

Apache Spark

유현석

Apache Spark 란?

- Lighting-fast unified analytics engine(빛처럼 빠른 통합 분석 엔진)
- 대규모 데이터 처리를 위한 통합 분석엔진
- Hadoop과 비교하면 Spark가 100배 빠르다.
- Hadoop은 disk에서 분산 데이터 처리(Distributed File System)
- Spark는 메모리에서 분산 데이터 처리(Distributed File System)
- Hadoop/Spark 둘다 병렬처리 구조



Apache Spark™ is a unified analytics engine for large-scale data processing.

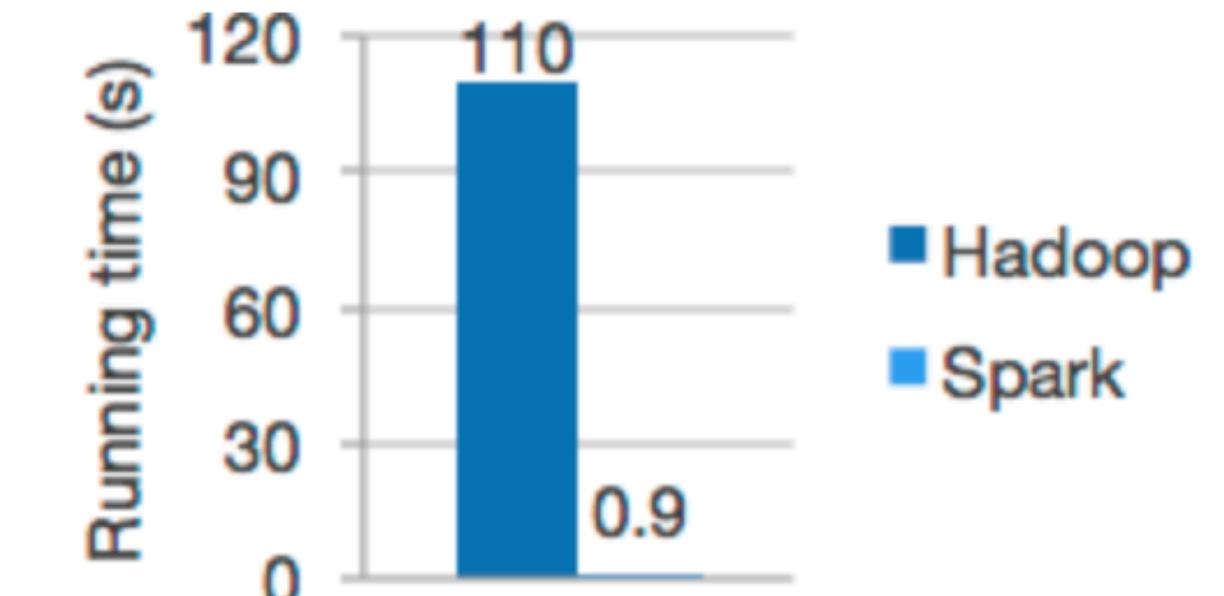
Apache Spark 란?

- <https://spark.apache.org/>

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

Ease of Use

Write applications quickly in Java, Scala, Python, R, and SQL.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

```
df = spark.read.json("logs.json")
df.where("age > 21")
    .select("name.first").show()
```

Spark's Python DataFrame API

Read JSON files with automatic schema inference

Apache Spark 란?

- 메인 언어는 Scala
- 여러가지 언어로 빠르게 적용할 수 있다.
 - Java, Scala, Python, R, and SQL
- Scala를 사용하는 것이 좀 더 빠르게 결과를 확인할 수 있다.

Apache Spark의 철학

- 통합
 - 빅데이터 애플리케이션 개발에 필요한 통합 플랫폼을 제공하자.
 - 간단한 데이터 읽기, SQL, 머신러닝, 스트림 처리를 API로 처리
 - 필요할 경우 Spark기반의 라이브러리를 만들수 있음
 - Spark API의 경우에는 사용자 애플리케이션에서 다른 라이브러리의 기능을 조합해 더 나은 성능을 발휘할 수 있도록 설계되어 있다.
 - 예) SQL 쿼리로 데이터를 읽고 ML 라이브러리로 머신러닝 모델을 평가해야 할 경우, Spark 엔진은 이 두 단계를 하나로 병합하고 데이터를 한번만 조회할 수 있게 해준다.

Apache Spark의 철학

- 컴퓨팅엔진
 - 스파크는 저장소 시스템의 데이터를 연산하는 역할만 수행할 뿐 영구 저장소 역할은 수행하지 않는다.
 - 대신, 분산처리 파일 시스템인 아파치 하둡(Apache Hadoop), 클라우드 기반의 애저 스토리지(Azure Storage), 아마존 S3(Amazon S3), 키/값 저장소인 아파치 카산드라(Apache Cassandra), 메시지 서비스인 아파치 카프카(Apache Kafka)등의 저장소 지원
 - 스파크는 데이터 저장 위치에 상관없이 처리에 집중하도록 만들어짐.
 - 스파크는 하둡 저장소와 잘 호환된다.
 - 하둡 아키텍처를 사용할 수 없는 환경에서도 많이 사용된다.

Apache Spark의 철학

- 라이브러리
 - 스파크 컴포넌트는 데이터 분석 작업에 필요한 통합 API를 제공하는 통합 엔진 기반의 자체 라이브러리이다.
 - 표준 라이브러리 + 외부 패키지
 - 표준라이브러리 : 스파크 엔진에서 제공
 - 외부 패키지 : 오픈소스 커뮤니티에서 서드파티 패키지 형태로 제공하는 라이브러리
 - 표준 라이브러리
 - Spark SQL : SQL과 구조화된 데이터를 제공
 - 머신러닝을 지원하는 MLlib
 - 스트림 처리 기능을 제공하는 스파크 스트리밍과 새롭게 선보인 구조적 스트리밍
 - 그래프 분석엔진인 GraphX 라이브러리를 제공
 - 외부 패키지
 - 다양한 저장소 시스템을 위한 커넥터(connector)
 - 머신러닝을 위한 알고리즘까지 수백개의 외부 오픈소스 라이브러리도 존재한다.

Apache Spark의 역사

- UC 버클리 대학교에서 2009년 스파크 연구 프로젝트로 시작됨.
- *Spark: Cluster Computing with Working Sets* 논문 발표로 알려짐
 - 마테이 자하리아(Matei Zaharia), 모샤라프 카우두리(Mosharaf Chowdhury), 마이클 프랭클린(Michael Franklin), 스콧 쉘커(Scott Shenker), 이온 스토이카(Ion Stoica)
 - 이상 AMPLab 소속
- ○ 문제
 - 첫번째문제: 클러스터 컴퓨팅이 엄청난 잠재력을 가지고 있다는 것, 하지만 경험이 없는 조직은 누군가가 성공하고 난 후에 사용할 수 있다는 것
 - 두번째 문제 : 맵리듀스 엔진을 사용하는 대규모 애플리케이션의 난이도와 효율성
- ○ 해결
 - 함수형 기반 API 설계
 - 연산 단계 사이에서 메모리에 저장된 데이터를 효율적으로 공유할 수 있는 새로운 엔진기반의 API 구현

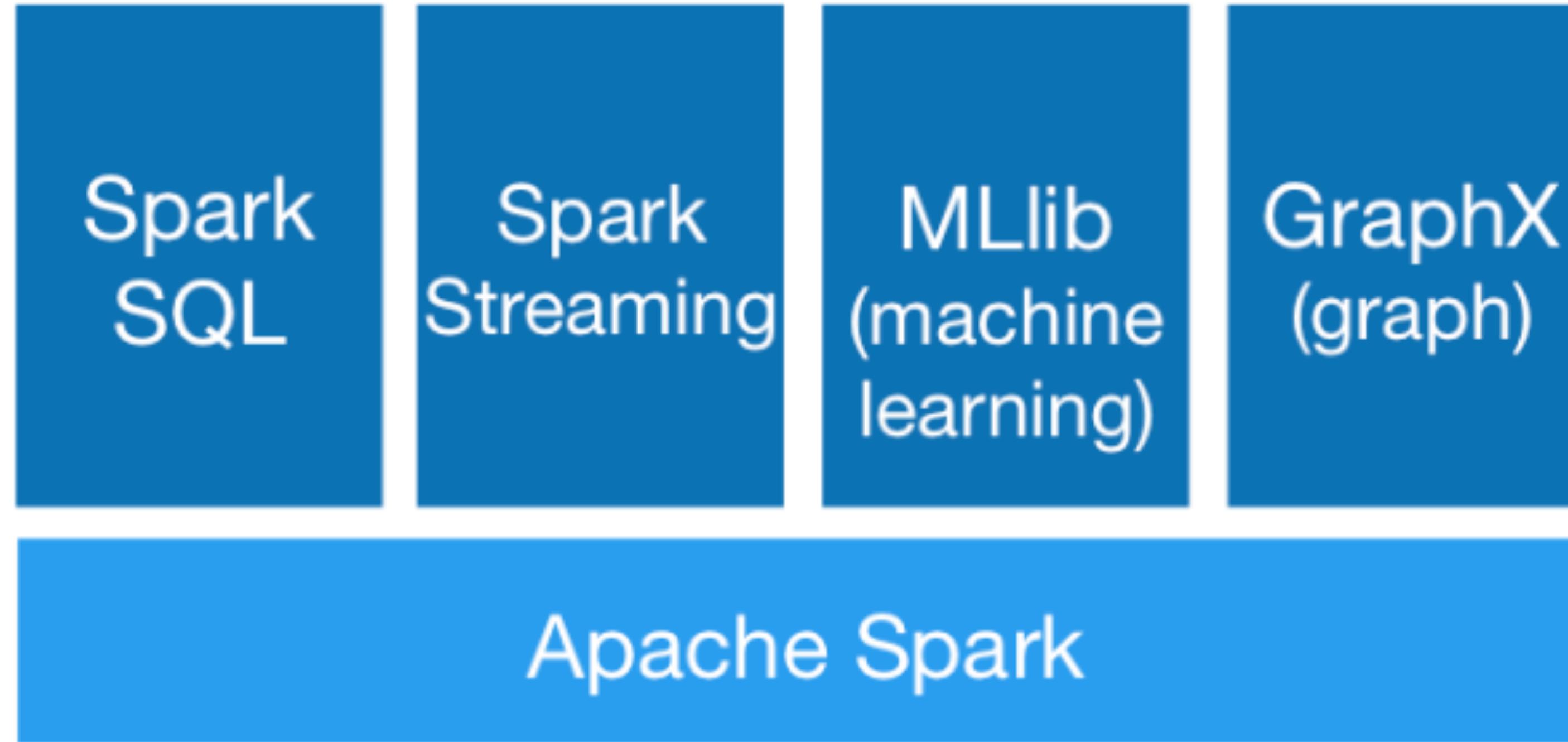
Apache Spark의 역사

- AMPLab팀은 스파크가 특정 업체에 종속되는 것을 막기위해 아파치 재단에 기부
- AMPLab팀은 프로젝트를 성장시키기위해 데이터브릭스를 설립
- 2014년 스파크 1.0 발표(by 아파치 스파크 커뮤니티)
- 2016년 스파크 2.0 발표(by 아파치 스파크 커뮤니티)
- Spark 초기버전 1.0 이전 : 함수형 연산
- Spark 1.0 부터 Spark SQL(구조화된 데이터 기반) 추가
- DataFrame, 머신러닝 파이프라인 그리고 자동 최적화를 수행하는 구조적 스트리밍등 더 강력한 구조체 기반의 신규 API들이 추가됨.

