

Spark & RDD

목차

1. Spark란?

- Hadoop Ecosystem
- Apache Spark란?
- Spark vs. Hadoop
- Spark ♥ Hadoop

2. Spark의 구조

- Spark Component
- Spark의 동작 구조 (Spark 클러스터 아키텍처)

3. Spark RDD

- Apache Spark의 3가지 자료구조
- Spark RDD란?
- RDD의 특징
- RDD의 연산

1. Spark란?



Spark란?

Hadoop Ecosystem

- **Hadoop**

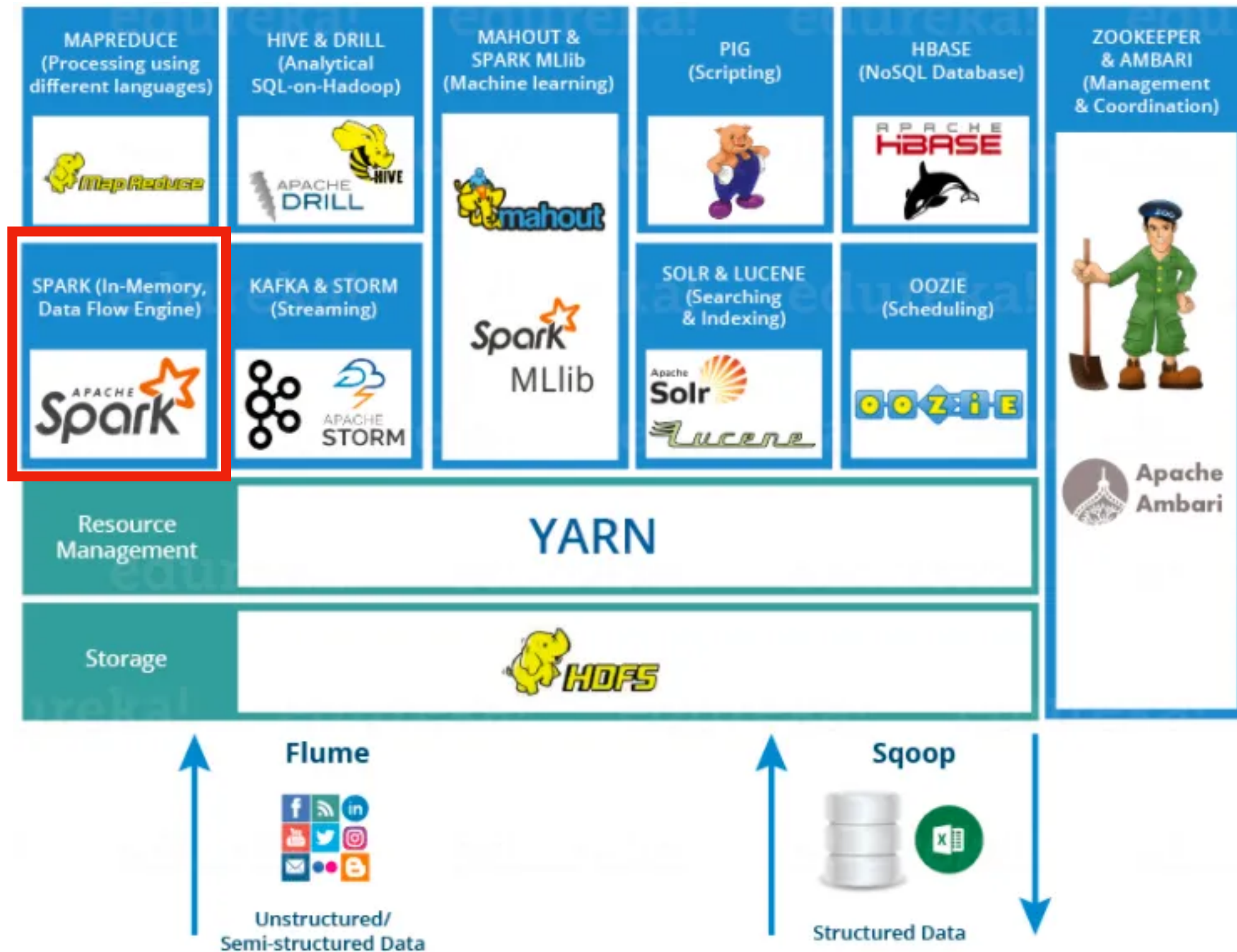
HDFS, YARN, MapReduce를 사용하는 대용량 분산처리 시스템

