
# RDD & DataFrame

이론: 엔지 19기 정성윤

실습: 엔지 19기 김지원

# 목차


**Part 1. Apache Spark**

1.1. ApacheSpark의 등장 배경
1.2. Spark와 Hadoop
1.3. Spark 특징
1.4. Apache Spark 동작 프로세스
1.5. SparkSession & SparkContext
1.6. Spark Component
1.7. Spark 기능
1.8. Apache Spark API

**Part 2. RDD**

2.1. RDD-resilient
2.2. RDD-Distributed
2.3. RDD 생성과 동작 원리
2.4. Narrow Transformation & Wide Transformation

**Part 3. DataFrame**

3.1. DataFrame 특징
3.2. RDD와 DataFrame
3.3. SparkSQL

# Part 1. Apache Spark

- Apache Spark는 오픈 소스 클러스터 컴퓨팅 프레임 워크로
  클러스터 환경에서 **데이터를 병렬로 처리하는 라이브러리의 집합으로 구성**.
- Apache Spark는 캘리포니아 대학교 버클리의 AMPLab에서 발표한 논문
  『Spark: Cluster Computing with Working Sets』를 통해 처음 세상에 알려짐.
- 스파크는 "빅데이터 애플리케이션 개발에 필요한 통합 플랫폼을 제공하자"
  라는 핵심 목표를 가지고 있음.
- 스파크는 데이터 읽기에서부터 SQL 처리, 머신러닝, 그리고 스트림 처리에 이르기까지
  **데이터 분석 작업을 같은 연산 엔진과 일관성 있는 API로 수행할 수 있도록 설계**되어 있음.

# 1.1. Apache Spark의 등장 배경



- Spark가 등장하기 전에는 HDFS와 MapReduce 엔진으로 분산 컴퓨팅을 대중화한 **하둡이 대중적으로 사용됨.**

- 그러나 **MapReduce는 Disk I/O를 기반**으로 동작하기 때문에 느리다는 단점을 가지고 있음.
  근 몇년간 생성되는 데이터가 많아졌고, 이를 실시간 처리하기 위한 수요 또한 많아 짐.
  이러한 수요에 맞춰, 보다 빠르게 처리하기 위해 **In-memory 기반의 Spark**가 주목 받게 됨.

- Spark는 Hadoop의 MapReduce보다 100배 이상의 속도를 냄.

# 1.2. Spark와 Hadoop


- Apache Spark는 하둡 에코 시스템의 **연산엔진을 대체**하는 플랫폼.

  - 하둡이 스파크를 반드시 사용하지 않아도 되듯이 스파크도 하둡 없이 사용이 가능.
  - 스파크에는 자체적인 **파일 시스템이 존재하지 않지만**, HDFS가 아니더라도 다른 클라우드 데이터 플랫폼과 호환이 가능.
  - 스파크는 본래 Hadoop의 연산엔진을 대체하기 위해 탄생한 플랫폼인 만큼 **HDFS을 사용하였을 때 가장 안정적**.

# 1.3. Spark 특징

- **Speed** : 인메모리(In-memory) 기반의 빠른 처리

- **Ease of Use** : 다양한 언어 지원(Java, Scala, Python, R, SQL)을 통한 사용의 편의성
  (spark는 본래 Scala를 기반으로 설계 되었기 때문에 Scala를 사용하는 것이 더 빠른 성능)