Apache Spark의 현재와 미래

- Spark의 고수준 스트리밍 엔진인 구조적 스트리밍(2016)을 기반으로 스파크의 영역을 꾸준히 넓혀가고 있다.
- 우버, 넷플릭스, NASA, CERN(유럽 입자 물리연구소) 같은 연구소에서 거대한 규모의 데이터 셋을 처리하기 위해 사용
- MIT, 하버드 같은 교육기관에서는 과학적 데이터 분석에 스파크를 사용

Apache Spark Process

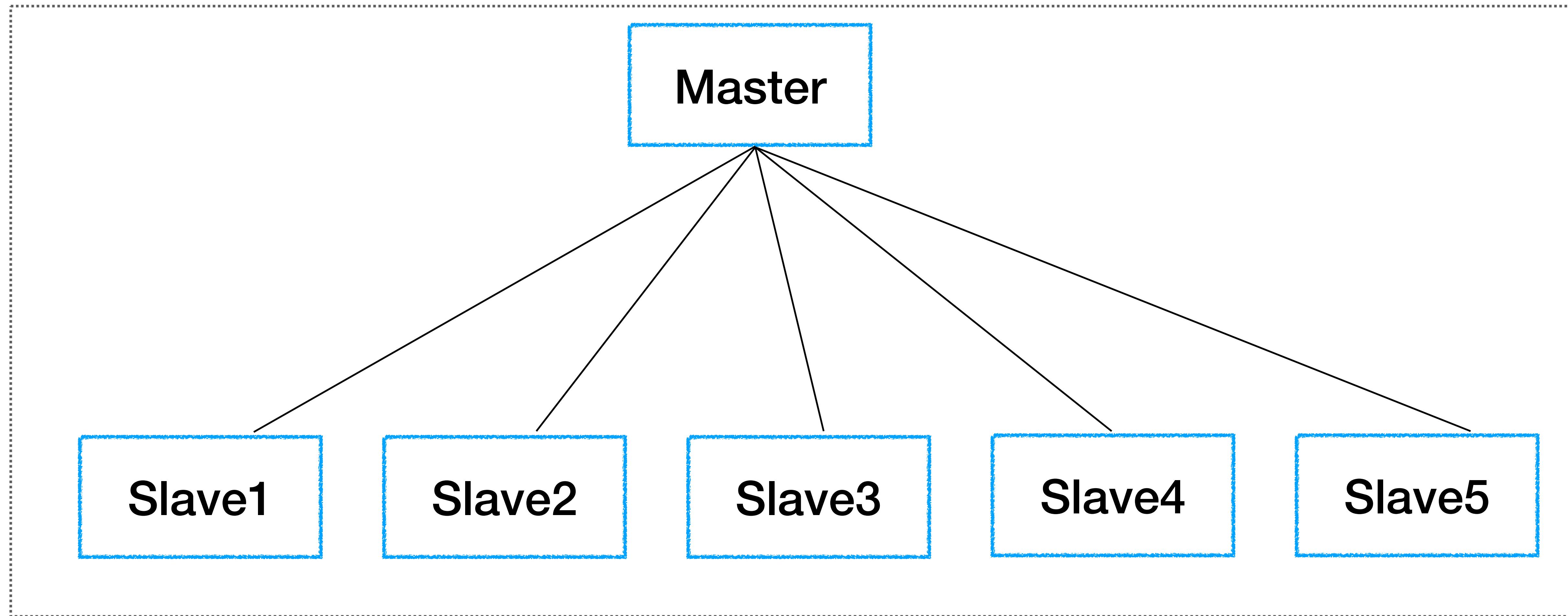


Apache Spark Process

- Spark의 데이터 형태는 RDD, Dataframe, Dataset 가 있다.
- Spark SQL은 RDD, Dataframe, Dataset을 쉽게 다룰 수 있다.
- Spark 위에 MLlib를 설치해서 사용할 수 있다.
- 리소스 매니저
 - Standalone Scheduler, YARN, Mesos 이 있음
 - YARN을 많이 사용하지만 선택해서 사용하면 된다.
 - Standalone일 경우에는 Standalone Scheduler가 관리를 해준다.

Apache Spark Process

Clusters



Apache Spark Process

- clusters : 여러 컴퓨터의 자원을 가지고 하나의 컴퓨터처럼 사용하는 것
- master와 slave 구조로 되어있다.
- 반장을 할 컴퓨터가 필요하다 -> master가 필요하다.
- 클러스터 관련 서비스
 - k8s(kubernetes)
 - Apache Spark
 - Apache Hadoop

Apache Spark 실행하기

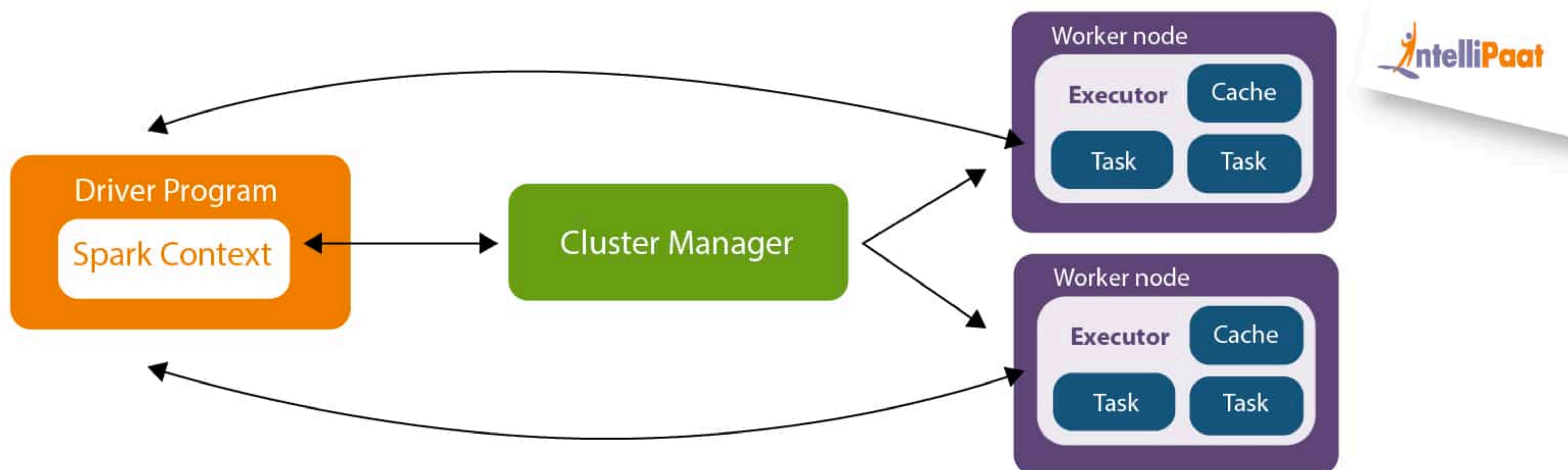
- 파이썬, 자바, 스칼라, R, SQL 언어에서 스파크를 사용할 수 있다.
- 스파크는 스칼라로 구현되어 자바 가상 머신 기반으로 동작한다.
- 따라서 노트북(Notebook) or 클러스터 환경에서 스파크를 실행하려면 자바를 설치해야 한다.
- 파이썬 API를 사용하려면, python 2.7버전 이상을 설치해야 한다.
- R을 사용하려면 R을 컴퓨터에 설치해야 한다.
- 스파크를 시작하는 두 가지 방법
 - Apache Spark를 노트북에 내려받아 설치
 - Spark 학습용 무료 클라우드 환경 웹 기반의 데이터 브릭스 커뮤니티