- **Hadoop Ecosystem**

Hadoop 환경에서 빅데이터 문제를 효율적으로 다루기 위해 만들어진 서브 프로젝트들의 집합

Spark란?

Hadoop Ecosystem



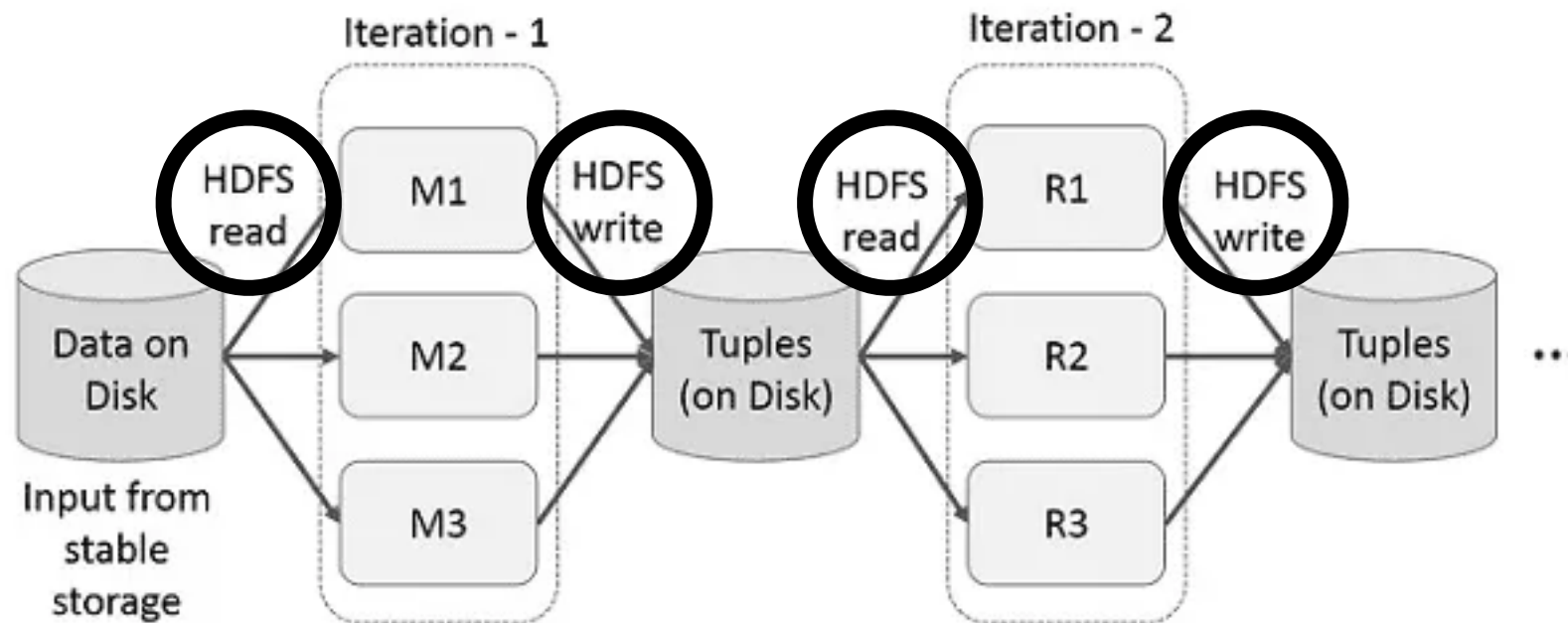
Spark란?