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Scala version 2.12.15 (Java HotSpot(TM) 64-Bit Server VM, Java 11.0.16.1)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Python version 3.9.12 (main, Jun  1 2022 06:34:44)
Spark context Web UI available at http://172.20.10.13:4040
Spark context available as 'sc' (master = local[*], app id = local-1663738106248).
SparkSession available as 'spark'.
>>>
```

- **Generality**: SQL, Streaming, 머신러닝, 그래프 연산 등의 다양한 컴포넌트 제공

- **Run Anywhere**: YARN, Mesos, Kubernetes 등 다양한 클러스터에서 동작,
  HDFS, Cassandra, HBase 등 다양한 파일 포맷 지원.

# 1.4. Apache Spark 동작 프로세스

- **Driver**

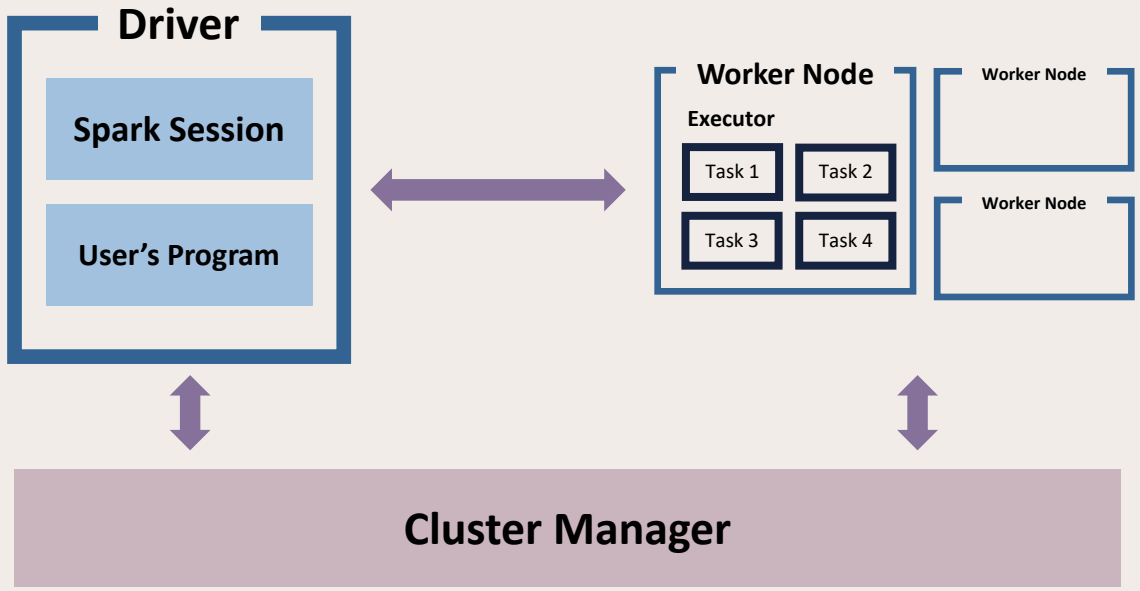
  - 콘솔 앱과 같은 프로그램과 Spark 세션으로 구성.
  - SparkSession은 프로그램을 처리하기 위해 하나의 작업을 **더 작은 작업으로 분할**하여 **Executor에게 보냄.**

- **Executors**

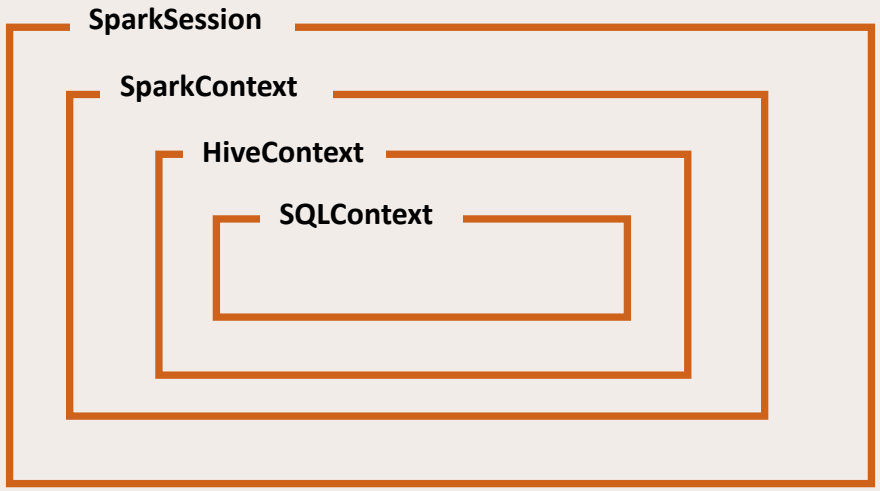
  각 executor와 worker node는 **드라이버로부터 작업을 받아** 해당 **작업을 실행.**

- **Cluster Manager** – ex) YARN, mesos

  - 리소스 할당 관리
  - 프로그램 분할 관리
  - 프로그램 실행 관리



# 1.5. SparkSession & SparkContext



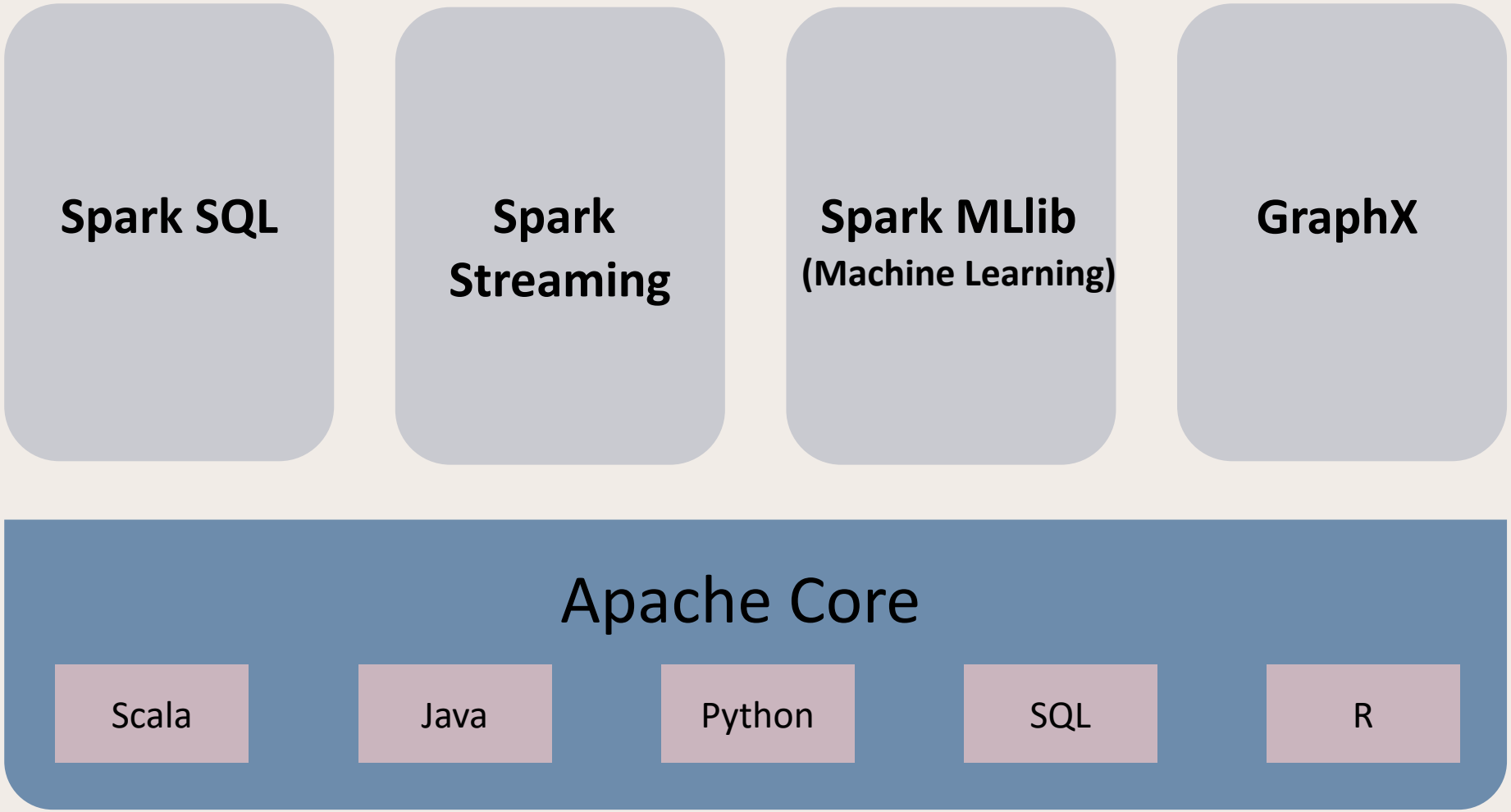

