Apache Spark을 활용한 실시간 데이터 처리 / 분석

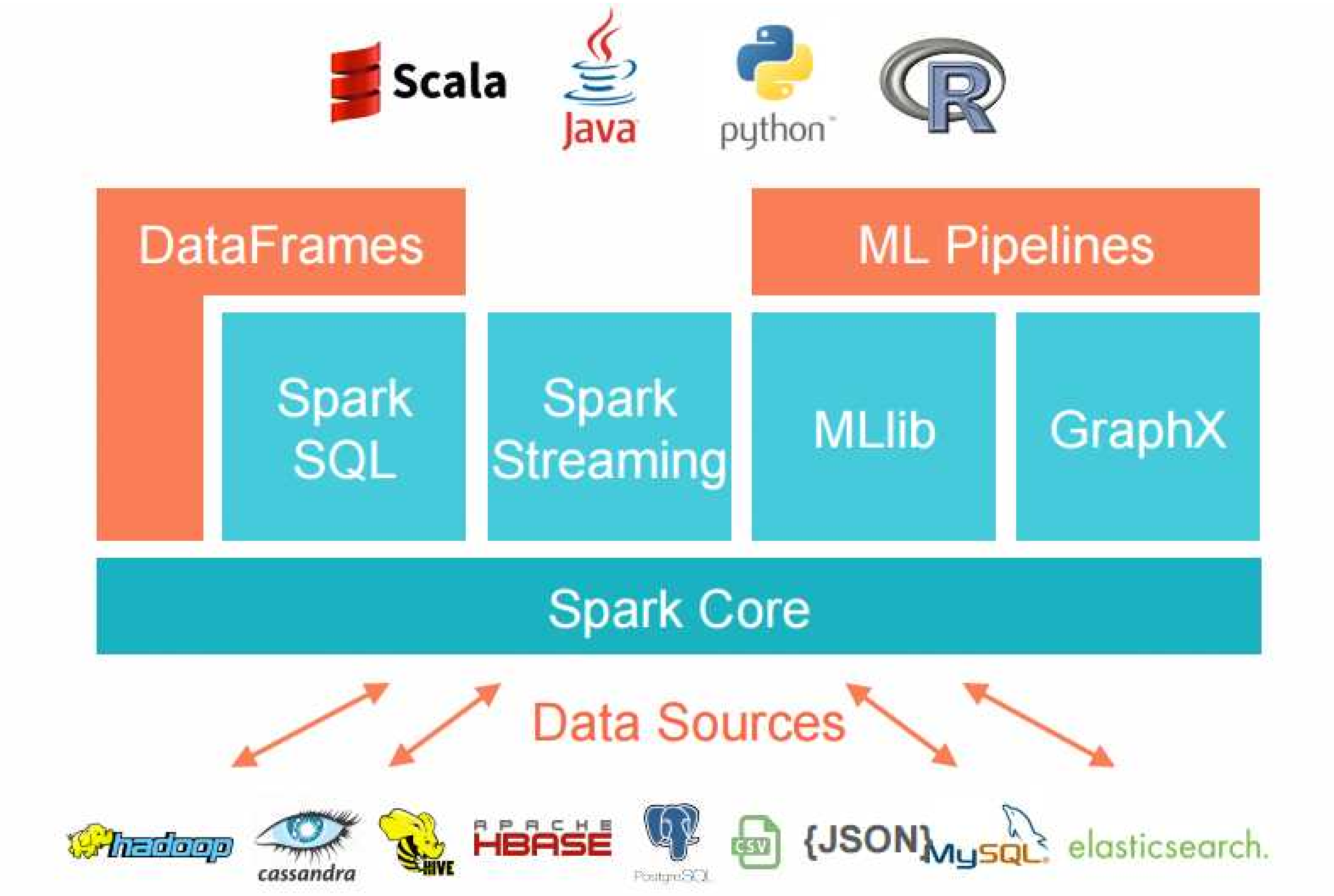
㈜라온비트 박진수

라온비트 RAONBIT

CONTENTS
PRESENTATION

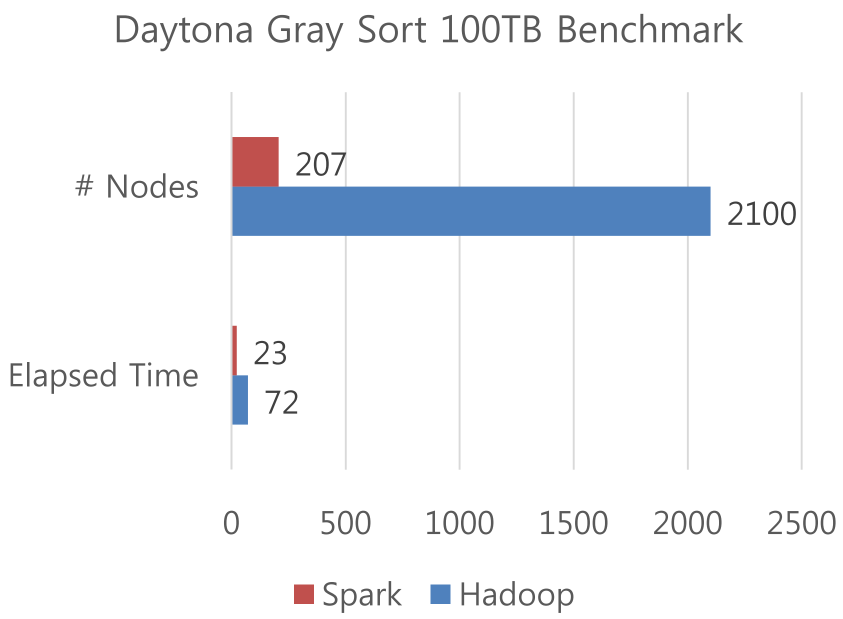
Apache Spark 개요

Overview



Scala
Java
python
R
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Sources
hadoop
cassandra
HIVE
APACHE HBASE
PostgreSQL
CSV
{JSON}
MySQL
elasticsearch.

Overview

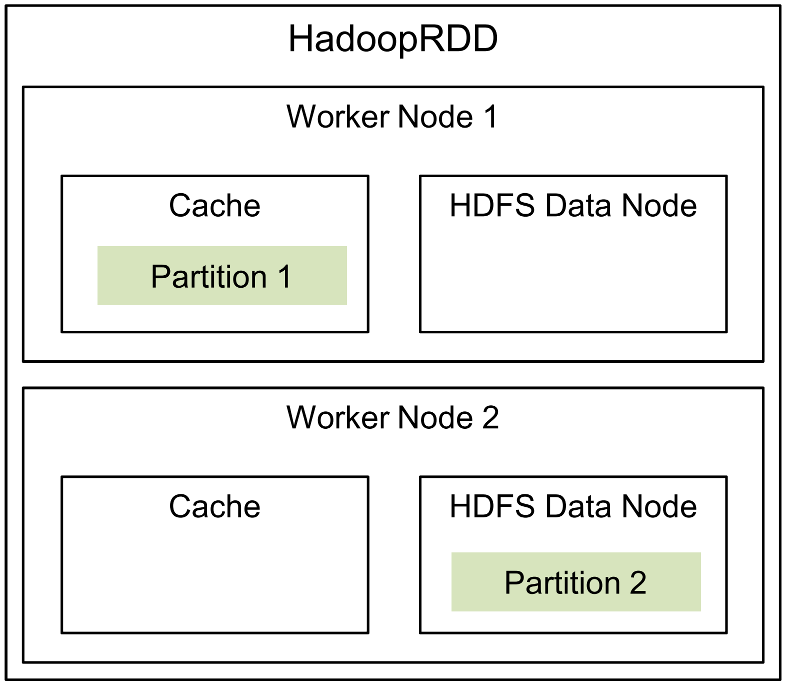


	Hadoop	Spark
Data Size	**102.5 TB**	**100 TB**
Elapsed Time	**72 mins**	**23 mins**
# Nodes	**2100**	**207**
# Cores	**50400**	**6592**
Rate	**1.42 TB/min**	**4.27 TB/min**
Rate/node	**0.67 GB/min**	**20.7 GB/min**
Sort Benchmark Daytona Rules	**Yes**	**Yes**