Apache Spark란?

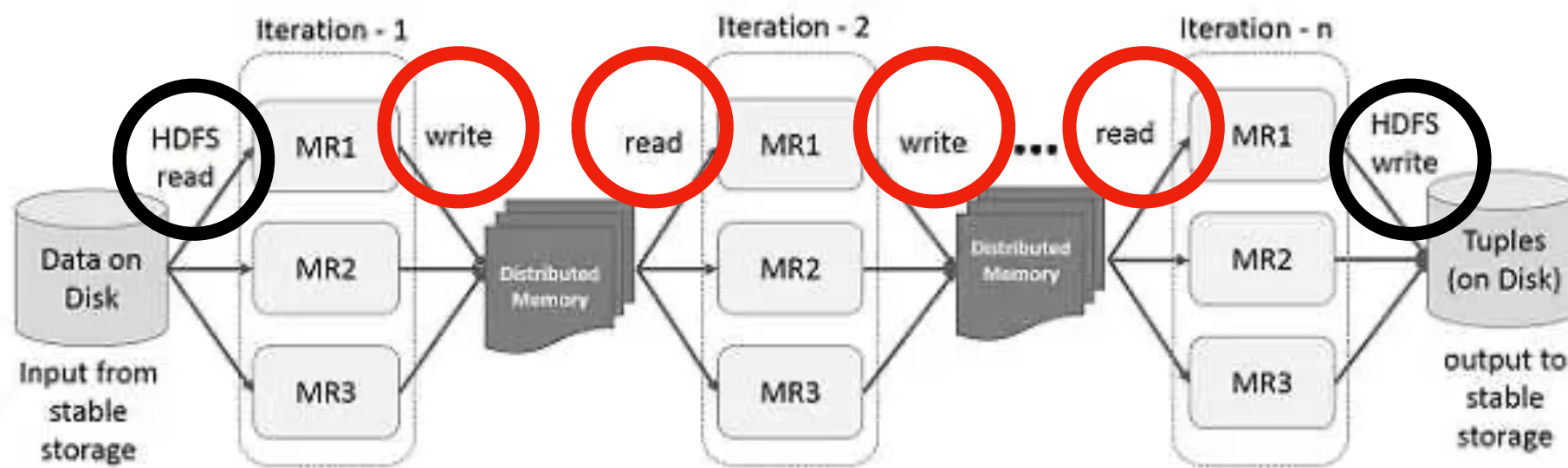
- 현재 매우 활발히 개발 및 사용되고 있는 범용 분산 클러스터 컴퓨팅 프레임워크
- 데이터 센터나 클라우드에서 대규모 분산 데이터 처리를 하기 위해 설계된 통합형 엔진
- **In-Memory 기반**의 대용량 데이터 고속 처리 엔진

Spark란?

Spark vs. Hadoop



Hadoop은
디스크 기반



Spark은
메모리 기반

Spark란?

Spark vs. Hadoop

- 하둡은 맵을 하든 리듀스를 하든 중간에 디스크를 거쳐야 함
-> hadoop의 시스템의 대부분은 HDFS 읽기 쓰기 작업을 수행하는데 90% 이상을 사용
- 스파크는 처음에는 디스크에서 불러오지만 **중간 작업은 메모리에서** 함
-> 메모리 기반의 빠른 연산, 반복 처리에 특히 강함

Spark란?

