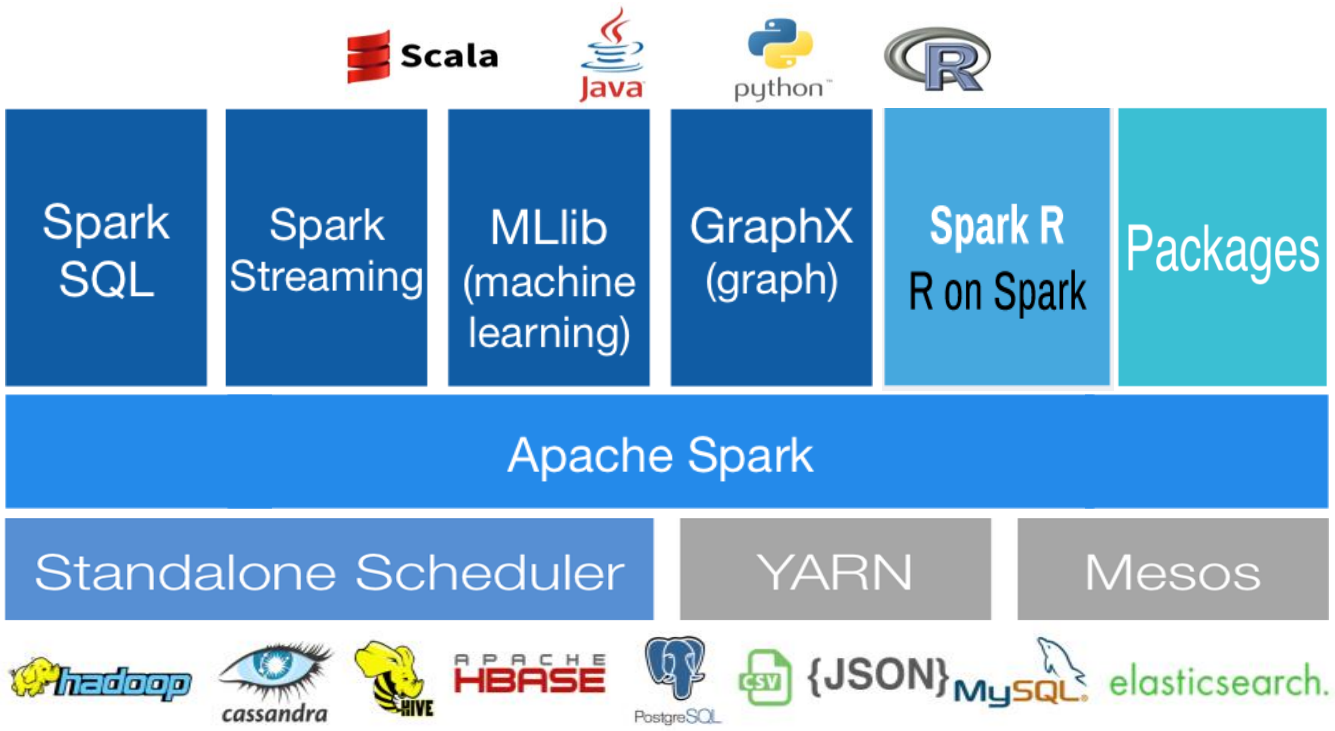
Apache Spark란?

- **Unified Engine** : Support end-to-end applications
- **High-level APIs** : Easy to use, rich optimizations
- **Integrate Broadly** : Storage systems, libraries, etc



Apache Spark란?

- Apache Spark는 대용량 데이터 프로세싱을 위한 빠르고 범용적인 인메모리 기반 클러스터 컴퓨팅 엔진
- 분산 메모리 기반의 빠른 분산 병렬 처리
- 배치, 대화형 쿼리, 스트리밍, 머신러닝과 같은 다양한 작업 타입을 지원하는 범용 엔진으로 Apache Hadoop과 호환
- Scala, Java, Python, R 기반 High-level APIs 제공



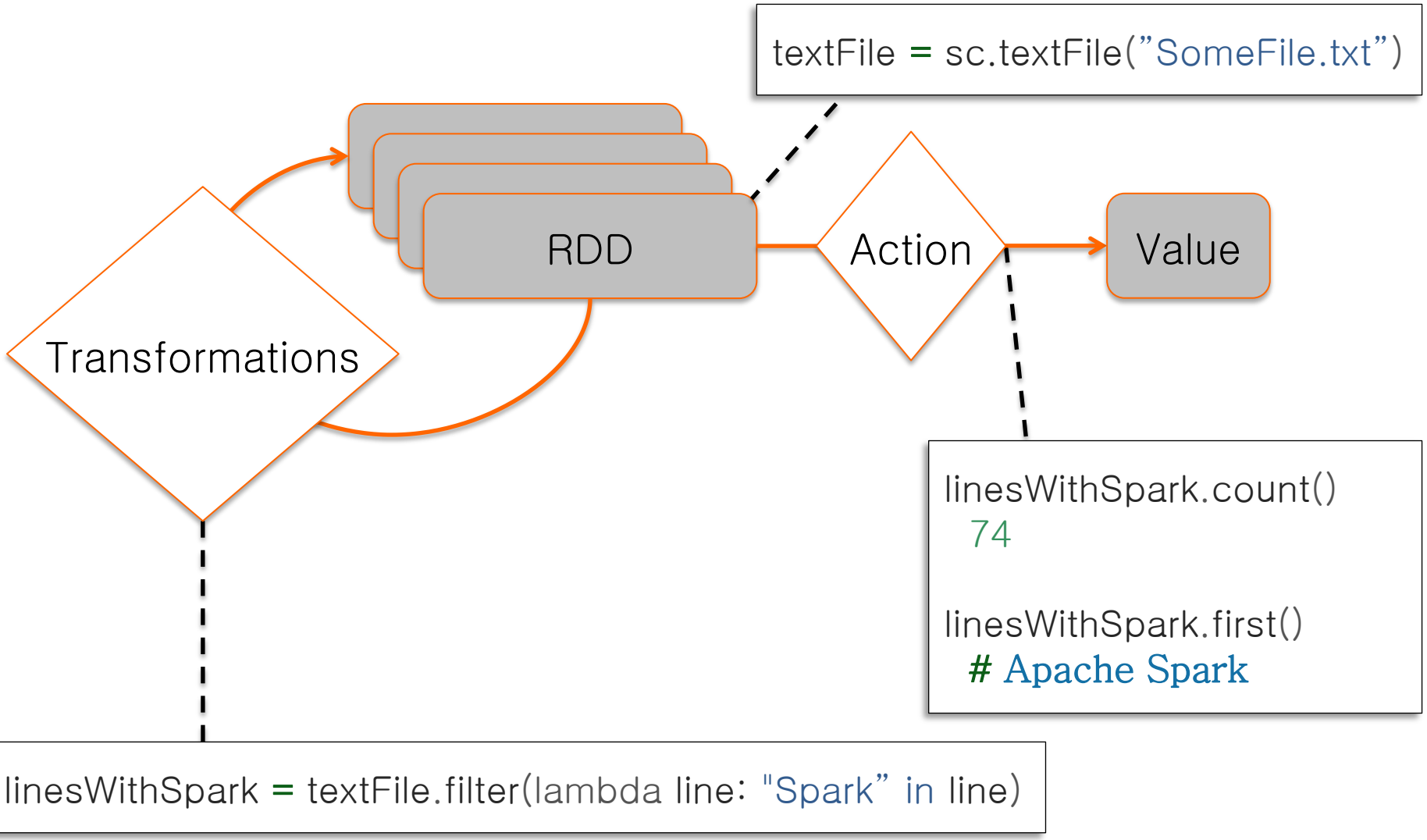
Apache Spark 특징

- In-Memory 컴퓨팅 (물론 Disk 기반도 가능)
- RDD (Resilient Distributed Dataset) 데이터 모델
- 다양한 개발 언어 지원 (Scala, Java, Python, R, SQL)
- Rich APIs 제공 (80여개 이상, 2~10x Less Code)
- General execution graphs => DAG (Directed Acyclic Graph) => Multiple stages of map & reduce
- Hadoop과의 유연한 연계 (HDFS, HBase, YARN and Others)
- 빠른 데이터 Processing (In-Memory Cached RDD, Up to 100x Faster)
- 대화형 질의를 위한 Interactive Shell (Scala, Python, R Interpreter)
- 실시간(Real-time) Stream Processing (vs. MapReduce for stored Data)
- 하나의 애플리케이션에서 배치, SQL 쿼리, 스트리밍, 머신러닝과 같은 다양한 작업을 하나의 워크플로우로 결합 가능
- Both fast to write and fast to run

RDD (Resilient Distributed Dataset)

- **Dataset** : 메모리나 디스크에 분산 저장된 변경 불가능한 데이터 객체들의 모음
- **Distributed** : RDD에 있는 데이터는 클러스터에 자동 분배 및 병렬 연산 수행
- **Resilient** : 클러스터의 한 노드가 실패하더라도 다른 노드가 작업 처리 (RDD Lineage, Automatically rebuilt on failure)
- **Immutable** : RDD는 수정이 안됨. 변형을 통한 새로운 RDD 생성
- **Operation APIs**
  : Transformations (데이터 변형, e.g. map, filter, groupBy, join)
  : Actions (결과연산 리턴/저장, e.g. count, collect, save)
- **Lazy Evaluation** : All Transformations (Action 실행 때까지)
- **Controllable Persistence** : Cache in RAM/Disk 가능 (반복 연산에 유리)

RDD 생성 > RDD 변형 > RDD 연산



Spark Language Support

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String x) {
    return x.contains("ERROR");
  }
}).count();
```

Python

```
lines = sc.textFile(...)
lines.filter(lambda x: "ERROR" in x).count()
```

Standalone Programs

- Scala, Java, Python, R

Interactive Shells

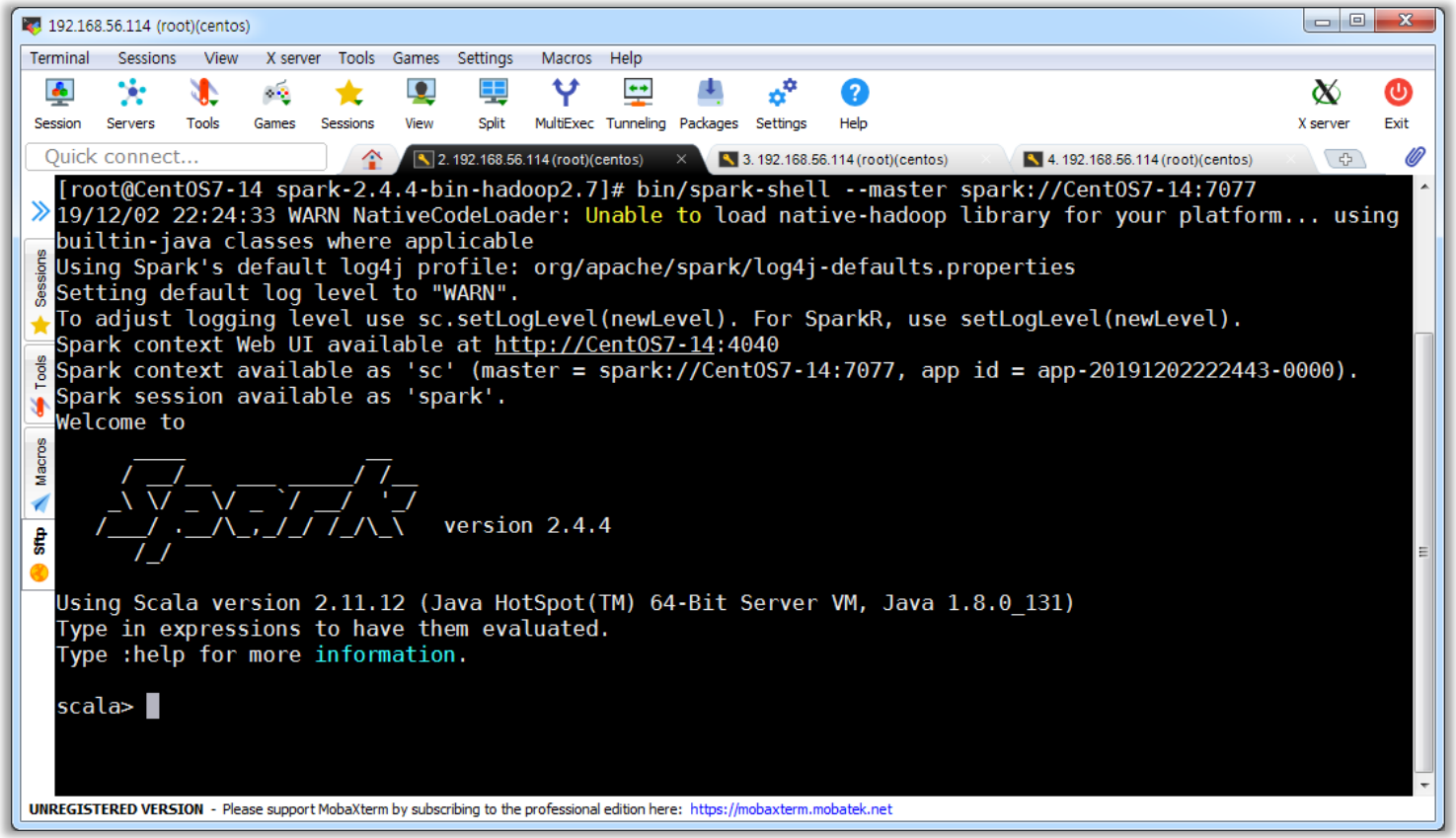
- Scala, Python, R

Performance

- Scala/Java are faster due to static typing
- …but Python is often fine

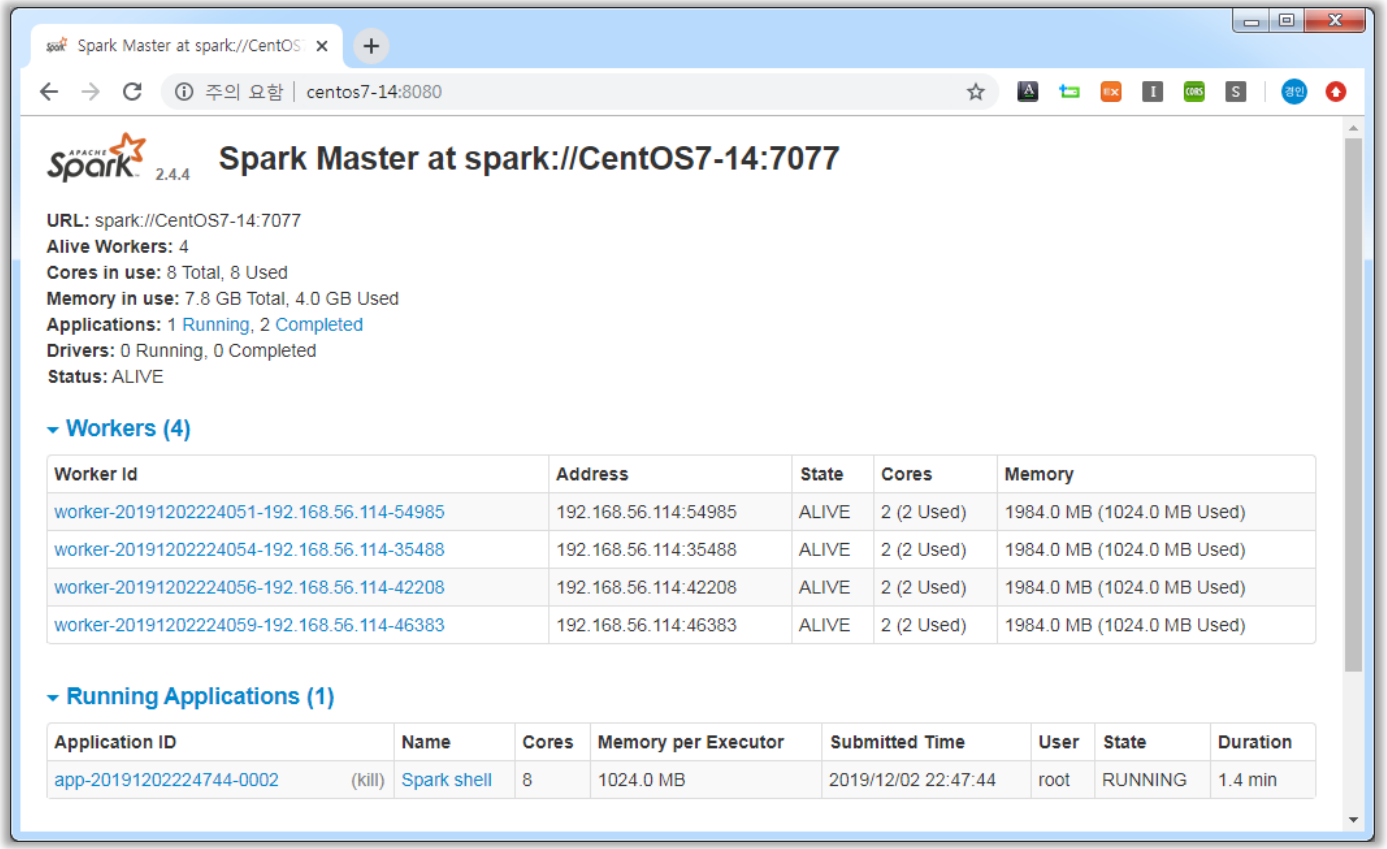
Interactive Shell

- The Fastest Way to Learn Spark
- Available in Python, Scala, R
- Runs as an application on an existing Spark Cluster…
- or Can run locally



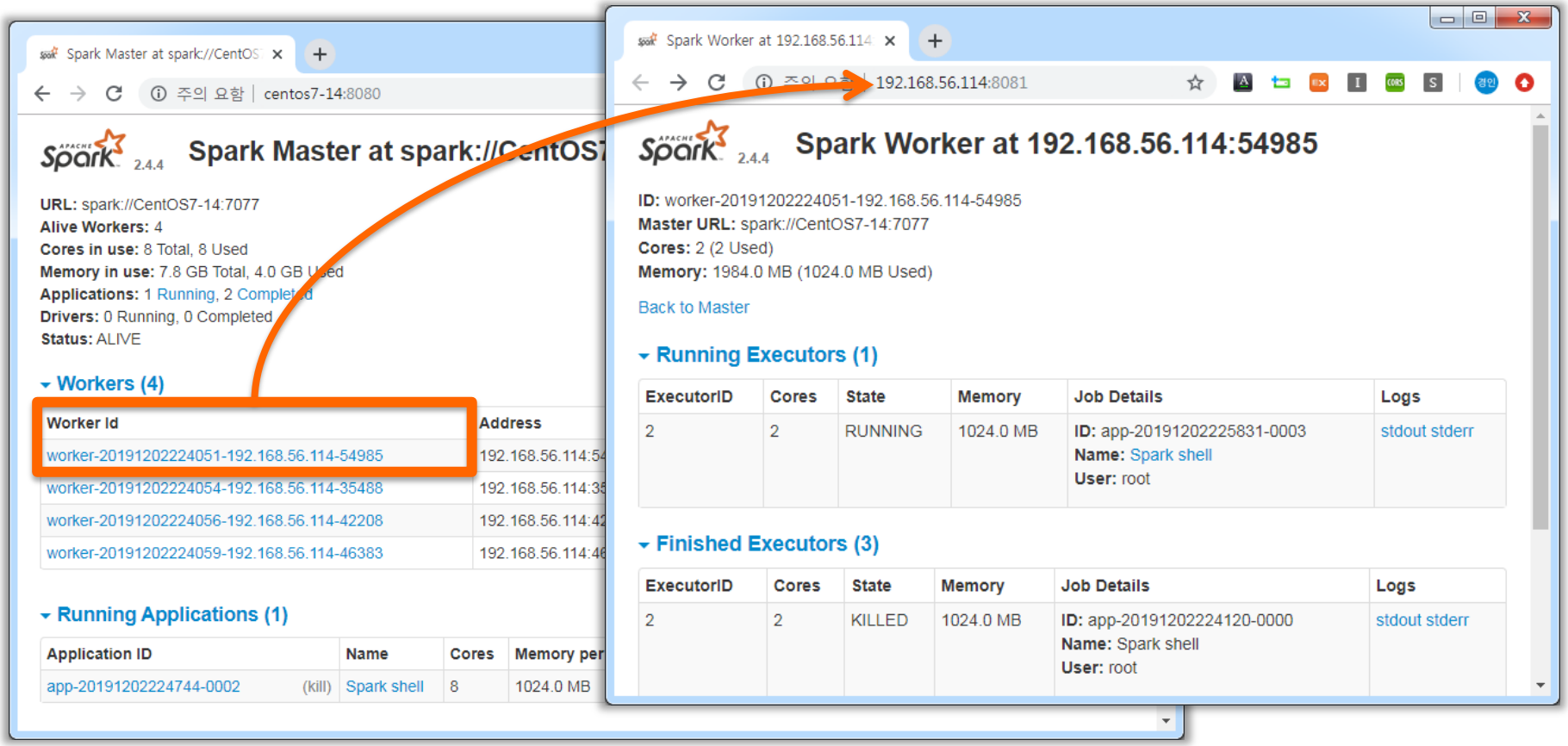
Administrative Web UIs

- **http://<Standalone Master>:8080 (by default)**
- http://<Worker>:8081 (by default)
- http://<Driver Node>:4040 (by default)
- http://<History Server>:18080 (by default)



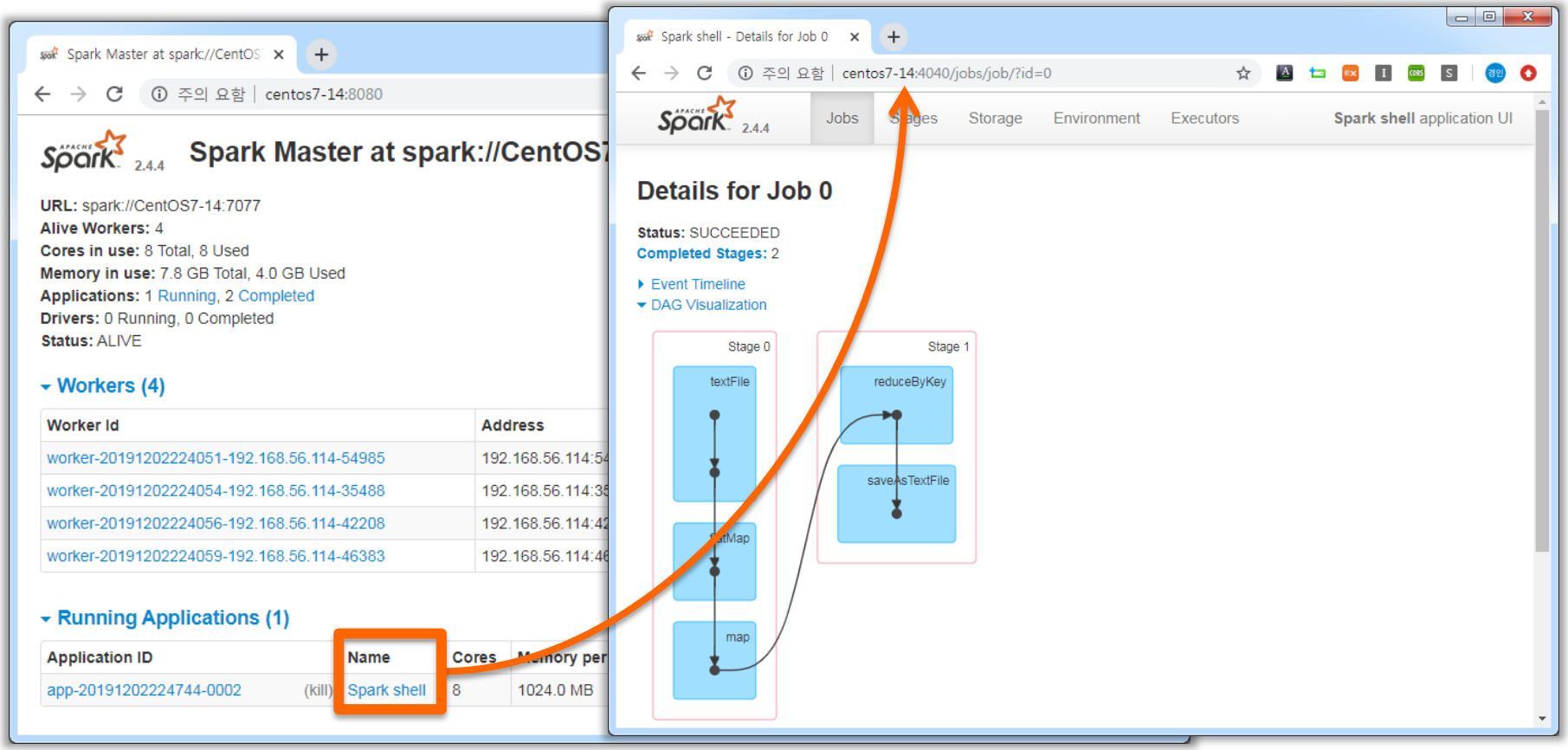
Administrative Web UIs

- http://<Standalone Master>:8080 (by default)
- **http://<Worker>:8081 (by default)**
- http://<Driver Node>:4040 (by default)
- http://<History Server>:18080 (by default)



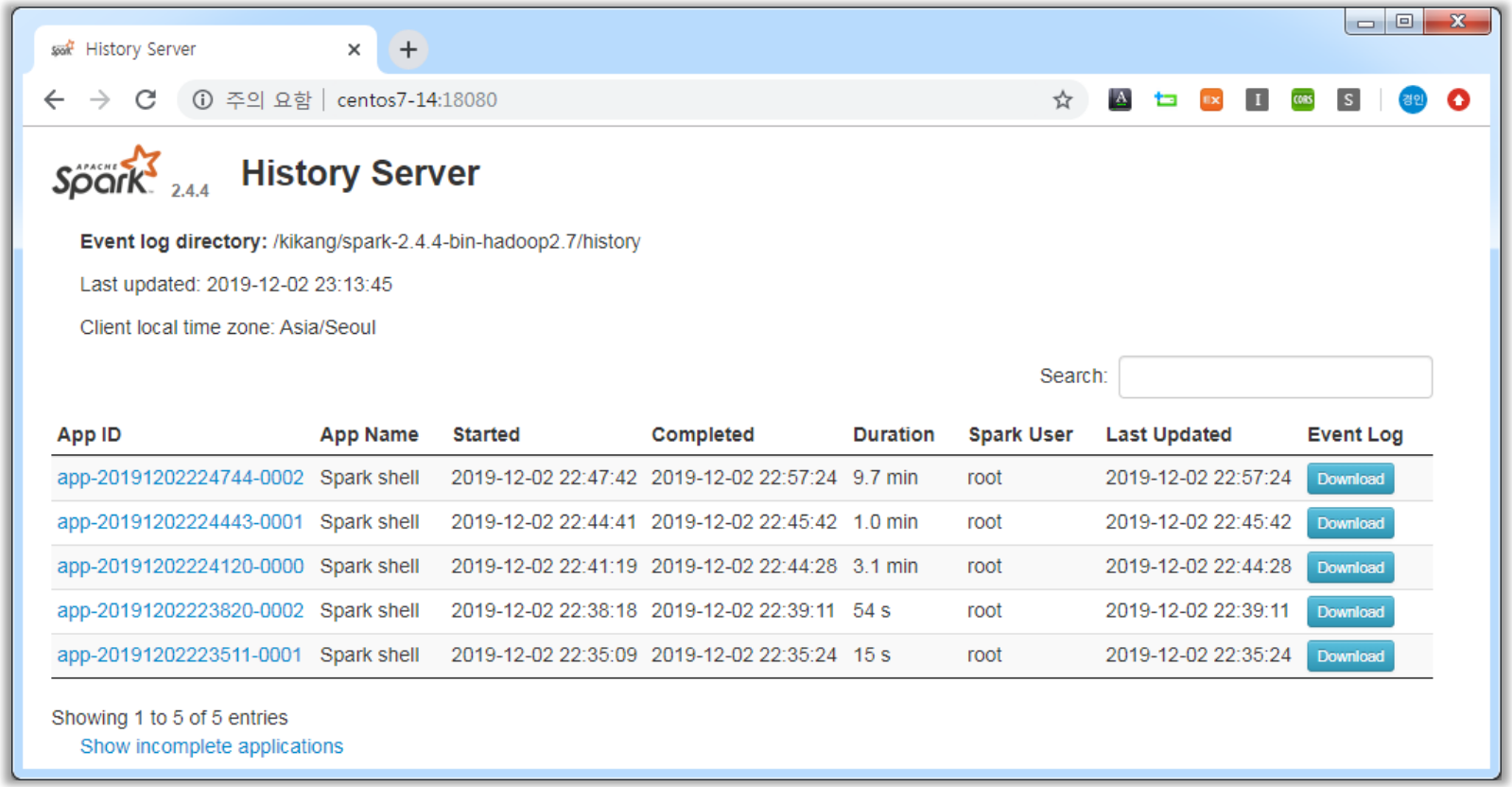
Administrative Web UIs

- http://<Standalone Master>:8080 (by default)
- http://<Worker>:8081 (by default)
- **http://<Driver Node>:4040 (by default)**
- http://<History Server>:18080 (by default)



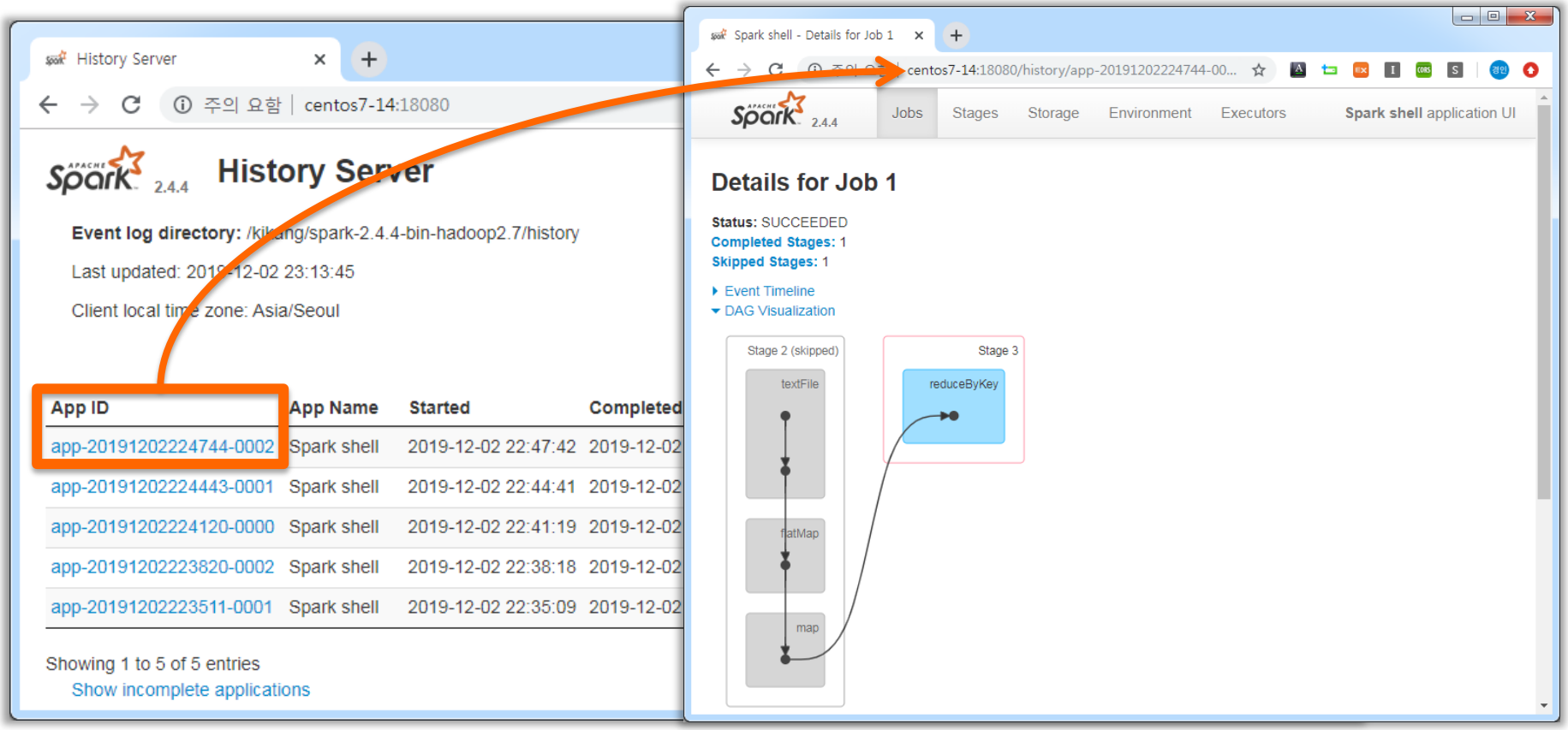
Administrative Web UIs

- http://<Standalone Master>:8080 (by default)
- http://<Worker>:8081 (by default)
- http://<Driver Node>:4040 (by default)
- **http://<History Server>:18080 (by default)**

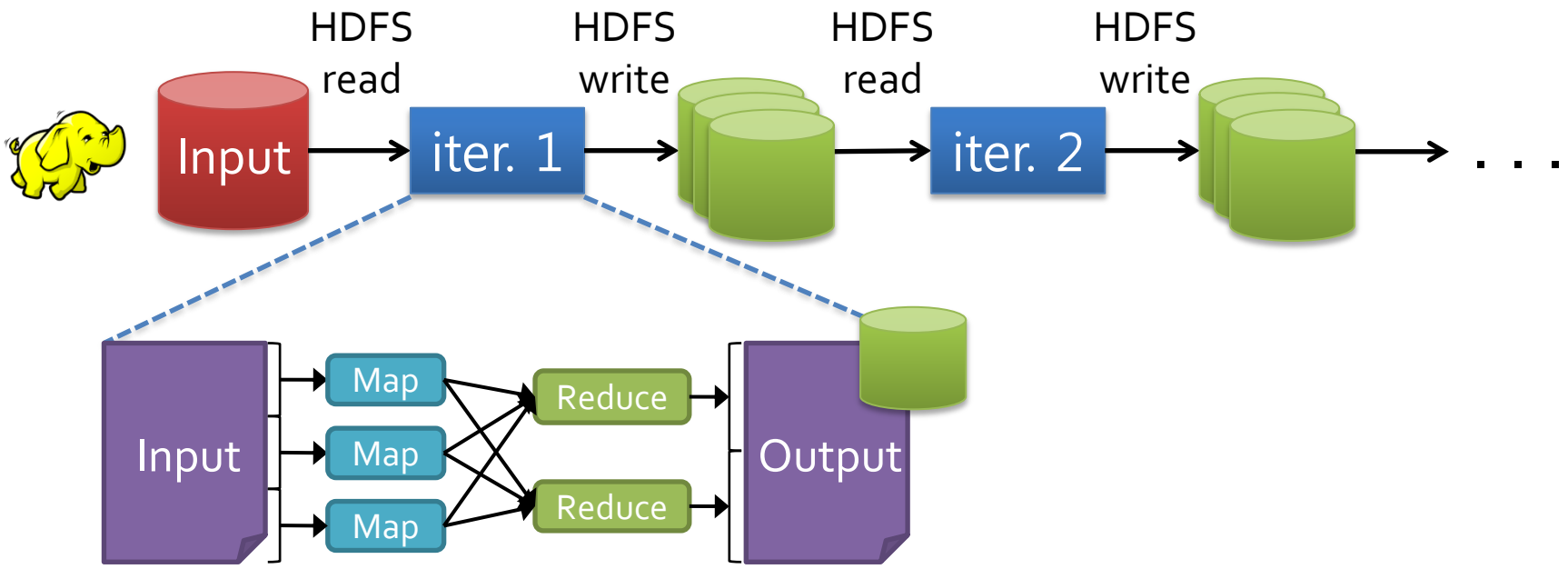


Administrative Web UIs

- http://<Standalone Master>:8080 (by default)
- http://<Worker>:8081 (by default)
- http://<Driver Node>:4040 (by default)
- **http://<History Server>:18080 (by default)**



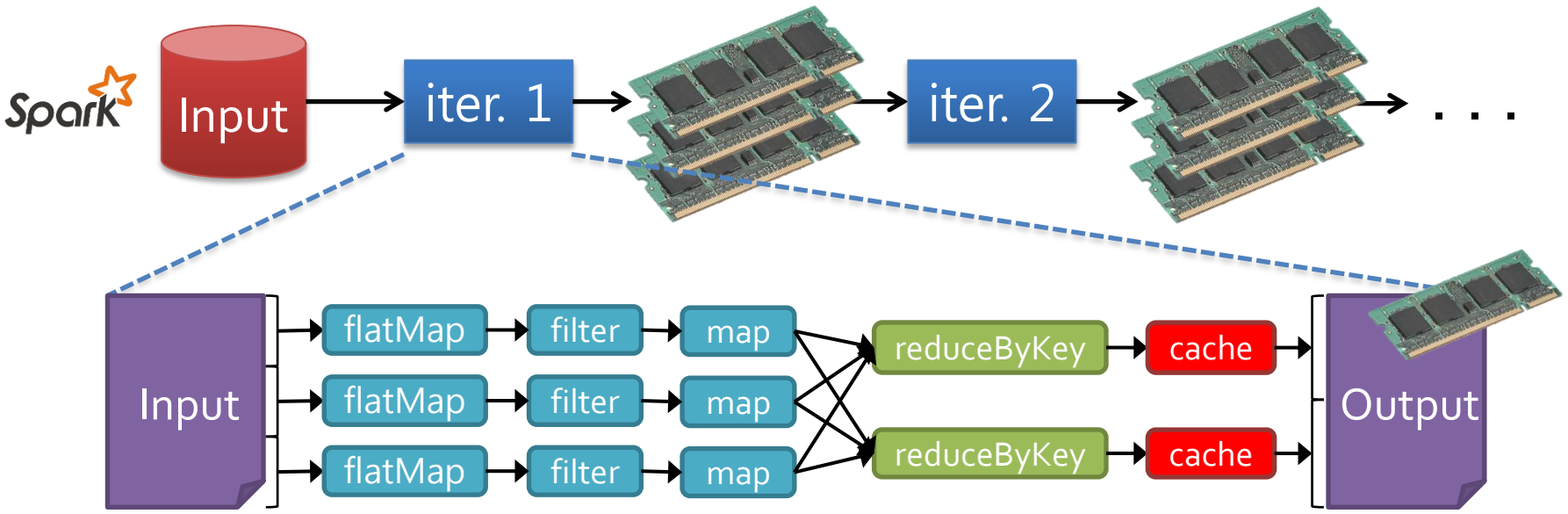
Hadoop – on Disk…. Limitations



- Slow due to replication, serialization, and disk IO
- Inefficient for

  : Iterative algorithms (Machine Learning, Graphs & Network Analysis)
  : Interactive Data Mining (R, Excel, Ad-hoc Reporting, Searching)

Spark – Solutions? In Memory + DAG….



- Load Data into Memory
- **10-100×** faster than disk
- Not tied to 2 stage Map/Reduce paradigm
  1. Extract a working set (ex. flatMap>filter>map>reduceByKey>….)
  2. Cache it (cache, persist)
  3. Query it repeatedly

Spark vs Hadoop – Speed (1/3)

- Spark won the **2014 Daytona Gray Sort 100TB Benchmark** (A team from Databricks including Spark committers)
- **Hadoop** : 2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks
- **Spark** : Amazon EC2 i2.8xlarge nodes x 32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD

	Data Size	Time	Nodes	Cores
Hadoop MR (2013)	102.5 TB	72 min	2,100	50,400 physical
Apache Spark (2014)	100 TB	23 min	206	6,592 virtualized

3X faster using **10X fewer machines** **on Disk**

Spark vs Hadoop – Speed (2/3)

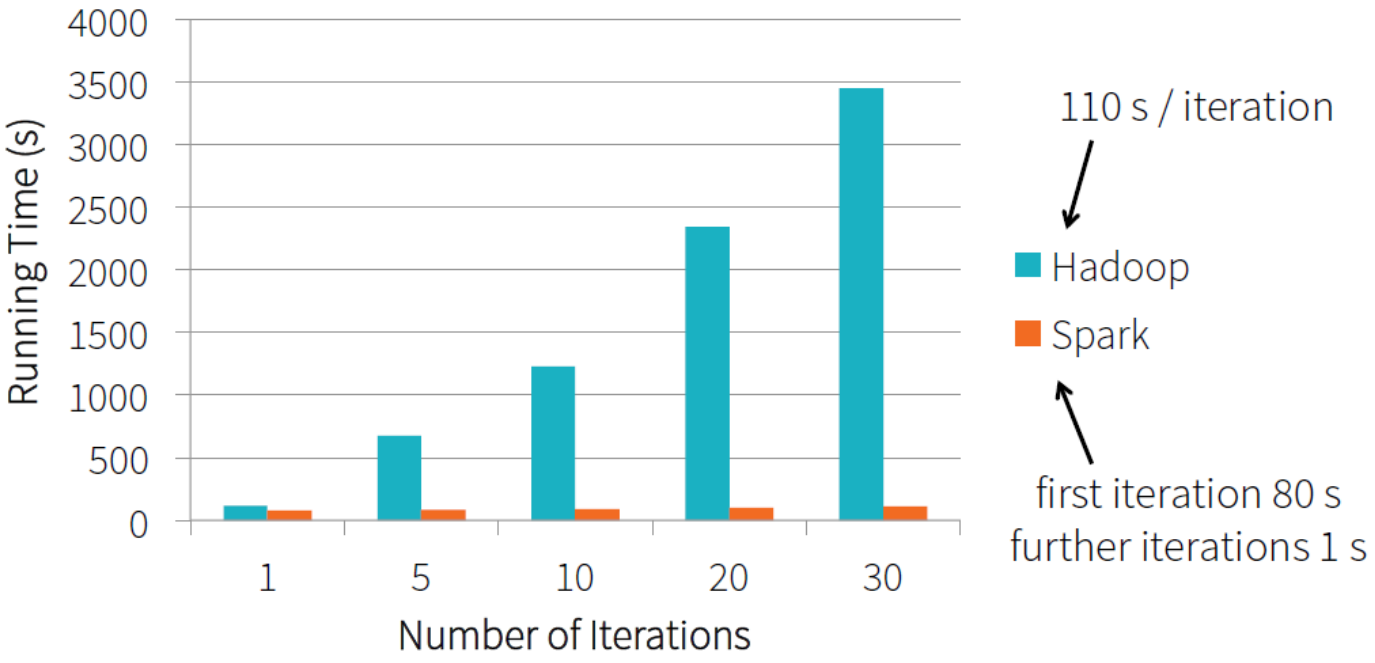
- Spark won the **2016 CloudSort Benchmark (Cost to sort 100TB of data)(both Daytona and Indy category)** (A joint team from Nanjing University, Alibaba Group, and Databricks)
- **University of California, San Diego** : 330 Amazon EC2 r3.4xlarge nodes x (16 vCores - 2.50Ghz Intel Xeon E5-2670 v2, 122 GB memory, 320GB SSD, 8x135GB EBS gp2)
- **NADSort(Spark)** : 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk)

3X more cost-efficient than the previous world record

Spark vs Hadoop – Speed (3/3)

Example: Logistic Regression

Iterative algorithm used in machine learning



Iterative Algorithm in Memory

Spark vs Hadoop - Ease of Use (1/2)

Example – Word Count

Hadoop MapReduce