```scala
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder
.appName("SparkSessionExample")
.master("local[4]")
.config("spark.sql.warehouse.dir", "target/spark-warehouse")
.enableHiveSupport()
.getOrCreate
```

< Builder Pattern >

- SparkSession은 스파크 응용프로그램의 통합 진입점으로 스파크의 기능들과 구조들이 상호작용하는 방식을 제공.
- SparkSession은 Builder Pattern을 사용해서 생성.
- SparkSession 이전에는 SparkContext와 SQLContext를 직접 생성하여 사용하였음. (SparkSession은 이 둘의 기능을 모두 사용 가능)
  - SparkContext: 스파크의 핵심 추상화 개념을 다루는 데 중점을 둠.
  - SQLContext: SparkSQL과 같은 고수준 API기능을 다루는 데 중점을 둠.
- SparkSession이 도입되면서 **SparkContext의 충돌**을 방지할 수 있음.

# 1.6. Spark Component



## Apache Spark Core

- Spark Component에 **필요한 기본 기능을 제공**.
- RDD로 다양한 연산 및 변환 메소드를 제공.
- HDFS, GlusterFS, Amazon S3등 **다양한 파일 시스템에 접근**할 수 있음.
- 공유변수(broad variable)와 누적변수(accumulator)를 통해 **컴퓨팅 노드간 정보 공유**.

## Spark SQL

- Spark 와 Hive SQL이 지원하는 SQL을 사용해 **대규모 분산 정형 데이터를 다룰 수 있음.**
- JSON 파일, Parquet 파일 등 **정형 데이터를 읽고 쓸 수 있음.**
- DataFrame과 Dataset에 적용된 연산을 일정 시점에 **RDD연산으로 변환**하여 **Spark Job 으로 실행**함.

## Spark Streaming

- **실시간 스트리밍 데이터**를 처리하는 프레임워크.
- HDFS, Apache Kafka, Apache Flume 등의 리소스를 사용할 수 있음.
- 다른 스파크 컴포넌트와 함께 사용할 수 있어