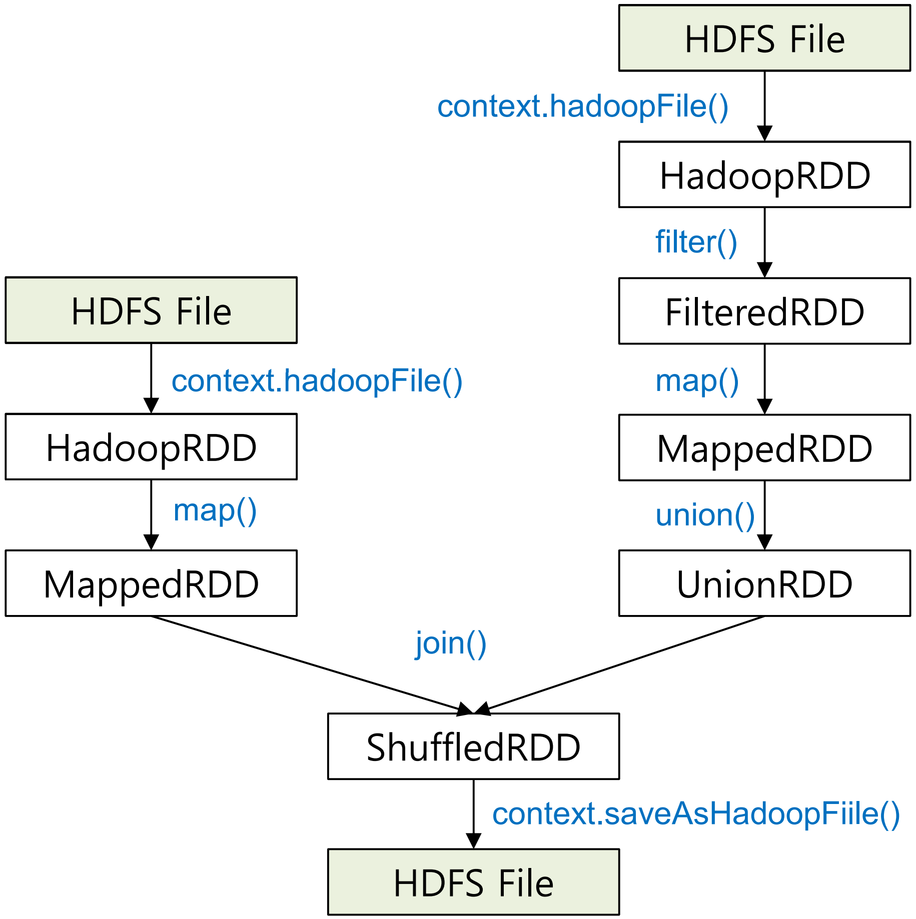
- Hadoop : 2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks
- Spark : Amazon EC2 i2.8xlarge nodes x
  32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD

RDD(Resilient Distributed Dataset)

- Spark의 핵심 추상화 기법
- Immutable, re-computable, fault-tolerant partitioned collections of records
- 클러스터 노드들 사이에 파티션을 표현
- Data Set의 병렬 처리를 가능하게 함
- 파티션은 메모리나 디스크에 존재



< RDD Partition >



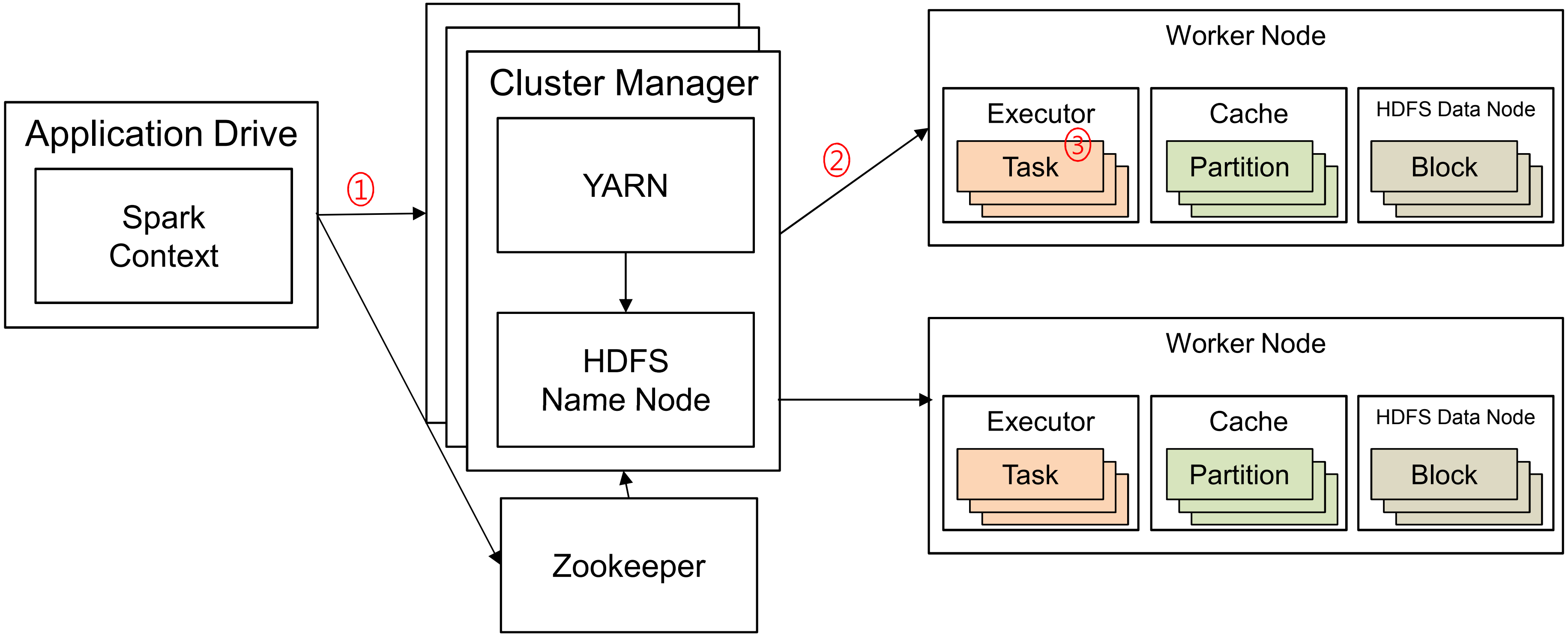
< RDD Lineage >

RDD Operations

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	:	$(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	:	$(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
Actions	$count()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Long}$
	$collect()$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]}$
	$reduce(f : (\mathrm{T}, \mathrm{T}) \Rightarrow \mathrm{T})$	:	$\mathrm{RDD[T]} \Rightarrow \mathrm{T}$
	$lookup(k : \mathrm{K})$	:	$\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]}$ (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	:	Outputs RDD to a storage system, *e.g.*, HDFS

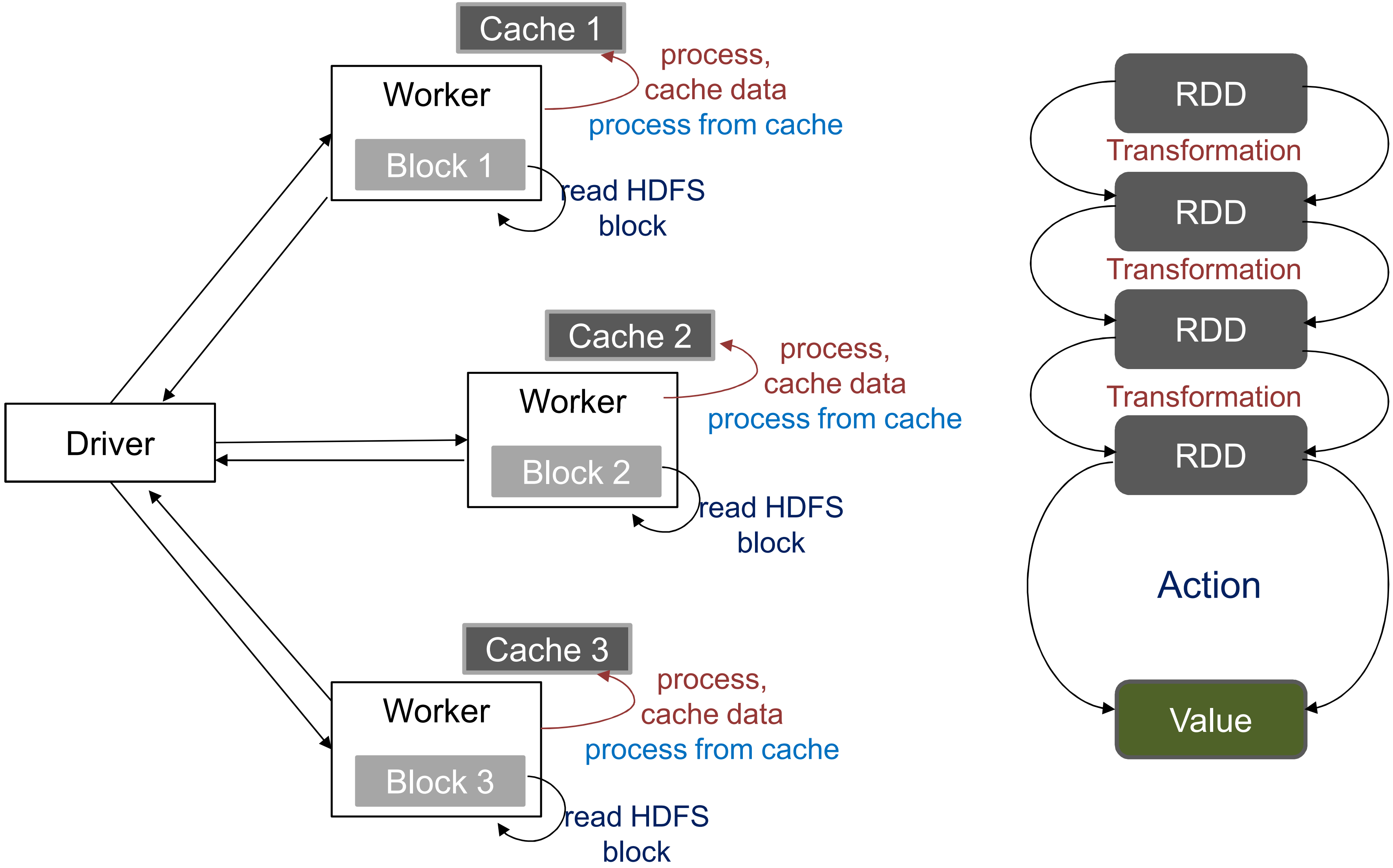
- lazy-evaluation : transformation 연산은 실제 데이터를 가져와서 RDD를 만드는 대신 RDD를 생성할 수 있는 lineage 정보만 생성, action 연산이 실행되면 그 때 실제 데이터를 가져와서 RDD를 생성하고 연산 수행
- 자원을 효율적으로 분배하여 사용 가능