```
public static class WordCountMapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}

public static class WorkdCountReduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Spark

```
val spark = new SparkContext(master, appName, [sparkHome], [jars])
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

2~5x Less Code

Spark vs Hadoop - Ease of Use (2/2)

Low level API vs. High Level API

Hadoop MapReduce

- map
- reduce

Spark

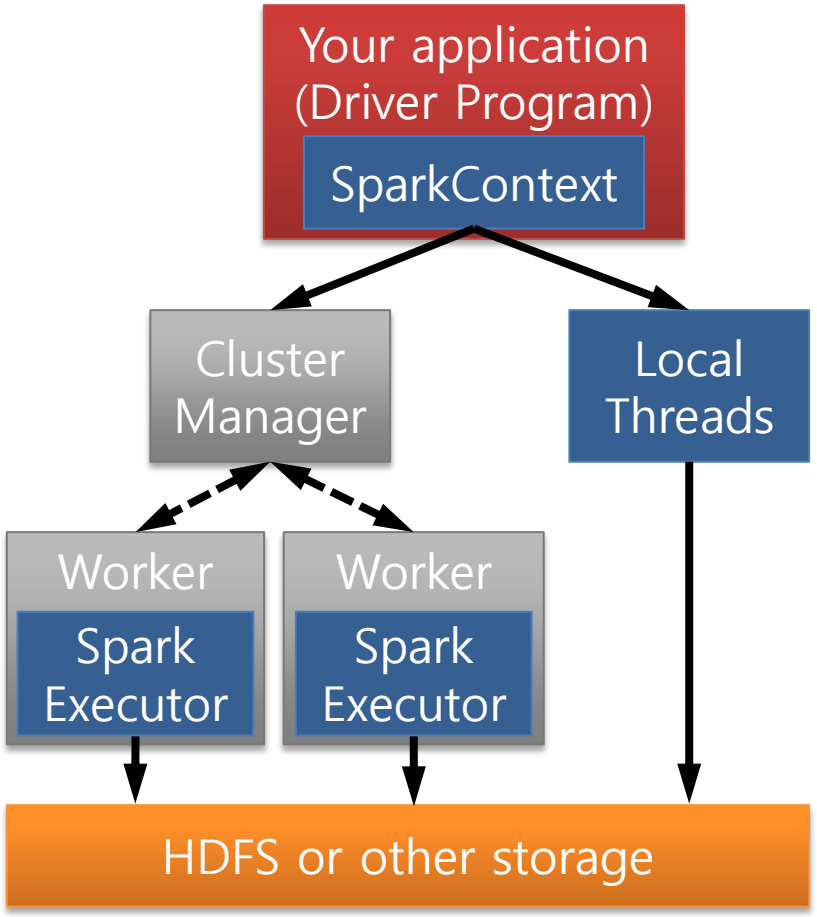
- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip

sample
take
first
partitionBy
mapWith
pipe
save ...

High Level API (Over 80 High-level Operators)

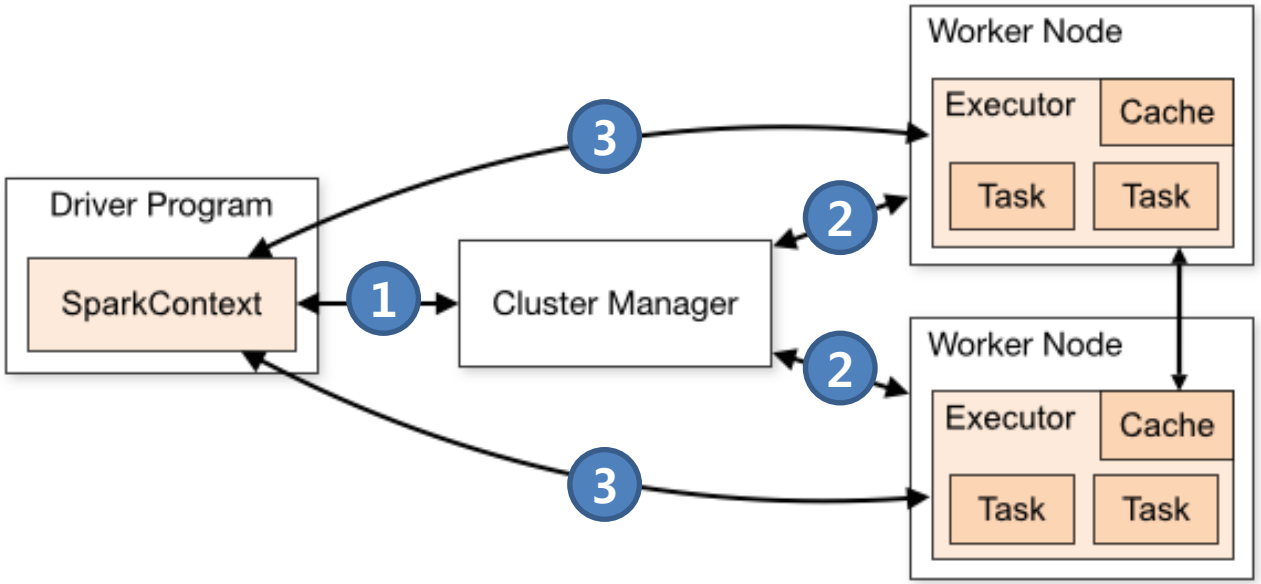
Spark Software Components

- Spark runs as a library in your program (one instance per app)
- **Spark Driver**
  : Client side application that creates Spark Context
- **Spark Context**
  : Talks to Spark Driver, Cluster Manager to launch Spark Executors
- Runs tasks locally or on a cluster manager
  : Standalone, Mesos, YARN
- **Spark Application** is two programs
  : Driver program
  : Workers program
- Worker programs run on cluster nodes(Executors) or in local threads

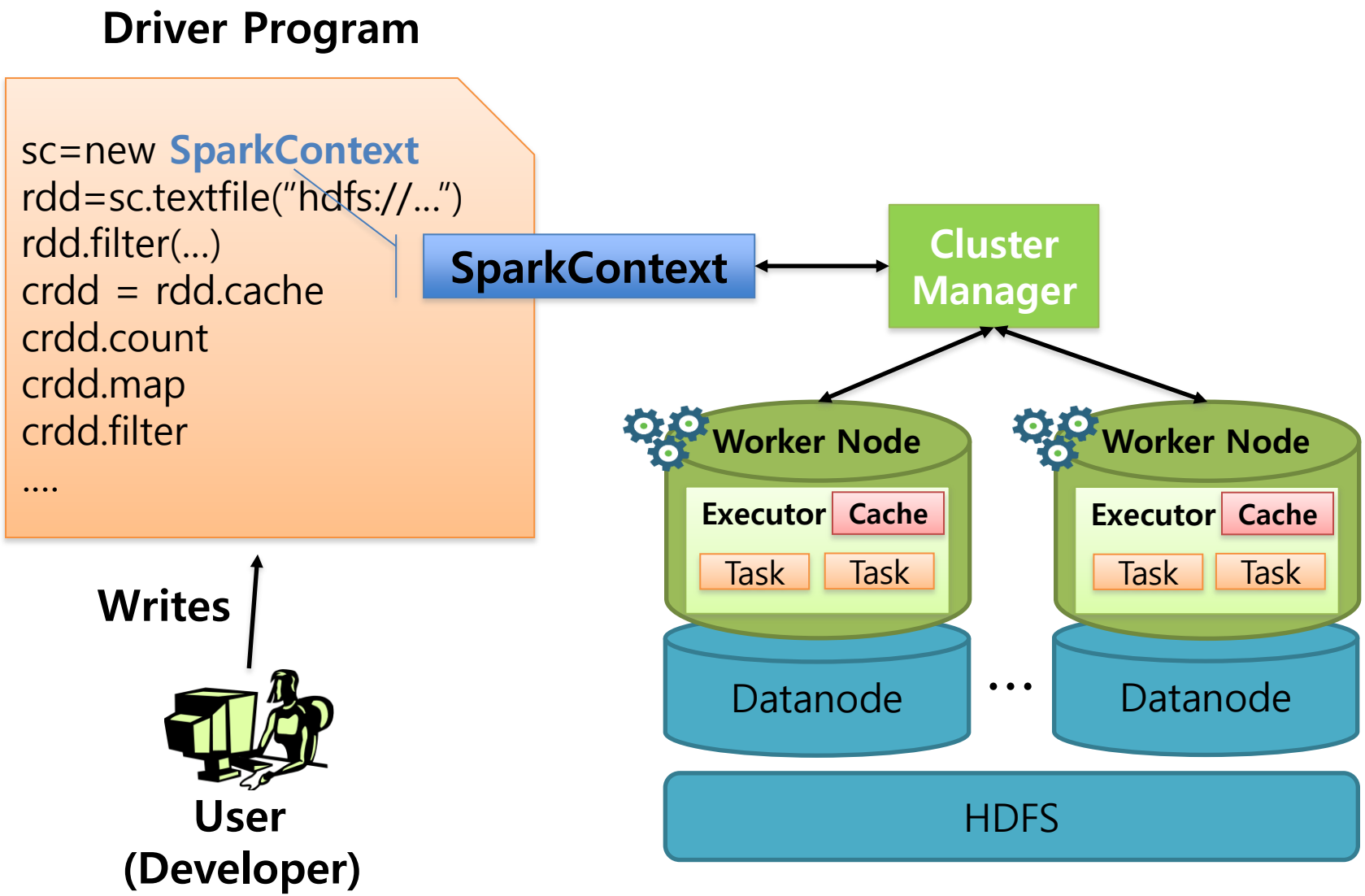


Spark Software Components

1. When you submit your program, the SparkContext will connect to cluster managers
2. Once connected, Spark was assigned with executors on nodes in the cluster
3. It sends your application code (defined by JAR or even Python files passed to SparkContext) to the executors. Then, SparkContext sends tasks for the executors to run and return the results.



Spark Programming Model - SparkContext



Spark Programming Model - RDD

Driver Program

```
sc=new SparkContext
rdd=sc.textfile("hdfs://...")
rdd.filter(...)
crdd = rdd.cache
crdd.repartition(100)
crdd.map
crdd.flatMap
....
```

RDD (Resilient Distributed Dataset)

- **Work with distributed collections as you would with local ones**
- Immutable Data structure
- In-memory (explicitly)
- Fault Tolerant
- Parallel Data Structure
- Controlled partitioning to optimize data placement
- Can be manipulated using rich set of operators.

Writes

User (Developer)

Spark Programming Model - RDD

- Programming Interface
  : Programmer can perform 3 types of operations

Transformations

- Create a **new dataset** from and existing one.
- **Lazy** in nature. They are executed only when some **action** is performed.
- Example :
  - Map(func)
  - Filter(func)
  - Distinct()

Actions

- **Returns** to the driver program a **value** or **exports** data to a **storage** system after performing a computation.
- Example:
  - Count()
  - Reduce(func)
  - Collect
  - Take()

Persistence

- For **caching** datasets in-memory for **future operations**.
- Option to store on **Disk or RAM** or mixed (Storage Level).
- Example:
  - Persist(Storage Level)
  - Cache()

RDD - Lazy Evaluation + No Cache

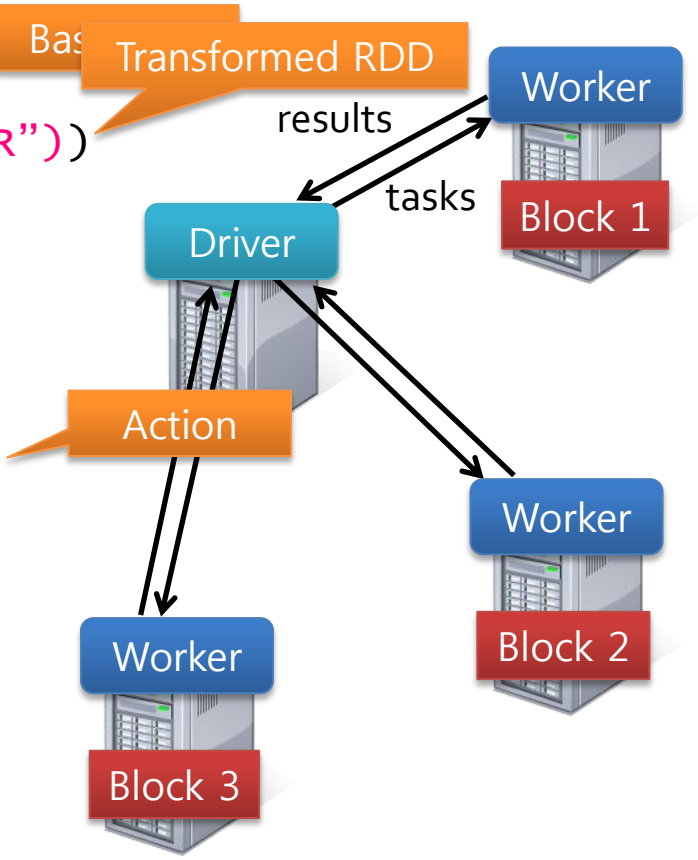
Example: Log Mining

- Load error messages from a log, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
```

```
// action #1
messages.filter(_.contains("foo")).count
// action #2
messages.filter(_.contains("bar")).count
```

Result: scaled to 1 TB data in **170 sec** for on-disk data



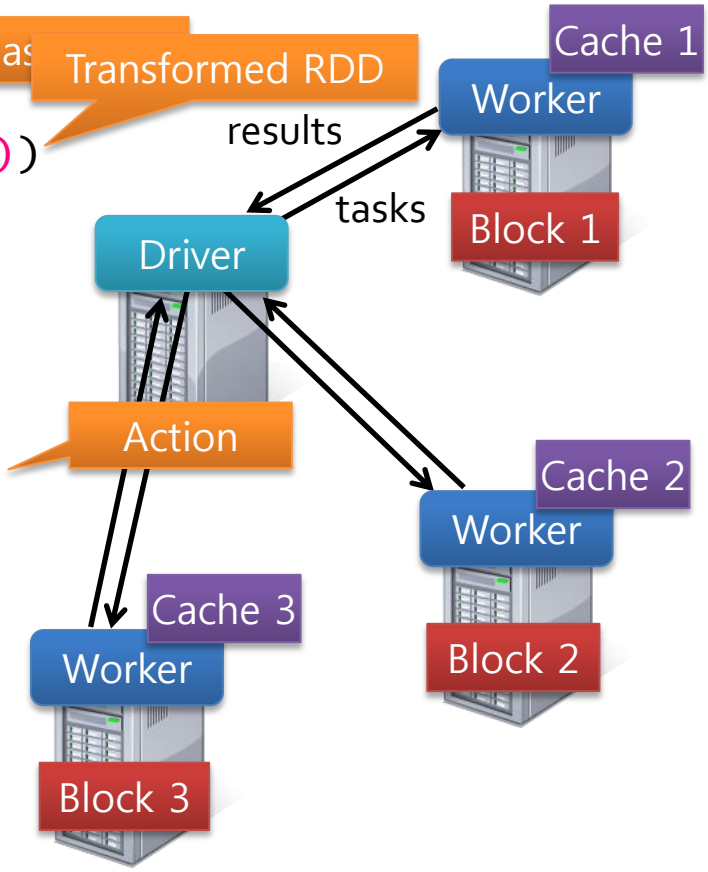
RDD - Lazy Evaluation + Cache

Example: Log Mining

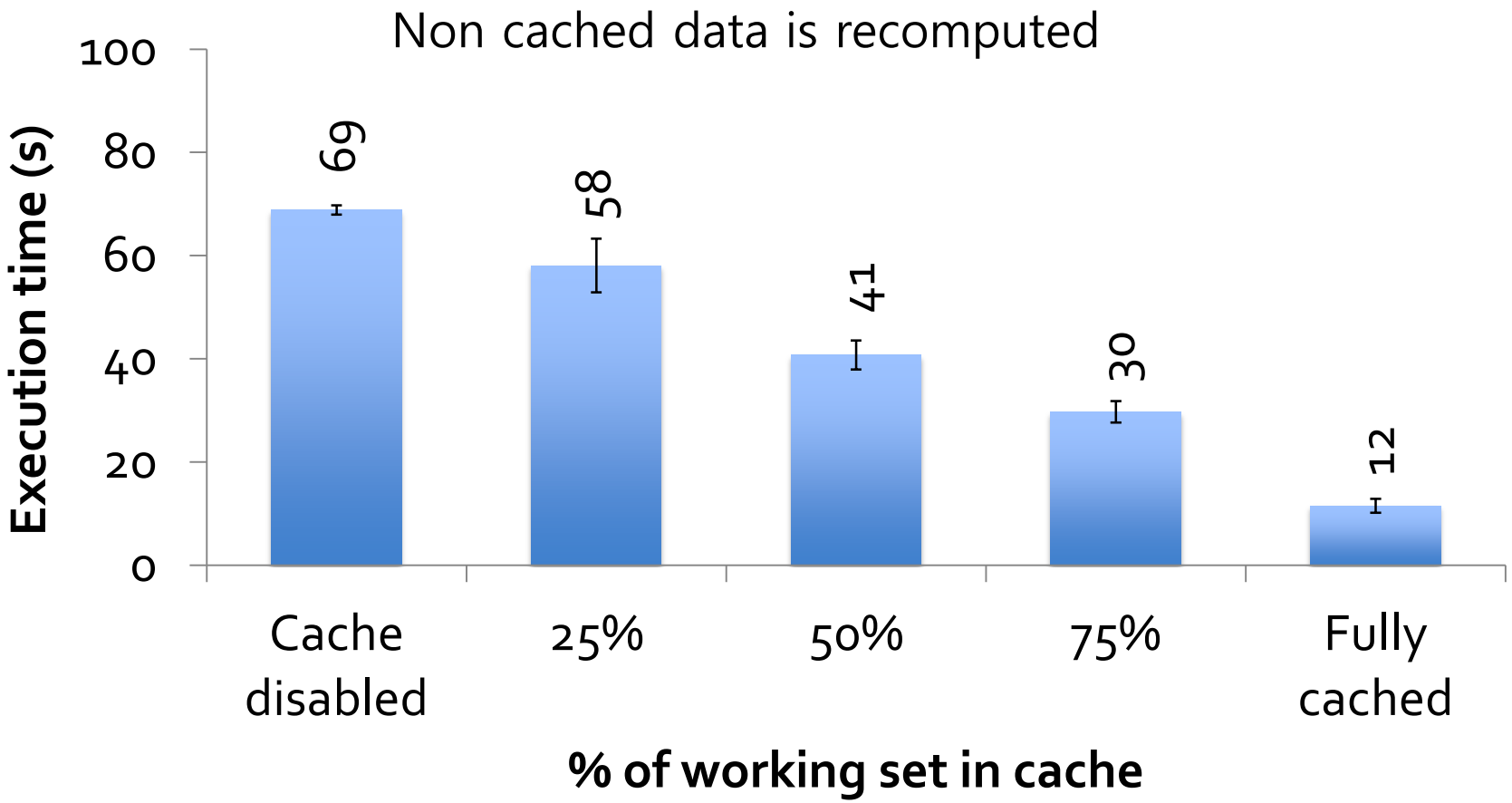
- Load error messages from a log **into memory**, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