  실시간 데이터 처리를 **머신러닝 작업, SQL 작업, 그래프 연산 등과 통합** 할 수 있음.

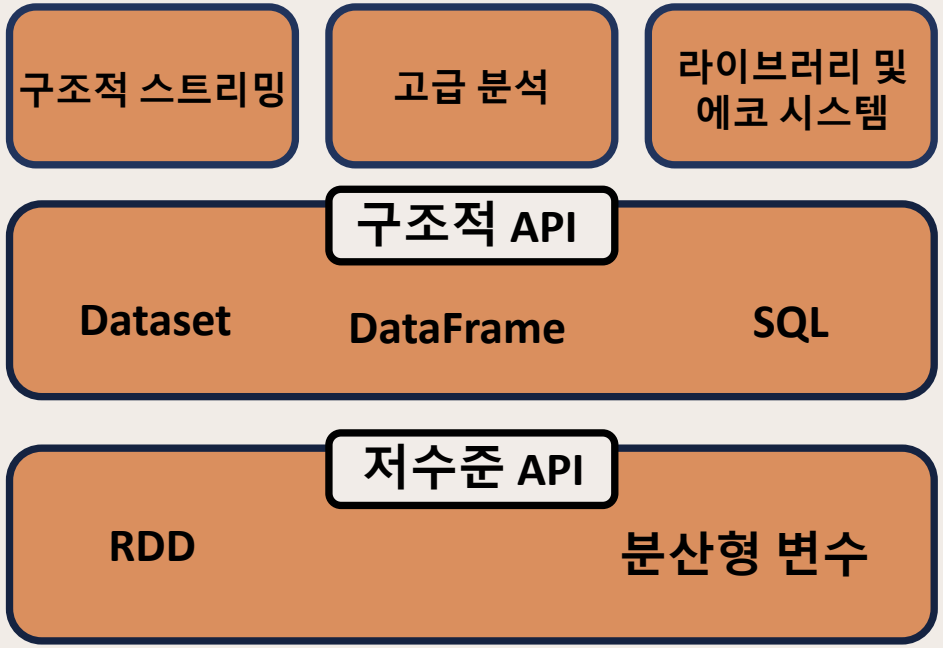
## Spark MLlib

- 머신 러닝 알고리즘 라이브러리로 RDD 또는 DataFrame의
  데이터셋을 변환하는 **머신러닝 모델을 구현**할 수 있음.

## Spark GraphX

- 그래프 RDD 형태의 그래프 구조를 만들 수 있는 기능을 제공.

# 1.7. Spark 기능




- **구조적 스트리밍 - 고급 분석 - 라이브러리 및 에코시스템**
- **구조적 API**: Dataset, **DataFrame**, SQL
- **저수준 API**: **RDD**, 분산형 공유 변수(broadcast, accumulator)

# 1.8. Apache Spark API

● Apache Spark에서는 RDD, DataFrame, DataSet 3가지의 API를 제공한다.

RDD

- 처음 출시된 스파크 1.0은 RDD API 를 이용하여 데이터를 처리하였음.
- RDD는 인메모리 데이터 처리를 통하여 **처리 속도를 높일 수** 있었지만,
테이블 조인 효율화 같은 처리를 사용자가 직접 제어해야 했기 때문에 **최적화에 어려움**을 겪었음.

DataFrame

- 데이터프레임은 스파크 1.3에서 처리 속도 증가를 위한 프로젝트 텅스텐의 일부로 소개되었음.
- 데이터를 **스키마 형태로 추상화** 하고, 카탈리스트 옵티마이저가 **쿼리를 최적화**하여 처리함.

Dataset

- 데이터 셋은 스파크 1.6에서 추가 되었음.
- 데이터의 타입체크, 데이터 직렬화를 위한 인코더, 카탈리스트 옵티마이저를 지원하여
데이터 처리 속도를 더욱 증가시킴.

※ 스파크 2.0에서 데이터프레임과 데이터셋을 통합함.

# Part 2. RDD(Resilient Distributed Dataset)

- RDD는 Spark 1.0 부터 도입 되었던 스파크의 철학이 담겨 있는 핵심 API
- RDD(Resilient Distributed Data)
  - **Resilient**: 회복력 있는, 탄력 있는, 변하지 않는
  - **Distributed**: 분산된
  - **Data**: 데이터

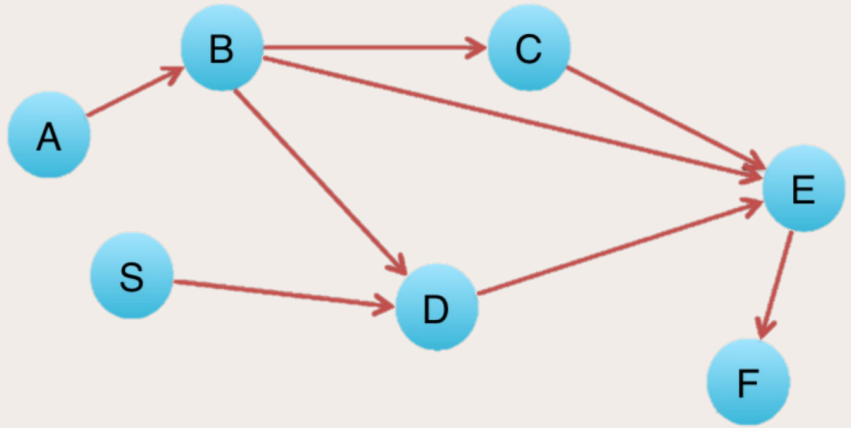
→ 여러 **분산 노드**에 걸쳐 저장되는, **변경 불가능**한 데이터의 집합.