Spark Deployment on Hadoop Cluster

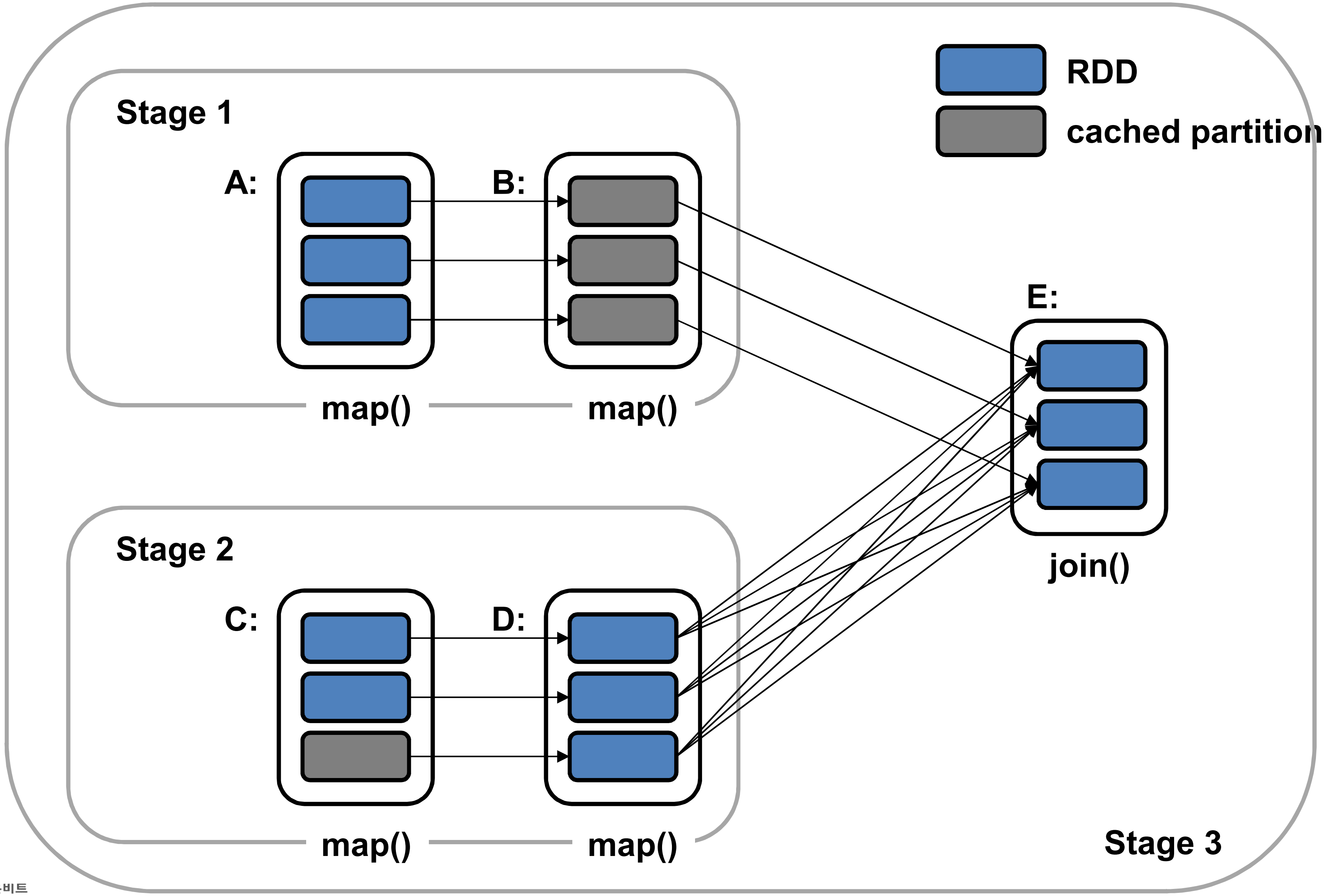


1. master는 어플리케이션들 사이에 리소스 배분하기 위해 cluster mange에 접근, 클러스터 상에서 task를 수행하고 데이터를 cache 할 수 있는 executor 획득
2. app code를 executor에 전송
3. task를 executor에 전송

Spark Application



Spark Application Operator Graph



MapReduce vs. Spark Application

```java
public class WordCount {
   public static class Map 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 tokenizer = new StringTokenizer(line);
           while (tokenizer.hasMoreTokens()) {
              word.set(tokenizer.nextToken());
               output.collect(word, one);
           }
        }
   }

  public static class Reduce 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));
       }
   }
```

```scala
object WordCount {

  def main(args: Array[String]) {
    val conf = new
SparkConf().setAppName("WordCount").
              setMaster("local[*]")
    val sc = new SparkContext(conf)
    val file = sc.textFile(args(0))
    val word = file.flatMap(_.split(" ")).map(w => (w,
1)).cache()
    word.reduceByKey(_ + _).saveAsTextFile(args(1))
  }

}
```

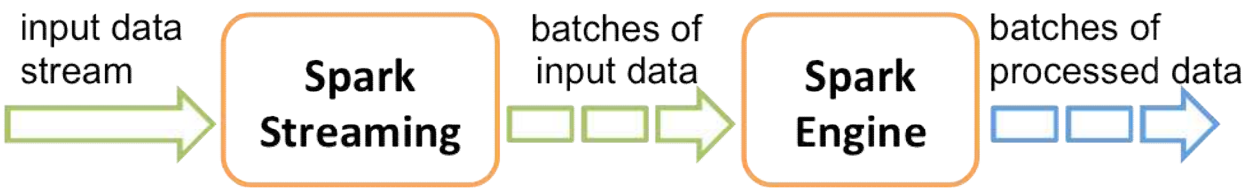
Apache Spark Streaming 개요

Overview

❑ 대규모의 실시간 데이터 처리를 위한 고성능의 장애 허용 framework, Spark Core 확장 API

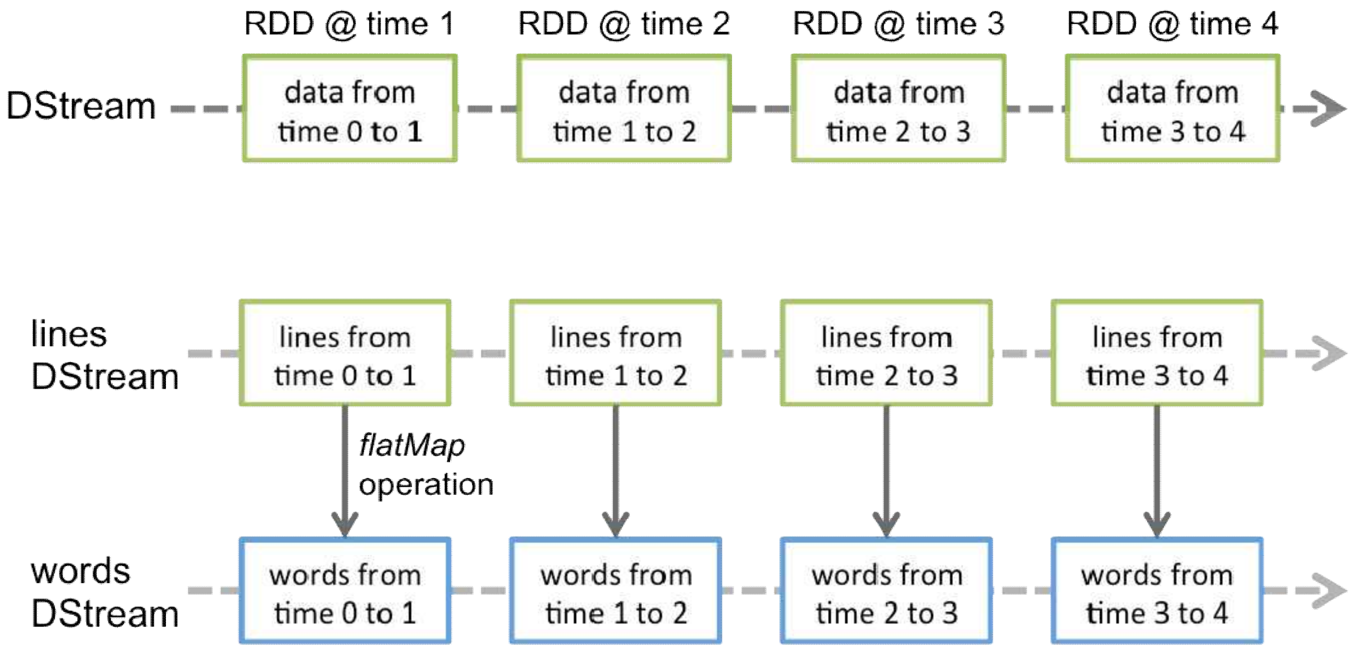
❑ Streaming 연산을 아주 작은 batch 작업의 연속으로 처리

- Scalable, Second-scale latencies
- Integrated with batch & interactive processing
- Simple programming model
- Efficient fault-tolerance in stateful computations