// action #1
cachedMsgs.filter(_.contains("foo")).count
// action #2
cachedMsgs.filter(_.contains("bar")).count
```

Result: scaled to 1 TB data in **5-7 sec**
(vs **170 sec** for on-disk data)



RDD - Behavior with Less RAM

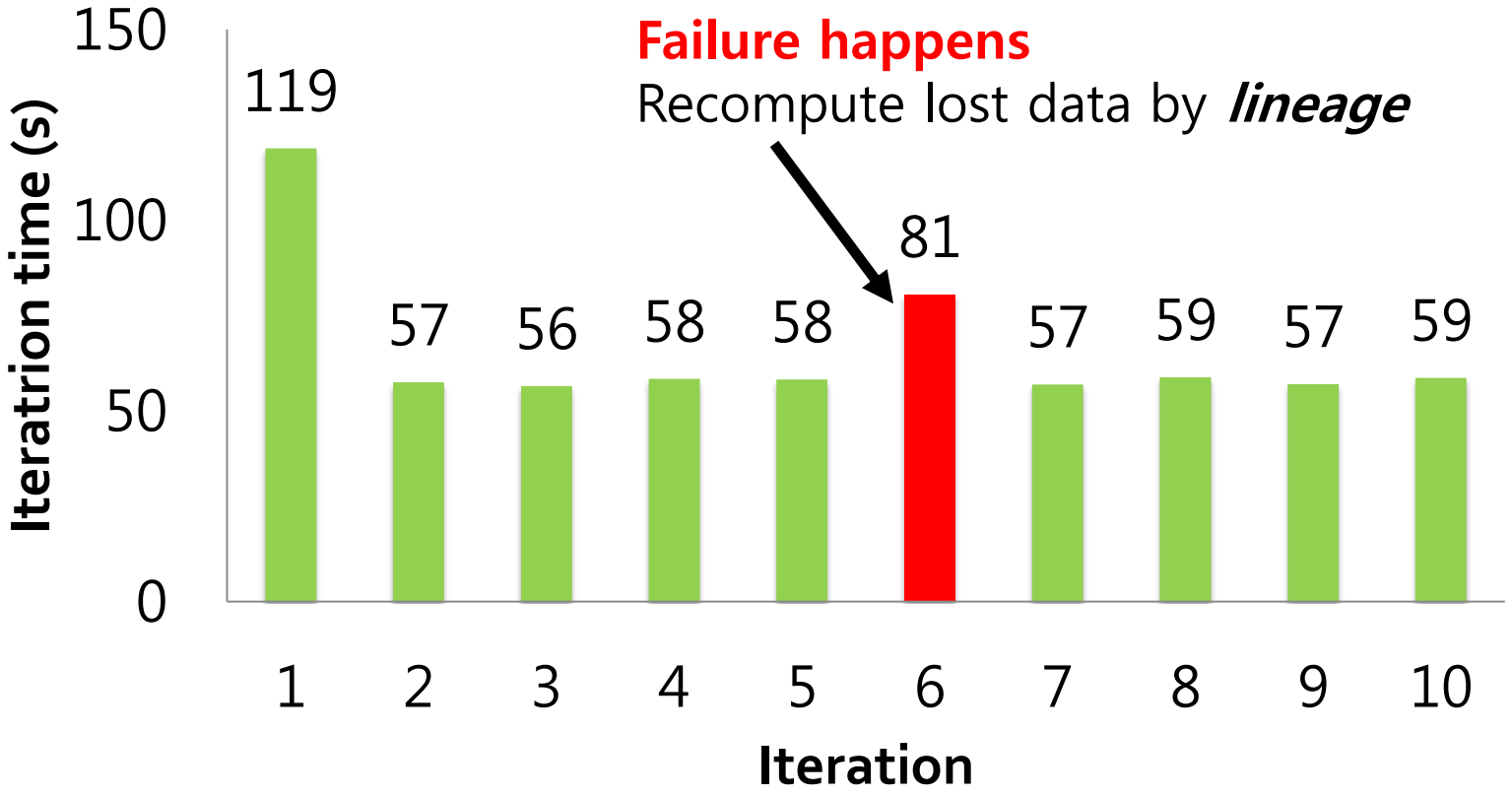


RDD - Fault Tolerance

- **Lineage** :
  RDDs track **the series of transformations** used to build them **to recompute lost data**

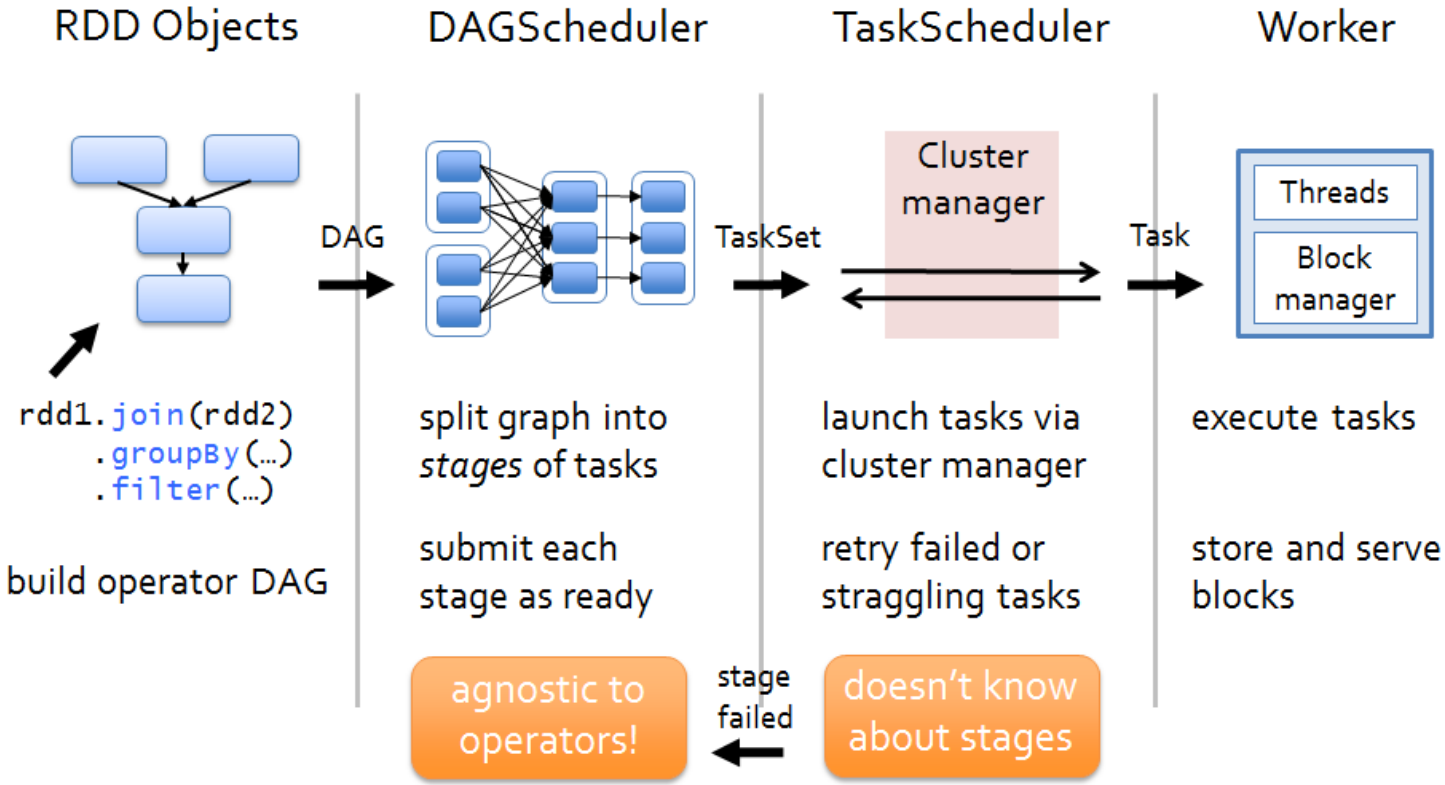
```
val msgs = sc.textFile("hdfs://...").filter(s => s.startsWith("ERROR"))
                                    .map(s => s.split("₩t")(2))
```

RDD - Fault Recovery Test



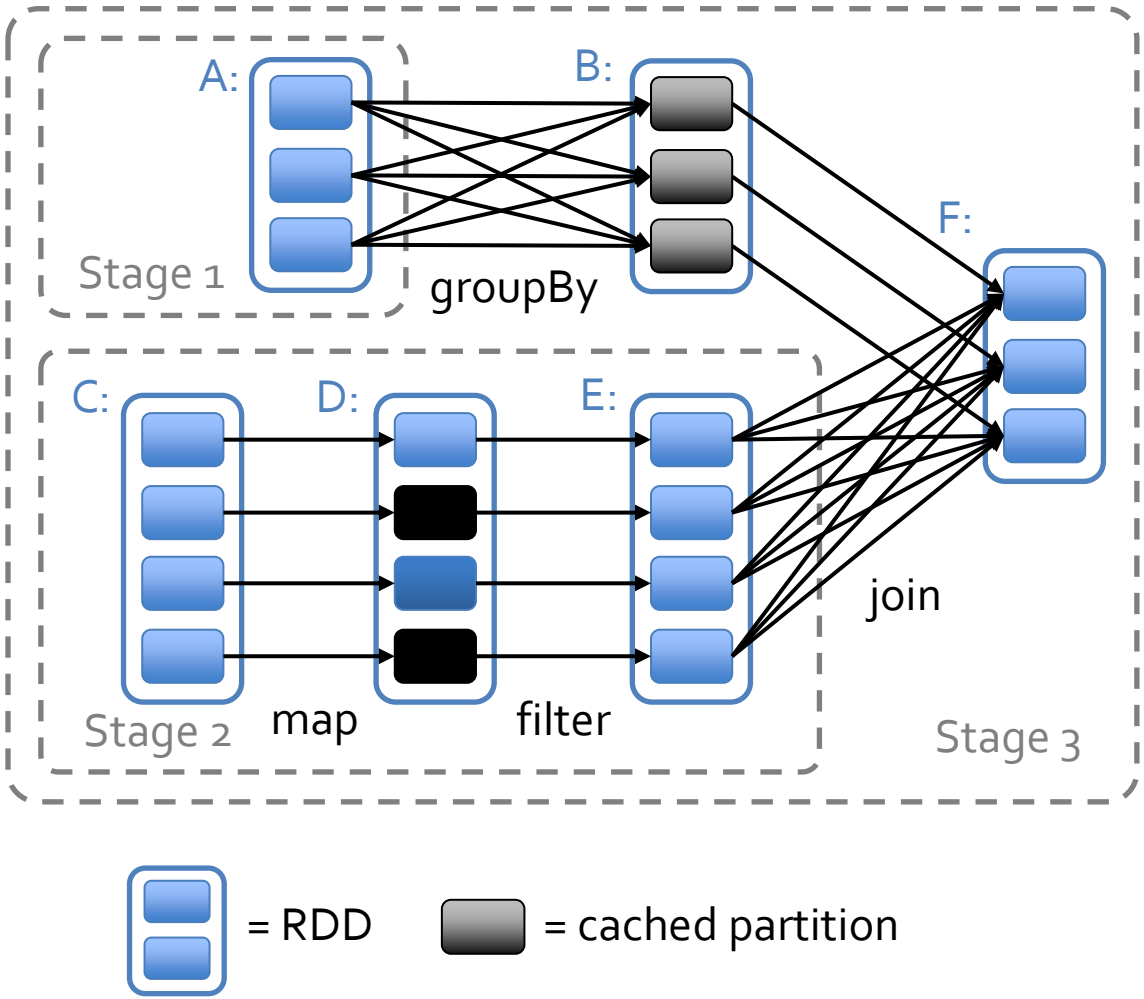
How Spark works

- User application create RDDs, transform them, and run actions.
- This results in a DAG (Directed Acyclic Graph) of operators.
- DAG is compiled into Stages
- Each Stage is executed as a series of Task (one Task for each Partition).



How Spark works

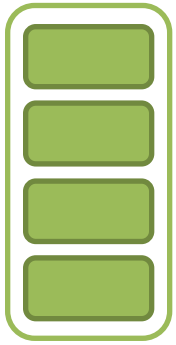
- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles



How Spark works - DAG

sc.textFile(“/wiki/pagecounts”)

RDD[String]



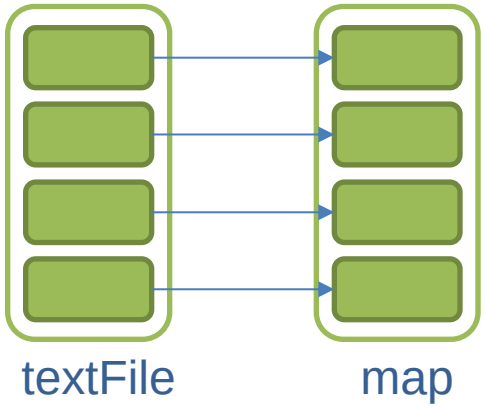
textFile

How Spark works - DAG

```
sc.textFile("/wiki/pagecounts")
    .map(line => line.split("\t"))
```

RDD[String]

RDD[List[String]]



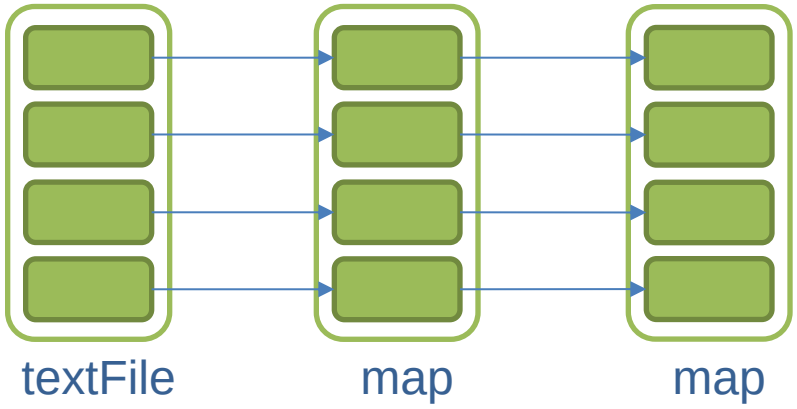
How Spark works - DAG

```
sc.textFile("/wiki/pagecounts")
    .map(line => line.split("\t"))
    .map(R => (R[0], int(R[1])))
```

RDD[String]

RDD[List[String]]

RDD[(String, Int)]



How Spark works - DAG

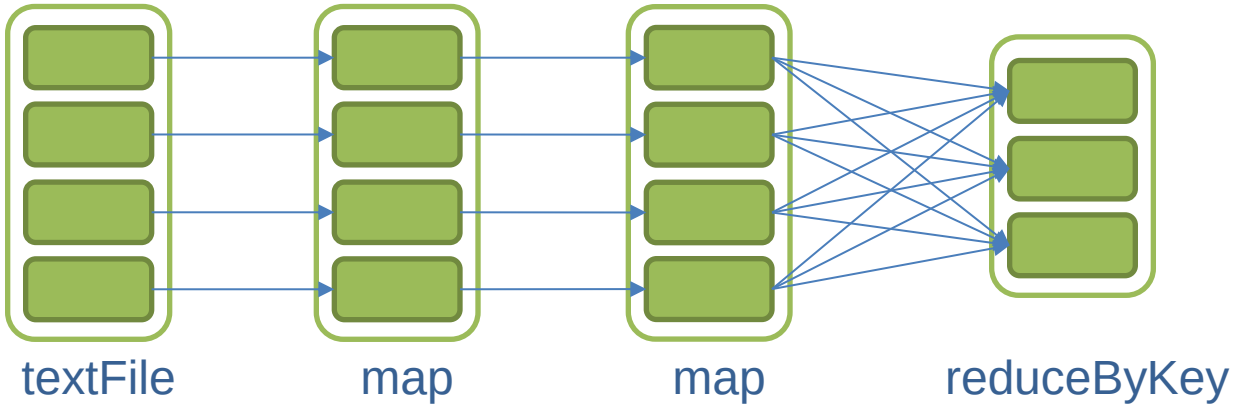

```
sc.textFile("/wiki/pagecounts")
    .map(line => line.split("\t"))
    .map(R => (R[0], int(R[1])))
    .reduceByKey(_+_, 3)
```

RDD[String]

RDD[List[String]]

RDD[(String, Int)]

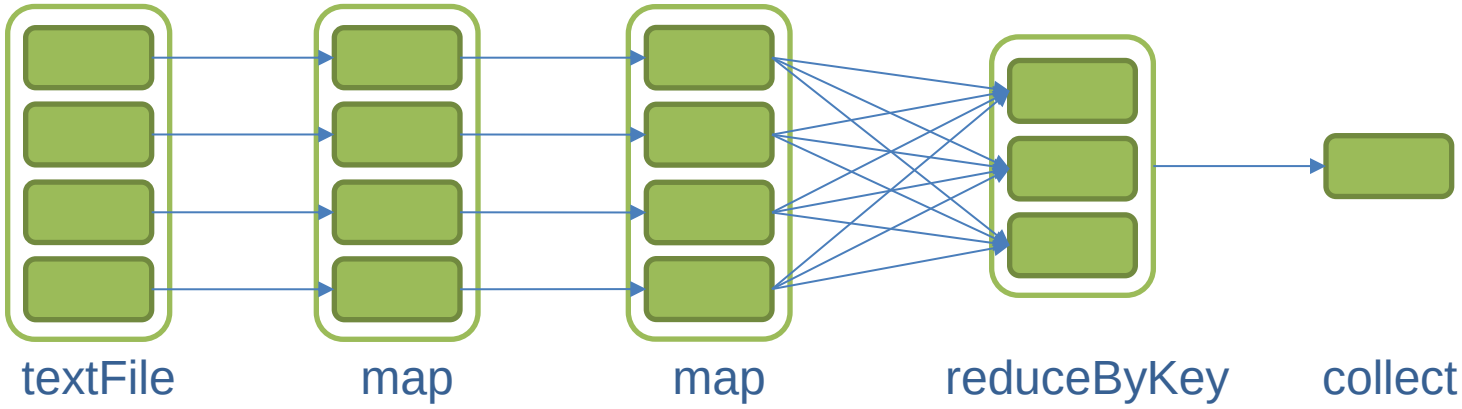
RDD[(String, Int)]



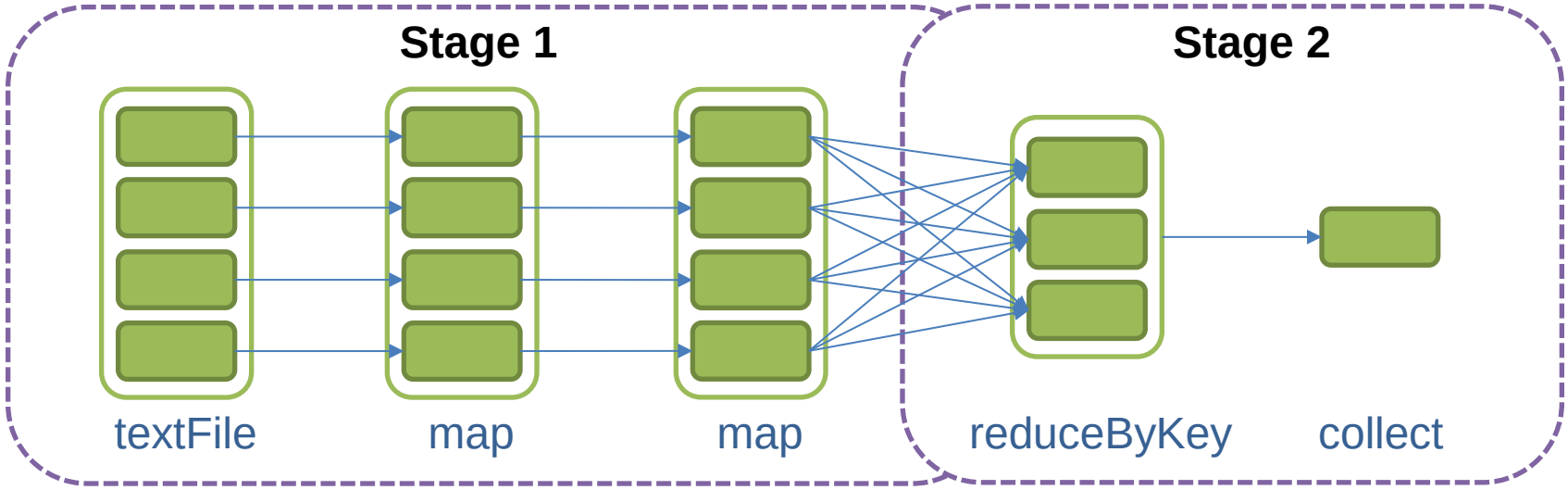
How Spark works - DAG