Apache Spark의 기본 아키텍처

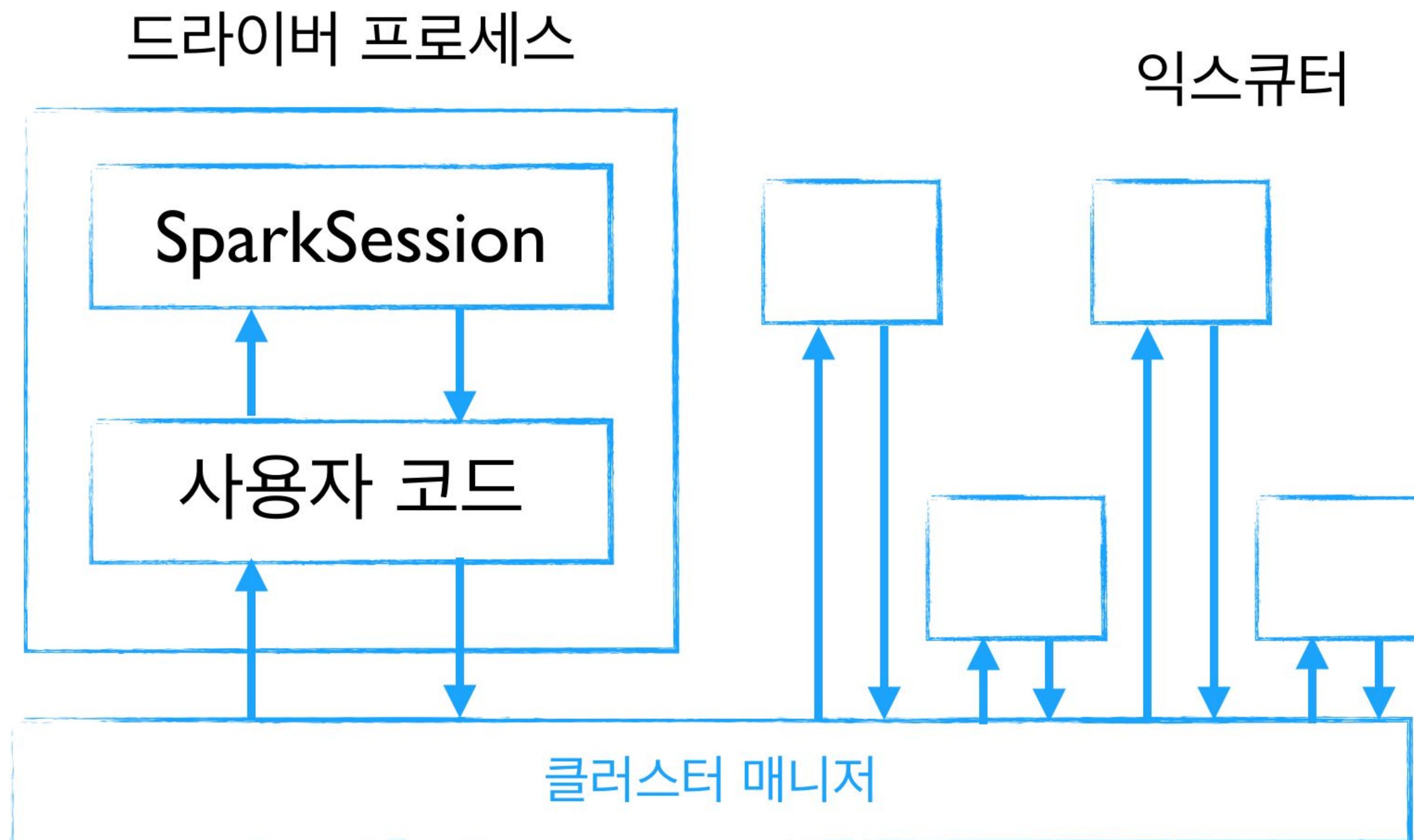
- 스파크는 데이터의 클러스터의 데이터 처리 작업을 관리하고 조율하는 프레임워크
- 클러스터: 여러 컴퓨터의 자원을 모아 하나의 컴퓨터처럼 사용할 수 있게 만들어 놓은 것
- 스파크 스탠드얼론(standalone) 클러스터 매니저, 하둡 YARN, 메소스(Mesos) 같은 클러스터
- 매니저에서 스파크 클러스터를 관리한다.
- 사용자-> 스파크 애플리케이션 제출 -> 클러스터 매니저 -> 자원 할당 -> 작업처리

Apache Spark의 기본 아키텍처

- <https://intellipaat.com/blog/tutorial/spark-tutorial/spark-architecture/>



Apache Spark Application Architecture



Apache Spark Application Architecture

- 클러스터 매니저가 물리적 머신을 관리하고, 스파크 애플리케이션에 자원을 할당
- 클러스터 매니저는 스파크 스탠드얼론 클러스터 매니저, 하둡 YARN, 메소스 중 하나 선택 가능
- 하나의 클러스터에서 여러 개의 스파크 애플리케이션을 실행 가능
- 위 그림에서는 클러스터 노드의 개념은 나타내지 않음
- 사용자는 각 노드에 할당할 익스큐터 수를 지정할 수 있다.

- 스파크는 클러스터 모드와 로컬 모드를 지원한다.
- 드라이버와 익스큐터는 단순 프로세스이므로 같은 머신이나 서로 다른 머신에서 실행 가능함.
- 단, 로컬 모드에서 실행을 하면 단일 머신에서 스레드 형태로 실행
- 드라이버는 스파크의 언어 API를 통해 다양한 언어로 실행 가능

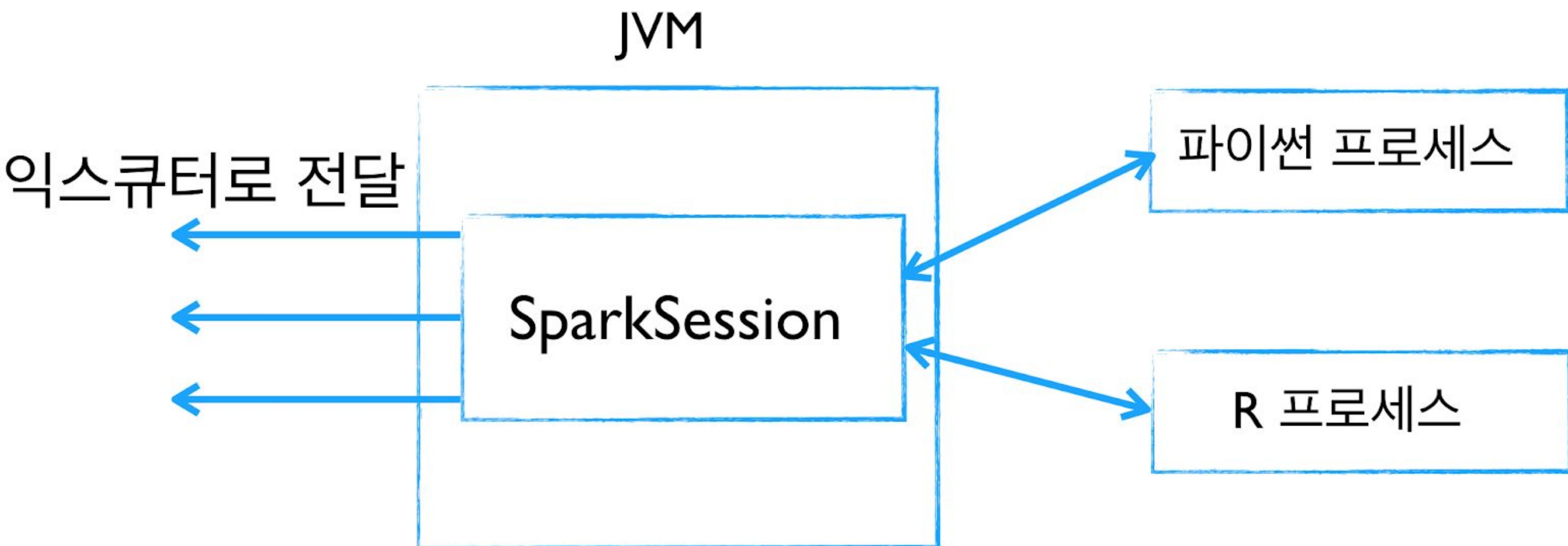
Apache Spark Application Architecture의 핵심

- 사용 가능한 자원을 파악하기 위해 클러스터 매니저를 사용
- 드라이버 프로세스는 주어진 작업을 완료하기 위해 드라이버 프로그램의 명령을 익스큐터에서 실행할 책임이 있다.

Apache Spark Application

- 구성 : 드라이버 프로세스(driver process) 하나, 다수의 익스큐터(executor) 프로세스
- 드라이버 프로세스(driver process)는 클러스터 노드 중 하나에서 실행되며, main() 함수 실행
- 드라이버 프로세스 : 스파크 애플리케이션의 심장
- 익스큐터(executor)
 - 드라이버 프로세스가 할당한 작업을 수행한다.
 - 드라이버가 할당한 코드를 실행하고 진행 상황을 다시 드라이버 노드에 보고 하는 역할

Apache Spark의 다양한 언어 API



Apache Spark의 다양한 언어 API

- 사용자는 스파크 코드를 실행하기 위해 `SparkSession` 객체를 진입점으로 사용할 수 있다.
- 파이썬이나 R로 스파크를 사용할 때는 JVM 코드를 명시적으로 작성하지 않는다.
- 스파크는 사용자를 대신해 파이썬이나 R로 작성한 코드를 익스큐터의 JVM에서 실행할 수 있는 코드로 변환한다.

Apache Spark에서 사용가능한 언어

- 스칼라 : 스파크는 스칼라로 개발되어 있으므로 스칼라가 스파크의 '기본' 언어이다.
- 자바 : 스파크가 스칼라로 되어 있지만, 스파크 창시자들은 자바를 이용해 스파크 코드를 작성할 수 있도록 심혈을 기울였다.
- 파이썬 : 파이썬은 스칼라가 지원하는 거의 모든 구조를 지원한다.
- SQL : ANSI SQL 2003 표준 중 일부를 지원한다. 분석가나 비프로그래머도 SQL을 이용해 스파크의 강력한 빅데이터 처리 능력을 쉽게 활용할 수 있다.
- R : 스파크에는 일반적으로 사용하는 두 개의 R 라이브러리가 있다. 하나는 스파크 코어에 포함된 SparkR이고, 다른 하나는 R 커뮤니티 기반 패키지인 sparklyr이다.

Apache Spark에서 사용가능한 언어

- 스칼라 : 스파크는 스칼라로 개발되어 있으므로 스칼라가 스파크의 '기본' 언어이다.
- 자바 : 스파크가 스칼라로 되어 있지만, 스파크 창시자들은 자바를 이용해 스파크 코드를 작성할 수 있도록 심혈을 기울였다.
- 파이썬 : 파이썬은 스칼라가 지원하는 거의 모든 구조를 지원한다.
- SQL : ANSI SQL 2003 표준 중 일부를 지원한다. 분석가나 비프로그래머도 SQL을 이용해 스파크의 강력한 빅데이터 처리 능력을 쉽게 활용할 수 있다.
- R : 스파크에는 일반적으로 사용하는 두 개의 R 라이브러리가 있다. 하나는 스파크 코어에 포함된 SparkR이고, 다른 하나는 R 커뮤니티 기반 패키지인 sparklyr이다.