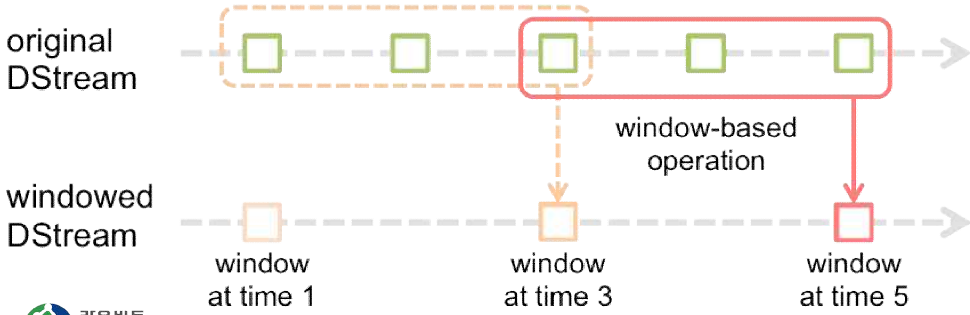
DStream(Discretized Stream)

- Spark Streaming의 Programming Model
  - Stream Data를 표현하는 연속된 RDD
  - RDD와 마찬가지로 input source에서 생성하거나 DStream을 transform 하여 생성
  - RDD 연산을 그대로 사용 – Batch(Historical) Data와 Stream Data를 동일한 방식으로 처리



Window Operations

Transformation	Window	Output
▪ map(func), flatMap(func), filter(func), count() ▪ repartition(numPartitions) ▪ union(otherStream) ▪ reduce(func), countByValue(), reduceBykey(func, [numTasks]) ▪ join(otherStream, [numTasks]), cogroup(otherStream, [numTasks]) ▪ transform(func) ▪ updateStateByKey(func)	▪ window(length, interval) ▪ countByWindow(length, interval) ▪ reduceByWindow(func, length, interval) ▪ reduceByKeyAndWindow(func, length, interval, [numTasks]) ▪ countByValueAndWindow(length, interval, [numTasks])	▪ Print() ▪ foreachRDD(func) ▪ saveAsObjectFiles(prefix, [suffix]) ▪ saveAsTextFiles(prefix, [suffix]) ▪ saveAsHadoopFiles(prefix, [suffix])



- window length : window의 기간
- sliding interval : window 연산이 수행되는 간격 (batch 크기의 배수로 지정해야 함)
- checkpointing 필수 : checkpointing duration은 sliding interval 의 5~10배가 적당

Input Source

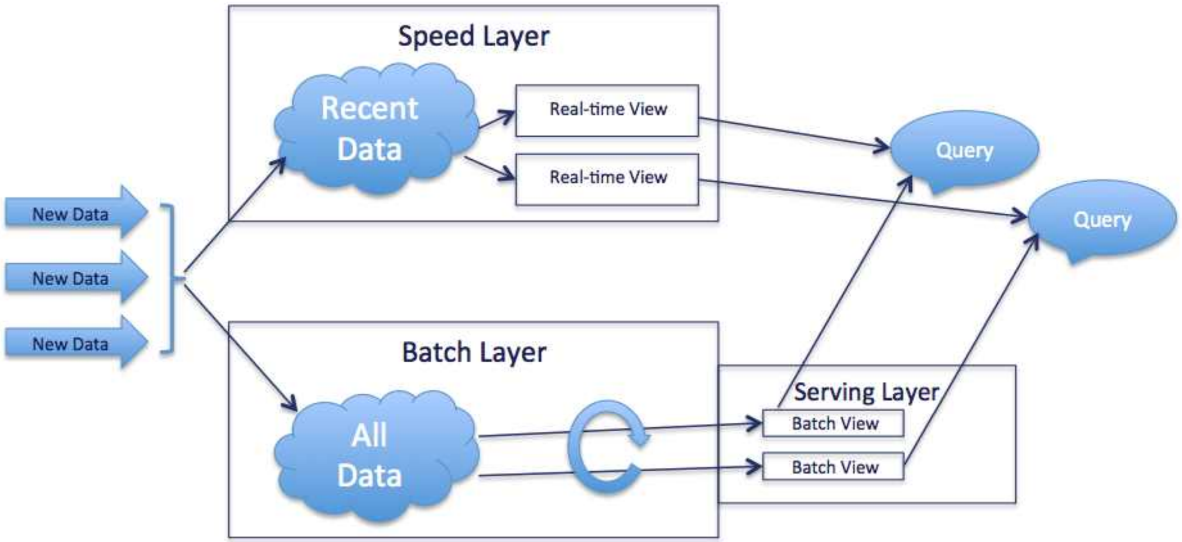
```
val lines = ssc.textFileStream(args(0))
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
```

```
 val stream = FlumeUtils.createStream(ssc, host, port, StorageLevel.MEMORY_ONLY_SER_2)
stream.count().map(cnt => "Received " + cnt + " flume events." ).print()
```

- actorStream
- socketTextStream
- socketStream
- rawSocketStream
- fileStream
- textFileStream
- queueStream

- Twitter(TwitterUtils)
- Kafka(KafkaUtils)
- Flume(FlumeUtils)
- Kinesis(KinesisUtils)
- MQTT(MQTTUtils)
- ZeroMQ(ZeroMQUtils)

Lambda Architecture




```
// batch data
val batchdata = sc.hadoopFile("data")

// streaming data
stream.transform( rdd => rdd.join(data).filter(...)
```

Integrated with Machine Learning

```
val trainingData = ssc.textFileStream(args(0)).map(Vectors.parse)

val testData = ssc.textFileStream(args(1)).map(LabeledPoint.parse)

val model = new StreamingKMeans()

                        .setK(args(3).toInt)

                        .setDecayFactor(1.0)

                        .setRandomCenters(args(4).toInt, 0.0)

model.trainOn(trainingData)

model.predictOnValues(testData.map(lp => (lp.label, lp.features))).print()
```

Integrated with Spark SQL

```
stream.map(rdd => rdd.regiesterTempTable("events"))
```

```
sqlContext.sql("select * from events")
```

Kafka Direct Stream API

- ❑ Receiver

val kafkaStream = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap)

- ❑ Direct(new in 1.3)

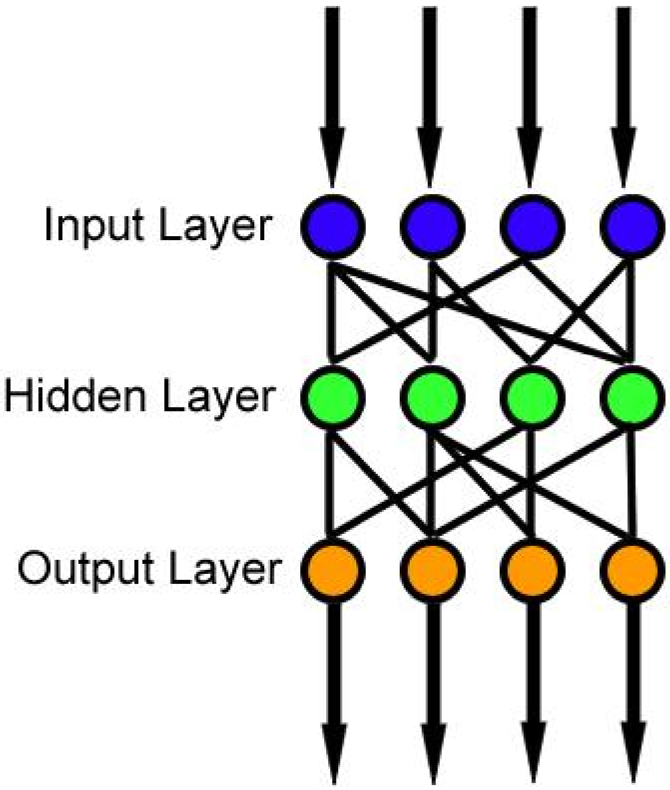
val kafkaStream = KafkaUtils.createStream[String, String, StringDecoder, String, Decoder](ssc, kafkaParams, topicsSet)

Kafka Direct Stream API

- ❑ Kafka를 파일 시스템처럼 취급
- ❑ receiver가 없고 kafka의 가장 최근 토픽 offset에 접근하여 파일을 읽는 것처럼 데이터를 가져옴
- ❑ Zookeeper대신 Spark Streaming 자체에서 kafka offset 정보 관리
- ❑ 보다 효율적이고 장애 허용에 뛰어나며 kafka 데이터를 정확히 한번만 받아옴

Spark Deep Learning

❑ Multilayer perceptron classifier(Spark 1.5.0에 추가)

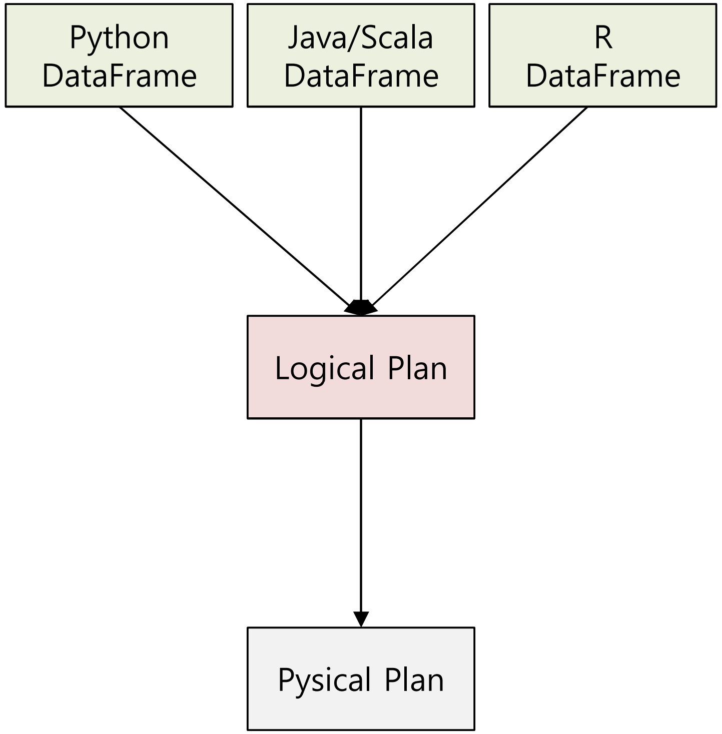


feedforward artificial neural network

```
val data = MLUtils.loadLibSVMFile(sc, "sample.txt").toDF()
val splits = data.randomSplit(Array(0.6, 0.4), seed = 1234L)
val train = splits(0) val test = splits(1)
val layers = Array[Int](4, 5, 4, 3)

val trainer = new MultilayerPerceptronClassifier()
                  .setLayers(layers)
                  .setBlockSize(128)
                  .setSeed(1234L)
                  .setMaxIter(100)
val model = trainer.fit(train)

val result = model.transform(test)
val predictionAndLabels = result.select("prediction", "label")
val evaluator = new MulticlassClassificationEvaluator()
                  .setMetricName("precision")

println("Precision:" + evaluator.evaluate(predictionAndLabels))
```