```
sc.textFile("/wiki/pagecounts")
    .map(line => line.split("\t"))
    .map(R => (R[0], int(R[1])))
    .reduceByKey(_+_, 3)
    .collect()
```

RDD[String]
RDD[List[String]]
RDD[(String, Int)]
RDD[(String, Int)]
Array[(String, Int)]

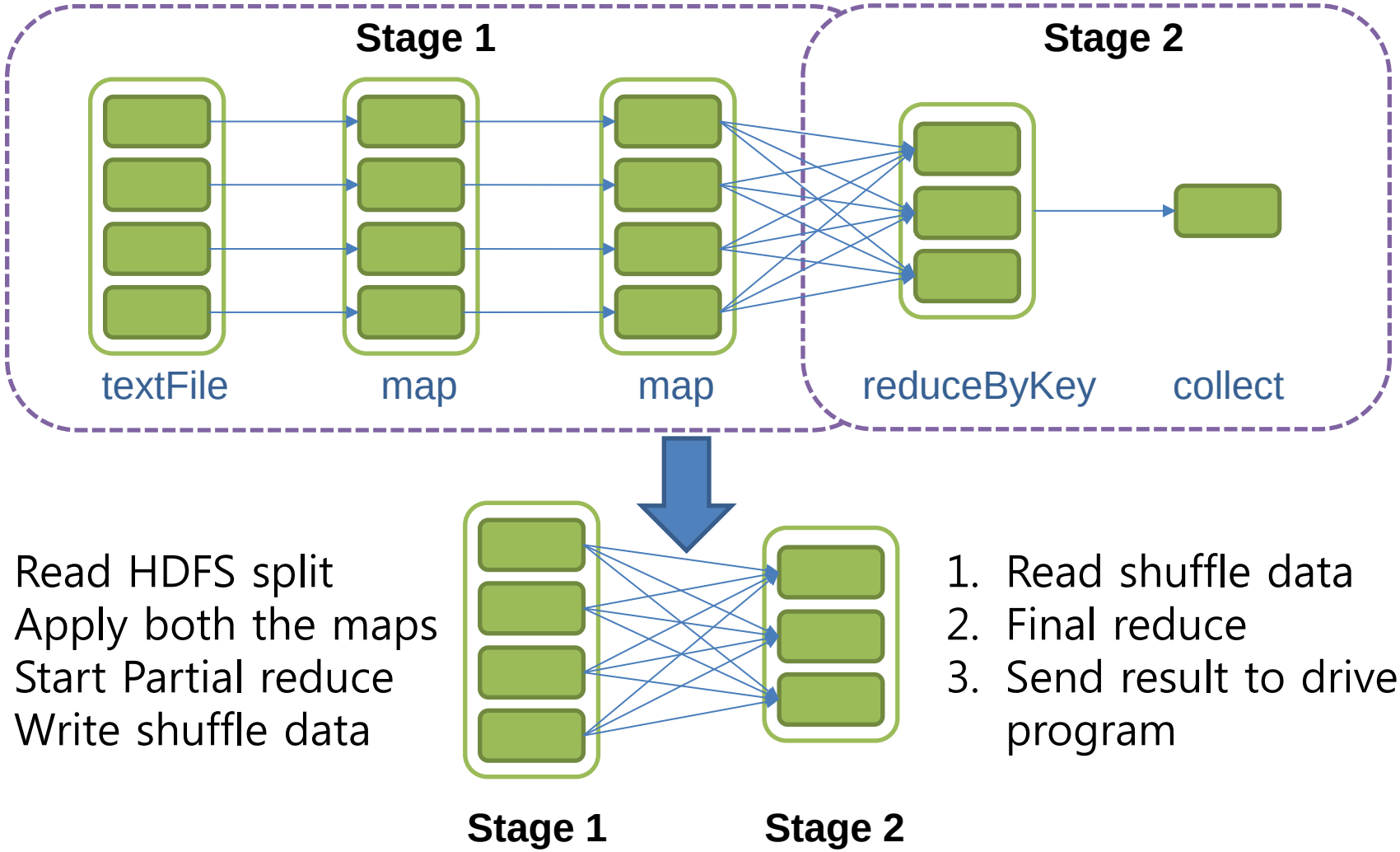


How Spark works - Execution Plan



- **Stages** are sequences of RDDs, that **don't have a Shuffle** in between

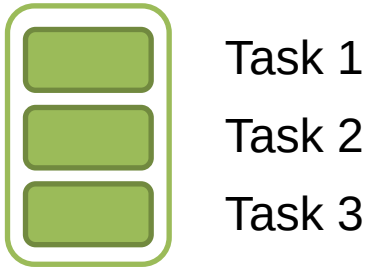
How Spark works - Execution Plan



1. Read HDFS split
2. Apply both the maps
3. Start Partial reduce
4. Write shuffle data

1. Read shuffle data
2. Final reduce
3. Send result to driver program

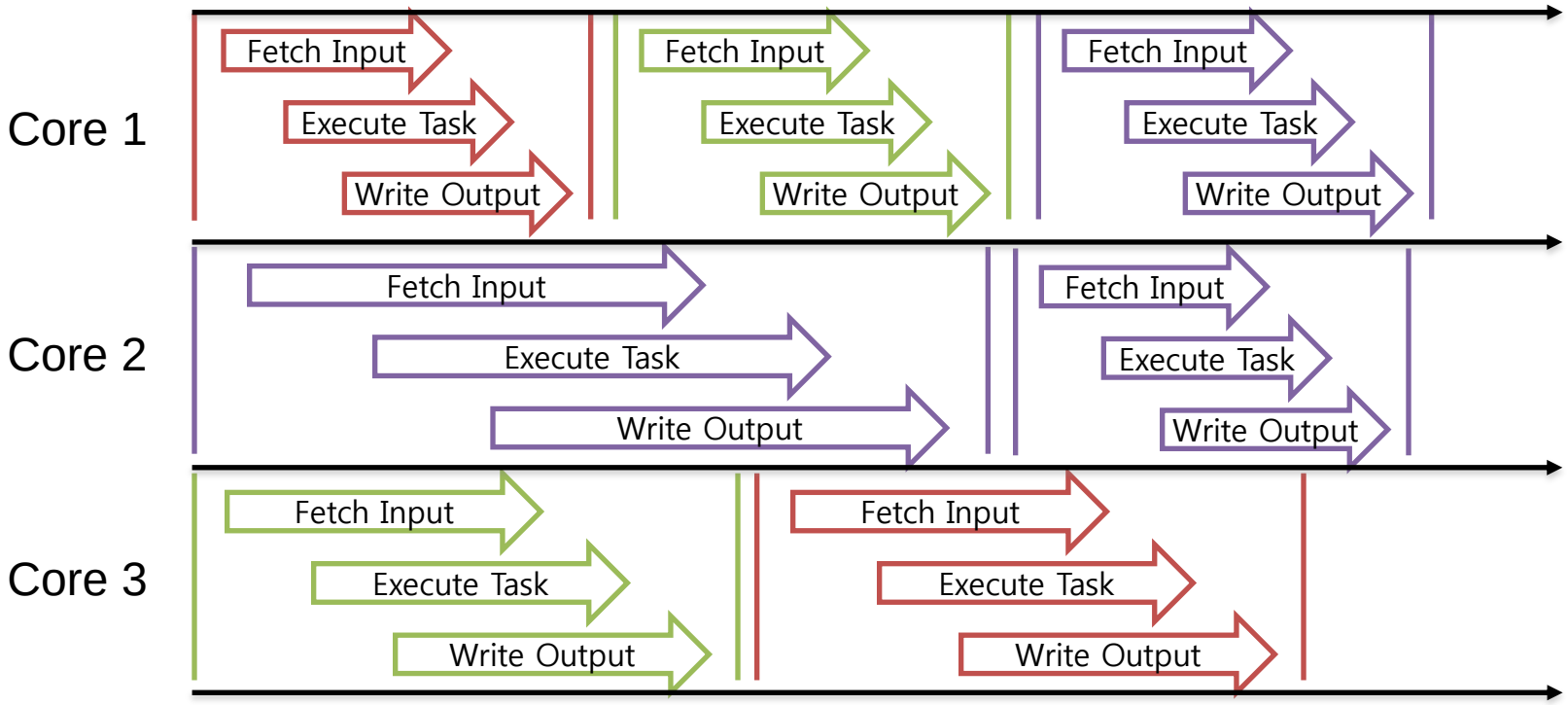
How Spark works – Stage Execution



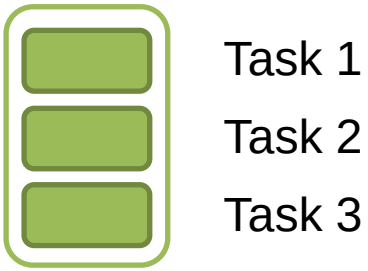
- Create a task for each Partition in the new RDD
- Serialize the Task
- Schedule and ship Tasks to Executors

And all this happens internally (you need to do nothing)

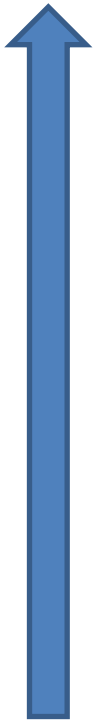
How Spark works – Executor, Task



- Executor(JVM 프로세스)에 3개 Core 할당
- Executor 내에서 3개 Task(쓰레드, 파티션) 병렬 실행



How Spark works - Summary of Components

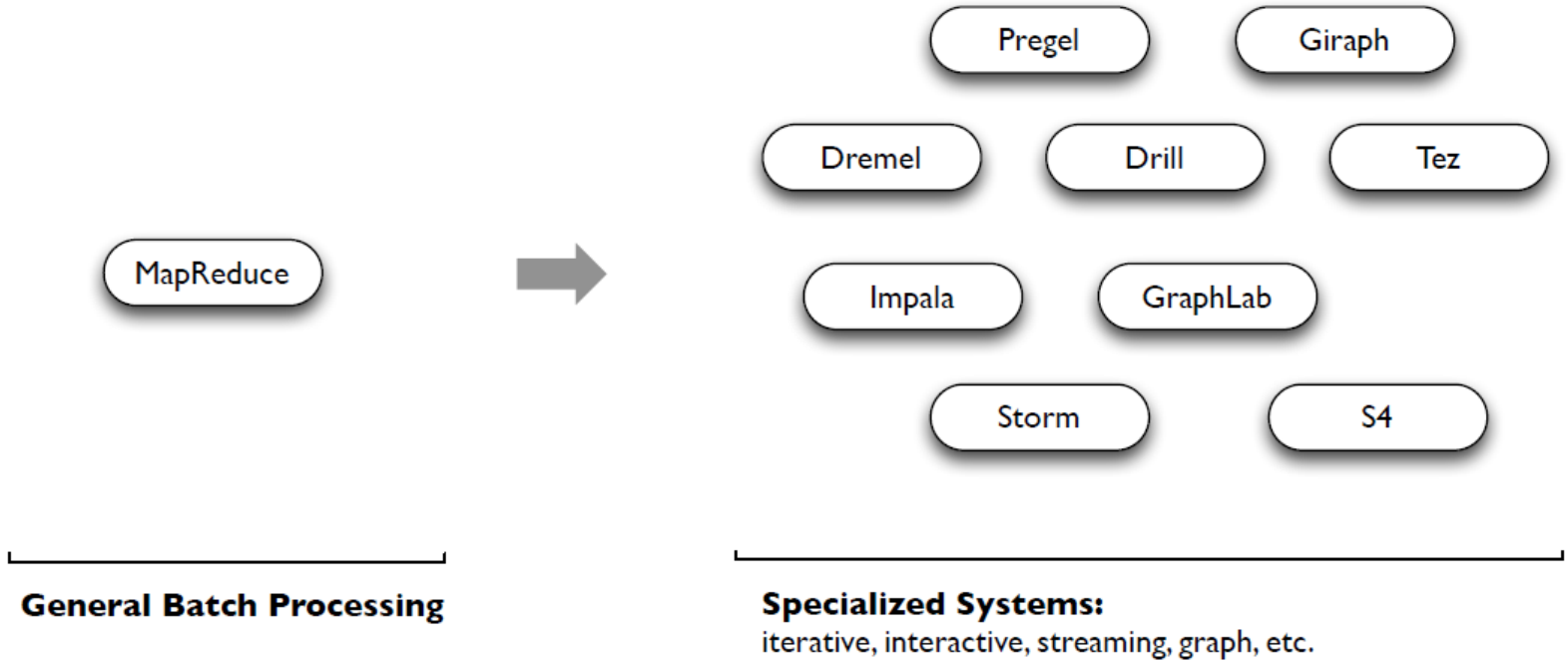


- **Task** : The fundamental unit of execution in Spark
- **Stage** : Set of Tasks that run parallel
- **DAG** : Logical Graph of RDD operations
- **RDD** : Parallel dataset with partitions

MapReduce limitations

- MapReduce use cases showed two major limitations :
  1. **difficultly** of programming directly in MR
  2. **performance bottlenecks**, or batch **not fitting the use cases**

- In short, MR doesn't compose well for large applications
- Therefore, **people built *specialized systems*** as workarounds…

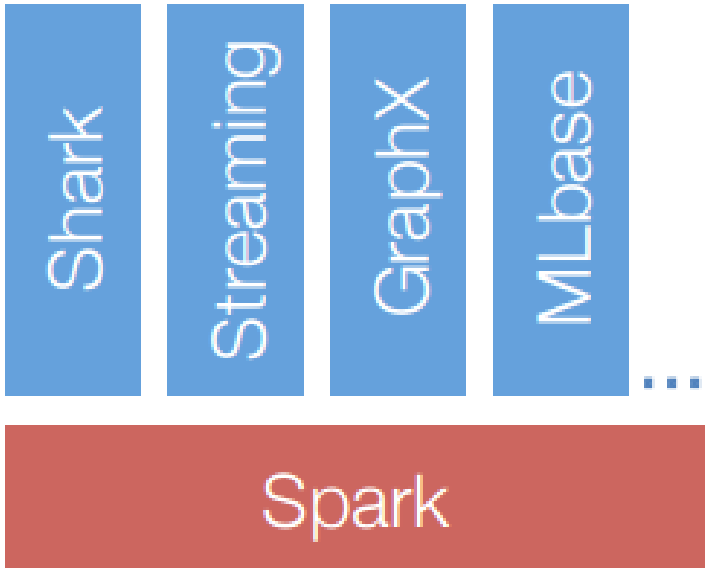


Spark's Approach

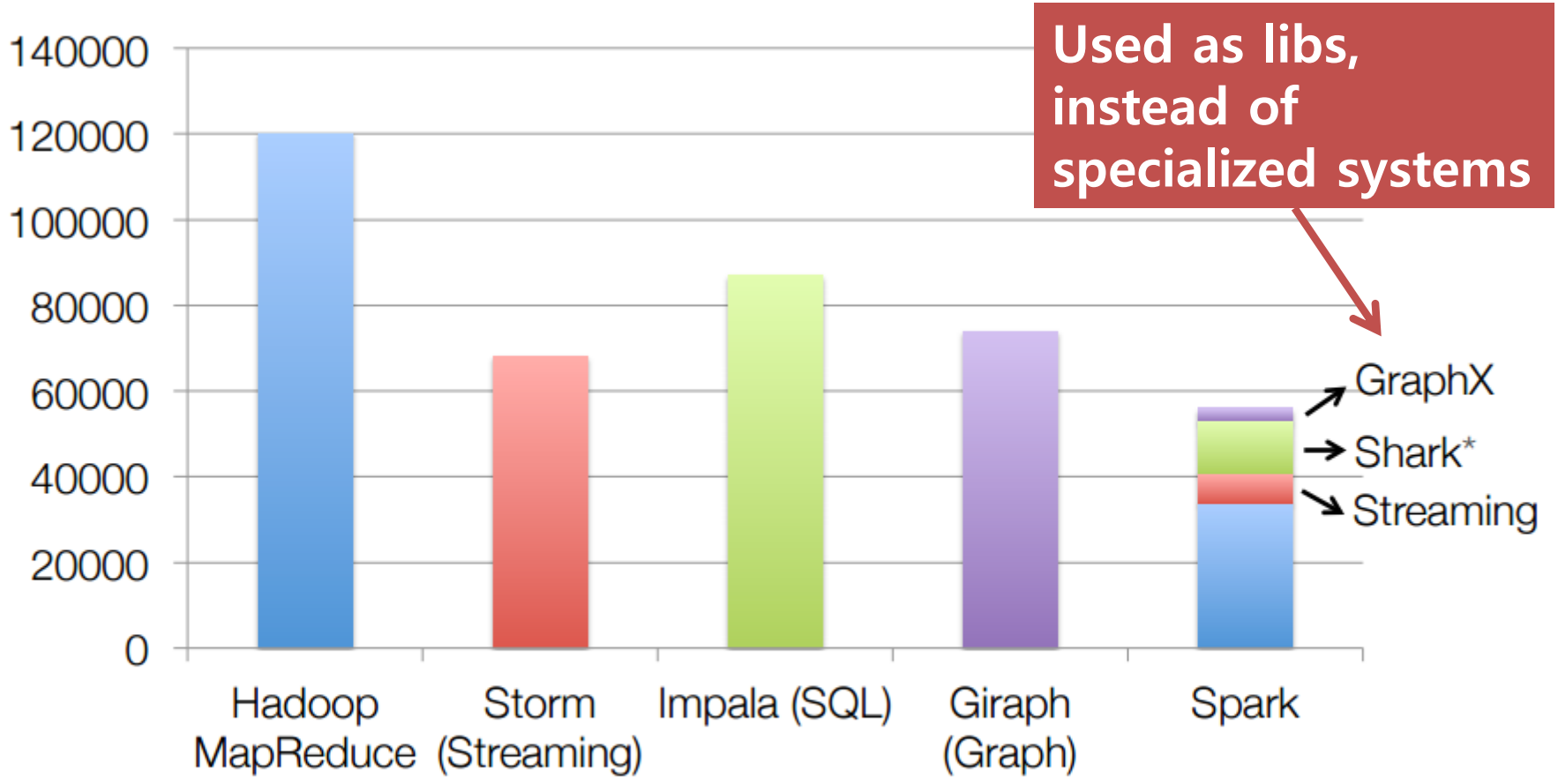
- Instead of various specialized systems, **generalize MapReduce** to support new apps **within same engine**

- Two additions are enough to express previous models
  - » **Fast data sharing**
  - » **General task DAGs**

- Unification has big benefits
  - » More efficient for engine
  - » Much simpler for users



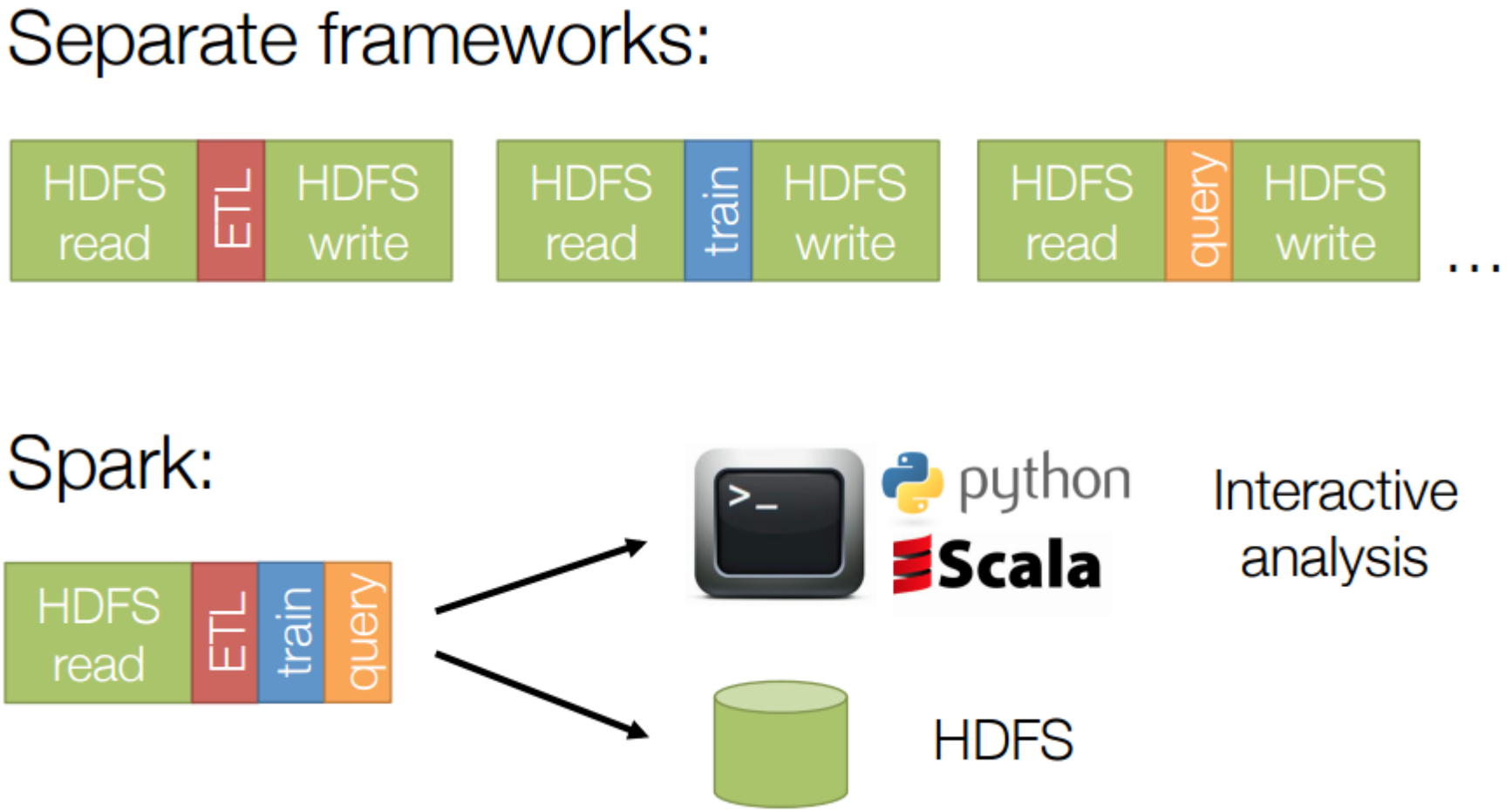
Spark's Code Size



non-test, non-example source lines

* also calls into Hive

What it Means for Users



- No copying or ETL of data between systems
- Combine processing types in one program

Combining Processing Types

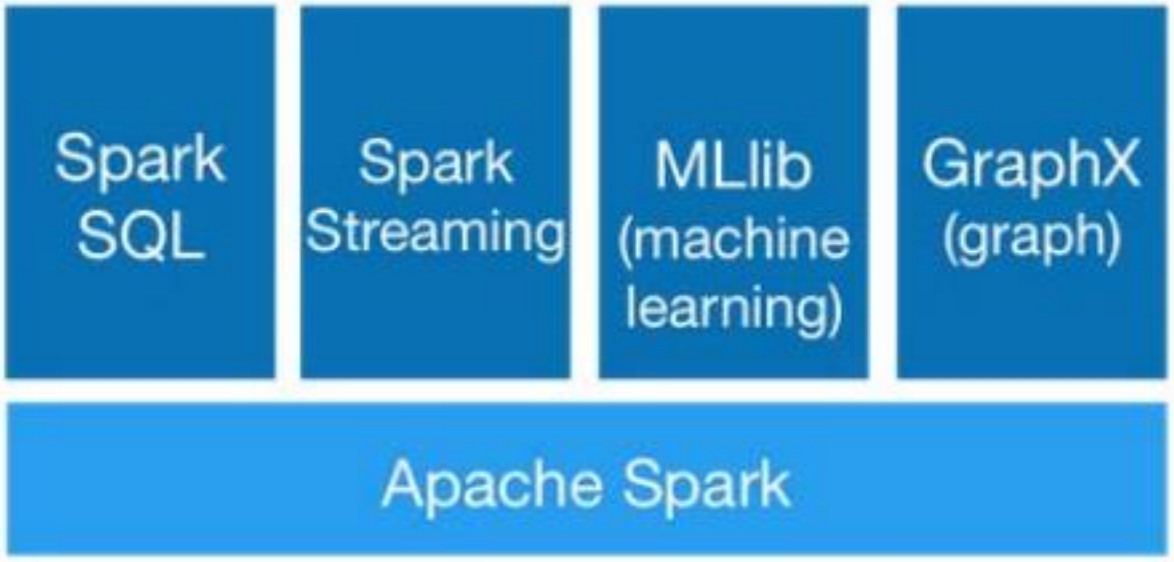
Spark SQL + Mllib + Spark Streaming….

```
// Spark SQL….
df.createOrReplaceTempView("historic_tweets")
val points = spark.sql(“select latitude, longitude from historic_tweets”)

// MLlib….
val kmeans = new KMeans().setK(10).setSeed(1L)
val model = kmeans.fit(points)

// Spark Streaming….
TwitterUtils.createStream(ssc, None)
            .map(t => (model.computeCost(t.location), 1))
            .reduceByKeyAndWindow(_ + _, Seconds(60))
```

Spark Components (Libraries)

Benefits of a Unified Platform

- No copying or ETL of data between systems
- Combine processing types in one program
- Code reuse
- One system to learn
- One system to maintain



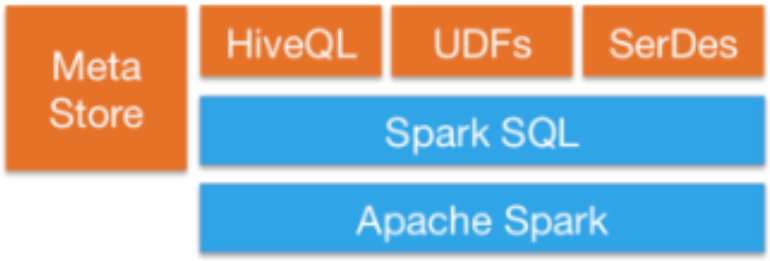
Spark Components - Spark SQL

- Query structured data inside Spark programs
- Access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC

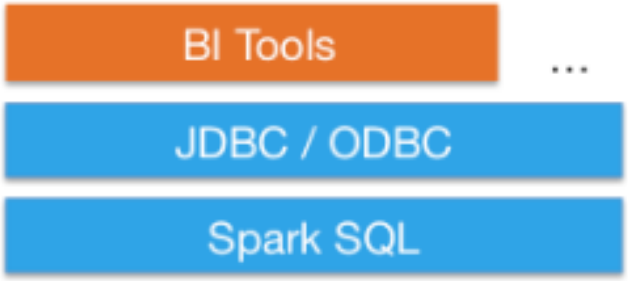
```
// Read JSON....
val df = spark.read.json("examples/src/main/resources/people.json")

// Register the DataFrame as a SQL temporary view....
df.createOrReplaceTempView("people")