Spark vs. Hadoop

	Apache Spark	Apache Hadoop (MapReduce)
핵심 개념	인메모리(메모리 기반) 연산	디스크 기반 배치 처리
속도	Hadoop보다 최대 100배 빠름 (메모리 활용)	디스크 I/O로 인해 상대적으로 느림
데이터 저장소	자체 스토리지가 없음 (HDFS, S3, Cassandra 등 외부 스토리지를 사용)	HDFS 사용 (내장 스토리지)
실시간 처리	✅ 실시간 스트리밍 지원 (Spark Streaming)	❌ 실시간 처리 불가능
유연성	SQL, 머신러닝, 그래프 분석 등 다양한 기능 제공	주로 배치(Batch) 처리에 최적화
코딩 방식	Scala, Python, Java, R 지원	주로 Java 기반
장점	빠른 속도, 유연한 API, 실시간 처리	데이터 내구성 보장, 안정적 배치 처리
단점	많은 메모리 필요, 높은 하드웨어 요구사항	속도가 느림, 실시간 처리 불가능

Spark란?

Spark ♥ Hadoop

- 그렇다고 Spark가 Hadoop을 대체하는 것은 아님. 보완하는 역할!
- Standalone Deployment / Hadoop Yarn Deployment / Spark In MapReduce(SIMR) 과 같은 방식으로 하둡이랑 같이 씬!



(a) Standalone



(b) Over Yarn



(c) Spark in
MR (SIMR)

Spark란?

Spark의 특징

- **In-Memory 기반**

메모리에 중간 결과를 저장하여 반복 작업의 처리 효율이 높음

- **다양한 라이브러리, 언어 제공**

Spark Streaming, SparkSQL, MLlib과 같은 다양한 대규모 데이터처리 환경에서 사용가능한 라이브러리 제공

Java, Scala, Python, R 등 다양한 언어를 지원. 단, 언어마다 처리하는 속도나 사용할 수 있는 기능이 살짝 다름(파이썬은 못 쓰는 자료구조 있음)