Apache Spark API

- 다양한 언어로 스파크를 사용할 수 있는 이유는 스파크가 크게 2가지 API를 제공하기 때문
 - 저수준의 비구조적(unstructured) API
 - 고수준의 구조적(structured) API
- 여기서는 고수준의 구조적 API를 중심으로 진행할 예정임

Apache Spark 시작하기

- `/bin/spark-shell` 명령을 사용해 스칼라 콘솔에 접속할 수 있다.
- 대화형 모드로 스파크를 시작하면 스파크 애플리케이션을 관리하는 `SparkSession`이 자동으로 생성된다.
- 스탠드얼론 애플리케이션으로 스파크를 시작하면 사용자 애플리케이션 코드에서 `SparkSession` 객체를 직접 생성해야 한다.

SparkSession

- 스파크 애플리케이션은 SparkSession이라 불리는 드라이버 프로세스로 제어한다.
 - SparkSession 인스턴스는 사용자가 정의한 처리 명령을 클러스터에서 실행한다.
 - 하나의 SparkSession은 하나의 스파크 애플리케이션에 대응한다.
 - 스칼라나 파이썬에서 SparkSession을 확인하기 위해 다음 코드를 실행해보자.
-
- spark
 - // 스칼라에서 실행
 - res0: org.apache.spark.sql.SparkSession =
org.apache.spark.sql.SparkSession@4751cd3
-
- // 파이썬에서 실행
 - <pyspark.sql.session.SparkSession object at 0x103767bd0>

SparkSession

- (code)
- // 스칼라 코드
- `val myRange = spark.range(1000).toDF("number")`
- // 파이썬 코드
- `myRange = spark.range(1000).toDF("number")`
- 이 코드는 한 개의 컬럼과 1000개의 로우로 구성되며 각 로우에는 0부터 999까지의 값이 할당되어 있다.
- 이 숫자들은 분산 컬렉션을 나타낸다.
- 클러스터 모드에서 실행하면 숫자 범위의 각 부분이 서로 다른 익스큐터에 할당된다.
- 이것이 스파크의 DataFrame이다.

DataFrame(데이터 프레임)

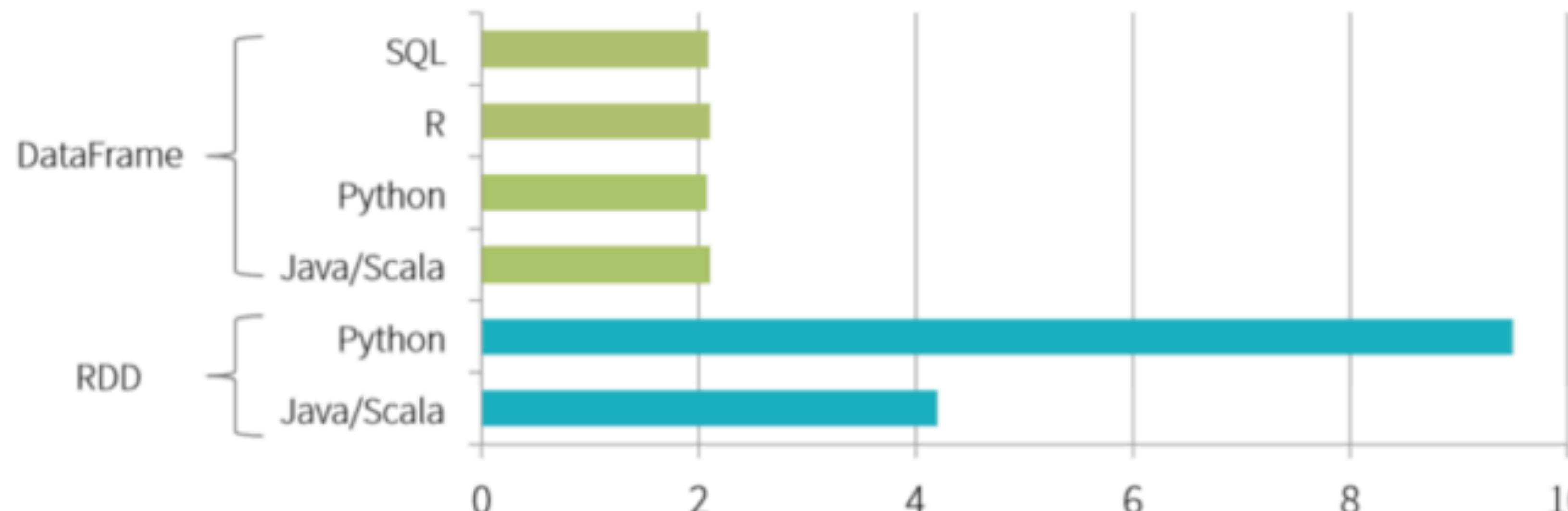
- DataFrame은 가장 대표적인 구조적 API이다.
- DataFrame은 테이블의 데이터를 로우와 컬럼으로 단순하게 표현한다.
- 컬럼과 컬럼의 타입을 정의한 목록을 스키마(Schema)라고 부른다.
- 스프레드 시트는 한 대의 컴퓨터에 있지만, 스파크의 DataFrame은 수천 대의 컴퓨터에 분산되어 있다.
- 여러 컴퓨터에 데이터를 분산하는 이유는? 단일 컴퓨터에 저장하기에는 데이터가 너무 크거나 계산에 너무 오랜 시간이 걸릴 수 있기 때문이다.
- 스파크는 Dataset, DataFrame, SQL 테이블 그리고 RDD라는 몇 가지 핵심 추상화 개념을 가지고 있다.(모두 분산 데이터 모음을 표현)
- 이 중 가장 쉽고 효율적인 DataFrame은 모든 프로그래밍 언어에서 사용할 수 있다.

DataFrame(데이터 프레임)

- SQL에 집중해서 데이터를 처리할 수 있다.
- spark에서도 optimize
- un type safe하기 때문에 데이터를 처리할때는 마음대로 변형이 가능한데, 실제로 결과를 도출해서 저장하거나 할 경우 type을 지정해줘야 해서 번거러워 질수 있다.

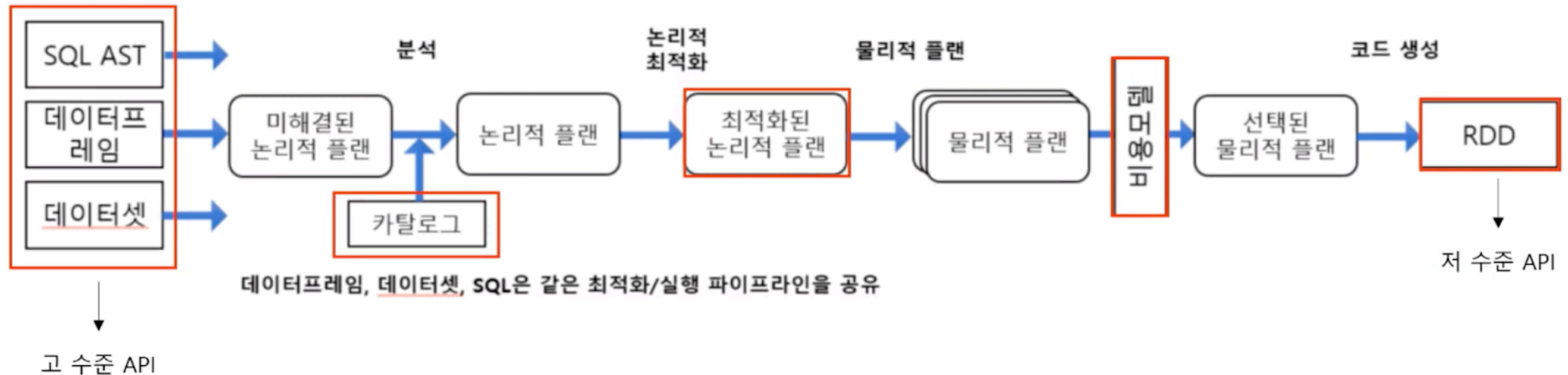
DatFrame(데이터 프레임)

Benefit of Logical Plan: Performance Parity Across Languages



DataFrame(데이터 프레임)