// Execute SQL query....
val sqlDF = spark.sql("SELECT * FROM people")
```

Spark Components - Spark SQL

- Query structured data inside Spark programs
- Access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC
- Run unmodified Hive queries on existing data
- JDBC and ODBC connectivity for business intelligence tools
- **Spark 2.0**
  : Supports both ANSI-SQL as well as Hive QL
  : Scalar Subquery support (NOT IN, IN, EXISTS, NOT EXISTS)
  : 2 - 10X performance up (whole stage code generation)

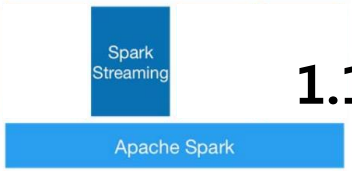


Spark SQL can use existing Hive metastores, SerDes, and UDFs.



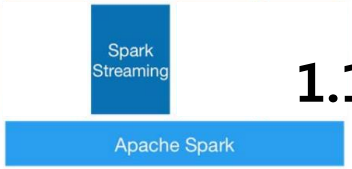
Use your existing BI tools to query big data.

Spark Components - Spark Streaming



- Spark Streaming expresses streams as a series of RDDs over time

Spark Components - Spark Streaming

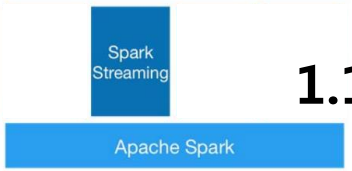


- Spark Streaming expresses streams as a series of RDDs over time
- Combine streaming with batch and interactive queries
- High Level APIs (Stateful, joins, aggregates, window etc.)

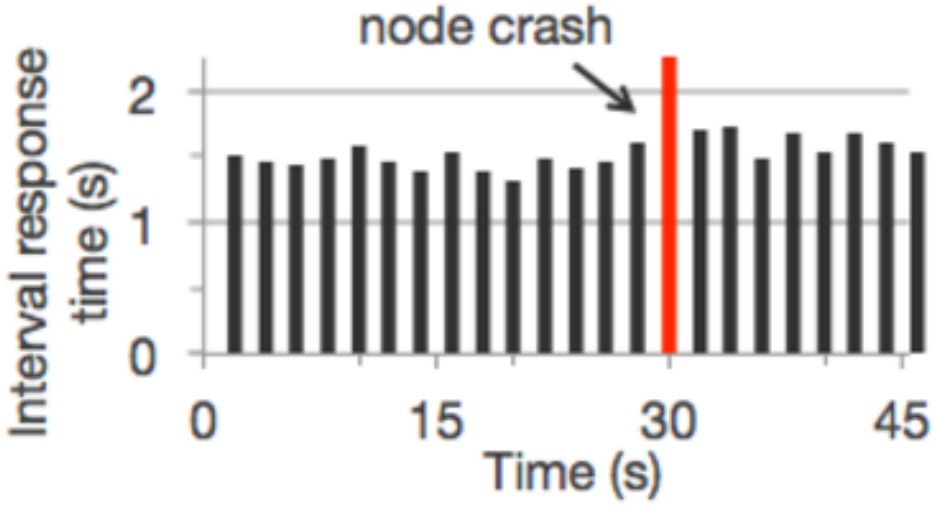
```
stream.join(historicCounts).filter {
  case (word, (curCount, oldCount)) =>
    curCount > oldCount
}
```

Find words with higher frequency than historic data

Spark Components - Spark Streaming

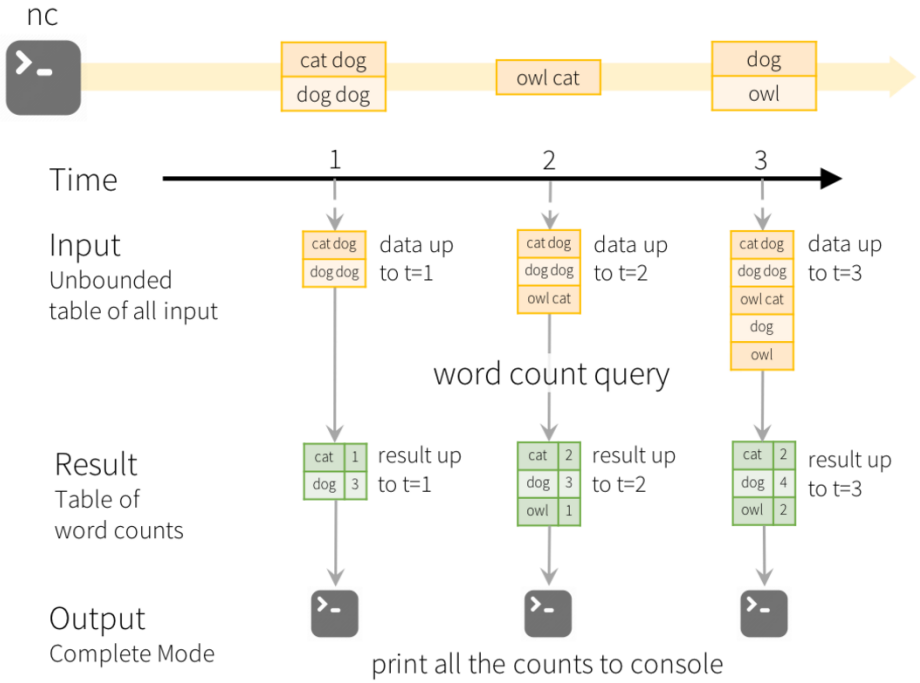


- Spark Streaming expresses streams as a series of RDDs over time
- Combine streaming with batch and interactive queries
- High Level APIs (Stateful, joins, aggregates, window etc.)
- Fault Tolerant (exactly once semantics achievable)

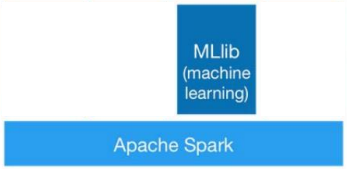


Spark Components - Spark Streaming

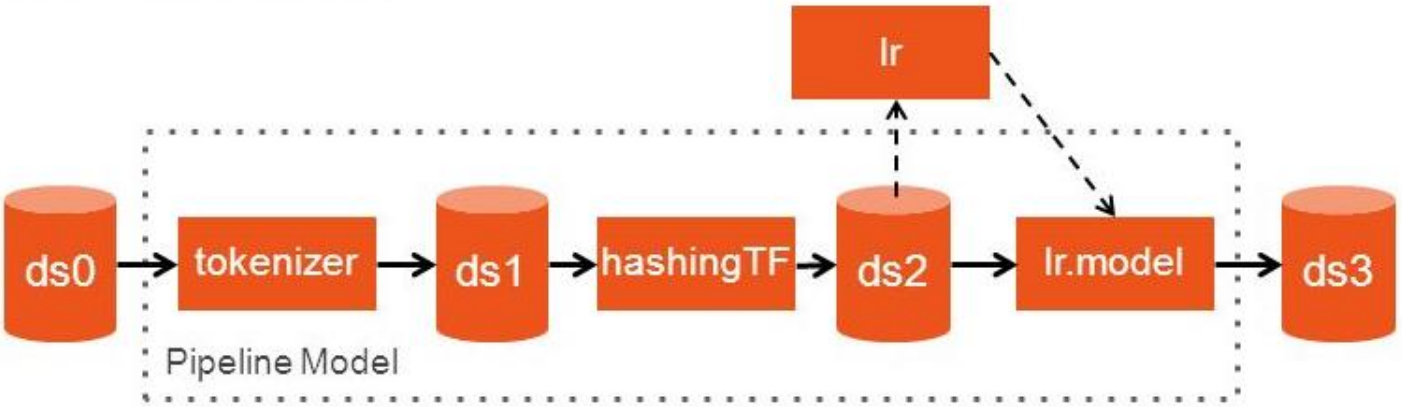
- Spark Streaming expresses streams as a series of RDDs over time
- Combine streaming with batch and interactive queries
- High Level APIs (Stateful, joins, aggregates, window etc.)
- Fault Tolerant (exactly once semantics achievable)
- **Spark 2.0**
  : Structured Streaming (ALPHA release) (on top of Spark SQL and the Catalyst optimizer)



Spark Components - Spark MLLib

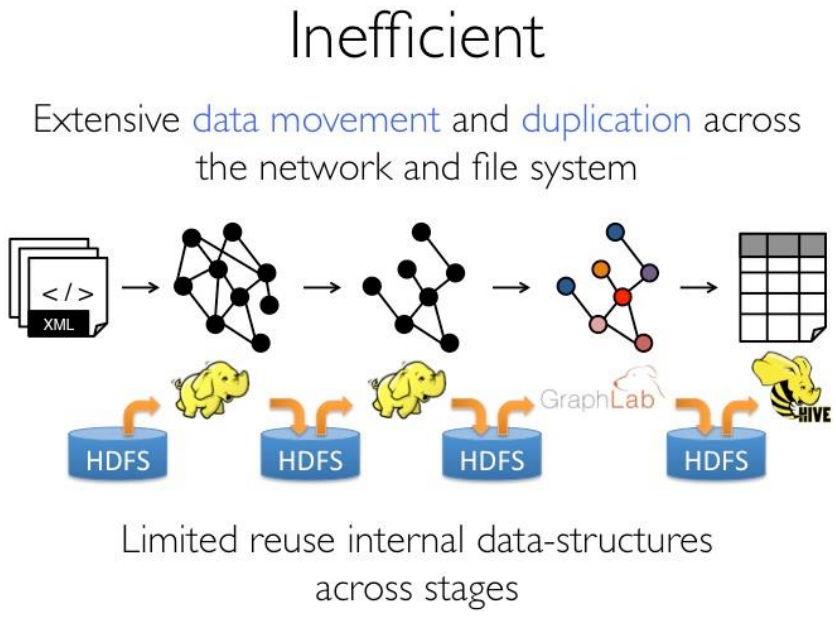


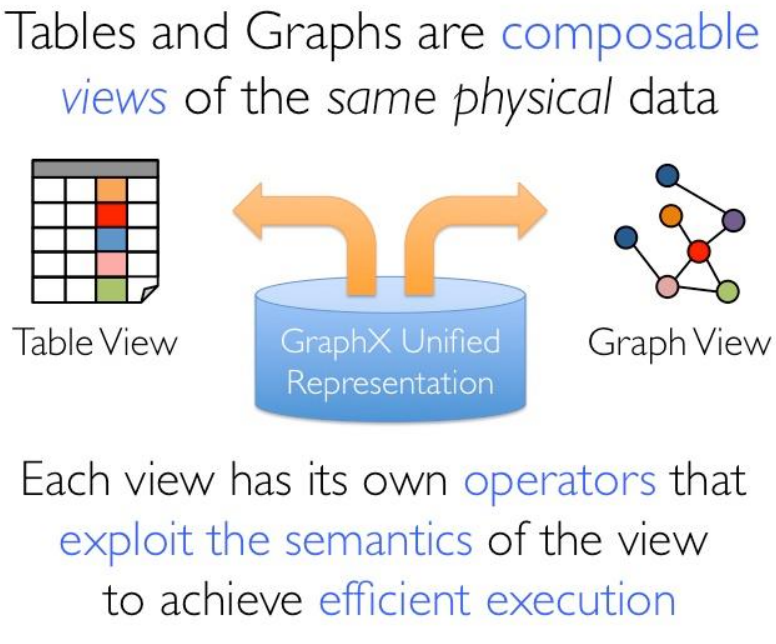
- Scalable machine learning library
- Fast iterative computation
- Interoperates with NumPy in Python and R libraries
- MLlib contains many algorithms and utilities
  : Classification, Regression, Decision trees, Recommendation
  : Clustering, Topic modeling
  : Model evaluation and hyper-parameter tuning
  : **ML Pipeline** construction
- **Spark 2.0**
  : ML persistence => saving and loading models and Pipelines
  : Added more algorithms for DataFrames-based, SparkR, PySpark



Spark Components - GraphX

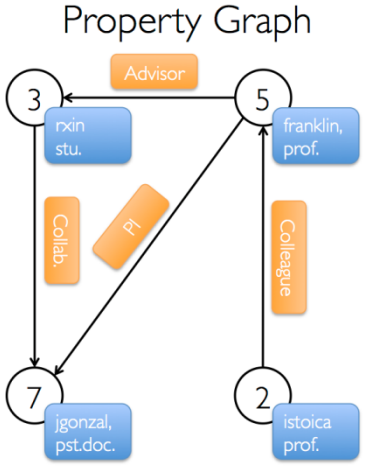
- Unifies graphs with RDDs of edges and vertices
  : Seamlessly work with both **graphs** and **collections(RDD)**
  : Custom iterative graph algorithms via Pregel API





Spark Components - GraphX

- Unifies graphs with RDDs of edges and vertices
  : Seamlessly work with both **graphs** and **collections(RDD)**
  : Custom iterative graph algorithms via Pregel API
- GraphX Algorithms (org.apache.spark.graphx.lib)
  : PageRank, Connected Components, Label Propagation
  : SVD++, Strongly Connected Components
  : Triangle Count, Shortest Paths

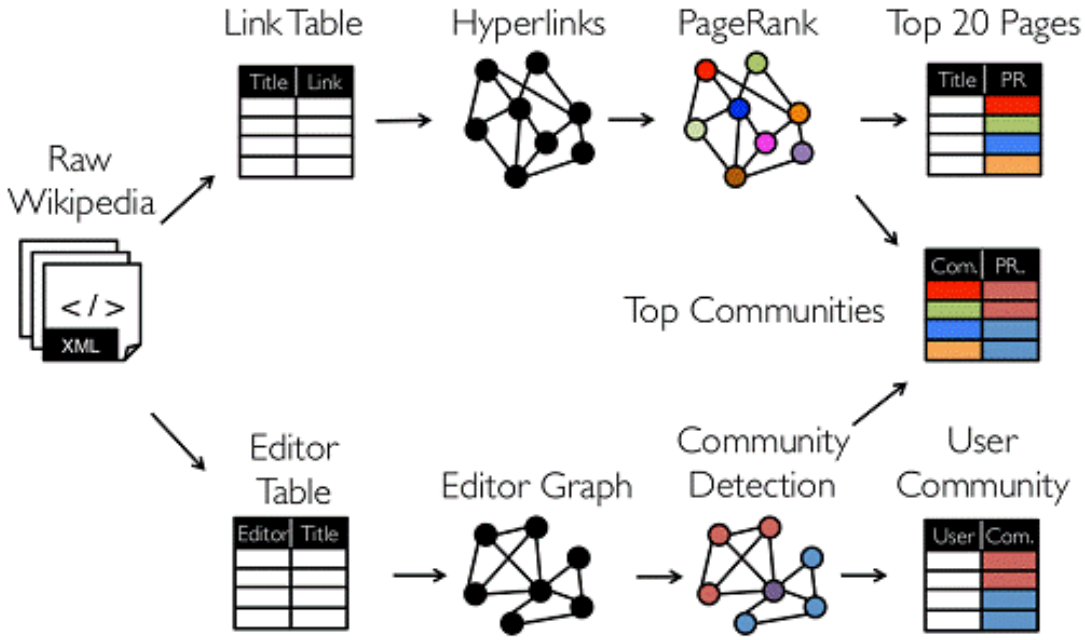


Vertex Table

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

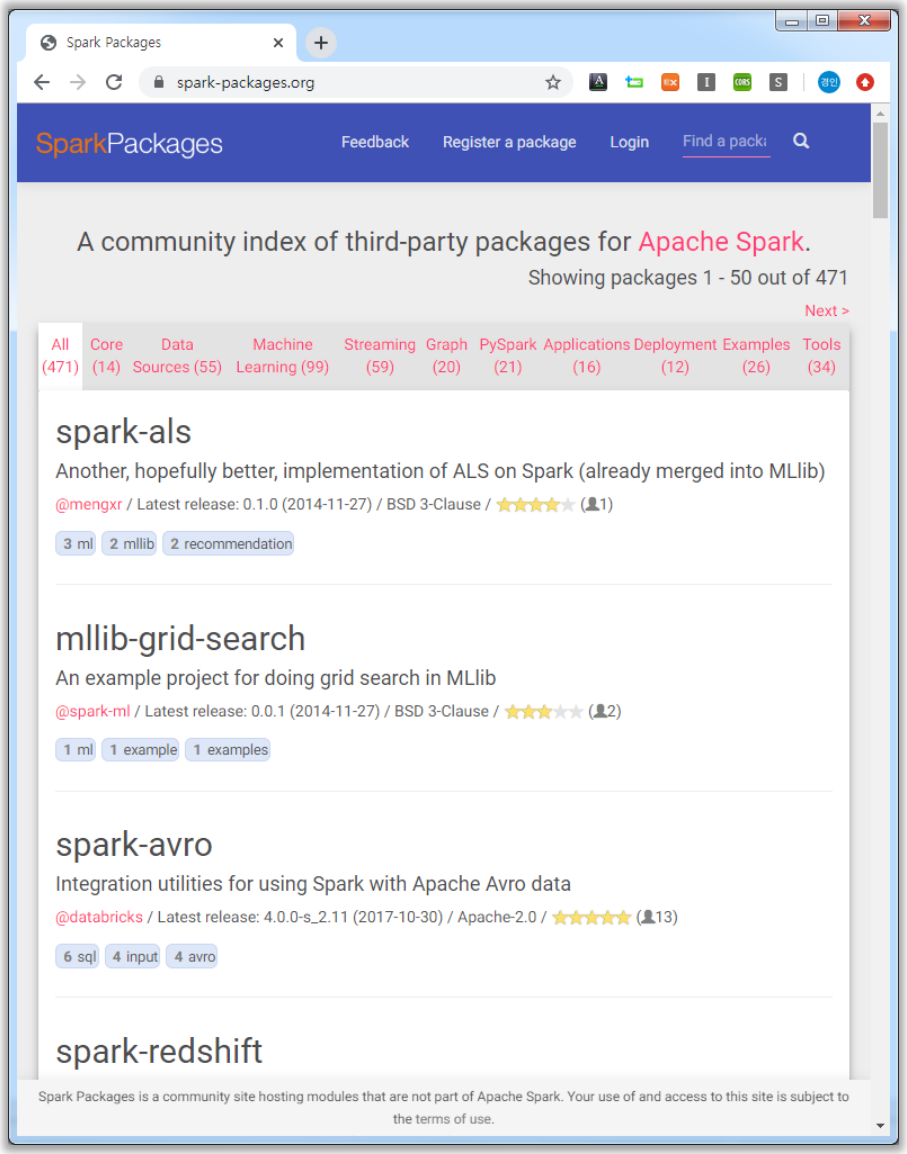
Edge Table

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



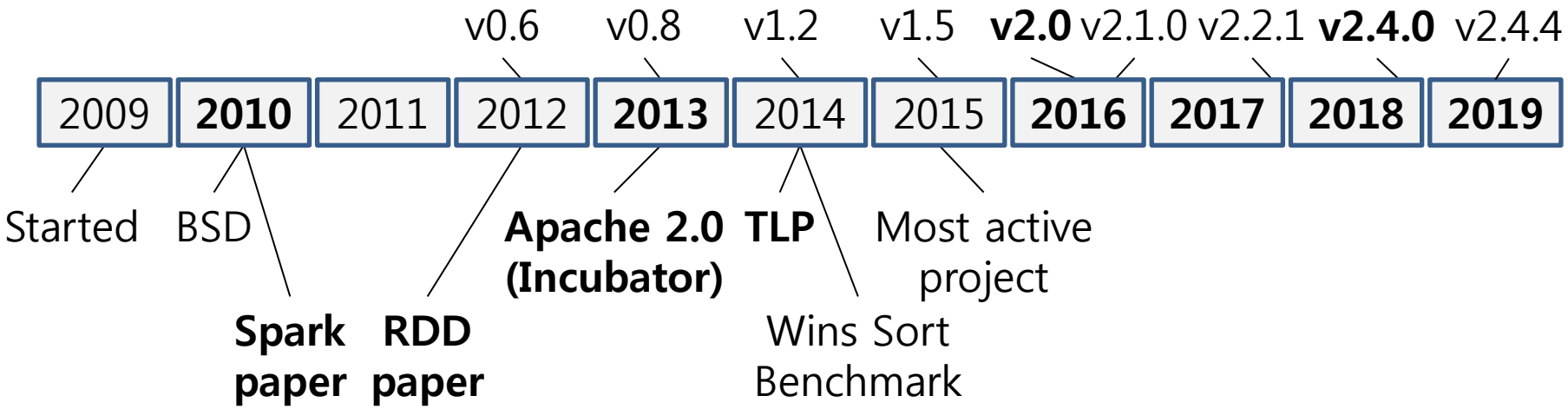
Spark Components 그 이외는?

- https://spark-packages.org
- Community index of third party packages and libraries
- **> $SPARK_HOME/bin/spark-shell --packages com.databricks:spark-csv_2.11:1.5.0**
- **spark-als**
  : already merged into MLlib
- **spark-csv**
  : now included in Spark 2.0
- **tensorframes**
  : TensorFlow on DataFrames



Spark History

- In **2009,** Spark started by **Matei Zaharia** at UC Berkeley's **AMPLab**
- In **2010**, Open sourced under a **BSD license**.
- In **2013**, switched to **Apache 2.0 license**.
- In February **2014**, Spark became a **Top-Level Apache Project**.
- In November **2014**, **Databricks** set **a new world record in large scale sorting** using Spark. (Daytona Gray Sort 100TB Benchmark)
- In 2015, Spark had in excess of **1000 contributors,** making it one of **the most active projects** in the Apache Software Foundation and one of **the most active open source big data projects**.



Spark 시작 – 2010, 2012 Papers

Matei Zaharia

- Spark Creator
- Co-founder and CTO of Databricks
- Assistant professor at Stanford University

2010

Spark: Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

act

duce and its variants have been highly successful
lementing large-scale data-intensive applications
modity clusters. However, most of these systems
lt around an acyclic data flow model that is not

MapReduce/Dryad job, each job must reload th
from disk, incurring a significant performance pe

- **Interactive analytics**: Hadoop is often used t
ad-hoc exploratory queries on large datasets, th
SQL interfaces such as Pig [21] and Hive [1]. Ic

http://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf

2012

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma,
Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

ct

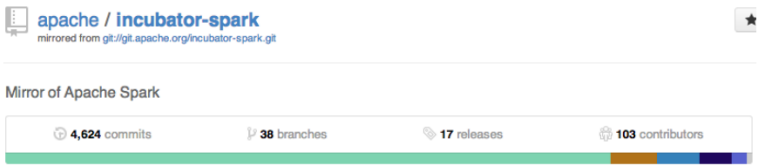
tion, which can dominate application execution ti

http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf

Spark 성장기 – 2013 Spark Summit ~

Development Community

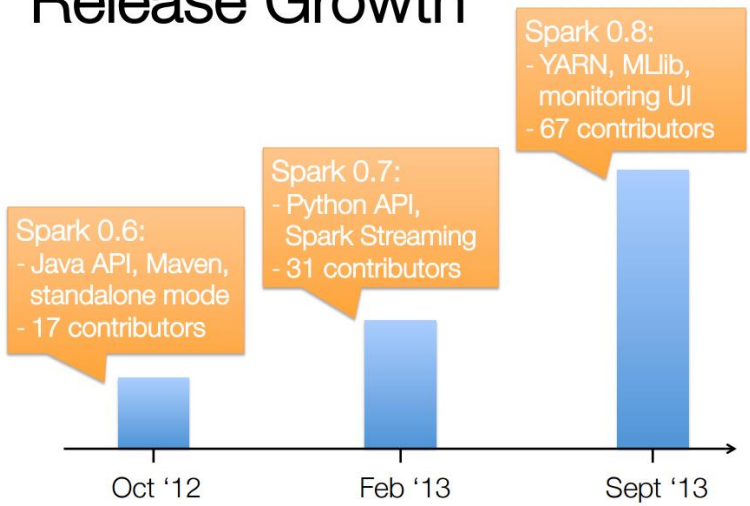
With over 100 developers and 25 companies, one of the most active communities in big data



Comparison: Storm (48), Giraph (52), Drill (18), Tez (12)