- **다양한 클러스터 매니저 지원**

다양한 클러스터 매니저 포맷을 지원하여 운영 시스템 선택에 다양성을 제공

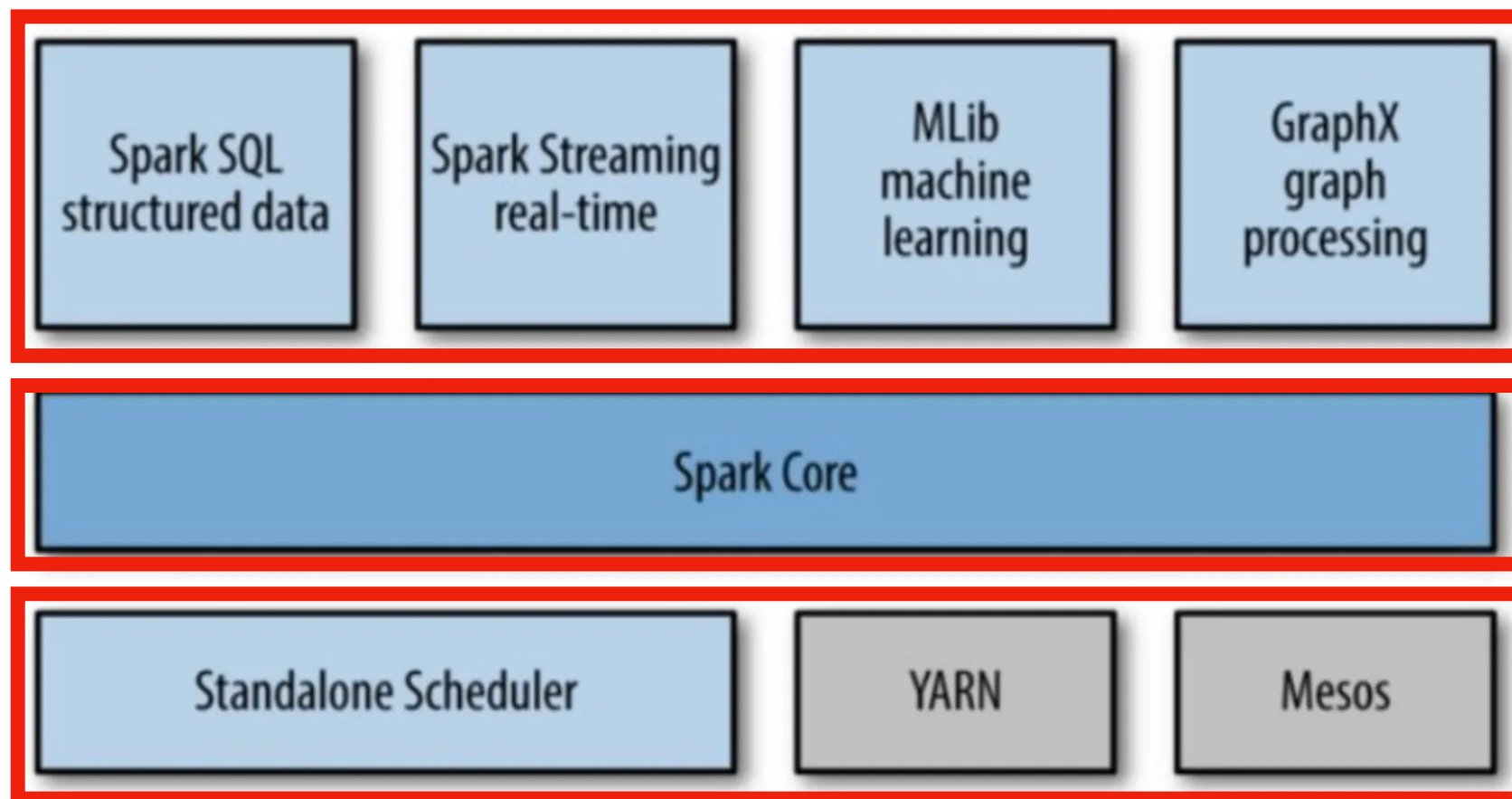
- **다양한 파일 포맷 지원 및 Hbase, Hive 등과 연동성**

기본적으로 txt, JSON, Parquet 등의 파일 포맷을 지원 + S3, HDFS 등의 파일 시스템, HBase, Hive와도 간단하게 연동 가능