# 2.1. RDD-Resilient

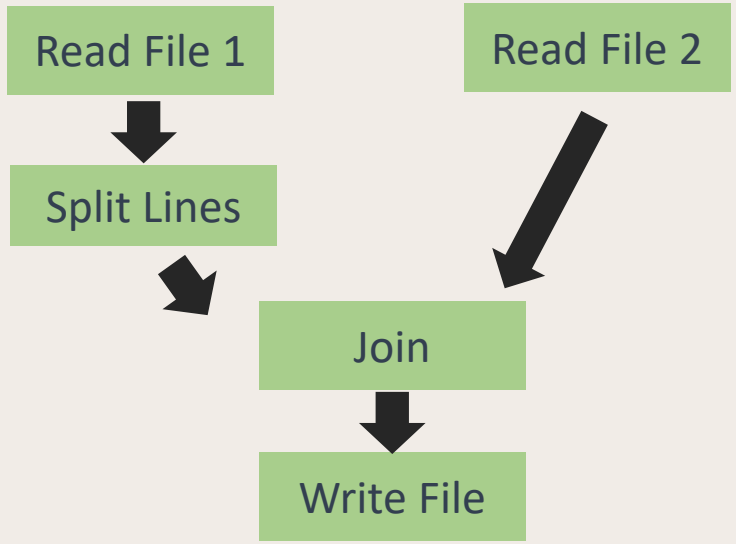
- RDD는 포함된 데이터를 저장해 두는 것이 아니고, **RDD를 생성하는 데 사용했던 작업 내용을 기억**하고 있는 것.
- 문제가 발생하면 전체 작업을 처음부터 다시 실행하는 대신 **문제가 발생했던 작업만 다시 수행하여 복구**함.

## RDD Lineage

- 이러한 구조를 RDD Lineage라고 불림. 이는 DAG(Directed Acyclic Graph)의 형태를 가짐.
- 노드 간의 **순환이 없고**, **일정 방향**을 가지기 때문에 각 노드 간에는 **의존성**이 있고, 노드 간의 **순서가 중요**한 형태라고 할 수 있음.




<DAG Graph>



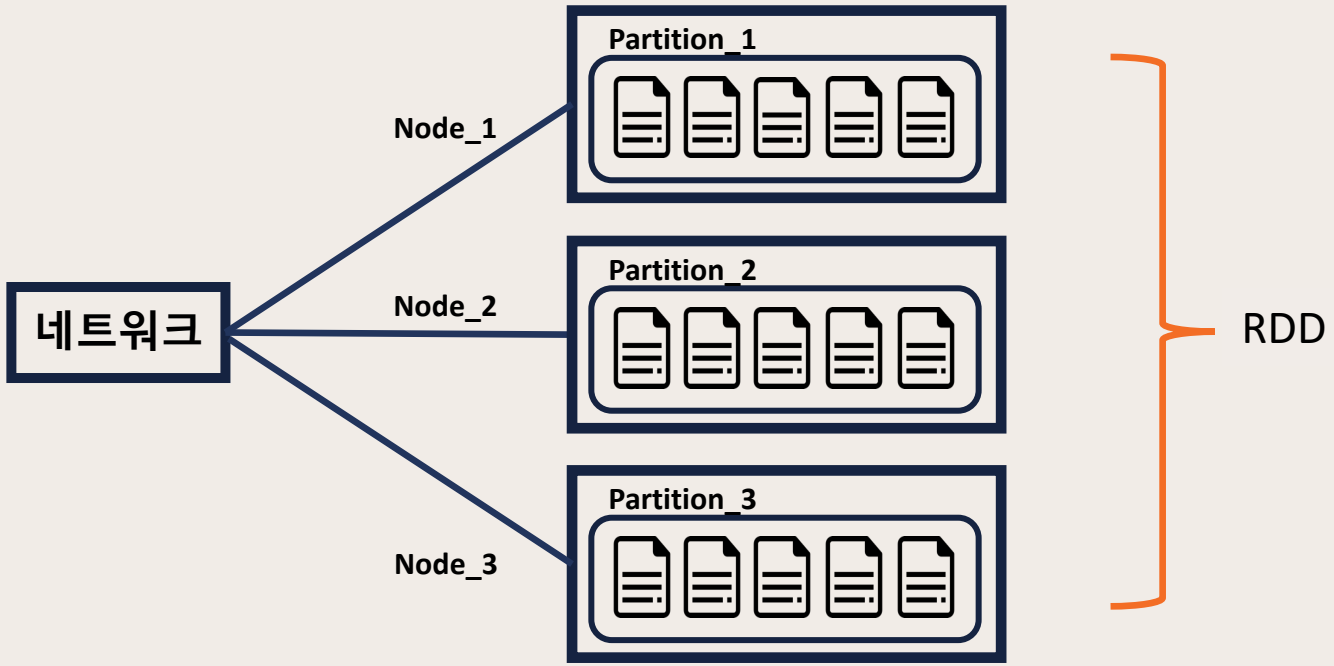

<DAG of Job>

# 2.2. RDD-Distributed

- RDD는 저장될 때 **여러 서버에 나누어 저장**되며, 처리할 때에는 각 서버에 저장된 데이터를 동시에 **병렬 처리** 함.

- **Partition**: RDD나 Dataset을 구성하고 있는 최소 단위의 객체.
  Partition은 **서로 다른 노드에서 분산처리** 됨.
  Spark에서는 하나의 최소 연산을 Task로 표현되는 데 이 하나의 Task에서는 하나의 Partition이 처리되며,
  하나의 Task는 하나의 Core가 연산함.

**1 Core = 1 Task = 1 Partition**

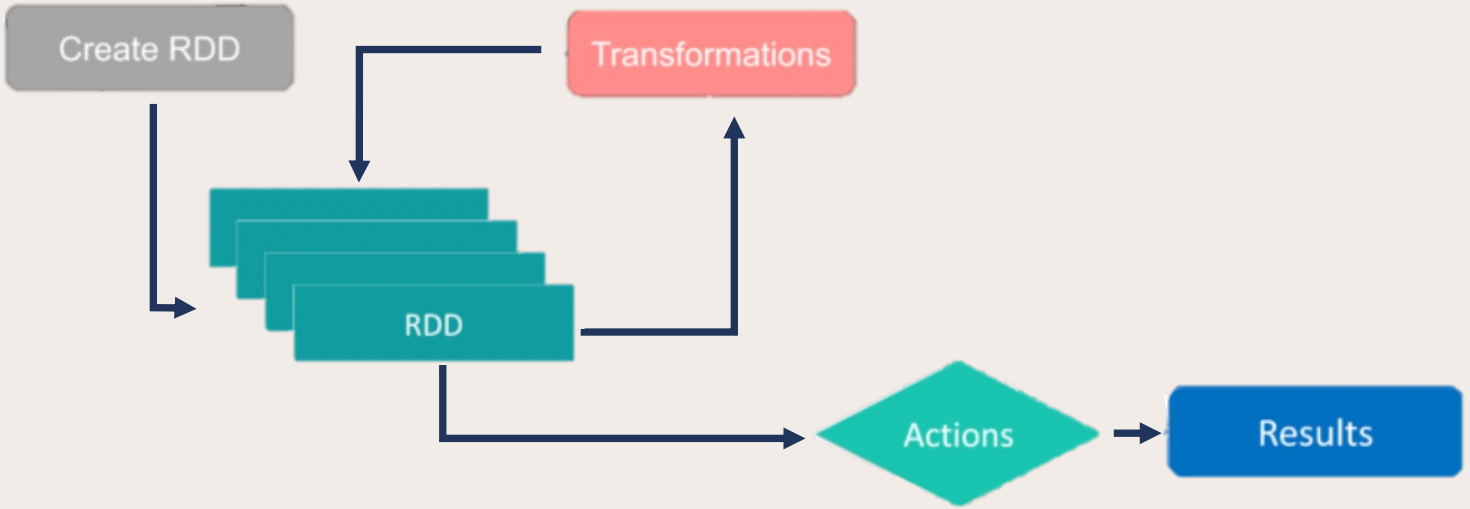



<15개의 데이터를 3개의 노드로 구성된 클러스터에 분산 저장하여 RDD를 구성함>

# 2.3. RDD 생성과 동작 원리

## RDD 생성

- **parallelize() 함수**:
  내부에서 만든 데이터 집합을 병렬화 하는 방법
- **.textFile() 함수**:
  외부의 파일을 로드 하는 방법




## RDD 동작

Transformation :

기존의 RDD에서 새로운 RDD를 만들어 내는 연산으로
Action을 하기 전까지 transformation은 일어나지 않는다.

Action :

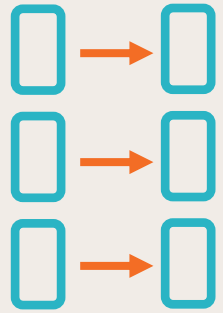
transformation으로 논리적인 실행 계획 후 action을 통해 transformation을 실행.