Past 6 months: more active devs than Hadoop MapReduce!

Release Growth



Some Community Contributions

YARN support (Yahoo!)

Columnar compression in Shark (Yahoo!)

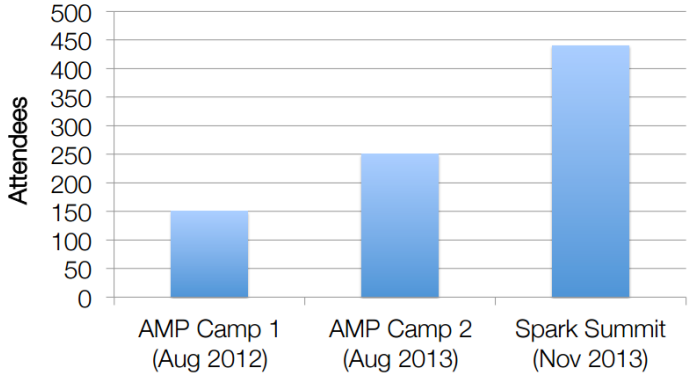
Fair scheduling (Intel)

Metrics reporting (Intel, Quantifind)

New RDD operators (Bizo, ClearStory)

Scala 2.10 support (Imaginea)

Conferences



Spark 현재 – 2019 Version's History

Version	Original release date	Latest version	Release date
0.5	2012-06-12	0.5.1	2012-10-07
0.6	2012-10-14	0.6.2	2013-02-07[36]
0.7	2013-02-27	0.7.3	2013-07-16
0.8	2013-09-25	0.8.1	2013-12-19
0.9	2014-02-02	0.9.2	2014-07-23
1.0	2014-05-26	1.0.2	2014-08-05
1.1	2014-09-11	1.1.1	2014-11-26
1.2	2014-12-18	1.2.2	2015-04-17
1.3	2015-03-13	1.3.1	2015-04-17
1.4	2015-06-11	1.4.1	2015-07-15
1.5	2015-09-09	1.5.2	2015-11-09
1.6	2016-01-04	1.6.3	2016-11-07
2.0	2016-07-26	2.0.2	2016-11-14
2.1	2016-12-28	2.1.3	2018-06-26
2.2	2017-07-11	2.2.3	2019-01-11
2.3	2018-02-28	2.3.3	2019-02-15
2.4	2018-11-02	2.4.4	2019-09-01[37]
Legend: Old version	Older version, still supported	**Latest version**	Latest preview version

Spark 현재 – 2019 Version's History

Apache Spark Documentation

Setup instructions, programming guides, and other documentation are available

- Spark 2.4.4
- Spark 2.4.3
- Spark 2.4.2
- Spark 2.4.1
- Spark 2.4.0
- Spark 2.3.4
- Spark 2.3.3
- Spark 2.3.2
- Spark 2.3.1
- Spark 2.3.0
- Spark 2.2.3
- Spark 2.2.2
- Spark 2.2.1
- Spark 2.2.0
- Spark 2.1.3
- Spark 2.1.2
- Spark 2.1.1
- Spark 2.1.0
- Spark 2.0.2
- Spark 2.0.1
- Spark 2.0.0
- Spark 1.6.3
- Spark 1.6.2
- Spark 1.6.1
- Spark 1.6.0
- Spark 1.5.2
- Spark 1.5.1
- Spark 1.5.0
- Spark 1.4.1
- Spark 1.4.0
- Spark 1.3.1
- Spark 1.3.0
- Spark 1.2.1
- Spark 1.1.1
- Spark 1.0.2
- Spark 0.9.2
- Spark 0.8.1
- Spark 0.7.3
- Spark 0.6.2

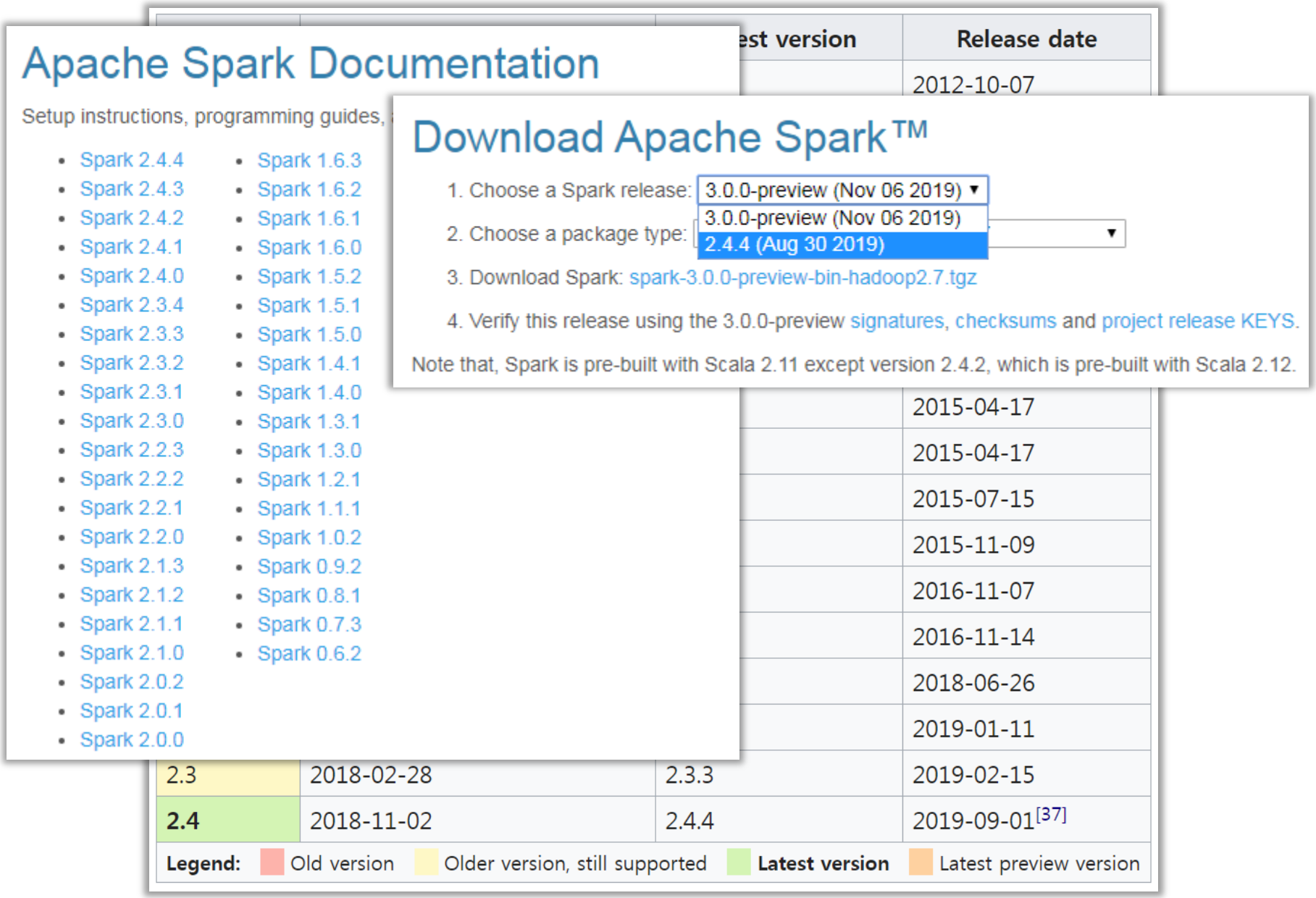
Documentation for preview releases:

- Spark 2.0.0 preview
- Spark 3.0.0 preview

		est version	Release date
			2012-10-07
			2013-02-07[36]
			2013-07-16
			2013-12-19
			2014-07-23
			2014-08-05
			2014-11-26
			2015-04-17
			2015-04-17
			2015-07-15
			2015-11-09
			2016-11-07
			2016-11-14
			2018-06-26
			2019-01-11
2.3	2018-02-28	2.3.3	2019-02-15
2.4	2018-11-02	2.4.4	2019-09-01[37]

Legend: Old version | Older version, still supported | **Latest version** | Latest preview version

Spark 현재 – 2019 Version's History



Apache Spark Documentation

Setup instructions, programming guides,

- Spark 2.4.4
- Spark 2.4.3
- Spark 2.4.2
- Spark 2.4.1
- Spark 2.4.0
- Spark 2.3.4
- Spark 2.3.3
- Spark 2.3.2
- Spark 2.3.1
- Spark 2.3.0
- Spark 2.2.3
- Spark 2.2.2
- Spark 2.2.1
- Spark 2.2.0
- Spark 2.1.3
- Spark 2.1.2
- Spark 2.1.1
- Spark 2.1.0
- Spark 2.0.2
- Spark 2.0.1
- Spark 2.0.0
- Spark 1.6.3
- Spark 1.6.2
- Spark 1.6.1
- Spark 1.6.0
- Spark 1.5.2
- Spark 1.5.1
- Spark 1.5.0
- Spark 1.4.1
- Spark 1.4.0
- Spark 1.3.1
- Spark 1.3.0
- Spark 1.2.1
- Spark 1.1.1
- Spark 1.0.2
- Spark 0.9.2
- Spark 0.8.1
- Spark 0.7.3
- Spark 0.6.2

Download Apache Spark™

1. Choose a Spark release: 3.0.0-preview (Nov 06 2019)
2. Choose a package type:
3. Download Spark: spark-3.0.0-preview-bin-hadoop2.7.tgz
4. Verify this release using the 3.0.0-preview signatures, checksums and project release KEYS.

Note that, Spark is pre-built with Scala 2.11 except version 2.4.2, which is pre-built with Scala 2.12.

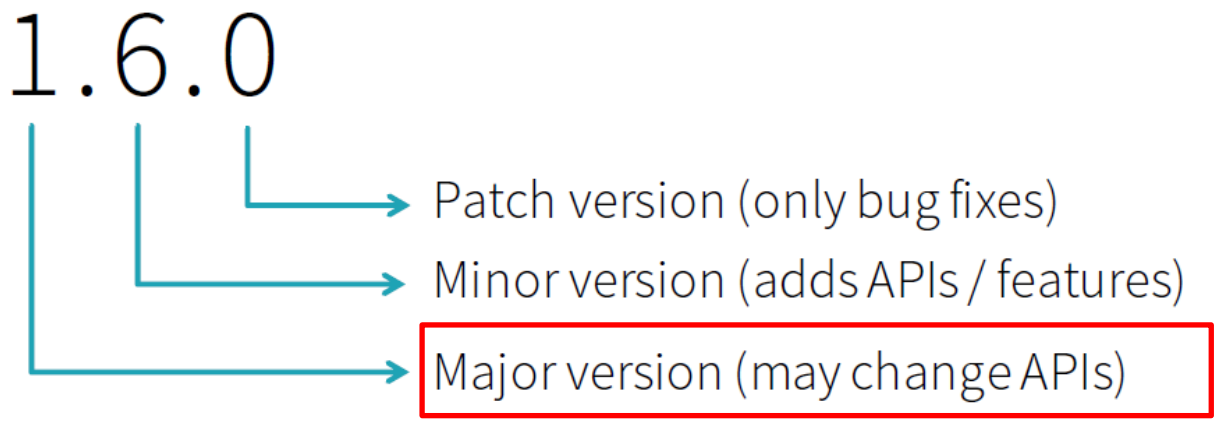
			2012-10-07
			2015-04-17
			2015-04-17
			2015-07-15
			2015-11-09
			2016-11-07
			2016-11-14
			2018-06-26
			2019-01-11
2.3	2018-02-28	2.3.3	2019-02-15
2.4	2018-11-02	2.4.4	2019-09-01[37]

Legend: Old version | Older version, still supported | **Latest version** | Latest preview version

Spark 현재 – Spark 2.4.4

- **Spark 1.6.2에서 Spark 2.0.0으로 점프! (Jul 26, 2016)**
- Spark 2.0.2 (Nov 14, 2016), Spark 1.6.3 (Nov 07, 2016)
- Spark 2.1.3 (Jun 26, 2018)
- Spark 2.2.3 (Jan 11, 2019)
- Spark 2.3.3 (Feb 15, 2019)
- Spark 2.4.0 (Nov 02, 2018), **현재 Spark 2.4.4 (Sep 01, 2019)**
- **Major version의 변화 (1.x.x => 2.x.x)**
- What happen?

Versioning in Spark



Spark 2.0 – Major Features

Faster

Tungsten Phase 2
speedups of 5-20x

Smarter

Structured Streaming

Easier

SQL 2003
& Unifying Datasets
and DataFrames

- **Faster**
  : 물리적 실행 영역을 다시 설계(Whole-stage Code Generation)
  : CPU 낭비시간 해소(가상함수 호출 시간, CPU Cache/RAM IO)
- **Smarter**
  : Streaming 엔진 이상의 역할(Continuous application)
  : DataFrame based Streaming(vs. 기존 RDD based Streaming)
- **Easier**
  : 표준 SQL 지원(서브쿼리도 지원하는 새로운 Ansi-SQL 파서 적용)
  : DataFrame/Dataset API 통합 (Java/Scala)
  : SparkSession (SQLContext, HiveContext 대체)

Notable Spark Users - Spark Summit 2015

Notable Spark Users - Spark Summit 2016

Large-Scale Usage

Largest cluster:
8000 Nodes (Tencent)

Largest single job:
1 PB (Alibaba, Databricks)

Top Streaming Intake:
1 TB/hour (HHMI Janelia Farm)

2014 On-Disk Sort Record
Fastest Open Source Engine for sorting a PB

Spark 적용 사례

Source: How Spark is Making an Impact at Goldman Sachs

- Started with Hadoop, RDBMS, Hive, HBASE, PIG, Java MR, etc.
- Challenges: Java, Debugging PIG, Code-Compile-Deploy-Debug
- Solution: Spark
  - Language Support: Scala, Java, Python, R
  - In-Memory: Faster than other solutions
  - SQL, Stream Processing, ML, Graph
  - Both Batch and stream processing

Spark 적용 사례

- Scale:
  - > 1000 nodes (20,000 cores, 100TB RAM)
- Daily Jobs: 2000-3000
- Supports: Ads, Search, Map, Commerce, etc.
- Cool project: Enabling Interactive Queries with Spark and Tachyon
  - >50X acceleration of Big Data Analytics workloads

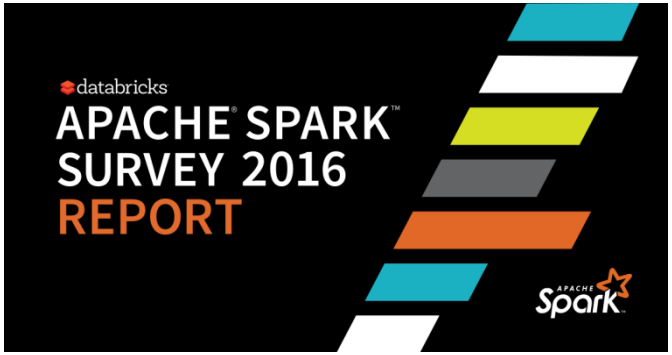
Spark 적용 사례

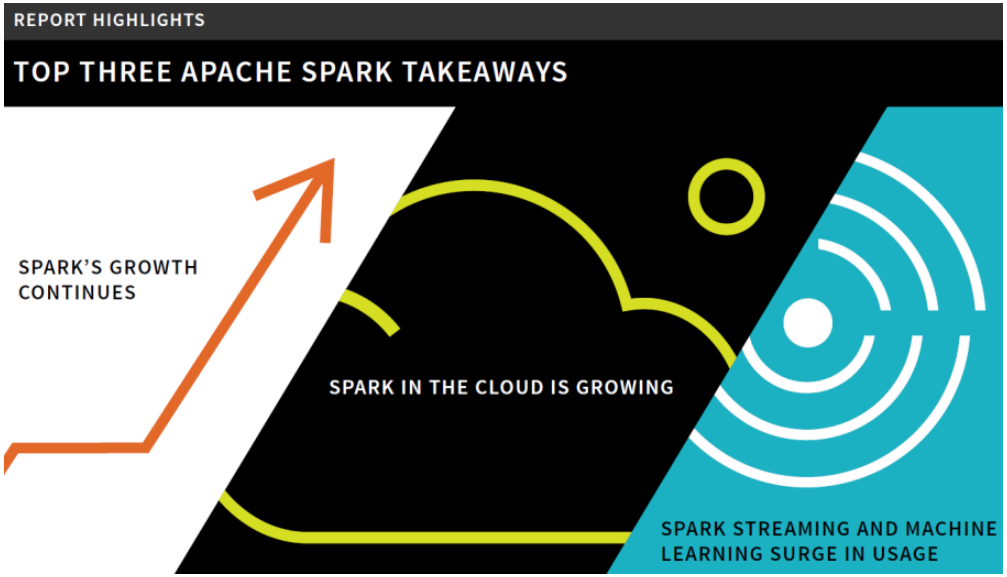
Source: Toyota Customer 360 Insights on Apache Spark and MLlib

- Performance
  - Original batch job: 160 hours