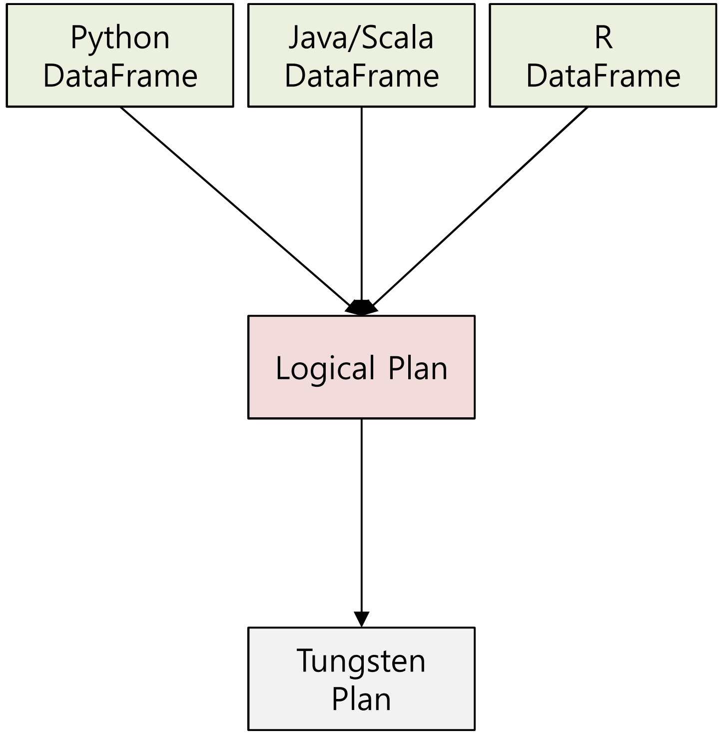
Spark의 현재와 미래

❑ DataFrame



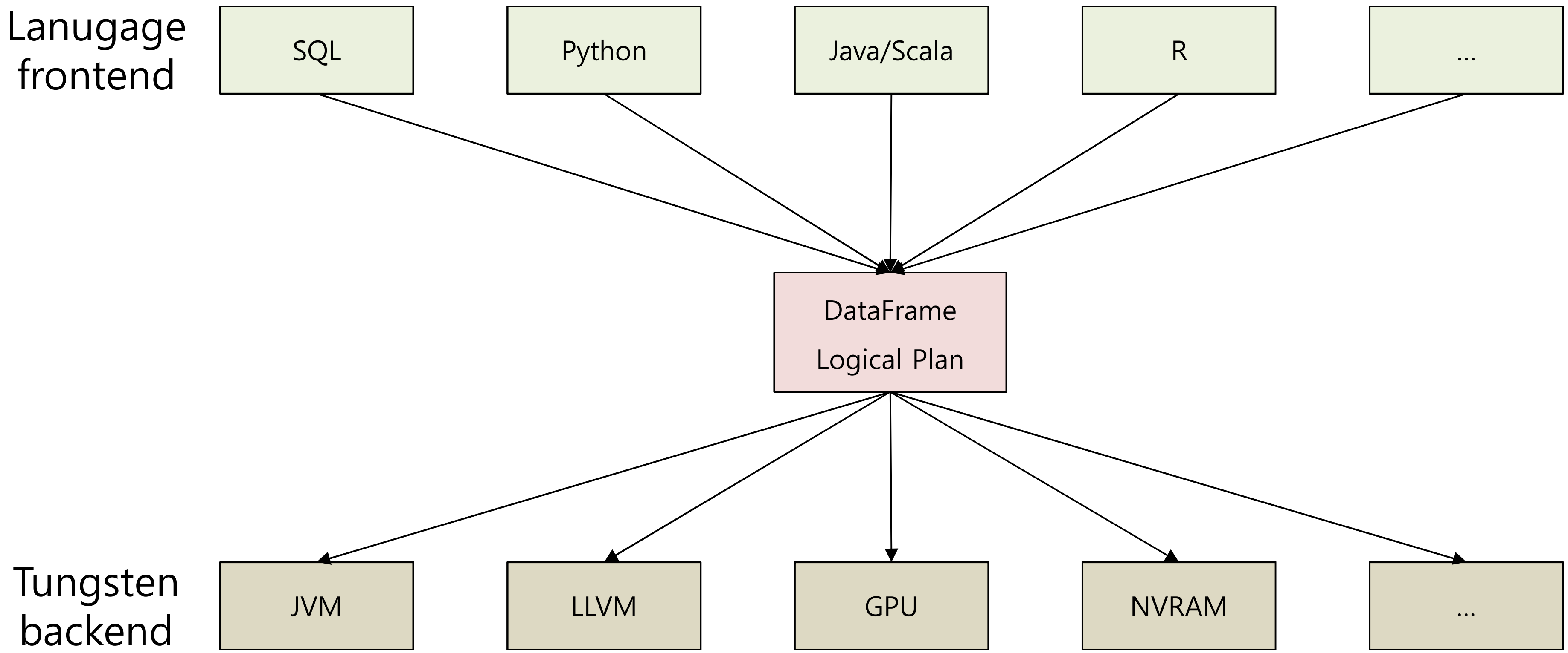
- Spark DF : scalability, multi-core, distributed
- R/Python DF : 다양한 libraries

❑ Tungsten project



- Runtime code generation
- Exploiting cache locality
- Off-heap memory management