→ **Lazy Evaluation (지연 연산)**

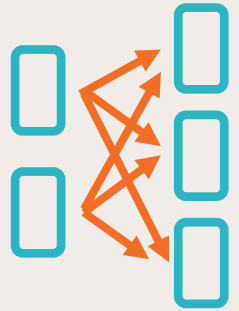
이와 같이 스파크가 action을 취하기 전까지 transformation을 쌓으며 기다리는 동작 방식

# 2.4. Narrow Transformation & Wide Transformation

- **Narrow Transformation**: 좁은 의존성을 가진
  - 각 transformation은 각 입력 파티션이 하나의 출력 파티션에만 영향을 미치는 변환을 의미함 (1:1 변환)
  - **메모리 내에서 수행(In - memory)**
  - ex) filter(), map(), flatMap(), sample() 등



- **Wide Transformation**: 넓은 의존성을 가진
  - 입력 파티션이 수많은 출력 파티션에 영향을 미침 이를 **Shuffle**이라 함 (1:N 변환)
  - Shuffle 을 수행하면 narrow tranformation과 달리 결과를 **Disk에 씀.**
  - ex) intersection(), join(), reduceByKey(), groupByKey(), takeSample()



※ 디스크에 접근이 많아지면 속도가 저하되기 때문에 효율적이지 못함.

→ **Shuffle의 횟수를 최적화**하는 것이 Spark 최적화의 핵심.

# Part 3. DataFrame

- Spark DataFrame 은 스파크에서 정형 데이터 처리를 위해 사용되는 데이터 셋 객체
- **RDD 기반으로 동작**하며, **Spark SQL 사용에 최적화** 되어 있음.

  → DataFrame의 Transformation을 실행하면 다수의 RDD Trasformation이 반환됨.
- R, Python Pandas, 스프레드 시트의 개념과 유사.

- DataFrame 관련 용어
  - 레코드와 컬럼: row 개념의 레코드와 수행할 연산 표현식을 나타내는 여러 컬럼으로 구성됨.
  - 스키마(schema): DataFrame의 컬럼명과 데이터 타입을 정의.
  - DataFrame Partitioning: DataFrame이 클러스터에서 물리적으로 배치되는 형태를 의미.
  - Partitioning Schema: 파티션을 배치하는 방법을 정의.

# DataFrame과 Schema

## DataFrame