- Plan Optimization & Execution

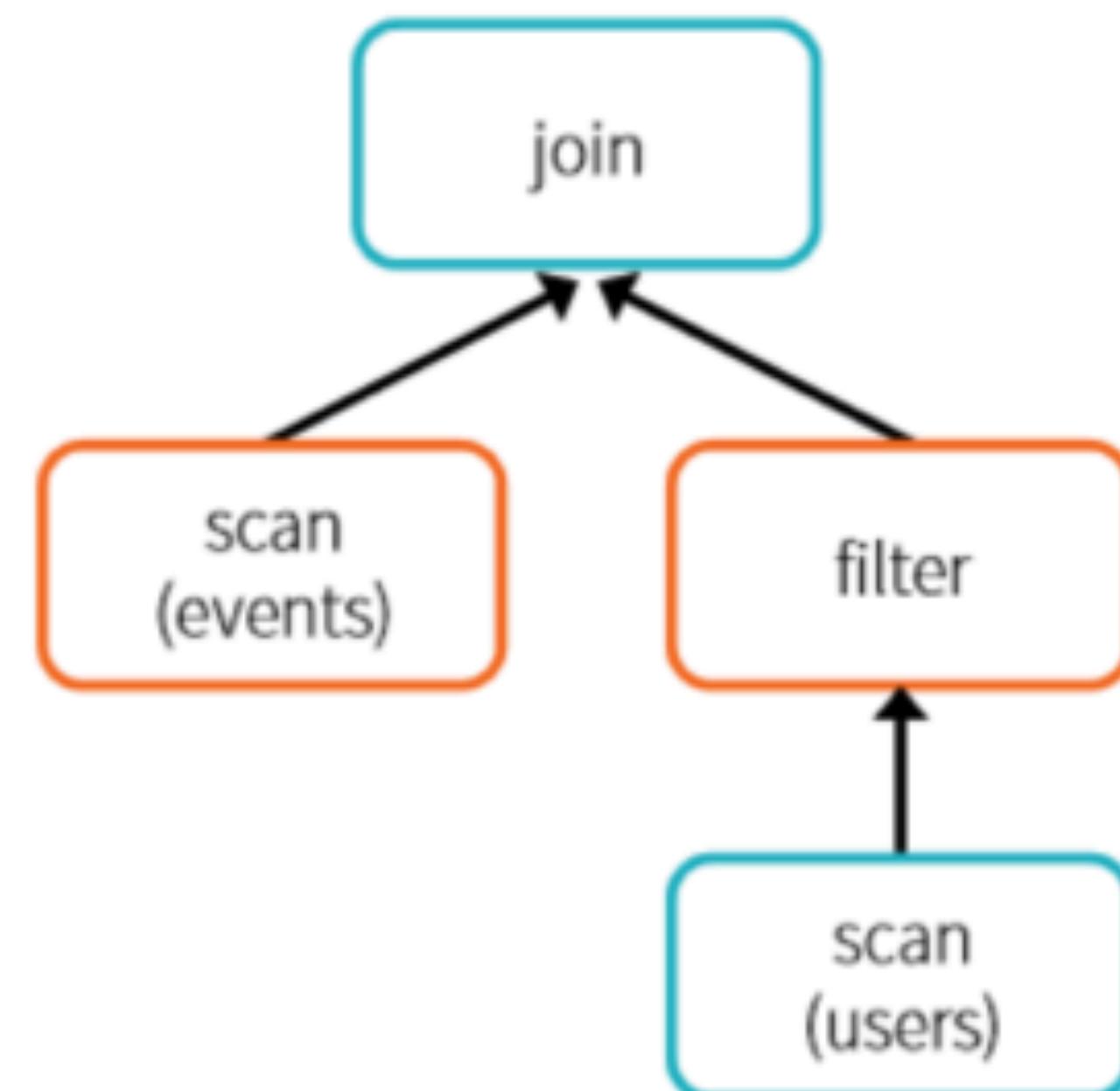


DatFrame(데이터 프레임)

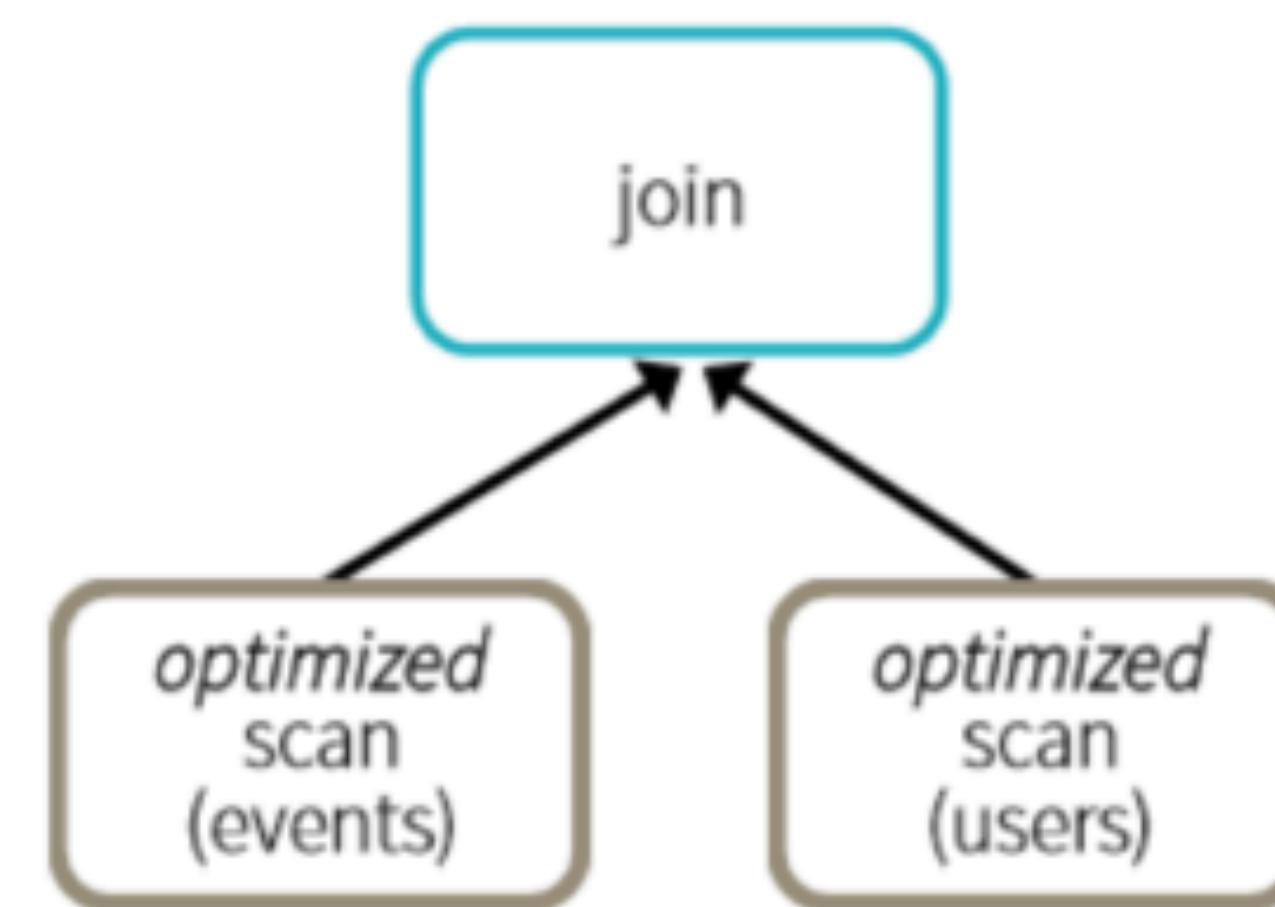
Logical Plan



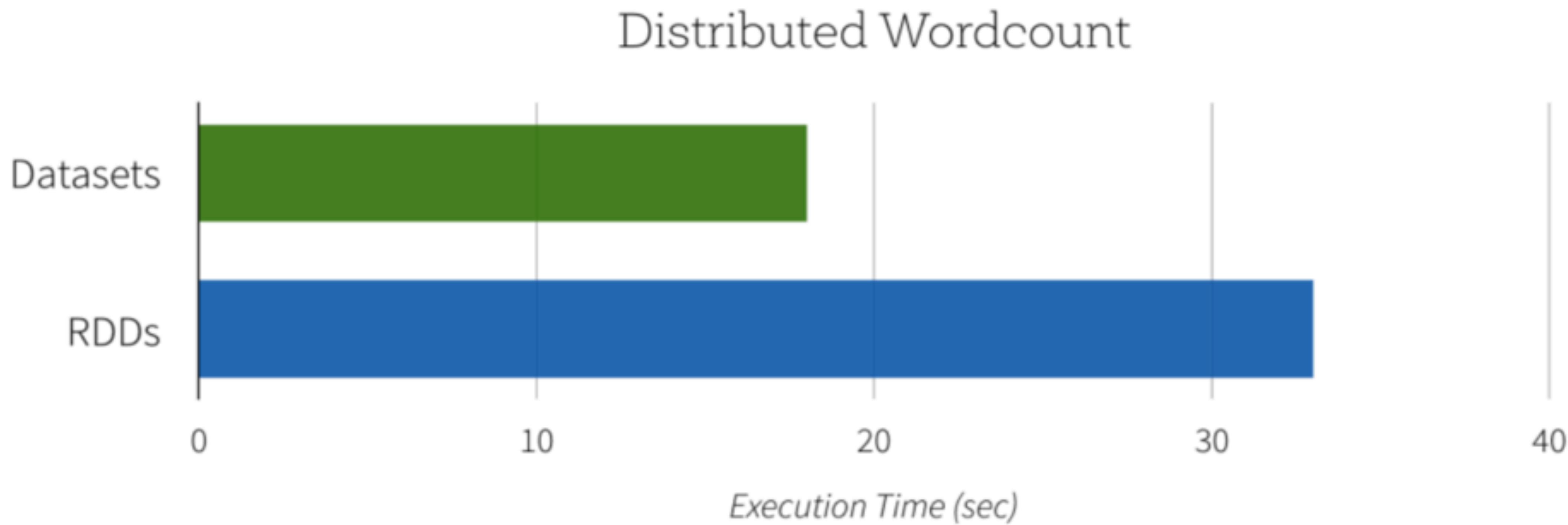
Physical Plan



Physical Plan
with Predicate Pushdown
and Column Pruning

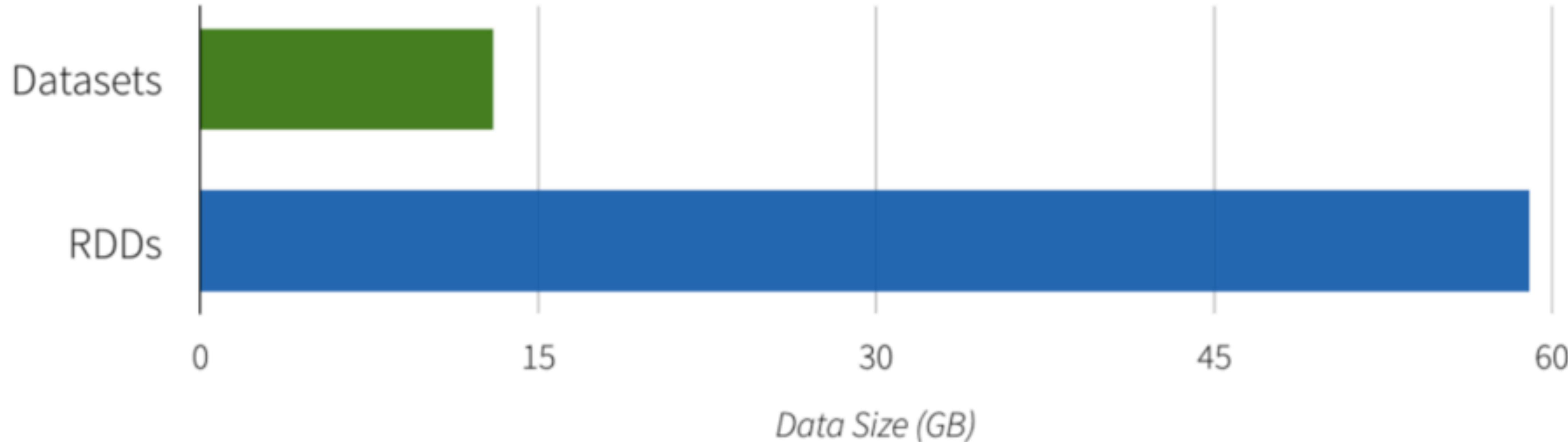


DatFrame(데이터 프레임)