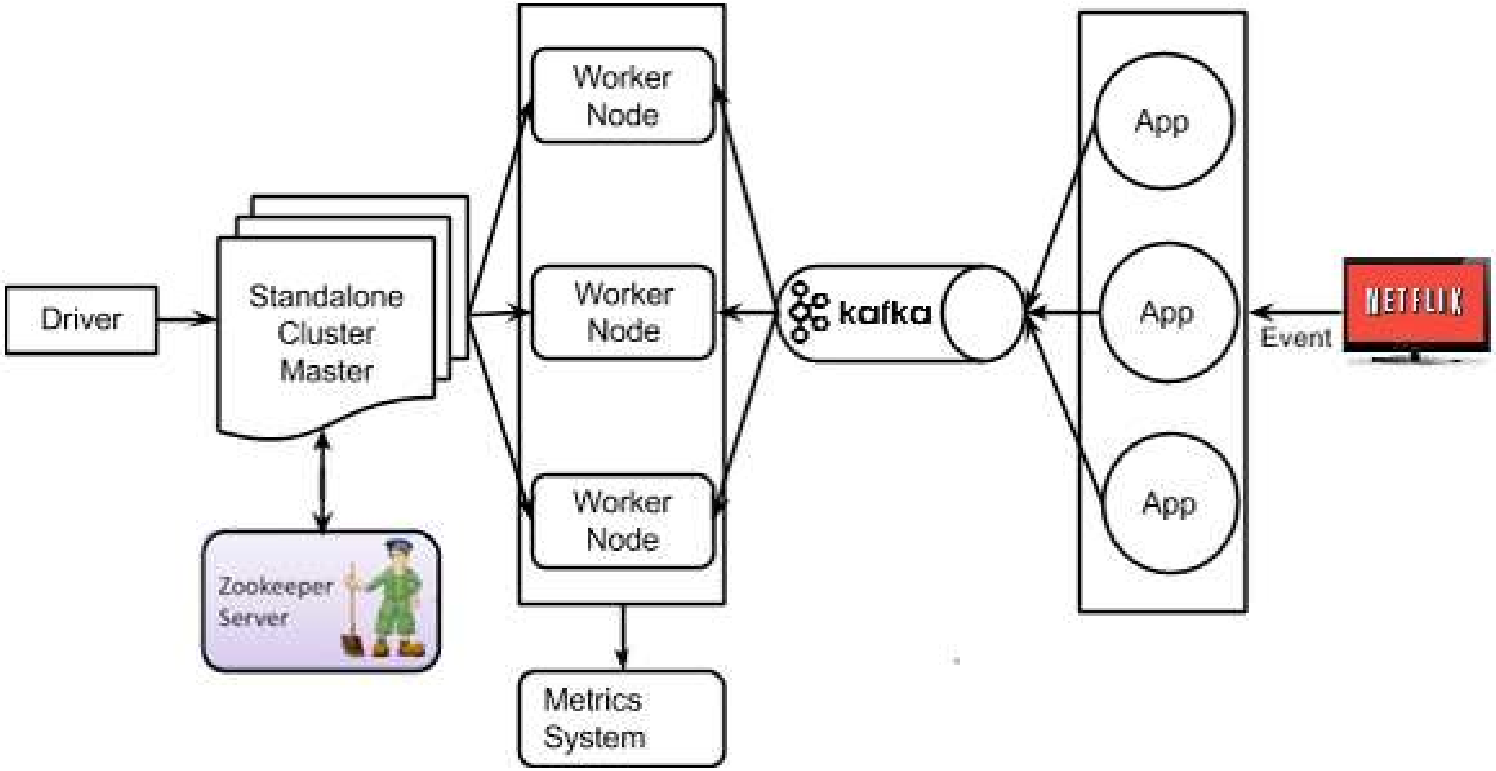
Tungsten project



RAONBIT

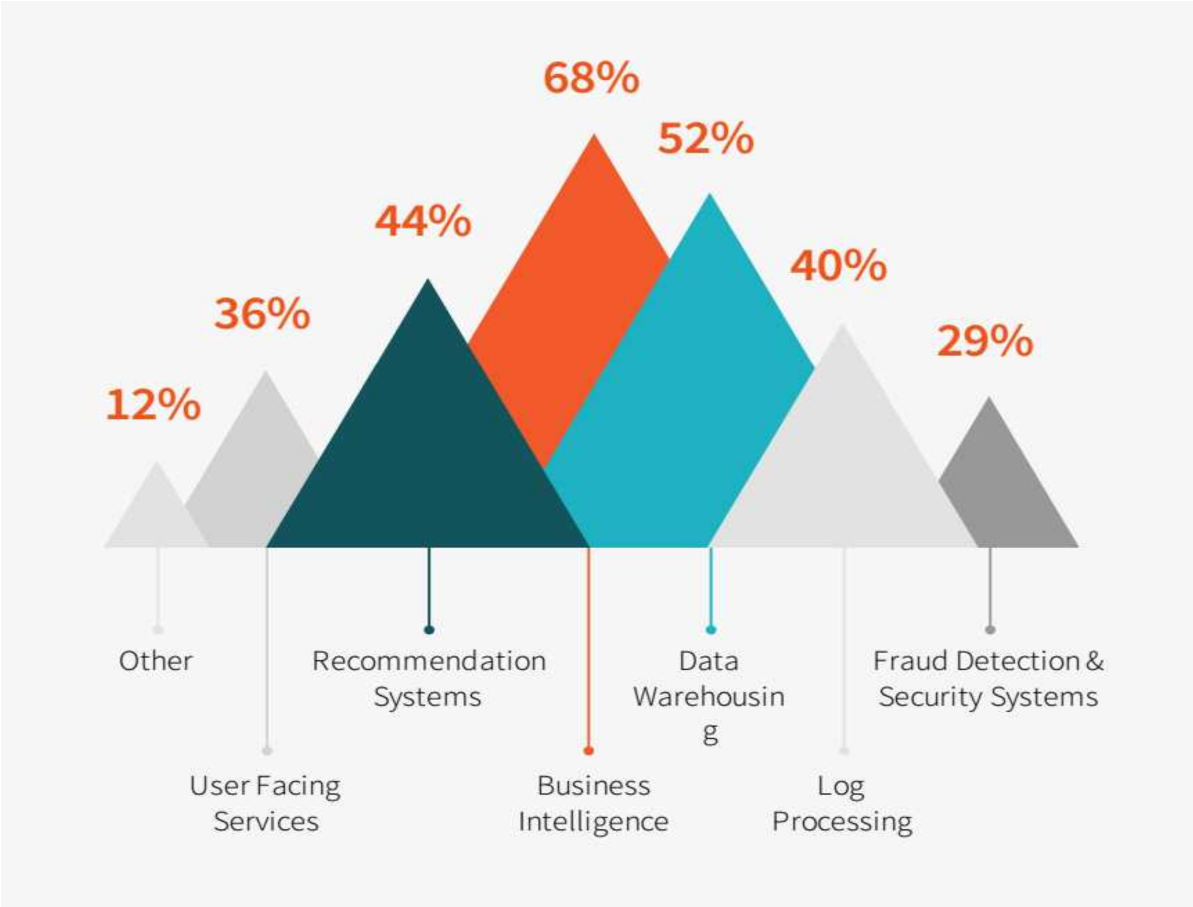
사용 사례

❑ Netflix



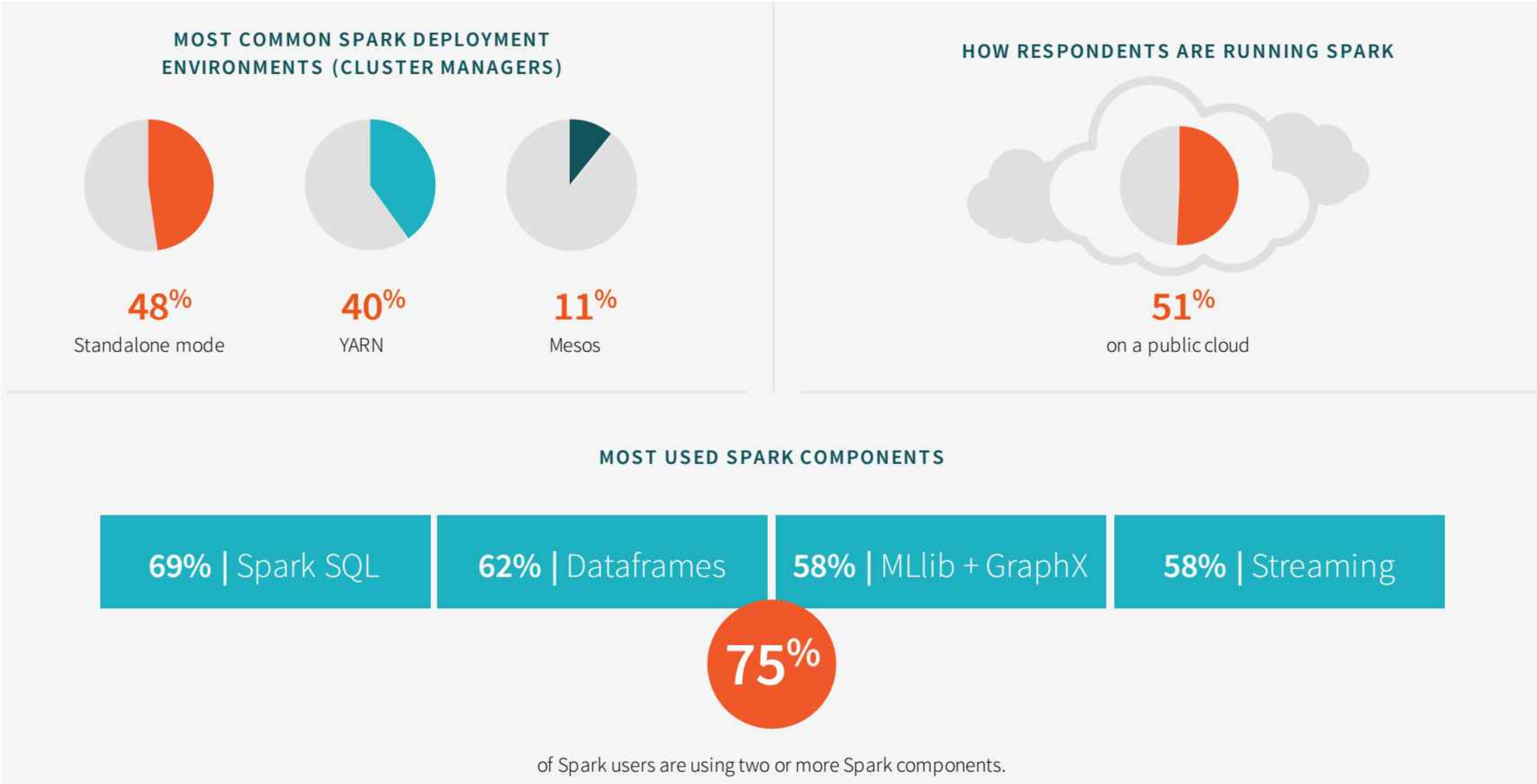
Apache Spark Survey(2015, Databricks)