  - Same Job re-written using Apache Spark: 4 hours
- Categorize
  - Prioritize incoming social media in real-time using Spark MLlib (differentiate campaign, feedback, product feedback, and noise)
  - ML life cycle: Extract features and train:
    - V1: 56% Accuracy -> V9: 82% Accuracy
    - Remove False Positives and Semantic Analysis (distance similarity between concepts)

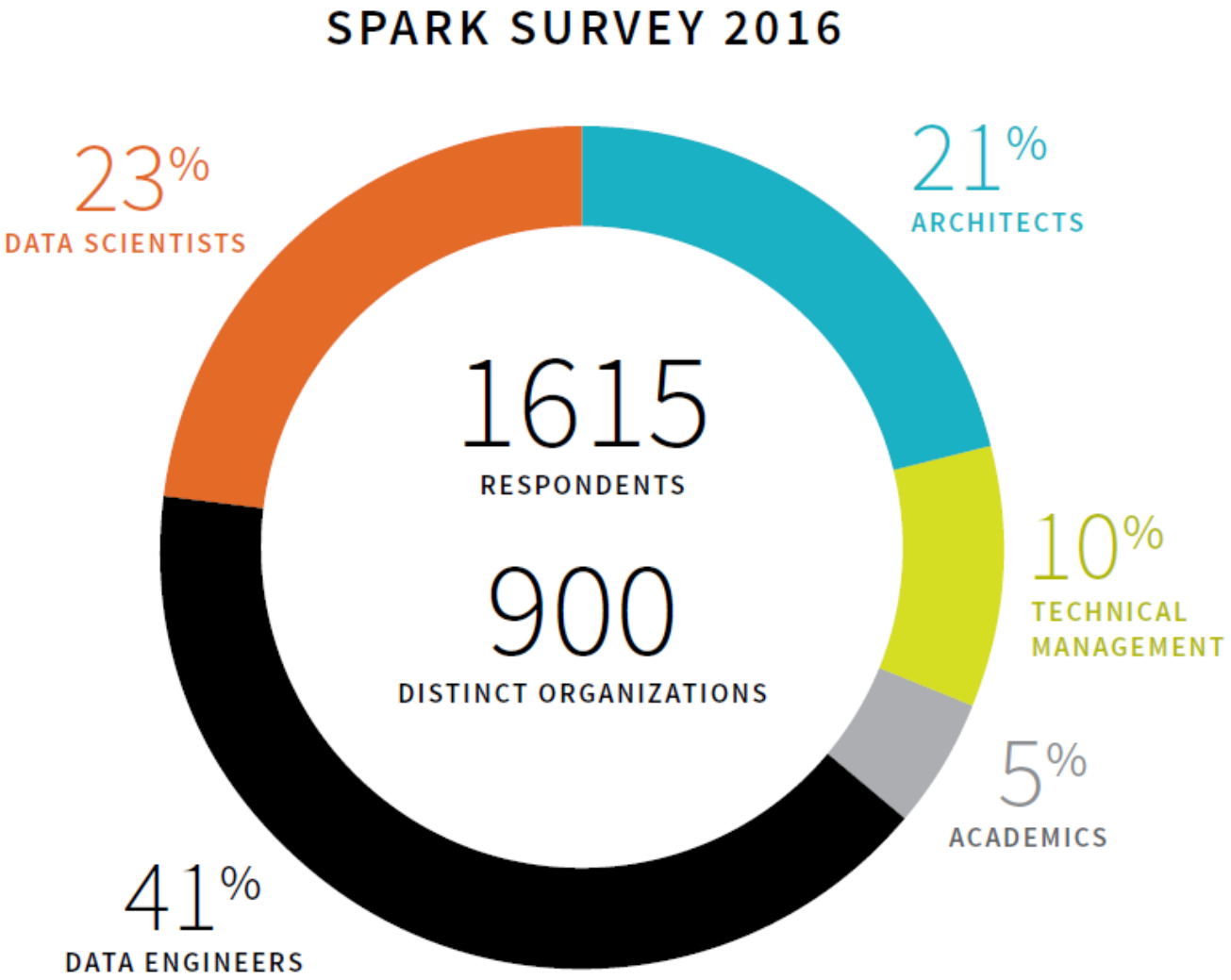
Spark Survey Report 2016 by Databricks

- In July 2016, Apache Spark Survey to identify insights on **how organizations are using Spark** and highlight growth trends since our last Spark Survey 2015.
- The 2016 survey results reflect answers from **900 distinct organizations** and **1615 respondents**, who were predominantly Apache Spark users.
- Apache Spark growth is pervasive as users embrace **public cloud**, **machine learning**, and **Spark Streaming**





Spark Survey Report 2016 by Databricks



Spark Survey Report 2016 by Databricks

SPARK MEETUP MEMBERS

+240%

2015	2016
66,000	225,000

CODE CONTRIBUTORS

+67%

2015	2016
600	1000

SPARK SUMMIT ATTENDEES

+30%

2015	2016
3912	5100

NUMBER OF COMPANIES AT SUMMITS

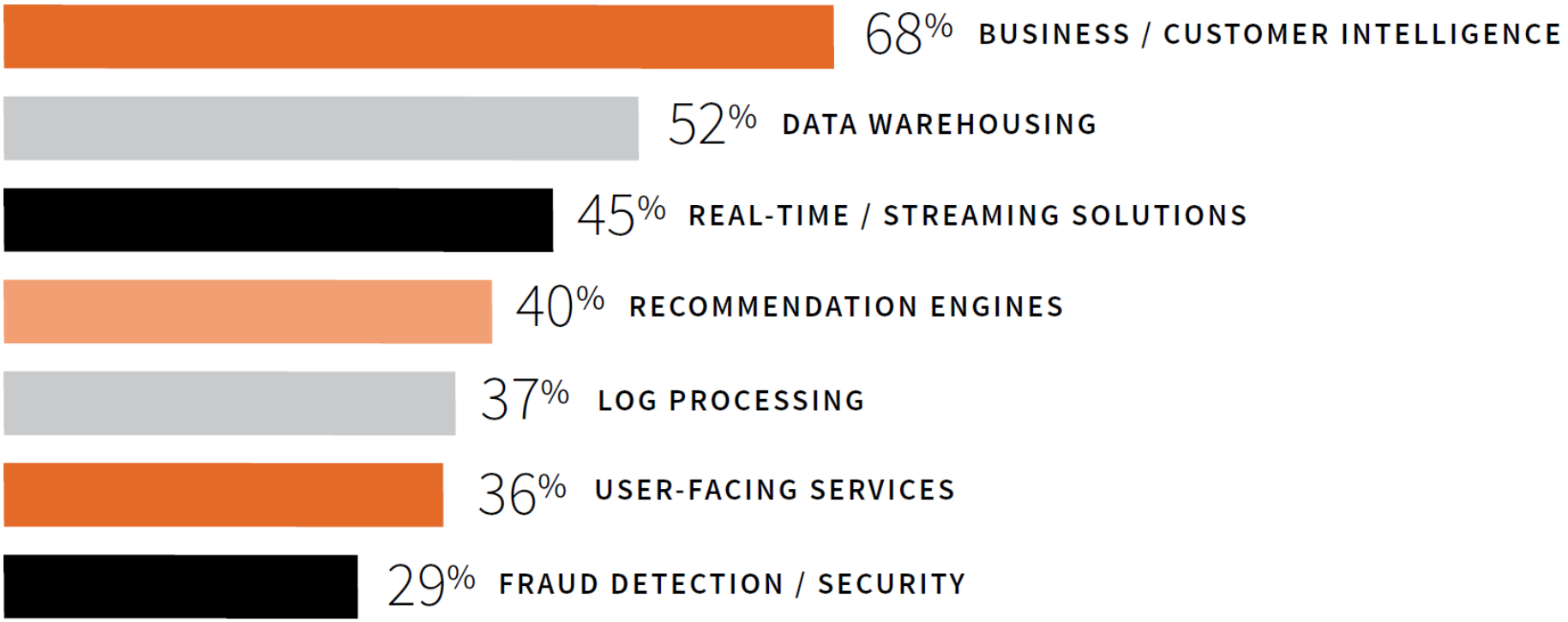
+57%

2015	2016
1144	1800

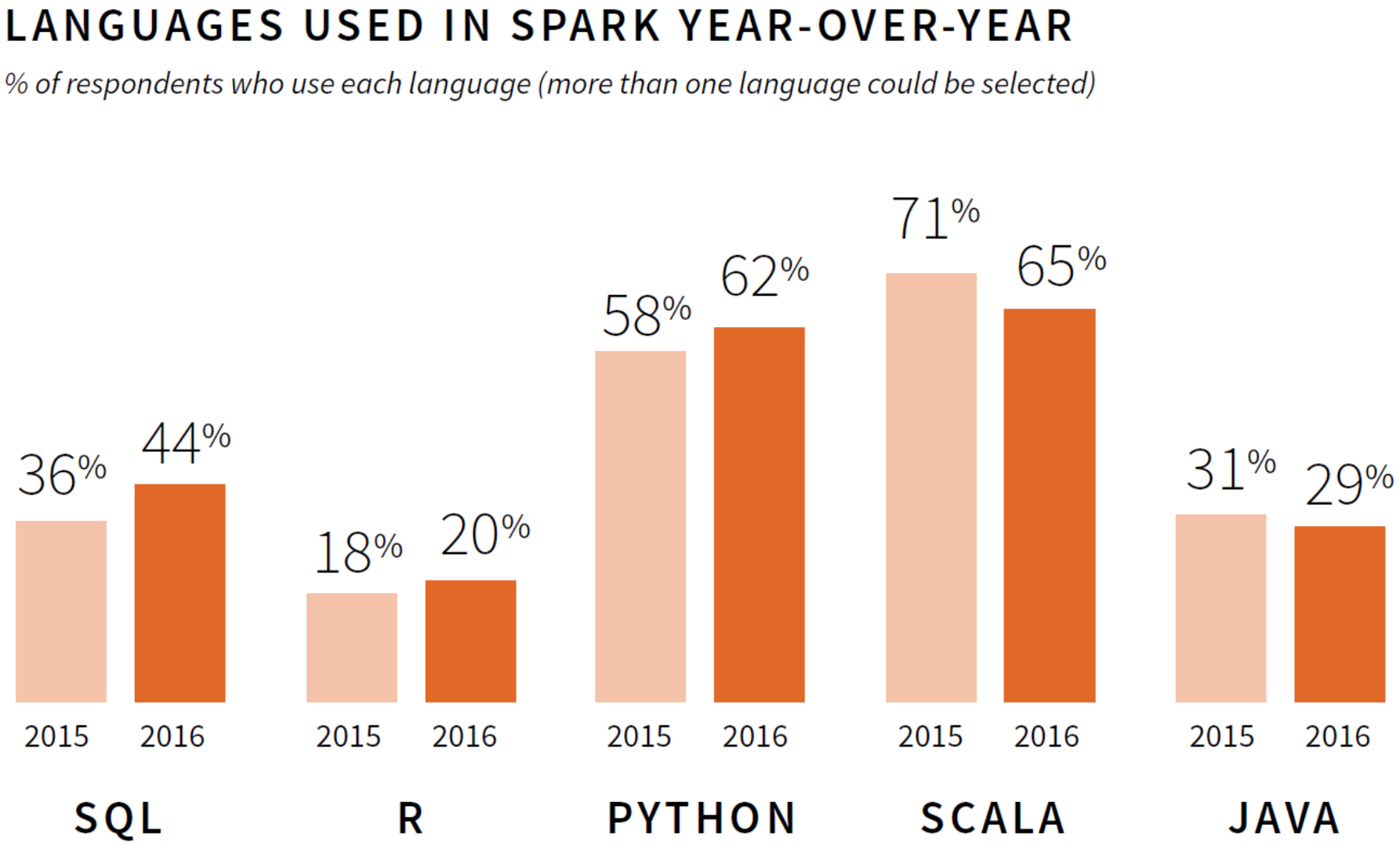
Spark Survey Report 2016 by Databricks

TYPES OF PRODUCTS BUILT

% of respondents who use Spark to create each product (more than one product could be selected)



Spark Survey Report 2016 by Databricks



Spark Survey Report 2016 by Databricks

NUMBER OF COMPONENTS USED

74%

of respondents

USE TWO OR MORE COMPONENTS

64%

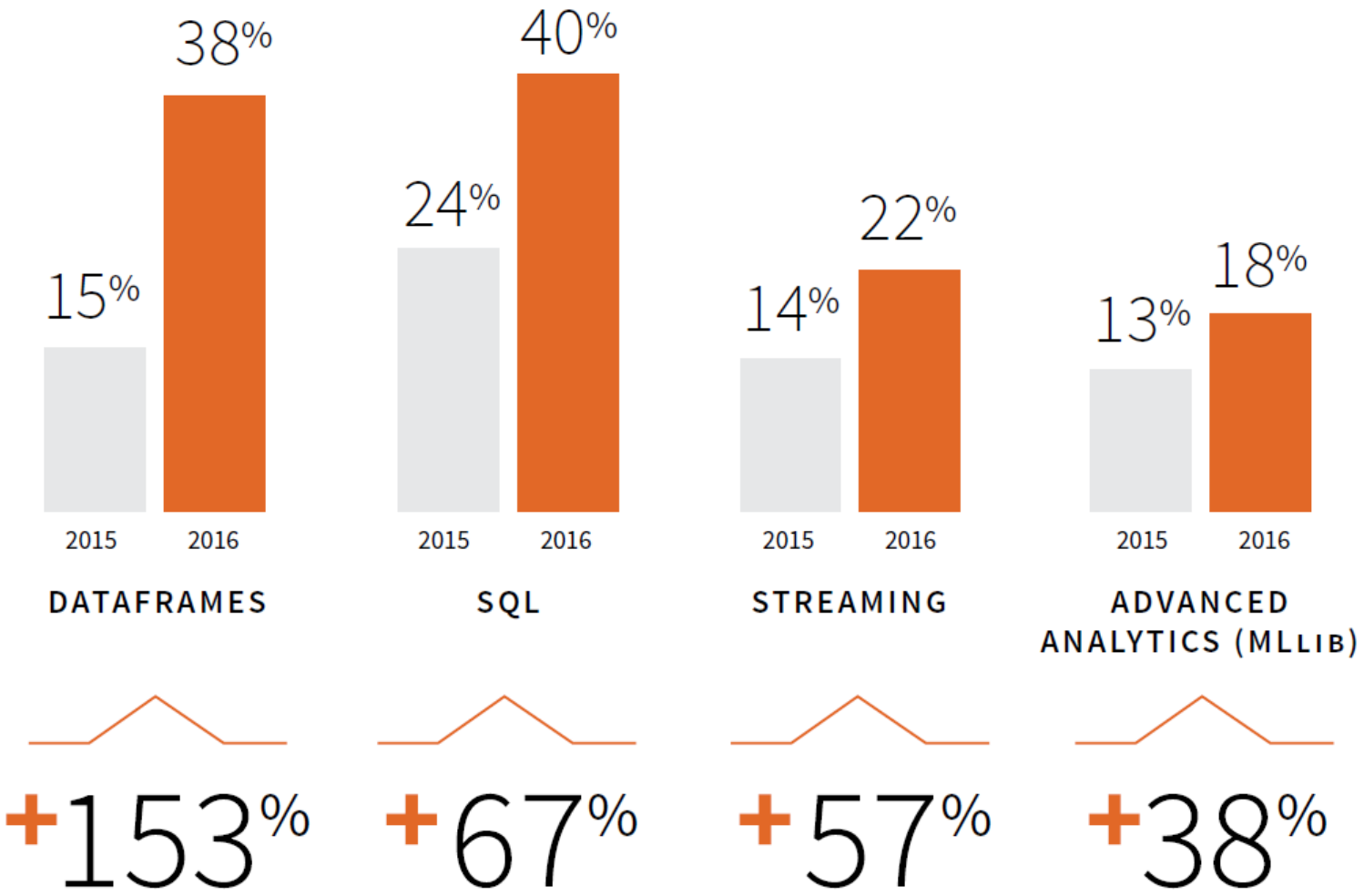
of respondents

USE THREE OR MORE COMPONENTS

Spark Survey Report 2016 by Databricks

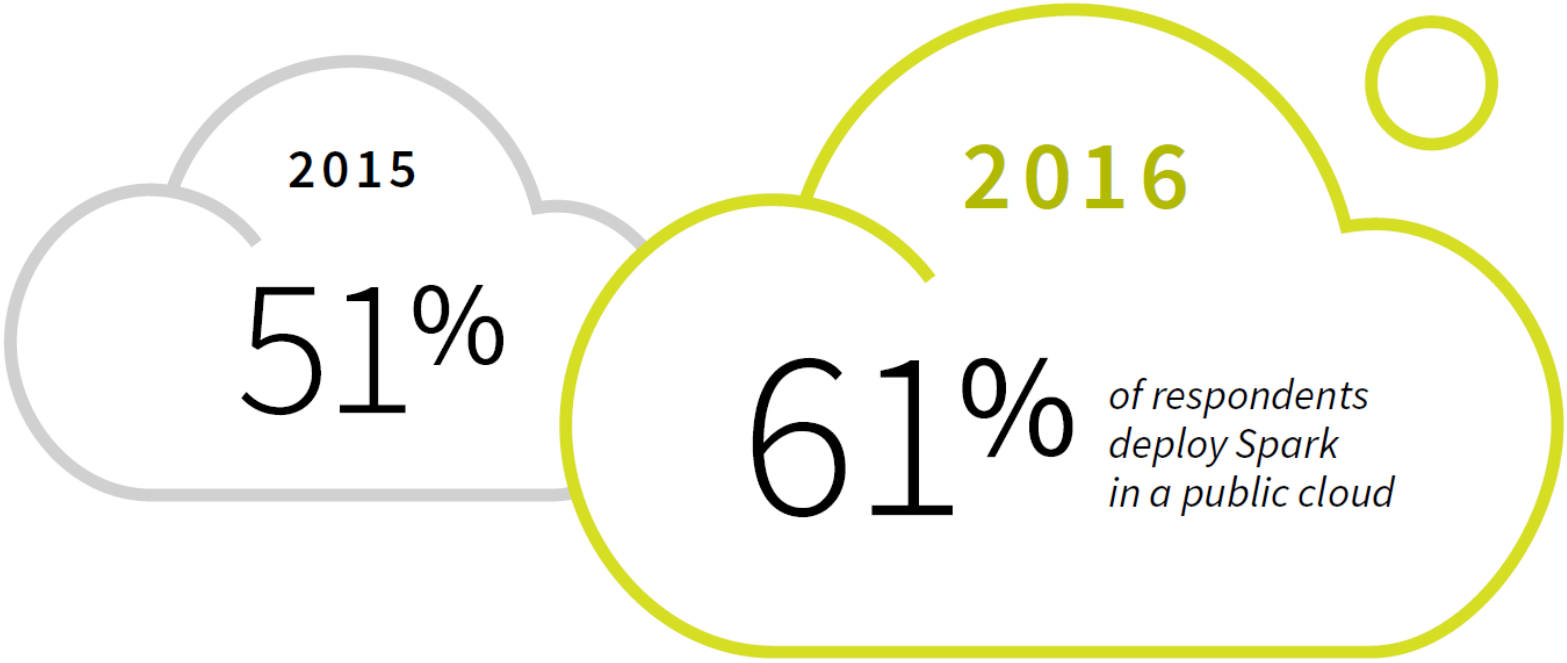
SPARK COMPONENTS USED IN PRODUCTION YEAR-OVER-YEAR

% of respondents who use each component in production (more than one component could be selected)



Spark Survey Report 2016 by Databricks

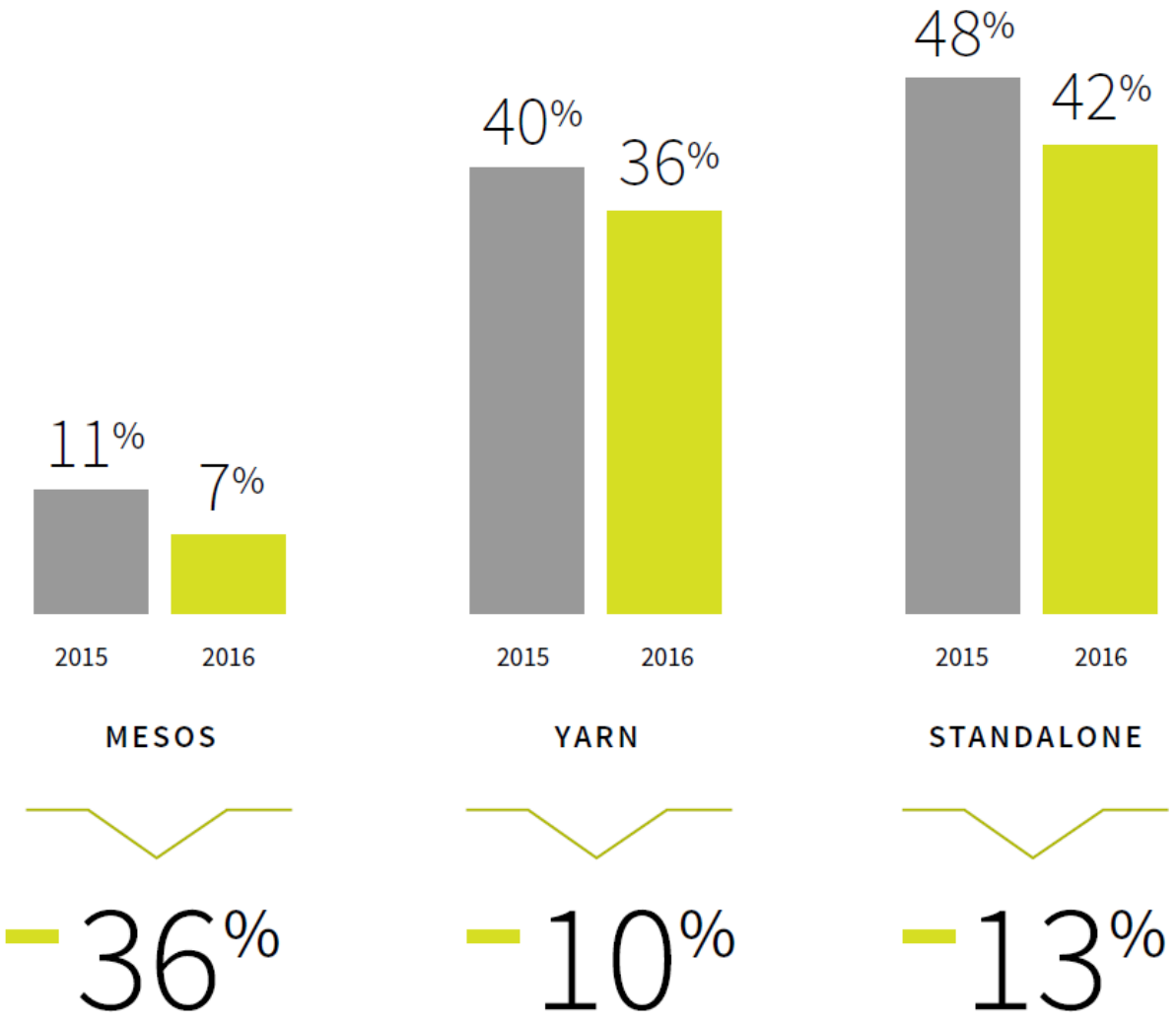
SPARK DEPLOYMENT IN PUBLIC CLOUDS HAS INCREASED BY 10% SINCE 2015.



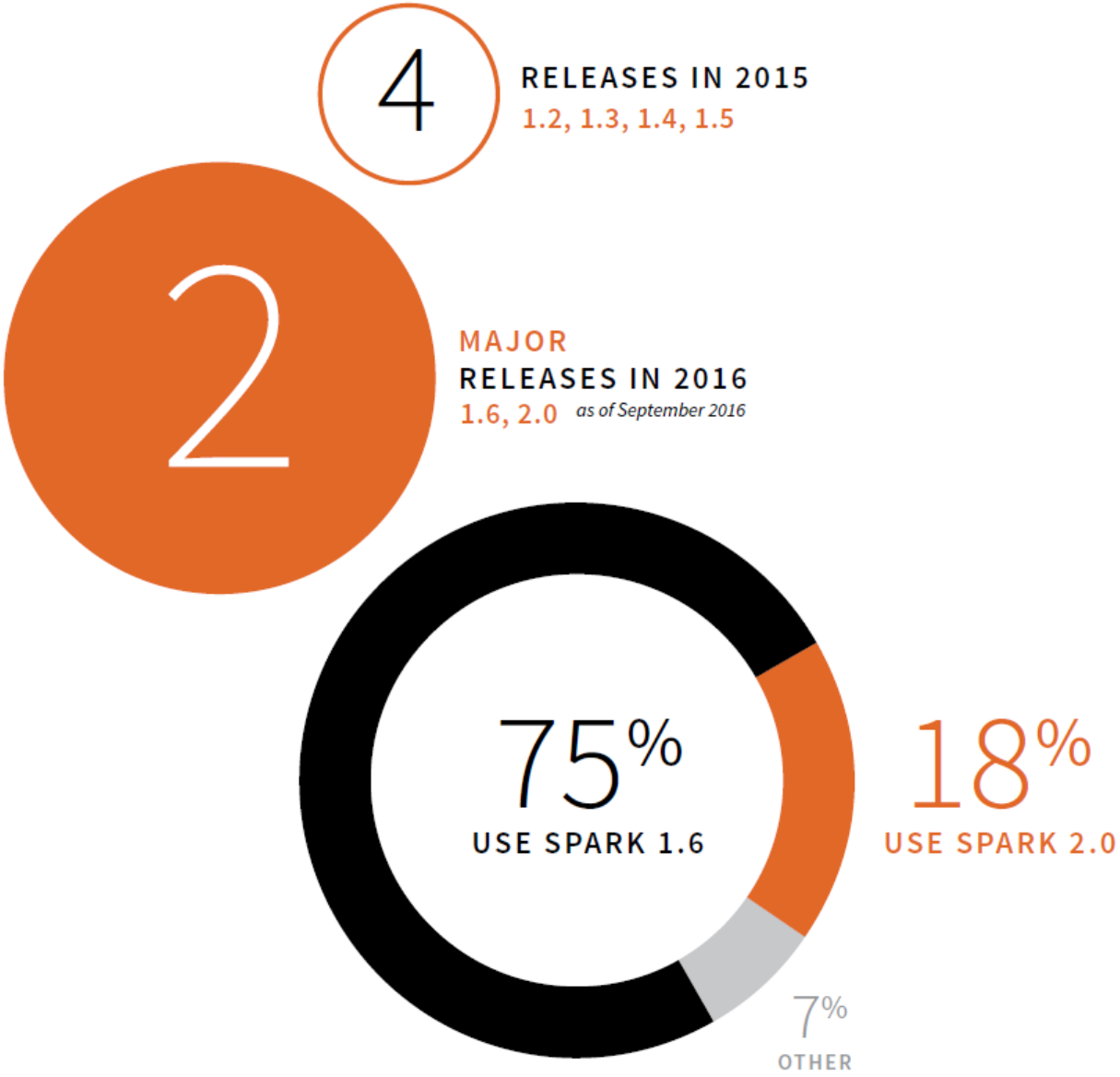
Spark Survey Report 2016 by Databricks

ON-PREMISES DEPLOYMENTS YEAR-OVER-YEAR

% of respondents who use each (more than one deployment could be selected)



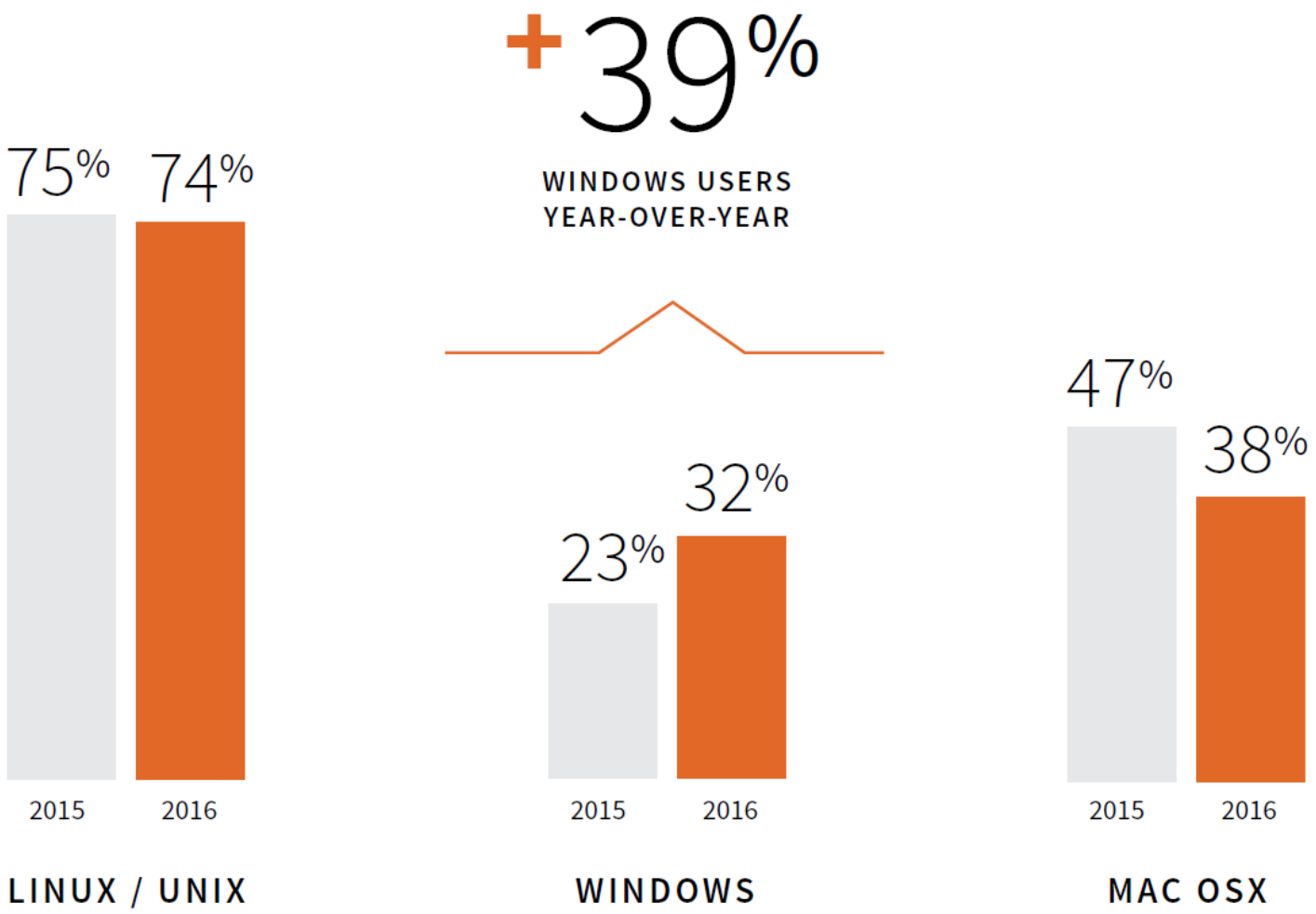
Spark Survey Report 2016 by Databricks



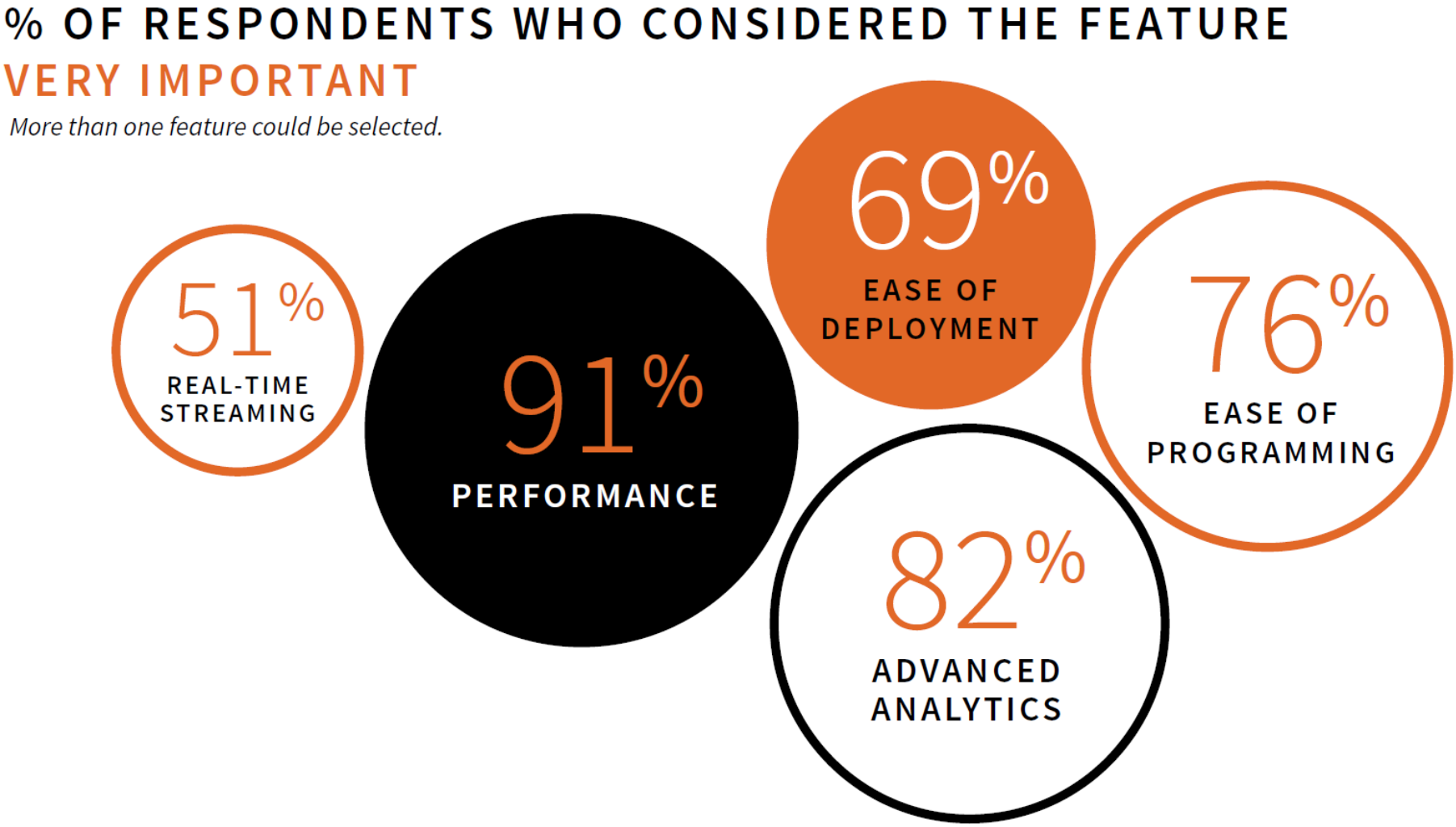
Spark Survey Report 2016 by Databricks

DEVELOPMENT ENVIRONMENTS

% of respondents who use each development environment (more than one environment could be selected)



Spark Survey Report 2016 by Databricks



Spark Survey Report 2016 by Databricks

WHICH OF THESE TECHNOLOGIES DO YOU CURRENTLY USE? *Select all that apply.*

82%

of respondents use

OPEN-SOURCE SQL DATABASES

73%

of respondents use

KEY-VALUE STORES (NoSQL)

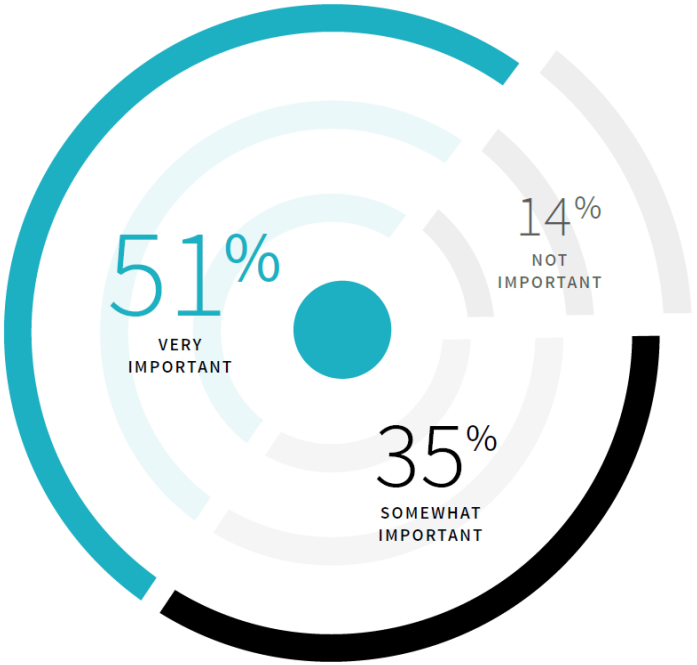
58%

of respondents use

PROPRIETARY SQL DATABASES

Spark Survey Report 2016 by Databricks

HOW IMPORTANT IS SPARK STREAMING TO YOUR USE CASE?



WHICH KINDS OF PRODUCTS DOES YOUR ORGANIZATION DEVELOP? *Select all that apply.*

29%

of respondents develop

FRAUD DETECTION / SECURITY PRODUCTS

45%

of respondents develop

REAL-TIME STREAMING PRODUCTS

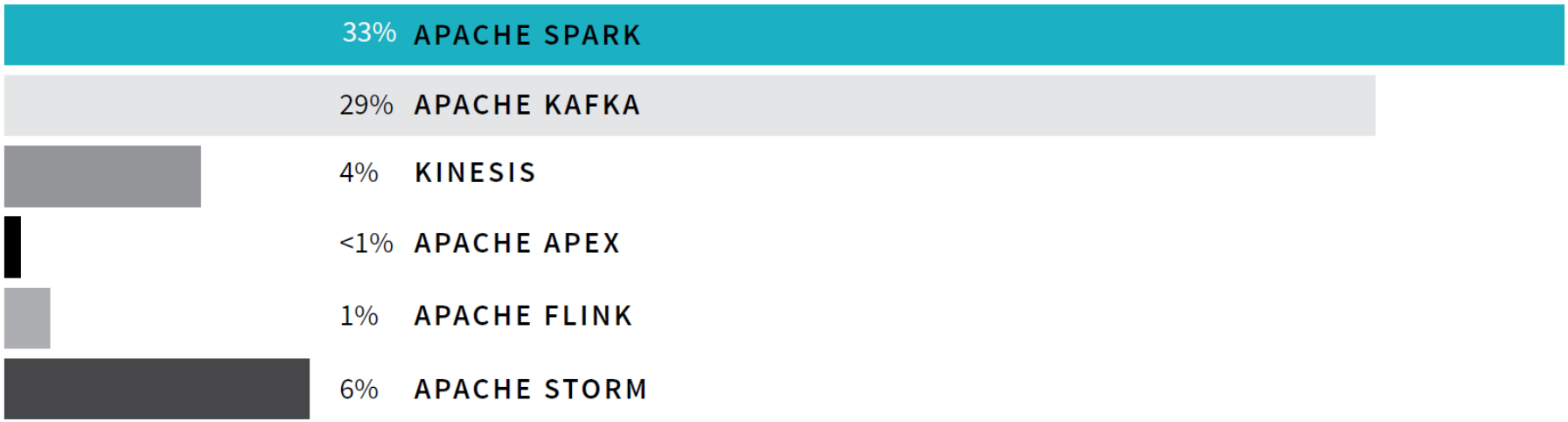
40%

of respondents develop

RECOMMENDATION ENGINE PRODUCTS

Spark Survey Report 2016 by Databricks

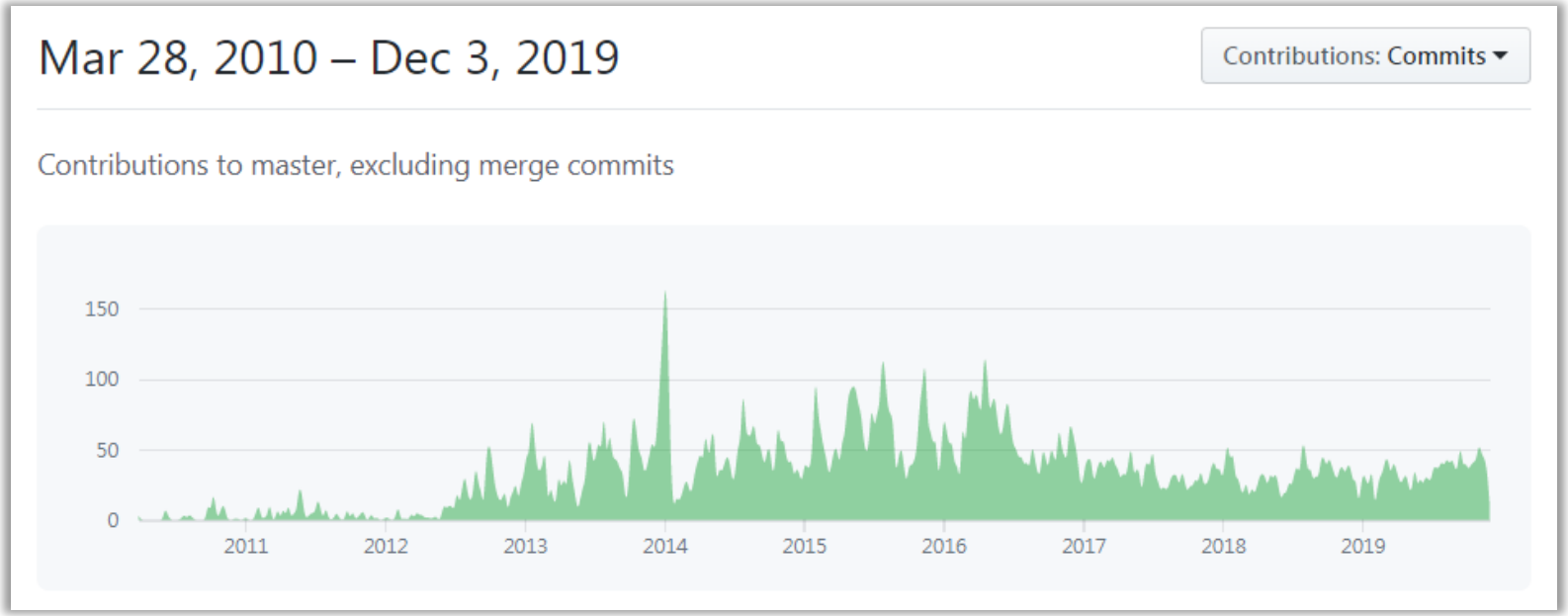
WHICH OF THESE TECHNOLOGIES DO YOU CURRENTLY USE A LOT FOR STREAMING AND/OR COMPLEX EVENT PROCESSING CASES? *Select all that apply.*



Note: Respondents were predominately Spark users.

Spark Community

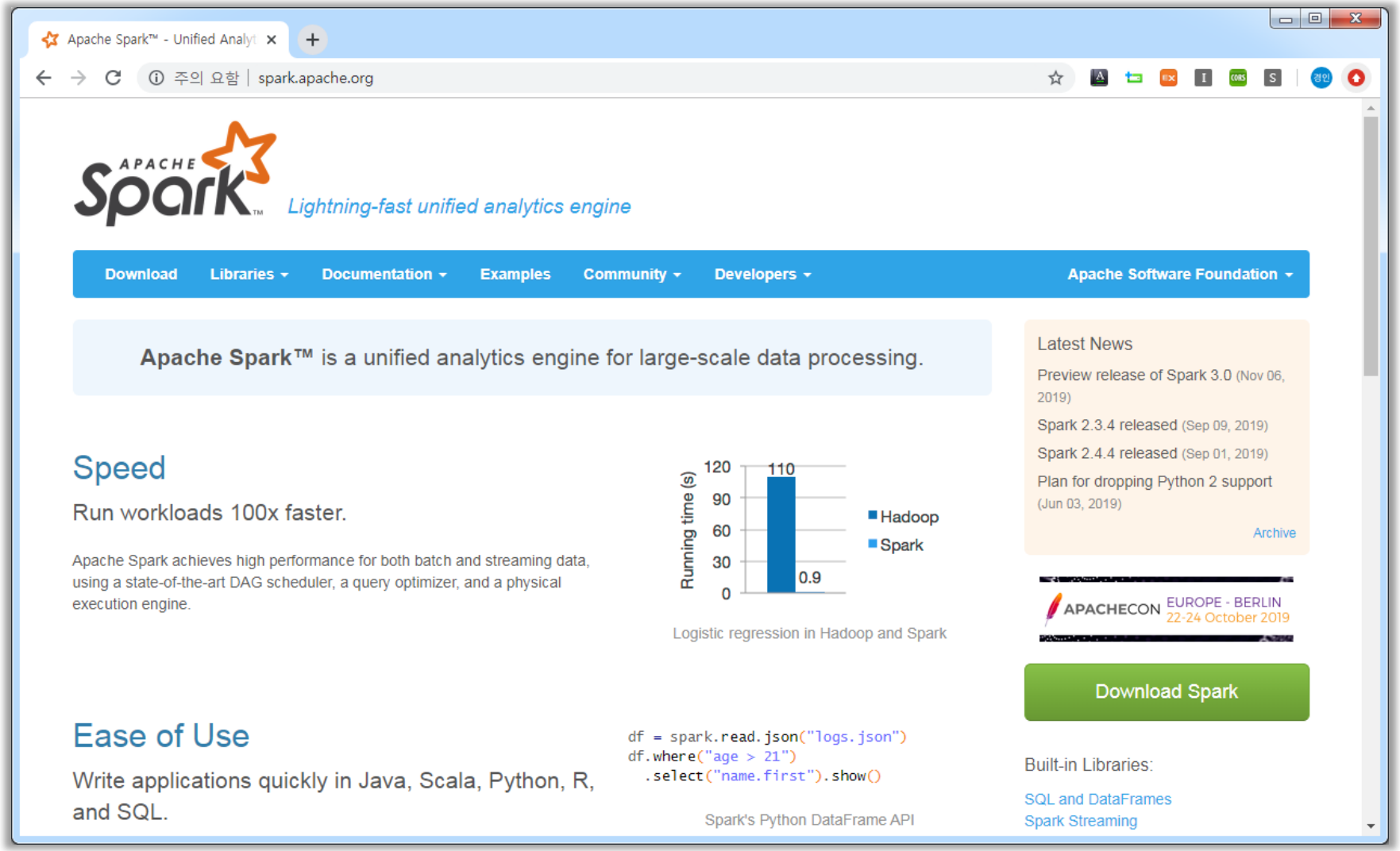
Growing community with 25+ companies contributing



amplab UC BERKELEY
databricks
YAHOO!
amazon web services
wanDISCO
intel
cloudera
阿里巴巴 Alibaba.com
Hortonworks
CLOUD PHYSICS
guavus Go Decisively
Adobe
AdMobius
ClearStory Data Now You See It
IBM
TAGGED
4B
stratio
CONVIVA
bizo
webtrends
ADATAO BIG DATA INTELLIGENCE
sharethrough
HHMI janelia farm research campus
OOYALA
quantiFind
tuplejump
MAPR TECHNOLOGIES

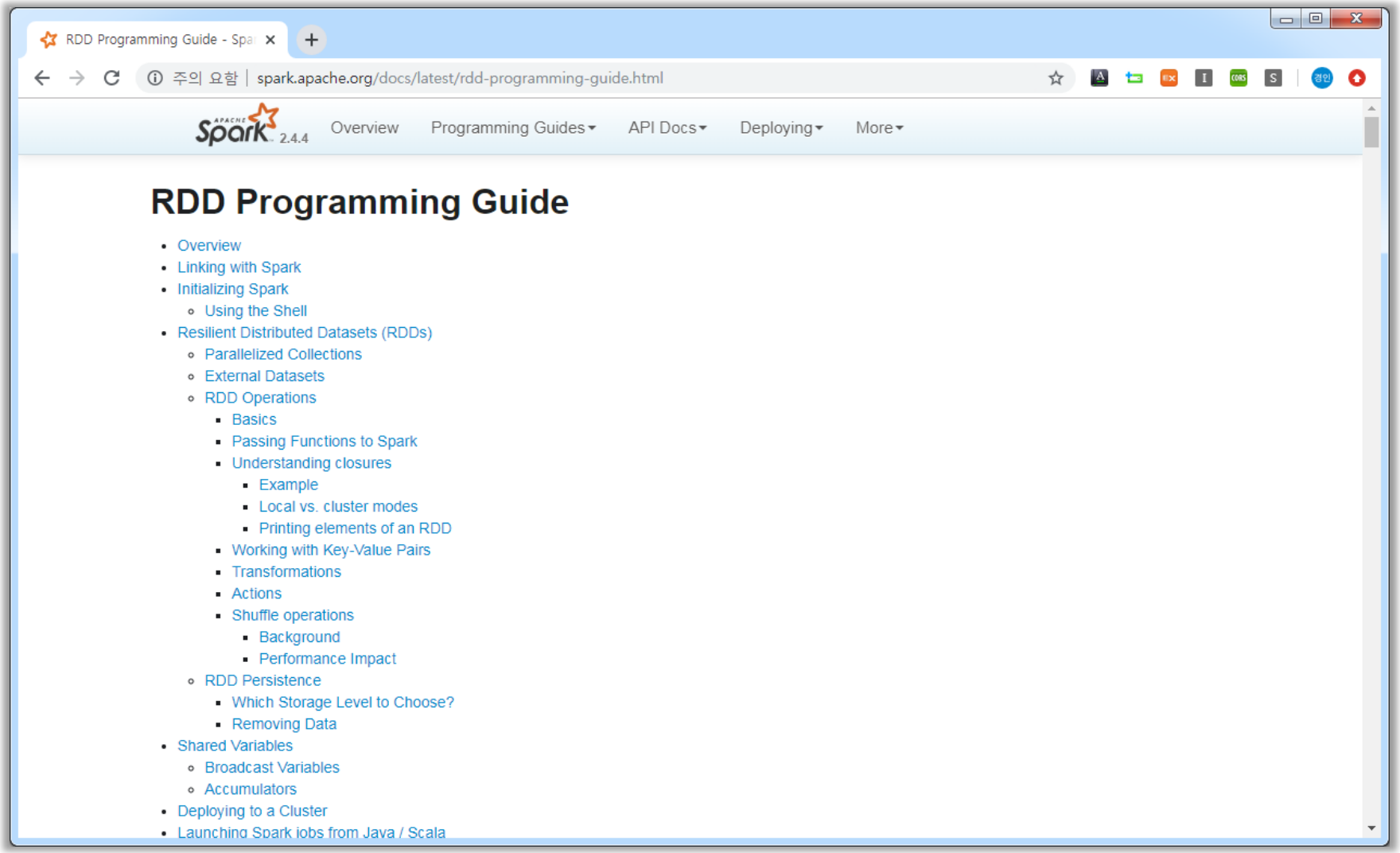
Apache Spark

1.1 Spark 개요



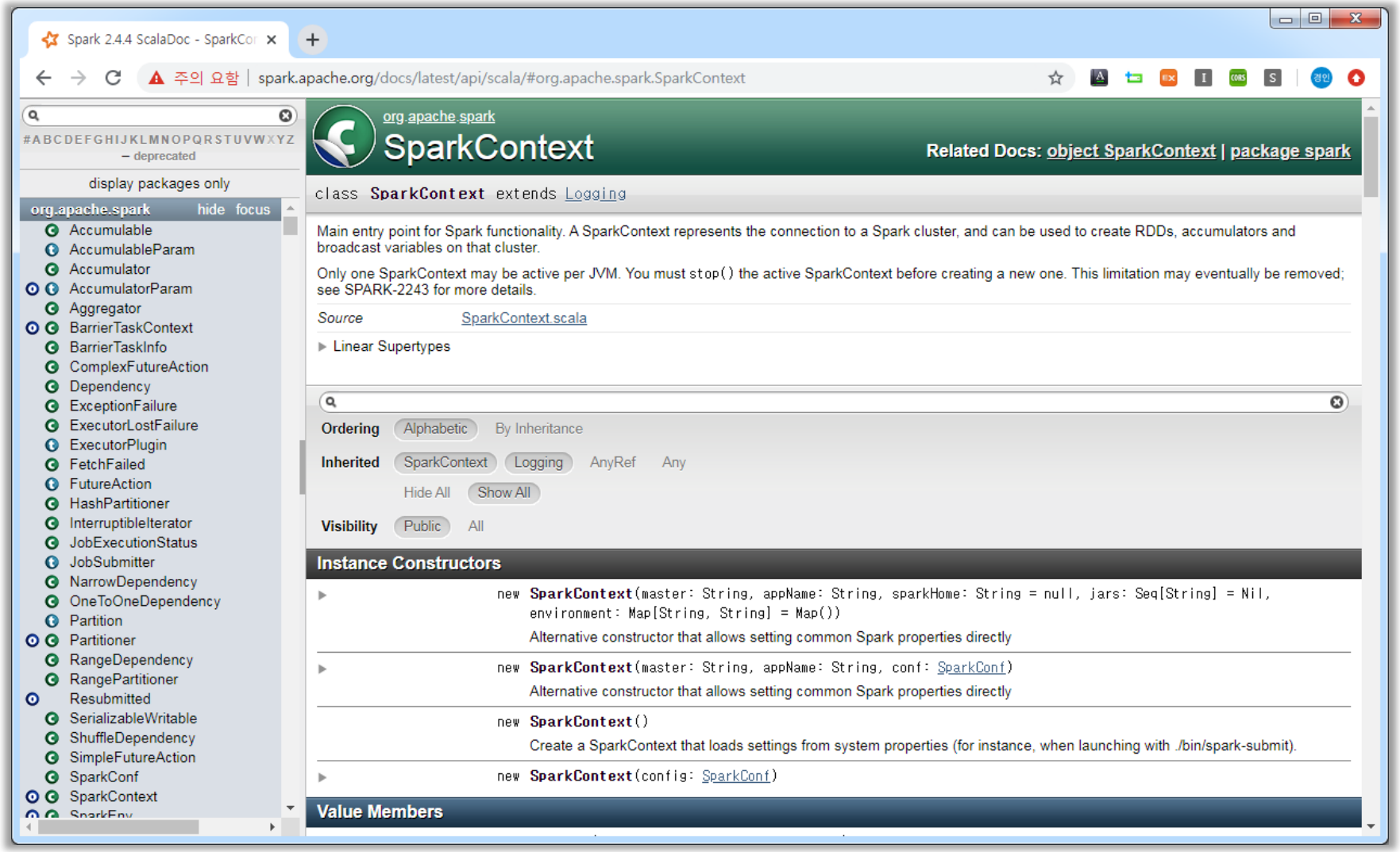
http://spark.apache.org/

Spark Programming Guides



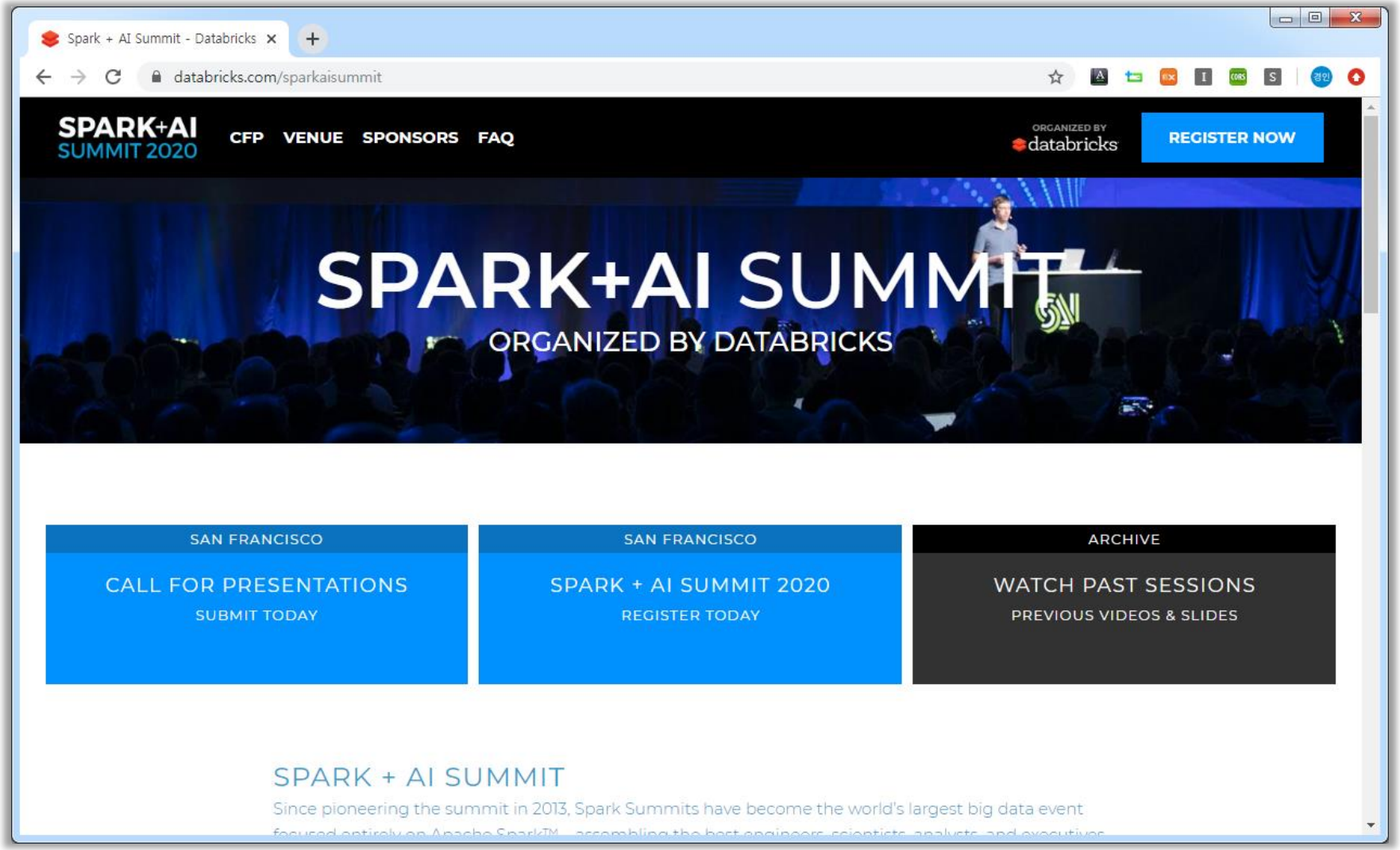
http://spark.apache.org/docs/latest/

Spark API Docs (Scala/Java/Python/R)



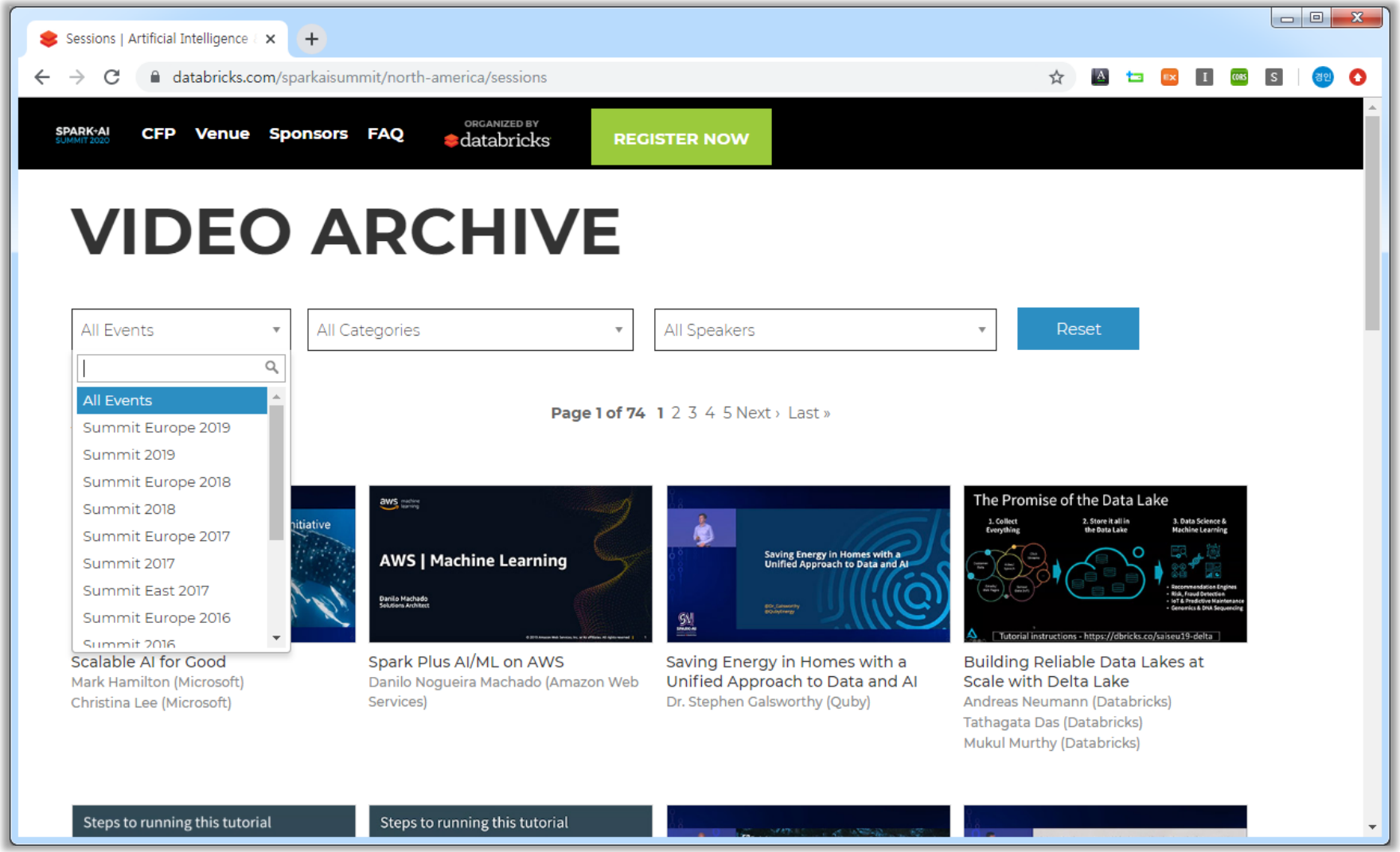
http://spark.apache.org/docs/latest/api/scala/

Spark+AI Summit



https://databricks.com/sparkaisummit

Spark+AI Summit

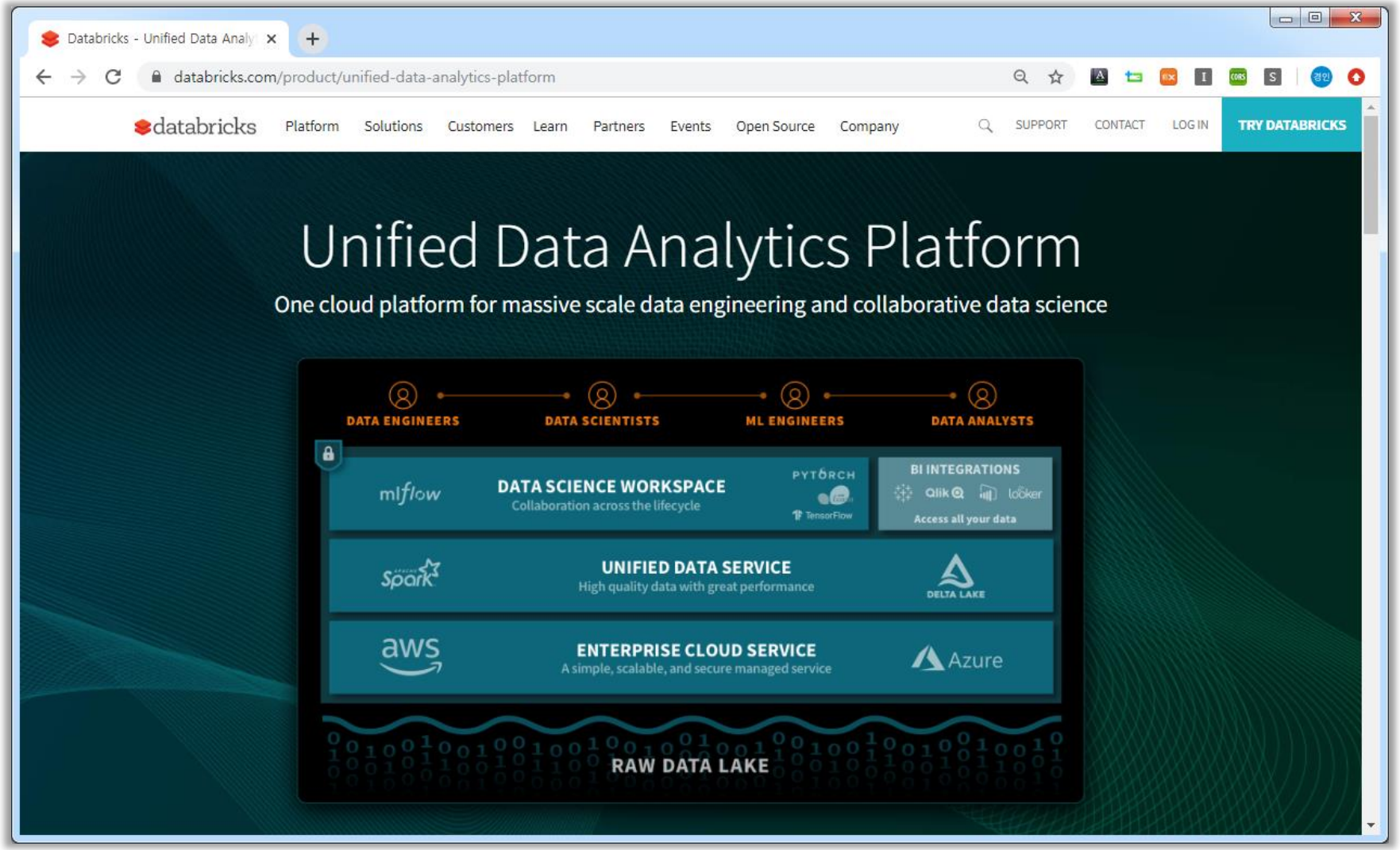


https://databricks.com/sparkaisummit

AMPLab BDAS (Berkeley Data Analytics Stack) 1.1 Spark 개요

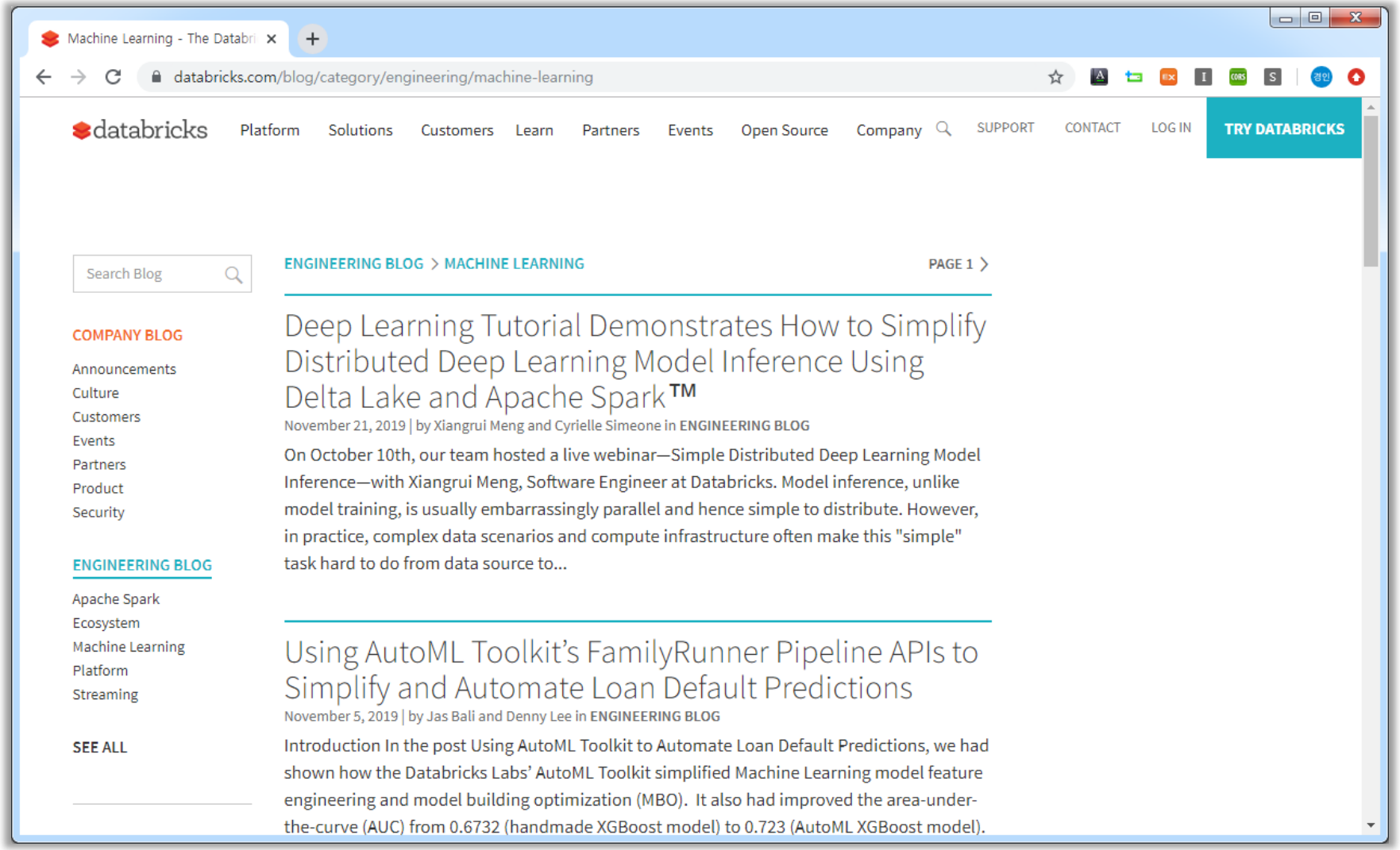
https://amplab.cs.berkeley.edu/software/

Databricks



https://databricks.com/

Databricks Blog

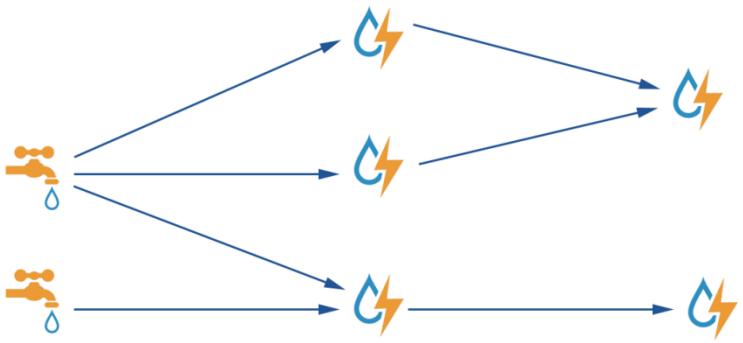


https://databricks.com/blog

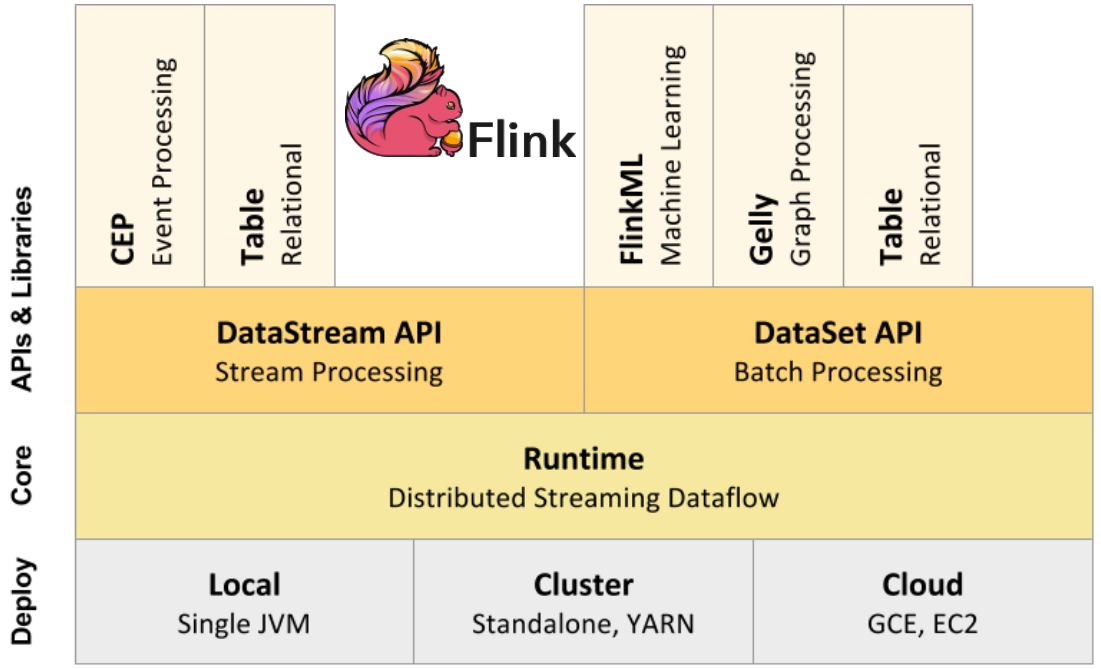
Competitors

- Apache Hadoop : 하둡과는 상생 관계
- Apache Storm : Stream Processing 분야에서 Spark Streaming과 경쟁
- Apache Flink : 신흥 기대주?

APACHE STORM™



http://storm.apache.org/



https://flink.apache.org/

1. Spark 이해하기

1.1 Spark 개요

1.2 Scala 개요

1.3 Spark 설치 및 실행

Spark with Scala vs. Java vs. Python

- **Spark** was originally **written in Scala**, which allows **concise function syntax** and **interactive use**
- **Java API added** for standalone applications
- **Python API added** more recently **along with an interactive shell**.

Why Scala?

- High-level language for the **JVM**
  : Object oriented + **functional** programming

- **Static typed**
  : Type Inference

- **Interoperates with Java**
  : Can use any Java Class
  : Can be called from Java code

Quick Tour of Scala

Declaring variables:

```
var x: Int = 7
var x = 7 // type inferred
val y = "hi" // read-only
```

Java equivalent:

```
int x = 7;

final String y = "hi";
```

Functions:

```
def square(x: Int): Int = x*x
def square(x: Int): Int = {
  x*x
}
def announce(text: String) =
{
  println(text)
}
```

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

Laziness

- **For variables** we can define **lazy val**, that are evaluated when called

  scala> lazy val x = 10 * 10 * 10 * 10 // long computation
  x: Int = <lazy>

- **For methods** we can define **call by value** and **call by name** for the parameters

  scala> def square(x**:** Double)…. **// call by value**

  scala> def square(x**: =>** Double)…. **// call by name**

- It changes the order the parameter are evaluated
- Ex) call by name vs. call by value

  scala> def callName(z: => Double) = {println("name")} **// call by name**
  scala> def callValue(z: Double) = {println("value")} **// call by value**
  scala> def printParam(z: Double): Double = {println(z); z} // for param
  scala> callName(printParam(4444))
  name
  scala> callValue(printParam(4444))
  4444.0
  value

Anonymous functions

- Lightweight anonymous functions
  scala> (x: Int) => x * x
  res1: Int => Int = <function1>

- This is a syntactic sugar for
  new Function1[Int ,Int] {
  ** def apply(x: Int): Int = x * x**
  }

- Calling the anonymous function
  scala> val square = (x: Int) => x * x
  square: Int => Int = <function1>
  scala> square(2)
  res2: Int = 4

Currying

- Converting a function **with multiple arguments** into a function **with a single argument** that **returns another function**.

```
scala> def gen(f: Int => Int)(x: Int) = f(x)

scala> def identity(x: Int) = gen(i => i)(x)
scala> def square(x: Int) = gen(i => i * i)(x)
scala> def cube(x: Int) = gen(i => i * i * i)(x)

scala> identity(2)
res4: Int = 2
scala> square(2)
res5: Int = 4
scala> cube(2)
res6: Int = 8

scala> def f100(f: Int => Int) = gen(f)(100)
scala> f100(x => x%5)
res7: Int = 0
```

Anonymous Parameters

- Define function
  scala> def doWithOneAndTwo(f: (Int, Int) => Int) = f

- **Explicit** type declaration
  scala> val call1 = doWithOneAndTwo(**(x: Int, y: Int)** => x + y)
  scala> call1(100, 100)

- The compiler expects 2 ints so x and y types are **inferred**
  scala> val call2 = doWithOneAndTwo(**(x, y)** => x + y)
  scala> call2(100, 100)

- Even more **concise** syntax by _(underscore)
  scala> val call3 = doWithOneAndTwo(_ + _)
  scala> call3(100, 100)

Returning multiple variables

- Return multiple variables by **Tuples**

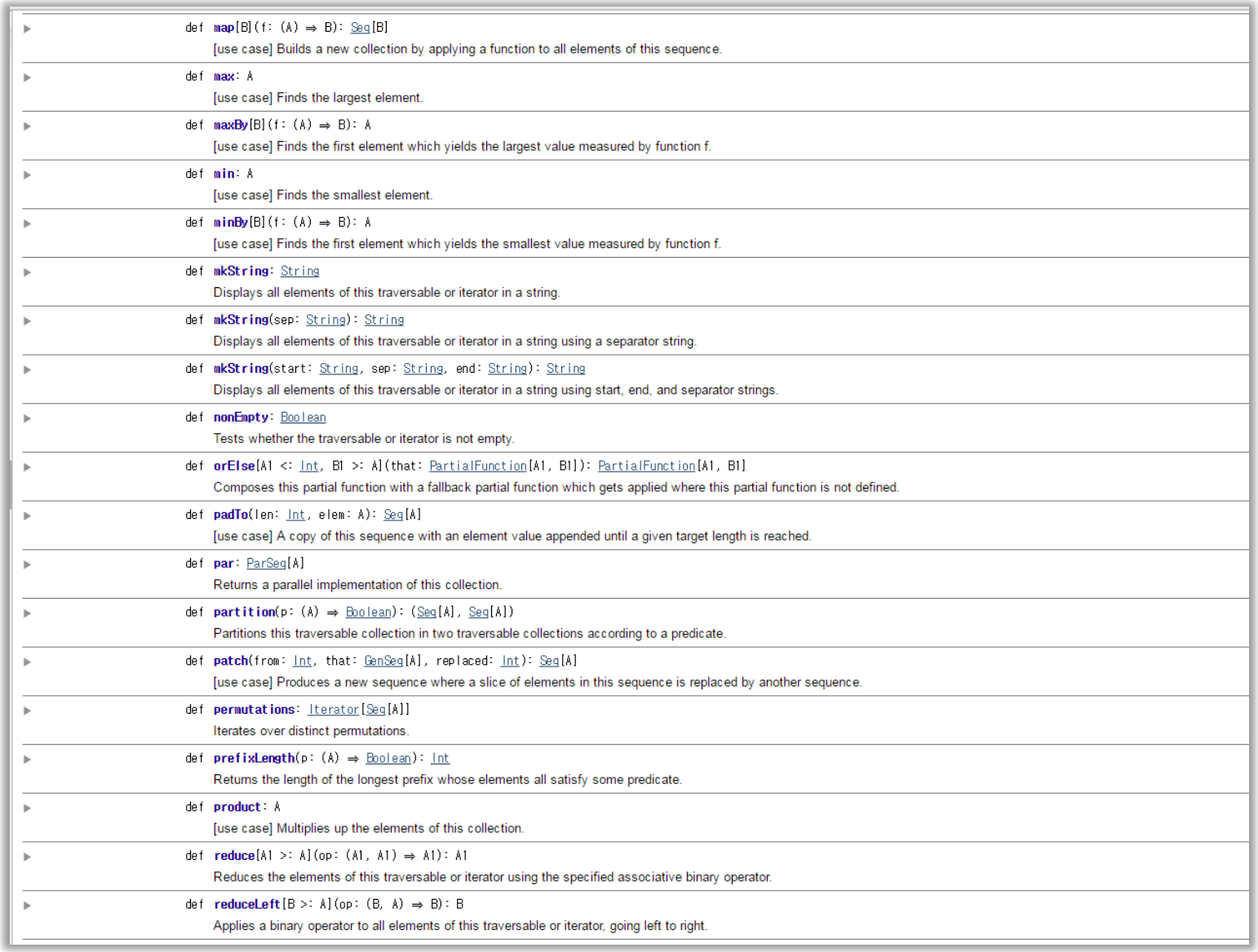
```
scala> def swap(x:String, y:String) = (y, x)
scala> val (a,b) = swap("hello", "world")
scala> println(a, b)
(world,hello)
```

High Order Functions

- Methods that take **as parameter functions**

```
scala> val list = (1 to 4).toList
scala> list.foreach( x => println(x))
scala> list.foreach(println)

scala> list.map(x => x + 2)
scala> list.map(_ + 2)

scala> list.filter(x => x % 2 == 1)
scala> list.filter(_ % 2 == 1)

scala> list.reduce((x,y) => x + y)
scala> list.reduce(_ + _)
```

Function Methods on Collections



def **map**[B](f: (A) ⇒ B): Seq[B]
[use case] Builds a new collection by applying a function to all elements of this sequence.

def **max**: A
[use case] Finds the largest element.

def **maxBy**[B](f: (A) ⇒ B): A
[use case] Finds the first element which yields the largest value measured by function f.

def **min**: A
[use case] Finds the smallest element.

def **minBy**[B](f: (A) ⇒ B): A
[use case] Finds the first element which yields the smallest value measured by function f.

def **mkString**: String
Displays all elements of this traversable or iterator in a string.

def **mkString**(sep: String): String
Displays all elements of this traversable or iterator in a string using a separator string.

def **mkString**(start: String, sep: String, end: String): String
Displays all elements of this traversable or iterator in a string using start, end, and separator strings.

def **nonEmpty**: Boolean
Tests whether the traversable or iterator is not empty.

def **orElse**[A1 <: Int, B1 >: A](that: PartialFunction[A1, B1]): PartialFunction[A1, B1]
Composes this partial function with a fallback partial function which gets applied where this partial function is not defined.

def **padTo**(len: Int, elem: A): Seq[A]
[use case] A copy of this sequence with an element value appended until a given target length is reached.

def **par**: ParSeq[A]
Returns a parallel implementation of this collection.

def **partition**(p: (A) ⇒ Boolean): (Seq[A], Seq[A])
Partitions this traversable collection in two traversable collections according to a predicate.

def **patch**(from: Int, that: GenSeq[A], replaced: Int): Seq[A]
[use case] Produces a new sequence where a slice of elements in this sequence is replaced by another sequence.

def **permutations**: Iterator[Seq[A]]
Iterates over distinct permutations.

def **prefixLength**(p: (A) ⇒ Boolean): Int
Returns the length of the longest prefix whose elements all satisfy some predicate.

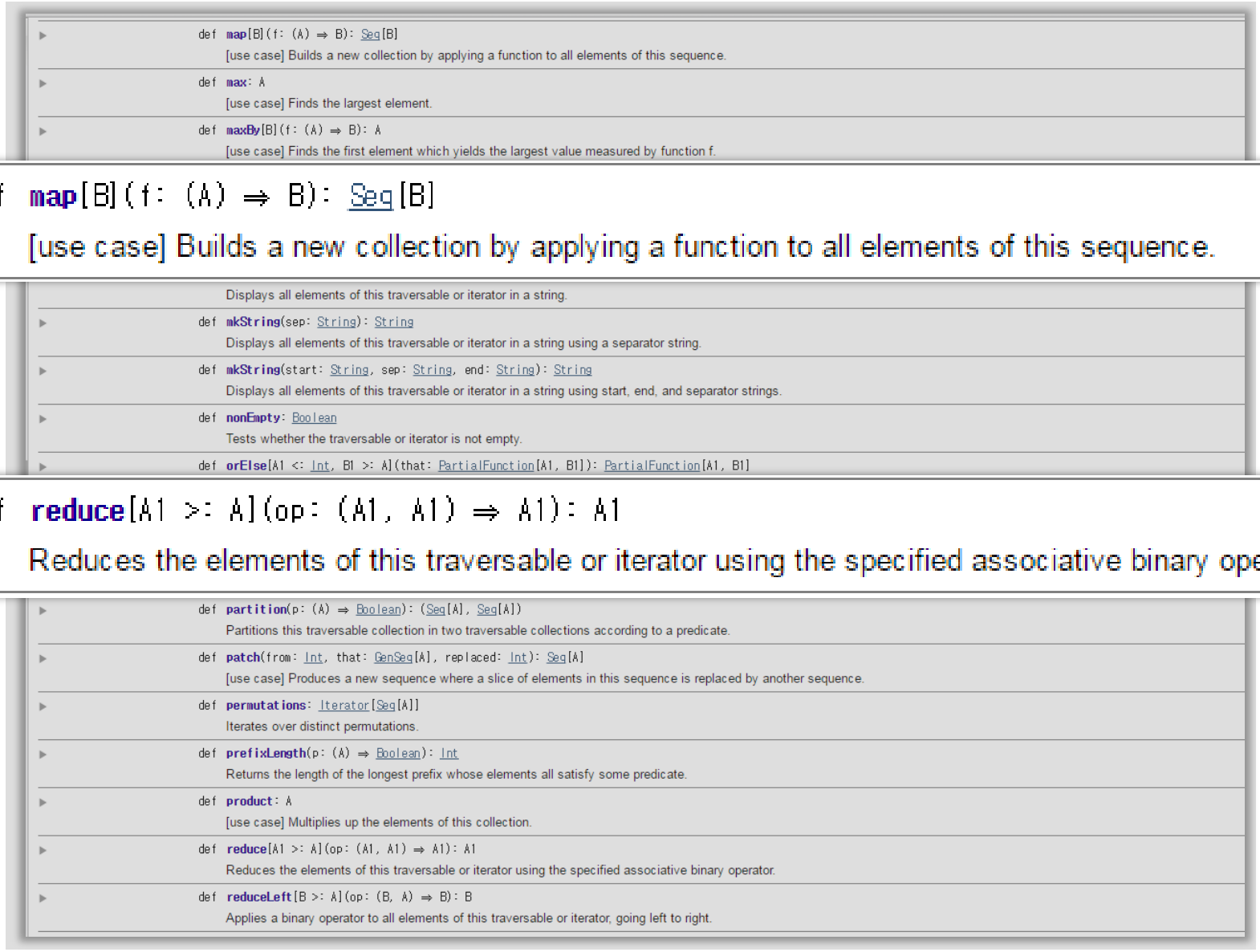
def **product**: A
[use case] Multiplies up the elements of this collection.

def **reduce**[A1 >: A](op: (A1, A1) ⇒ A1): A1
Reduces the elements of this traversable or iterator using the specified associative binary operator.

def **reduceLeft**[B >: A](op: (B, A) ⇒ B): B
Applies a binary operator to all elements of this traversable or iterator, going left to right.

http://www.scala-lang.org/api/2.11.6/index.html#scala.collection.Seq

Function Methods on Collections



def **map**[B](f: (A) ⇒ B): Seq[B]

[use case] Builds a new collection by applying a function to all elements of this sequence.

def **map**[B](f: (A) ⇒ B): Seq[B]

[use case] Builds a new collection by applying a function to all elements of this sequence.

Displays all elements of this traversable or iterator in a string.

def **mkString**(sep: String): String

Displays all elements of this traversable or iterator in a string using a separator string.

def **mkString**(start: String, sep: String, end: String): String

Displays all elements of this traversable or iterator in a string using start, end, and separator strings.

def **nonEmpty**: Boolean

Tests whether the traversable or iterator is not empty.

def **orElse**[A1 <: Int, B1 >: A](that: PartialFunction[A1, B1]): PartialFunction[A1, B1]

def **reduce**[A1 >: A](op: (A1, A1) ⇒ A1): A1

Reduces the elements of this traversable or iterator using the specified associative binary operator.

def **partition**(p: (A) ⇒ Boolean): (Seq[A], Seq[A])

Partitions this traversable collection in two traversable collections according to a predicate.

def **patch**(from: Int, that: GenSeq[A], replaced: Int): Seq[A]

[use case] Produces a new sequence where a slice of elements in this sequence is replaced by another sequence.

def **permutations**: Iterator[Seq[A]]

Iterates over distinct permutations.

def **prefixLength**(p: (A) ⇒ Boolean): Int

Returns the length of the longest prefix whose elements all satisfy some predicate.

def **product**: A

[use case] Multiplies up the elements of this collection.

def **reduce**[A1 >: A](op: (A1, A1) ⇒ A1): A1

Reduces the elements of this traversable or iterator using the specified associative binary operator.

def **reduceLeft**[B >: A](op: (B, A) ⇒ B): B

Applies a binary operator to all elements of this traversable or iterator, going left to right.

http://www.scala-lang.org/api/2.11.6/index.html#scala.collection.Seq

1. Spark 이해하기

1.1 Spark 개요

1.2 Scala 개요

1.3 Spark 설치 및 실행

[LAB] Spark 설치

- **Spark 2.4.4** (Sep 01, 2019)
- Spark runs on **Java 8+**, Python 2.7+/3.4+ and R 3.1+.
- For the Scala API, Spark 2.4.4 uses **Scala 2.11**.
- You will need to use a compatible Scala version (2.11.x).

http://spark.apache.org/docs/latest/

[LAB] Java 8 구성

- Java 8 다운로드, 압축해제, PATH 설정, 설치확인
- # cd /kikang
- # wget --no-cookies --no-check-certificate --header "Cookie: oraclelicense=accept-securebackup-cookie" "http://download.oracle.com/otn-pub/java/jdk/8u131-b11/d54c1d3a095b4ff2b6607d096fa80163/jdk-8u131-linux-x64.tar.gz"
- # tar xvfz jdk-8u131-linux-x64.tar.gz
- # vi ~/.bash_profile

  # added by kikang....
  export JAVA_HOME=/kikang/jdk1.8.0_131
  export PATH=$JAVA_HOME/bin:$PATH
- # source ~/.bash_profile
- # java -version

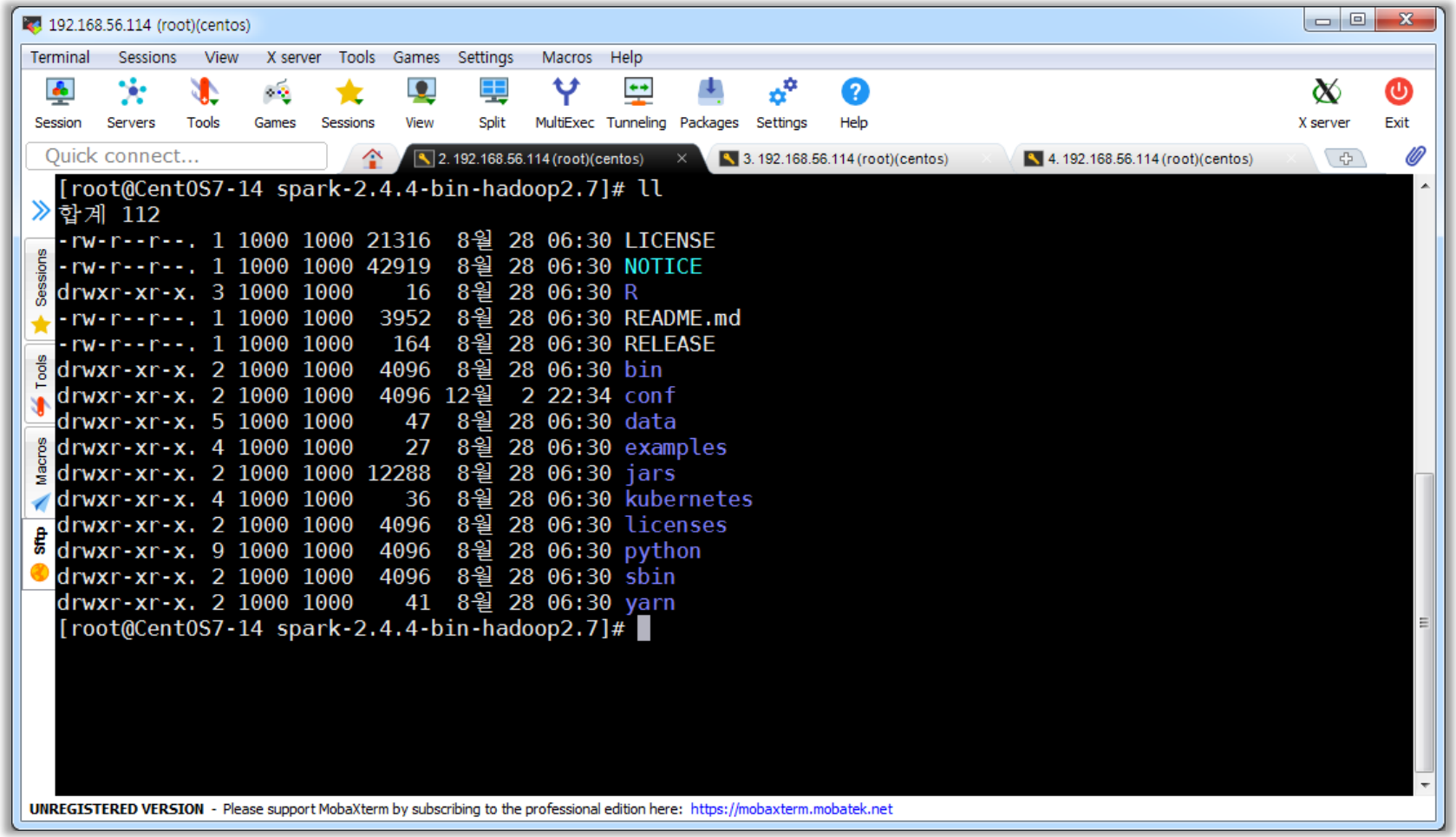
[LAB] Spark 2.4.4 구성

- Spark 2.4.4 바이너리 파일 다운로드, 압축해제, Spark-Shell 실행확인
- # cd /kikang
- # wget http://mirror.apache-kr.org/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
- # tar xvfz spark-2.4.4-bin-hadoop2.7.tgz
- # cd /kikang/spark-2.4.4-bin-hadoop2.7
- # ./bin/spark-shell

http://spark.apache.org/downloads.html

[LAB] Spark Directory 탐색

- Spark 2.4.4의 주요 Directory 탐색 (bin, conf, examples, sbin)
- # cd /kikang/spark-2.4.4-bin-hadoop2.7



[LAB] Spark-Shell 기본

- Scala용 Spark-Shell 실행, 프로세스 확인, 웹 UI 확인
- # cd /kikang/spark-2.4.4-bin-hadoop2.7
- # ./bin/spark-shell
- scala> sc
- scala> spark
- scala> sc.master
- SparkSubmit 프로세스 확인
  # jps
- 드라이버 프로그램용 웹 UI 확인
  http://192.168.56.114:4040

[LAB] RDD 생성

- 드라이버 프로그램 내 데이터를 이용하여 RDD 생성 및 필터링
- scala> val data = 1 to 10000
- scala> val distData = sc.parallelize(data)
- scala> distData.filter(_ < 10).collect()
- 웹 UI 확인

[LAB] Lazy Evaluation

- 외부 데이터(파일)를 이용하여 RDD 생성 및 Lazy Evaluation 확인
- scala> val data = sc.textFile("README.txt") // 잘못된 파일명
- scala> val distData = data.map(r => r + "_map")
- scala> distData.count // InvalidInput 예외 발생
- scala> val data = sc.textFile("README.md") // 파일명 정정
- scala> val distData = data.map(r => r + "_map")
- scala> distData.count

[LAB] Lineage 확인, Partition 개수 확인

- scala> distData.toDebugString
- scala> distData.getNumPartitions

[LAB] Shuffle by Repartition

- scala> val data = sc.textFile("README.md")
- scala> data.getNumPartitions
- scala> val distData = data.map(r => r + "_map")
- scala> distData.getNumPartitions // 파티션 개수 변화 없음
- scala> val newData = distData.repartition(10)
- scala> newData.getNumPartitions // 파티션 개수 변경됨
- scala> newData.toDebugString // Lineage 확인
- scala> newData.count
- 웹 UI 확인 (셔플데이터 Disk 기록 확인)
- scala> newData.collect.foreach(println)
- 웹 UI 확인 (Stage Skip 확인)
- scala> data.saveAsTextFile("dataREADME.md")
- 저장된 파일 내용 확인 (Partition 별로 원본 파일 내용 유지되어 있음)
- scala> newData.saveAsTextFile("newData")
- 저장된 파일 내용 확인 (Partition 별로 원본 파일 내용 섞여 있음)

[LAB] Shuffle by Repartition

- scala> val intRDD2 = sc.parallelize(1 to 20)
- scala> intRDD2.saveAsTextFile("intRDD2")
- scala> val intRDD5 = intRDD2.repartition(5)
- scala> intRDD5.saveAsTextFile("intRDD5")
- 저장된 파일 내용 확인 (Partition 별 데이터 확인)

[LAB] Partitions for Files

- scala> val rdd = sc.textFile("README.md")
- scala> rdd.getNumPartitions
- scala> val rdd2 = sc.textFile("NOTICE")
- scala> rdd2.getNumPartitions
- scala> val rdd3 = sc.textFile("README.md,NOTICE")
- scala> rdd3.getNumPartitions
- scala> val rdd4 = sc.textFile("LI*,NO*,RE*")
- scala> rdd4.getNumPartitions
- scala> rdd.saveAsTextFile("rdd")
- scala> rdd2.saveAsTextFile("rdd2")
- scala> rdd3.saveAsTextFile("rdd3")
- scala> rdd4.saveAsTextFile("rdd4")
- 웹 UI 확인 (Task = Partition 개수 확인) (rdd4 => partition별 데이터 unbalance. 특정 partition으로 데이터가 치우칠 수 있음)
- 저장된 파일 내용 확인 (rdd4 => File별로 Partition 구성되어 있음)

[LAB] Cache

- scala> val data = sc.textFile("README.md")
- scala> val distData = data.map(r => r + "_map")
- scala> distData.name = "distData" // 이름부여
- scala> distData.cache // 캐시
- scala> distData.take(5) // distData.top(5)과는 다른 결과 나옴....
- 웹 UI 확인 (Storage탭에서 Fraction Cached 부분이 100%가 아님을 확인)
- 웹 UI 확인 (Task도 1개만 실행되었음을 확인)
- scala> distData.collect
- 웹 UI 확인 (Storage탭에서 Fraction Cached 부분이 100%임을 확인)
- 웹 UI 확인 (Task도 모두 실행되었음을 확인)
- scala> distData.getStorageLevel // 캐시 StorageLevel 확인
- scala> distData.unpersist() // 캐시삭제
- 웹 UI 확인 (Storage탭에서 캐시삭제 확인)

[LAB] Cache Size(Int vs String vs Object) 1.3 Spark 설치 및 실행

- scala> val intRdd = sc.parallelize(1 to 10000)
- scala> intRdd.name = "intRdd"
- scala> intRdd.cache
- scala> intRdd.count
- scala> val strRdd = intRdd.map(_.toString)
- scala> strRdd.name = "strRdd"
- scala> strRdd.cache
- scala> strRdd.count
- scala> case class strCase (str: String)
- scala> val strCaseRdd = intRdd.map(x => strCase(x.toString))
- scala> strCaseRdd.name = "strCaseRdd"
- scala> strCaseRdd.cache
- scala> strCaseRdd.count
- 웹 UI 확인 (Storage탭에서 캐시 Size 확인)
- Object(Header 16 bytes), String(Overhead 40 bytes, 2 bytes/char)

[LAB] Get Cached RDD by Name

- scala> val data = sc.textFile("README.md")
- scala> val distData = data.map(r => r + "_map")
- scala> data.setName("data") //이름부여
- scala> distData.name = "distData" // 이름부여
- scala> data.cache // 캐시
- scala> distData.cache // 캐시
- scala> sc.getPersistentRDDs // 캐시 설정된 RDD 목록 확인….
- scala> distData.collect // action 수행….
- 웹 UI 확인 (distData만 action 실행했는데 data, distData 둘 다 캐시됨)
- scala> sc.getPersistentRDDs.filter(x => x._2.name.equals("data")).foreach(x => x._2.unpersist()) // 특정 캐시("data")만 제거….
- scala> sc.getPersistentRDDs.foreach(x => x._2.unpersist()) // 모든 캐시 제거….
- scala> distData.collect // action 수행….
- 웹 UI 확인 (캐시 없음. 기존 캐시 모두 unpersist를 통해 설정 해제 됨)

[LAB] Word Count

- scala> val f = sc.textFile("README.md")
- scala> val wc = f.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
- scala> wc.saveAsTextFile("wc_out.txt")
- 저장된 파일 확인
- scala> wc.collect.foreach(println)
- 웹 UI 확인 (Stage Skip 확인)