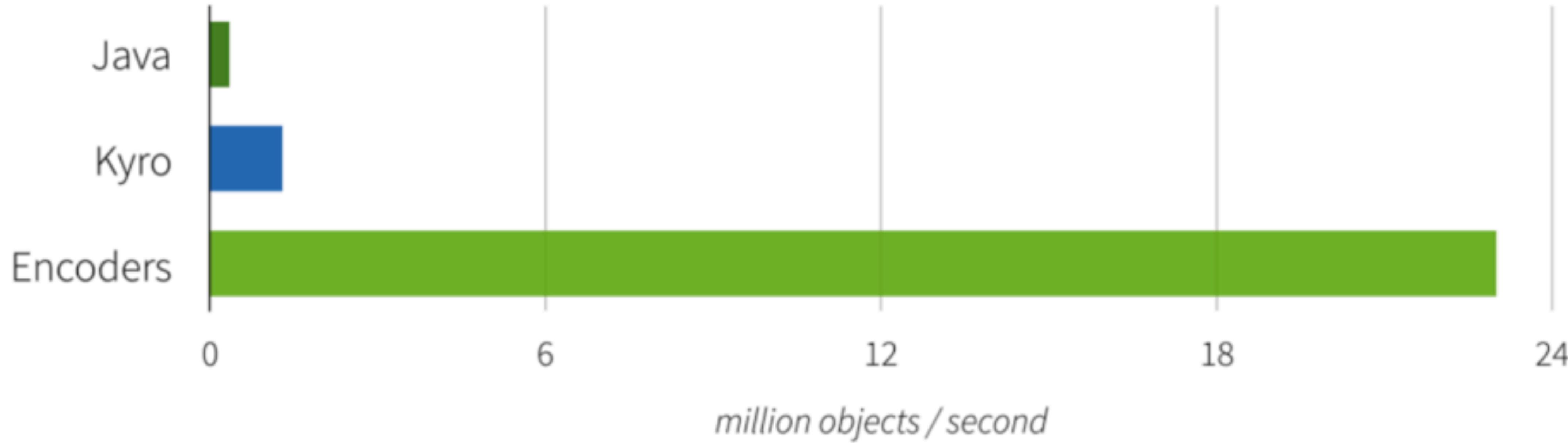
DatFrame(데이터 프레임)

Memory Usage when Caching



DataFrame(데이터 프레임)

Serialization / Deserialization Performance



DataFrame/SQL

- `createOrReplaceTempView`를 사용해서 DataFrame을 테이블이나 뷰로 만들어 보자.
 - `flightData2015.createOrReplaceTempView("flight_data_2015")`
- SQL로 데이터를 처리해보기
- 의미는 같은데, 구현과 정렬 방식이 다른 DataFrame구문을 살펴보자.
- 실행계획은 SQL에서 한 것과 같습니다.
- 실행계획은 트랜스포메이션의 지향성 비순환 그래프(Directed Acyclic Graph, DAG)이며, 액션이 호출되면 결과를 만들어 낸다.
- DAG의 각 단계는 불변성을 가진 신규 DataFrame 이다.

DataFrame/SQL

- DataFrame의 변환 흐름
 - csv -> read -> dataframe
 - -> group by -> dataframe group
 - -> sum -> dataframe
 - -> 컬럼명 변경 -> dataframe
 - -> sort -> dataframe
 - -> limit -> dataframe
 - -> collect -> Array(...)

DataFrame/SQL

- (소스코드-scala)
- import org.apache.spark.sql.functions.desc
- (flightData2015
 - .groupBy("DEST_COUNTRY_NAME")
 - .sum("count")
 - .withColumnRenamed("sum(count)", "destination_total")
 - .sort(desc("destination_total"))
 - .limit(5)
 - .show()

Dataset

- 타입 안정성을 제공해 주는 구조적 API
- Java와 스칼라의 정적 데이터 타입에 맞는 코드
- 타입 안정성을 지원하며 동적 타입 언어인 파이썬과 R에서는 사용할 수 없다.
- Dataset의 API는 DataFrame의 레코드를 사용자가 자바나 스칼라로 정의한 클래스에 할당하고 자바의 ArrayList 또는 스칼라의 Seq 객체 등의 고정 타입형 컬렉션으로 다룰 수 있는 기능을 제공한다.

Dataset

- Dataset API는 초기화에 사용한 클래스 대신 다른 클래스를 사용해 접근할 수 없다.
- (타입 안정성을 보장해 주기때문에)
- Dataset 클래스는 내부 객체의 데이터 타입을 매개변수로 사용한다.
- Java : Dataset<T> , 스칼라 : Dataset[T]
- collect 메소드나 take 메소드를 호출하면 DataFrame을 구성하는 Row 타입의 객체가 아닌 Dataset에 매개변수로 지정한 타입의 객체를 반환한다.
- 하여, 코드 변경없이 타입 안정성을 보장할 수 있다.
- 하여, 로컬이나 분산 클러스터 환경에서 데이터를 안전하게 다룰 수 있다.

구조적 스트리밍

- 스파크 2.2버전에서 안정화된 고수준 API
- 구조적 스트리밍을 사용하면 구조적 API로 개발된 배치 모드의 연산을 스트리밍 방식으로 실행할 수 있으며, 자연 시간을 줄일 수 있다.
- 구조적 스트리밍은 배치처리용 코드를 일부 수정하여 스트리밍 처리를 수행하고 값을 빠르게 얻을 수 있다는 장점이 있다.

Partition(파티션)

- 스파크는 모든 익스큐터가 병렬로 작업을 수행할 수 있도록 파티션이라 불리는 청크 단위로 데이터를 분할한다.
- 파티션은 클러스터의 물리적 머신에 존재하는 로우의 집합을 의미한다.
- DataFrame의 파티션은 실행중에 데이터가 컴퓨터 클러스터에서 물리적으로 분산되는 방식을 나타낸다.
- 파티션이 하나라면, 스파크에 수천 개의 익스큐터가 있더라도 병렬성은 1이 된다.
- 또한 수백개의 파티션이 있더라도 익스큐터가 하나밖에 없다면 병렬성은 1이 된다.
- DataFrame을 사용하면 파티션을 수동 혹은 개별적으로 처리할 필요가 없다.
- 물리적 파티션에 데이터 변환용 함수를 지정하면 스파크가 실제 처리 방법을 결정한다.
- RDD 인터페이스를 이용하는 저수준 API 역시 제공된다.

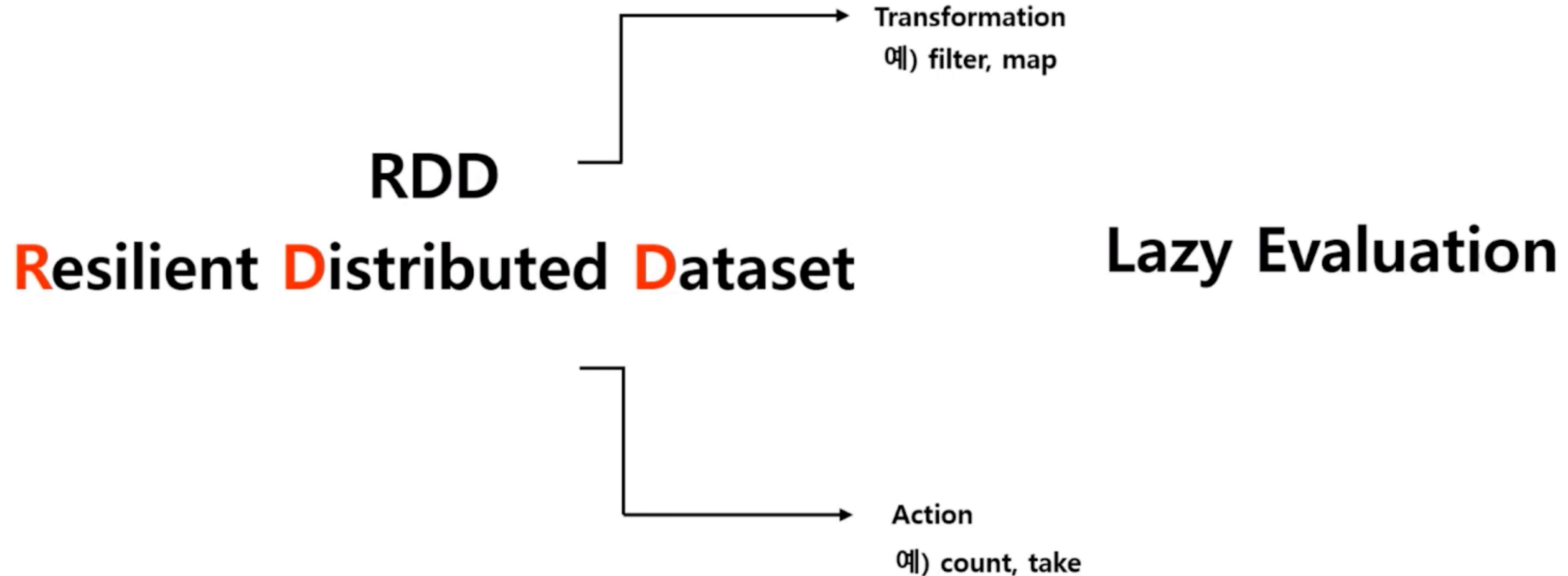
Spark 용어

- driver program : 프로그래밍 언어의 main()함수와 같은 역할
- SparkSession : 어떻게 스파크 클러스터에 접근할 수 있는지를 알려주는 Object or session
- cluster manager : 클러스터에 필요한 자원들을 찾아줌
- worker node : 실제 작업을 수행하는 노드
- job : 사용자 입장에서의 작업의 단위(task의 조합)
- task : executor에 할당되는 작업의 단위
- executor : task를 수행하는 프로세스