❑ Product Type



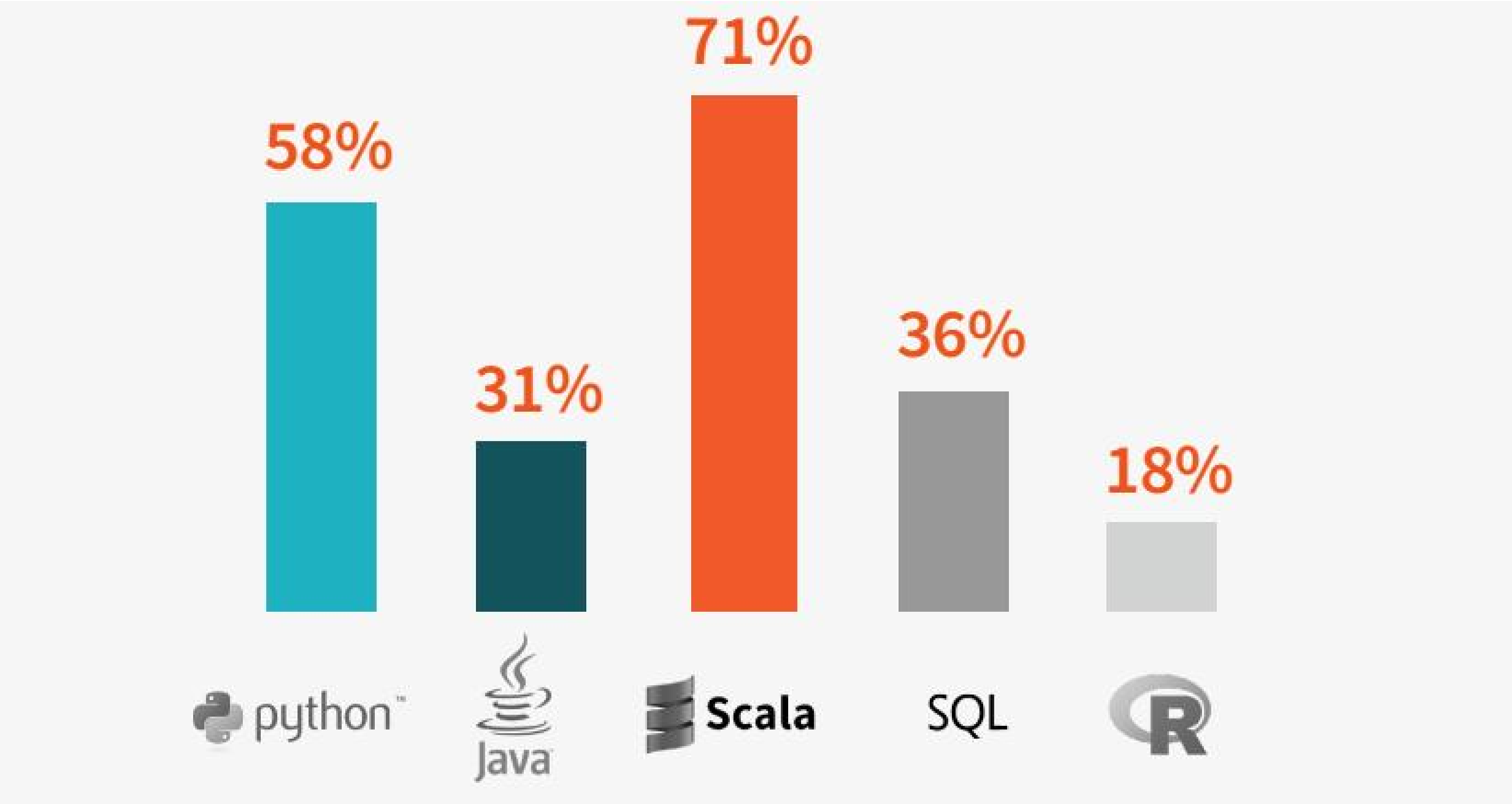
Apache Spark Survey(2015, Databricks)

❑ 사용 형태



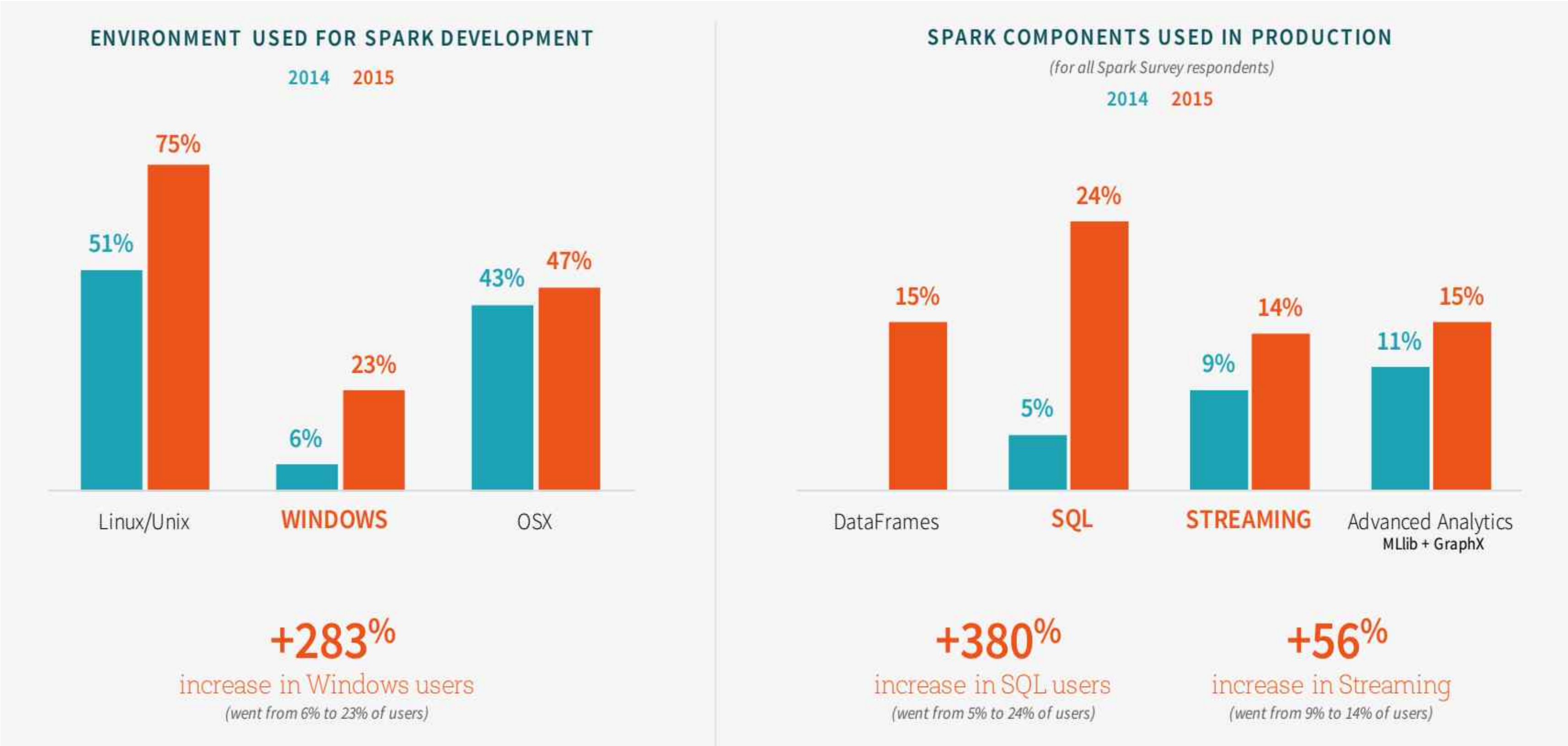
Apache Spark Survey(2015, Databricks)

❑ Programming Language



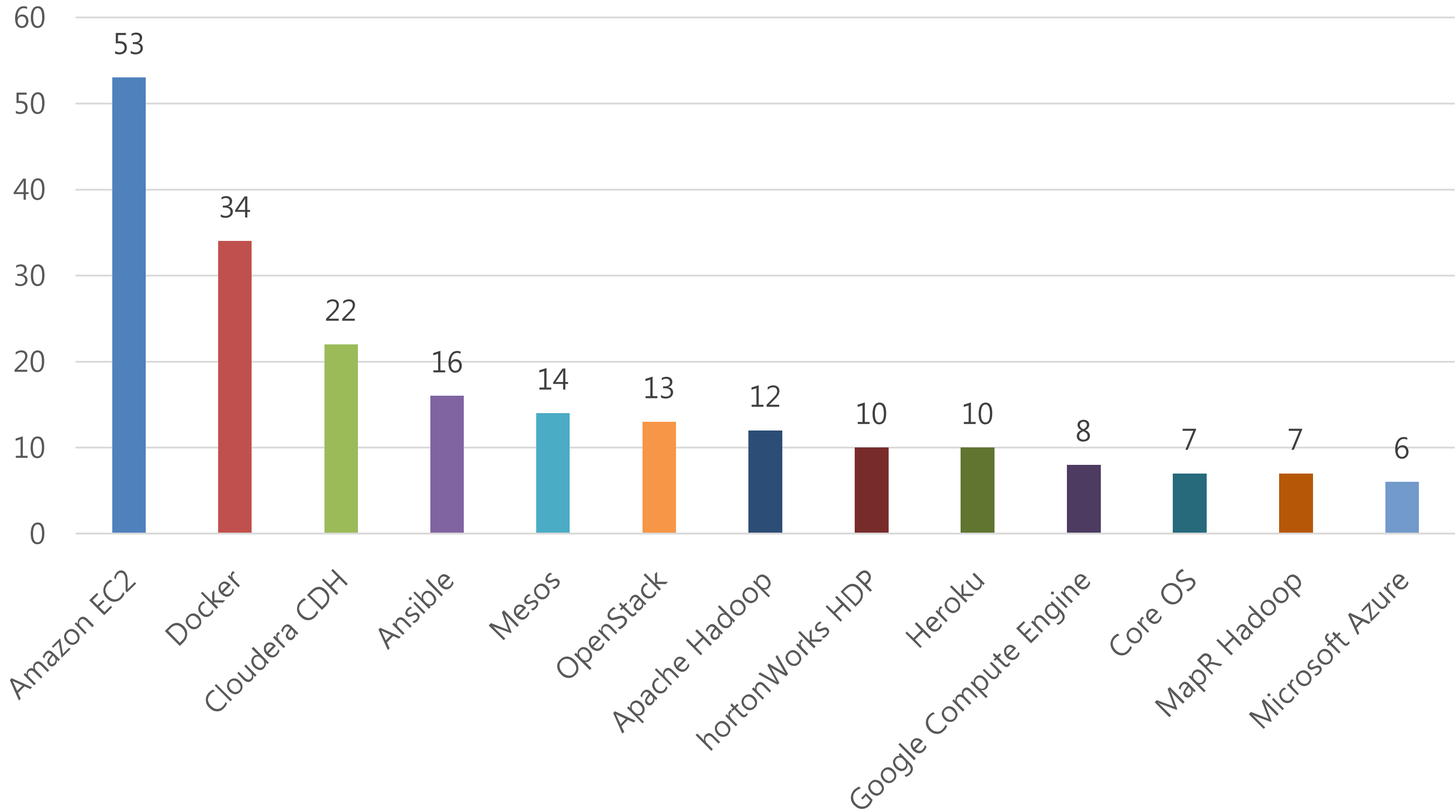
Apache Spark Survey(2015, Databricks)