```
+-------+------+---------+------+--------+
|   name|ticker|  country| price|currency|
+-------+------+---------+------+--------+
| Google| GOOGL|      USA|  2984|     USD|
|Netflix|  NFLX|      USA|   645|     USD|
| Amazon|  AMZN|      USA|  3518|     USD|
|  Tesla|  TSLA|      USA|  1222|     USD|
|Tencent|  0700|Hong Kong|   483|     HKD|
| Toyota|  7203|    Japan|  2006|     JPY|
|Samsung|005930|    Korea| 70600|     KRW|
|  Kakao|035720|    Korea|125000|     KRW|
+-------+------+---------+------+--------+
```

## DataFrame Schema

```
root
 |-- name: string (nullable = true)
 |-- ticker: string (nullable = true)
 |-- country: string (nullable = true)
 |-- price: long (nullable = true)
 |-- currency: string (nullable = true)
```

# 3.1. DataFrame 특징

- **구조화된(Structured) 데이터 구조**

  : DataFrame은 **구조화된 데이터를 다루기 쉽게** 하기 위해 만들어진 데이터 구조.

  사용자는 **SparkSQL**을 통해 데이터 처리 가능

- **GC(Garbage Collection) 오버헤드 감소**:

  RDD는 데이터를 메모리에 저장하지만, DataFrame은 데이터를 **오프-힙 영역에 저장**.

- **직렬화 오버헤드 감소**:

  DataFrame은 오프-힙 메모리를 사용한 직렬화를 통하여 오버헤드를 크게 감소.

- Fl**exibilithy & Scalabilty**:

  DataFrame은 CSV, Cassandra 등 **다양한 형태의 데이터를 지원**하기 때문에 사용자 입장에서 효율적이다.

※ GC(Garbage Collection) : 사용하지 않는 객체를 자동으로 메모리에서 해제

# 3.2. RDD와 DataFrame

RDD의 장단점

1) RDD는 메모리나 디스크 저장 공간이 충분하지 않으면 동작하지 않음.

2) RDD는 스키마(DataBase 구조) 개념이 별도로 없음.

→ RDD는 구조화 데이터와 비구조화 데이터를 함께 저장하기 때문에 **효율성이 떨어짐.**

3) RDD는 별도의 내장된 최적화 엔진이 없기 때문에 **사용자가 직접 최적화** 해야함.

→ 이로 인해 개발자가 **모든 값을 수동적으로 정의** 해야 하고, 이는 **잠재적인 문제점을 야기**할 수 있음.

4) 하지만, 객체를 사용하기 때문에 사용자가 **원하는 포맷으로 데이터를 저장**할 수 있다는 장점이 있음.

결론

- Apache Spark는 **공식적으로 RDD보다는 구조적 API(DataFrame) 사용을 권장함.**

- 대부분의 상황에서는 구조적 API인 DataFrame을 사용하지만, 이를 통해 모든 문제를 해결할 수 있는 것은 아님.

→ 상황에 따라 RDD를 사용할 줄도 알아야함.

# 3.3. Spark SQL

- Spark SQL은 스파크의 컴포넌트 중 하나로 Spark의 구조화 데이터인 **DataFrame의 데이터 처리**를 위한 Spark 모듈.
- 하둡 상의 데이터를 기반으로 작성된 Hive 쿼리에 비하여 최대 100배까지 빠른 성능을 가능하게 함.
- Spark SQL에는 비용 기반 최적화 프로그램, 열형식 스토리지, 코드 생성도 포함되어 있어 **쿼리속도가 빠름**.

  → RDD API 와 다르게 **Spark SQL은 내부적으로 질의 최적화가 잘되어 있음.**

- 동시에 Spark 엔진을 사용해 수천개의 노드, 여러 시간의 쿼리 규모로 확장할 수 있어
  쿼리 중 내결함성을 100%을 보장함.

- Spark SQL의 사용
  - Parquet 파일에서 Hive 테이블로 **관계형 데이터 가져오기**
  - 가져온 데이터 및 기존 RDD에 대하여 **SQL 쿼리 진행**
  - Hive 테이블이나 Parquet 파일로 **RDD 쓰기**

# 참고 자료

<참고 도서>

- 스파크 완벽 가이드 (빌 체이버스, 마테이 자하리아 지음)
- 스파크를 다루는 기술 (페타 제체비치, 마르코 바나치 지음)
- 스파크 입문(사루타 고스케 외 3명 지음)

<기업 공식 문서>

- https://spark.apache.org/
- https://www.databricks.com/kr/
- https://learn.microsoft.com/ko-kr/dotnet/spark/what-is-spark
- https://cloud.google.com/learn/what-is-apache-spark?hl=ko

<개발 블로그>

- https://artist-developer.tistory.com/
- https://rollingsnowball.tistory.com/
- https://velog.io/@busybean3
- https://wikidocs.net/book/2350

Thank you.