RDD(Resilient Distributed Datatsets)

- immutable collection
 - parallel processing
 - fault tolerant
 - 데이터 형이 정해져 있음
 - type safe
-
- 병렬로 처리가 된다.
 - data -> worker node 1 data
 - -> worker node 2 data
 - -> worker node 3 data

RDD(Resilient Distributed Datatsets)



RDD(Resilient Distributed Datasets)

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

RDD(Resilient Distributed Datatsets)

- RDD의 장점
 - type safety => compile time check
 - low level => fine tuning
 - (if 본인이 전문가 일경우, 구루일 경우)
- RDD의 단점
 - 느리다.(다른언어로 접근했을때)

Transformation

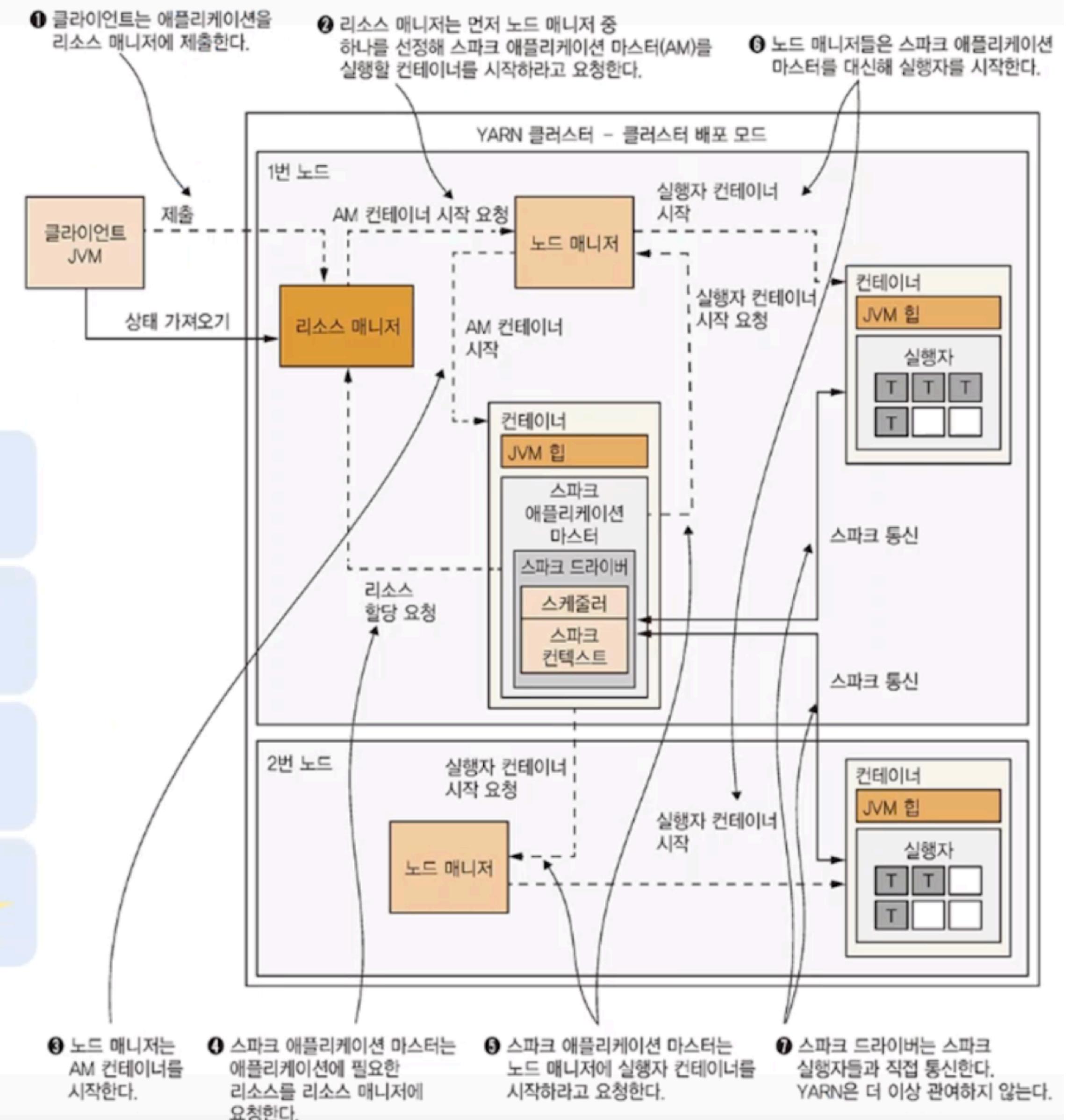
- 데이터를 내가 원하는 모양으로 조각하는 것
- transformation의 결과는 RDD
- `val learnKingText = spark.sparkContext.textFile("/data/pg1532.txt")`
- job -> 아무것도 없음 , 실제 실행이 안됨.
- lazy 된다.

Action

- `learKingText.count()`
- 조작한 데이터를 바탕으로 결과를 얻음
- Spark UI : `http://localhost:4040 -> job`
- node 1(머신 1) - driver program == job