❑ 2014년 ⟶ 2015년



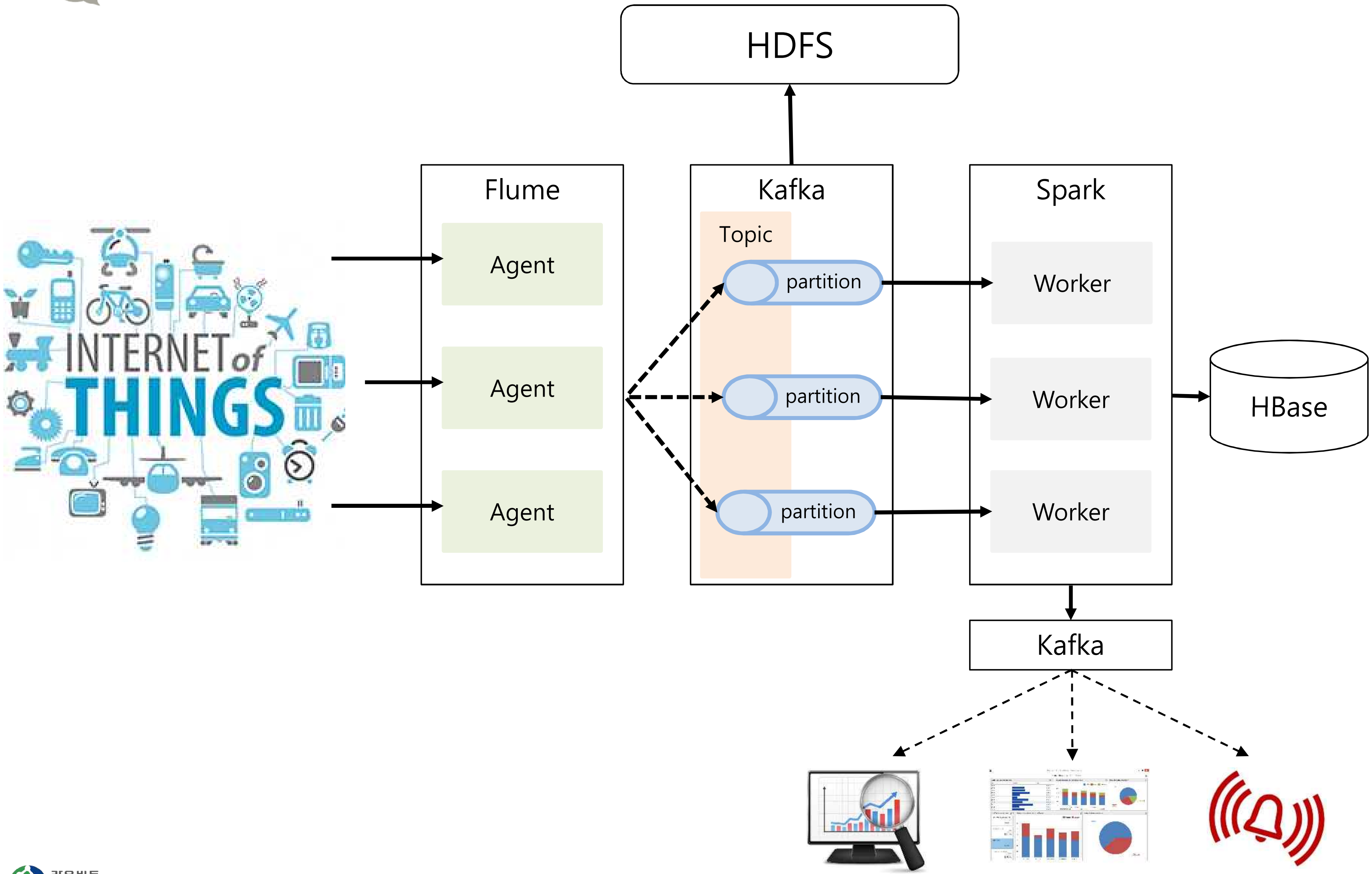
Apache Spark Survey(2014, Typesafe)

❑ Production Infrastructure

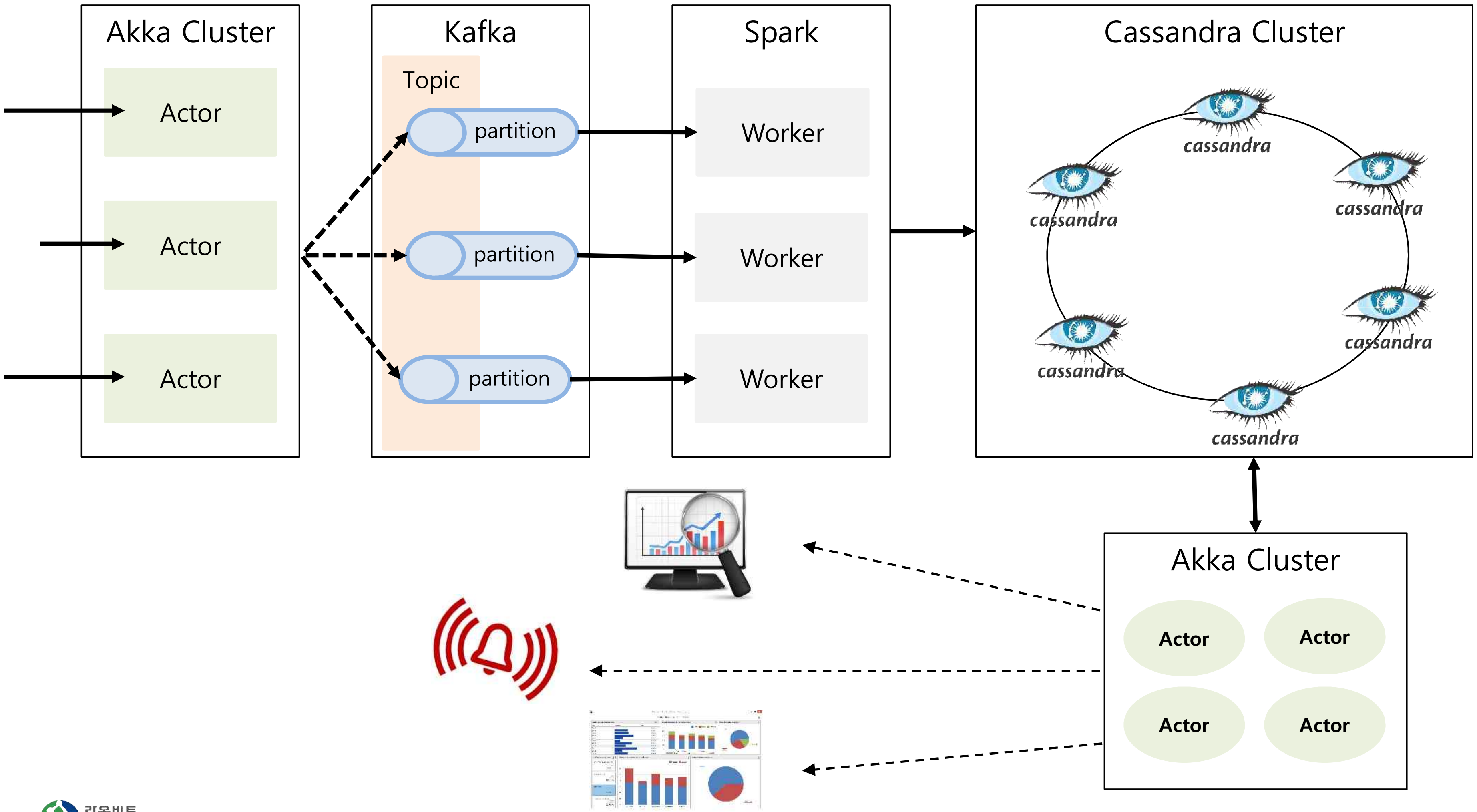


대용량 실시간 처리를 위한 Architecture 구성

Basic Architecture



Raonbit Architecture



Demo

Demo system architecture

Data Source **Spark streaming** **Spark SQL** **Pusher** **Daum Map API**

국가교통정보센터
National Transport Information Center

- 고속도로 공사정보
- 일반국도 공사정보
- 고속도로 사고정보
- 일반국도 사고정보
- 도로구간 평균속도

❑ 사용자 정의 입력 소스

- 공사/사고 정보 입력

REST Stream

\+

REST Stream

\+

REST Stream

\+ union

TCP Stream

Data Handling

Integrated Stream

source : https://bitbucket.org/raonbit/sparkstreamingdemo
plot demo : https://plot.ly/~kyhleem/219/streamtest-data2/

THANK YOU!

Contact

- 대표번호 070-4116-2876
- FAX 070-8838-2876
- E-Mail jinsu.park@raonbit.com
- Homepage http://www.raonbit.com