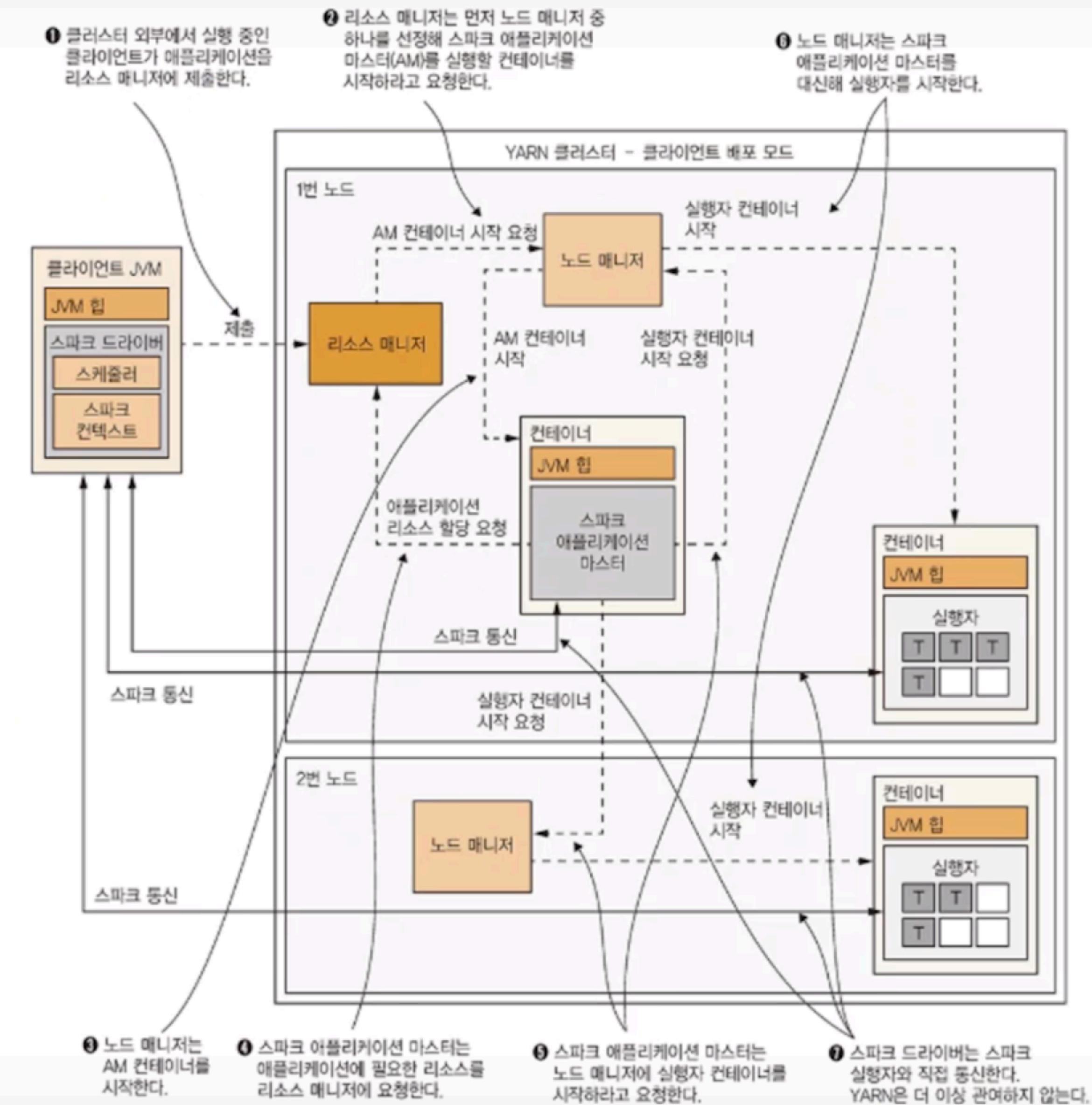
Cluster mode

Cluster Configuration on Spark – cluster mode



Client mode

Cluster Configuration on Spark – client mode



Cluster mode

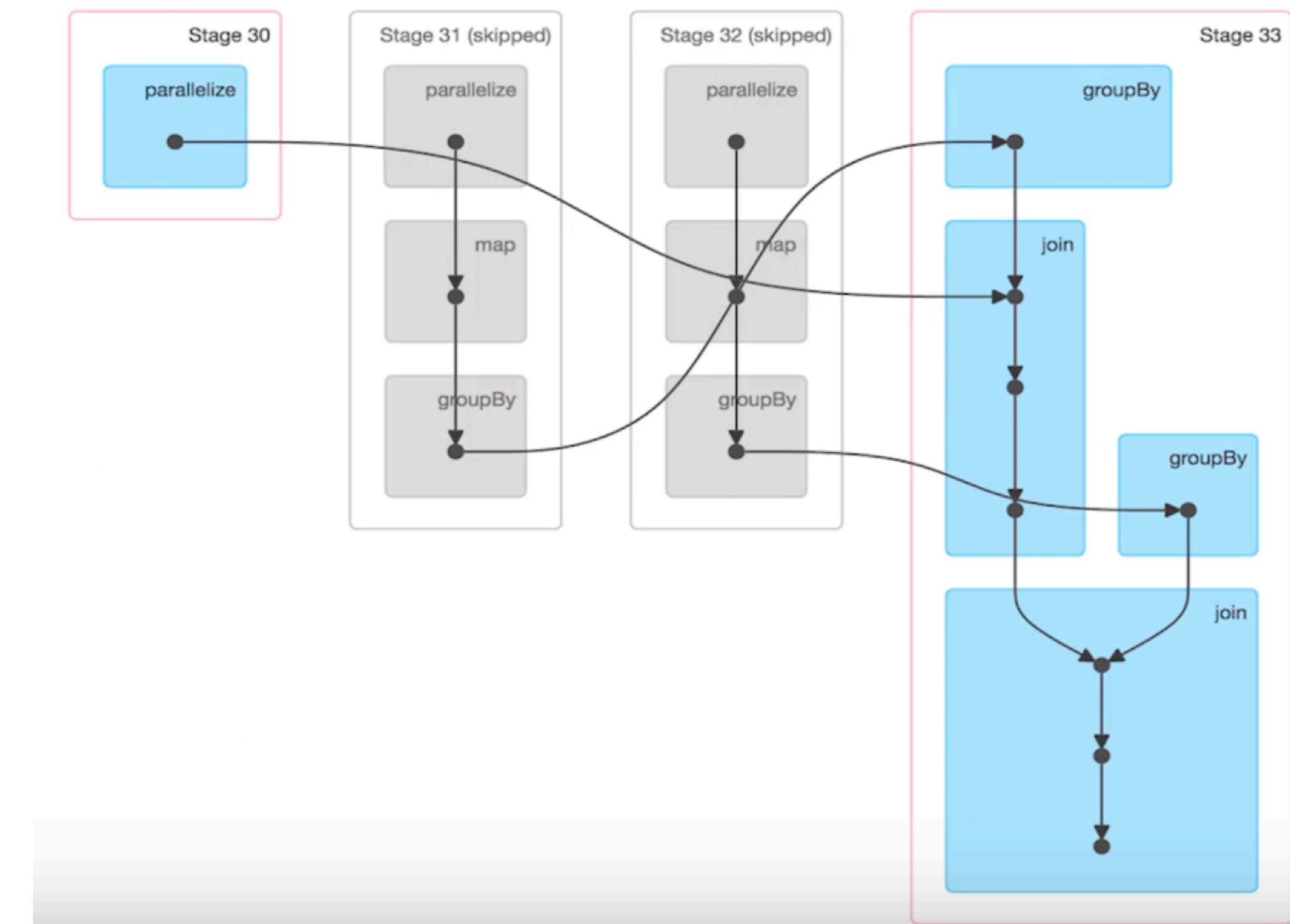
Cluster Configuration on Spark – cluster mode



Details for Job 18

Status: SUCCEEDED
Completed Stages: 2
Skipped Stages: 2

- ▶ Event Timeline
- ▼ DAG Visualization



Apache Spark Dataframe 실습 |

- Zeppelin 에 연동해서 실습할것

The screenshot shows the Zeppelin web interface at localhost:8080/#/. The top navigation bar includes links for '개인', '강의', '디자인', '소프트웨어 캠퍼스', '기타2', 'JS Bin - Collaborati...', '빅데이터', and '(19) (기계 학습, Mac...'. The main header features the Zeppelin logo and navigation tabs for 'Notebook' and 'Job'. A search bar is on the right. The main content area displays a large 'Welcome to Zeppelin!' message, followed by a description: 'Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!'. Below this, there are sections for 'Notebook' (with 'Import note' and 'Create new note' buttons), 'Help' (link to documentation), 'Community' (links to 'Mailing list', 'Issues tracking', and 'Github'), and a sidebar with a 'Filter' input and a list of notebooks: 'Spark-The-Definitive-Guide-Dependency', 'Spark-The-Definitive-Guide-Java', 'Spark-The-Definitive-Guide-Python' (with edit and delete icons), 'Spark-The-Definitive-Guide-R' (with edit and delete icons), 'Spark-The-Definitive-Guide-Scala', 'Spark-The-Definitive-Guide-Sql', 'Zeppelin Tutorial', and 'Trash'.

Apache Spark Dataframe 실습 I

- Zeppelin 환경 구축
- Docker
 - 컨테이너 기반의 오픈소스 가상화 플랫폼
 - 컨테이너는 HostOS 위에 구성된 고립된 환경을 의미한다.
 - Host OS 환경과 다른 컨테이너의 환경에 영향을 받지 않는 OS 공간이다.

Apache Spark Dataframe 실습 1

- Docker 설치하기
 - 먼저 <https://hub.docker.com/> 에 가입을 한다.
 - google 검색 : docker download
 - window 10 Home 의경우 : docker toolbox download
 - download후 클릭해서 설치하면 된다.

Apache Spark Dataframe 실습 I

- Docker 이미지 실행하기
 - window 10 Home의 경우 docker terminal에서 아래 명령어 실행
 - window professional 일경우 docker 실행후 cmd창에서 실행
 - Docker 이미지 내려 받기
 - docker pull docker.io/rheorl08/spark_the_definitive_guide_practice
 - Docker 이미지 실행 하기
- docker run -p 8080:8080 -p 4040:4040 rheorl08/spark_the_definitive_guide_practice
- 웹 브라우저를 통해 제플린 UI에 접속하기
- <http://localhost:8080>

Apache Spark Dataframe 실습 |

- http://localhost:8080

The screenshot shows the Zeppelin web-based notebook interface at <http://localhost:8080>. The browser address bar shows the URL. The Zeppelin header includes a logo, the word "Zeppelin", and navigation links for "Notebook" and "Job". A search bar is also present. The main content area features a large, stylized blue feather graphic on the right. The left sidebar contains a "Notebook" section with "Import note" and "Create new note" buttons, and a "Filter" input field. Below these are several notebook entries with icons for edit and delete:

- Spark-The-Definitive-Guide-Dependency
- Spark-The-Definitive-Guide-Java
- Spark-The-Definitive-Guide-Python + ⚒
- Spark-The-Definitive-Guide-R + ⚒
- Spark-The-Definitive-Guide-Scala
- Spark-The-Definitive-Guide-Sql
- Zeppelin Tutorial

The main content area also includes sections for "Help" (link to documentation) and "Community" (links to Mailing list, Issues tracking, and Github).

Apache Spark Dataframe 실습 1

- Zeppelin 에 연동해서 실습할것
- sc : spark context가 sc로 자동으로 등록되어 있는 것을 볼 수 있다.

Apache Spark Dataframe 실습 I

- [실습I] 실습 예제에 있는 파일을 Dataframe으로 변환해보자.
- val data = sc.textFile("/data/flight-data/csv/2015-summary.csv")
- 데이터를 RDD 형태로 불러왔다.
- 데이터 10개를 불러와 보자.

```
| spark
```

```
res65: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@4ff9b516
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:38:24 AM.

```
| sc
```

```
res67: org.apache.spark.SparkContext = org.apache.spark.SparkContext@7eb3311
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:38:26 AM.

```
| val data = sc.textFile("/data/flight-data/csv/2015-summary.csv")
```

```
data: org.apache.spark.rdd.RDD[String] = /data/flight-data/csv/2015-summary.csv MapPartitionsRDD[83] at textFile at <console>:25
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:38:29 AM.

```
| data.take(10)
```

```
res70: Array[String] = Array(DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count, United States,Romania,15, United States,Croatia,1, United States,India,62, United States,Singapore,1, United States,Grenada,62, Costa Rica,United States,588, Senegal,United States,40)
```

Took 1 sec. Last updated by anonymous at February 10 2020, 6:38:37 AM.

Apache Spark Dataframe 실습 |

- 데이터를 보기 좋게 출력해보고, 데이터의 모양을 보고 나눠보고, DataFrame으로 만들어 보자

```
data.take(10).foreach{println}
```

```
DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count
United States,Romania,15
United States,Croatia,1
United States,Ireland,344
Egypt,United States,15
United States,India,62
United States,Singapore,1
United States,Grenada,62
Costa Rica,United States,588
Senegal,United States,40
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:38:40 AM.

```
val datasplit = data.map(_.split(", "))

datasplit.take(10)
```

```
datasplit: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[84] at map at <console>:27
res74: Array[Array[String]] = Array(Array(DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count), Array(United States,Romania,15), Array(Egypt,United States,15), Array(United States,India,62), Array(United States,Singapore,1), Array(United States,Grenada,62), Array(Costa Rica,United States,588), Array(Senegal,United States,40))
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:38:59 AM.

```
val flightData2015 = (spark
  .read
  .option("inferSchema", "true")
  .option("header", "true")
  .csv("/data/flight-data/csv/2015-summary.csv"))
```

```
flightData2015: org.apache.spark.sql.DataFrame = [DEST_COUNTRY_NAME: string, ORIGIN_COUNTRY_NAME: string ... 1 more field]
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:39:10 AM.

Apache Spark Dataframe 실습 1

- 데이터를 보기 좋게 출력해보고, 데이터의 모양을 보고 나눠보고, DataFrame으로 만들어 보자

```
flightData2015.show
```

```
+-----+-----+-----+
| DEST_COUNTRY_NAME | ORIGIN_COUNTRY_NAME | count |
+-----+-----+-----+
| United States | Romania | 15 |
| United States | Croatia | 1 |
| United States | Ireland | 344 |
| Egypt | United States | 15 |
| United States | India | 62 |
| United States | Singapore | 1 |
| United States | Grenada | 62 |
| Costa Rica | United States | 588 |
| Senegal | United States | 40 |
| Moldova | United States | 1 |
| United States | Sint Maarten | 325 |
| United States | Marshall Islands | 39 |
| Guyana | United States | 64 |
| Malta | United States | 1 |
| Anguilla | United States | 11 |
```

Took 0 sec. Last updated by anonymous at February 10 2020, 6:39:31 AM.