2. Spark의 구조

Spark의 구조

Spark Component



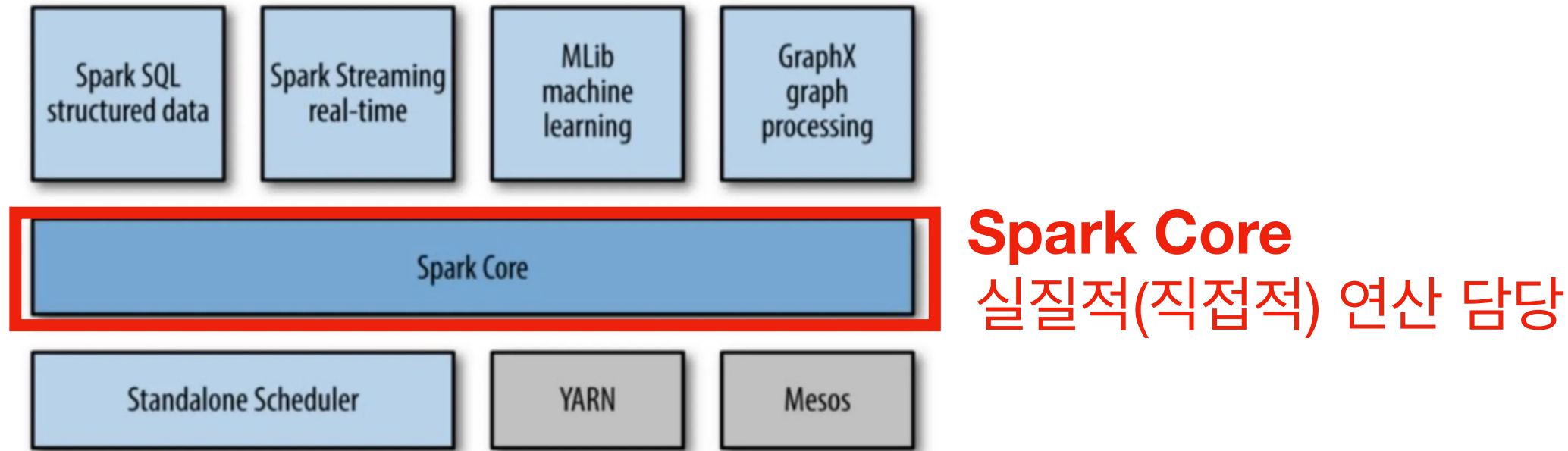
Spark Library
부가적 연산 담당

Spark Core
실질적(직접적) 연산 담당

Cluster Manager
리소스 관리

Spark의 구조

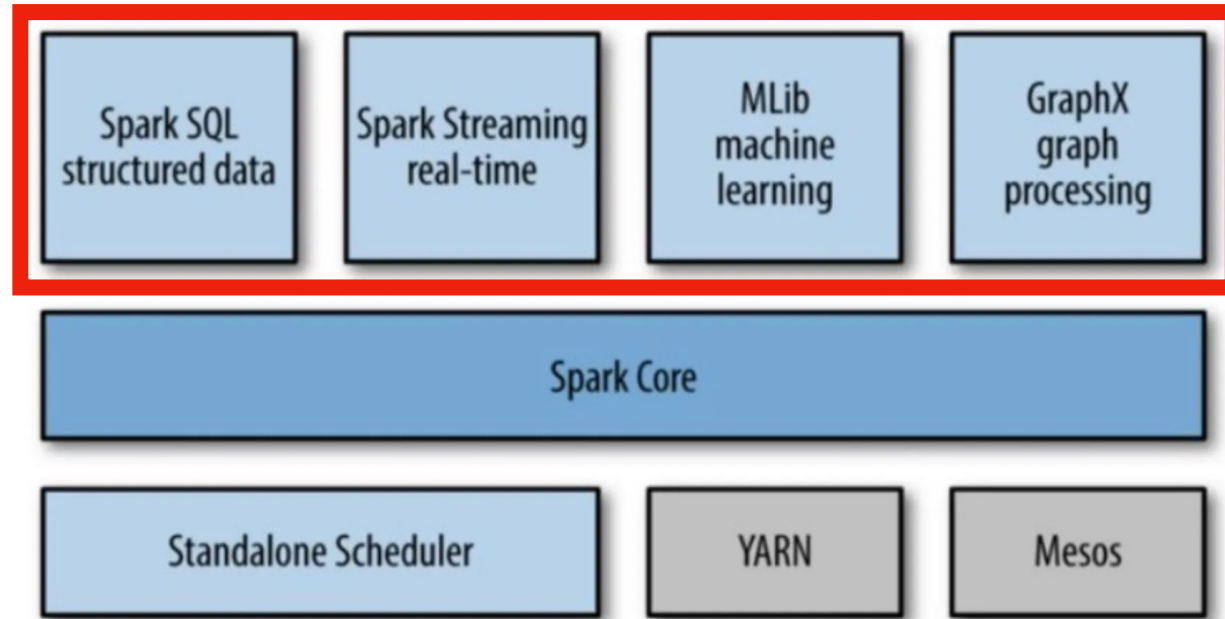
Spark Component



- Spark의 핵심, 클러스터 환경에서 데이터 처리를 위한 **기본적인 API** 제공
- 메인 컴포넌트로서, 작업 스케줄링 / 메모리 관리 / 장애 복구와 같은 기본적인 기능 제공
- RDD같은 Spark만의 자료구조를 이용해 분산 환경에서 연산을 처리

Spark의 구조

Spark Component

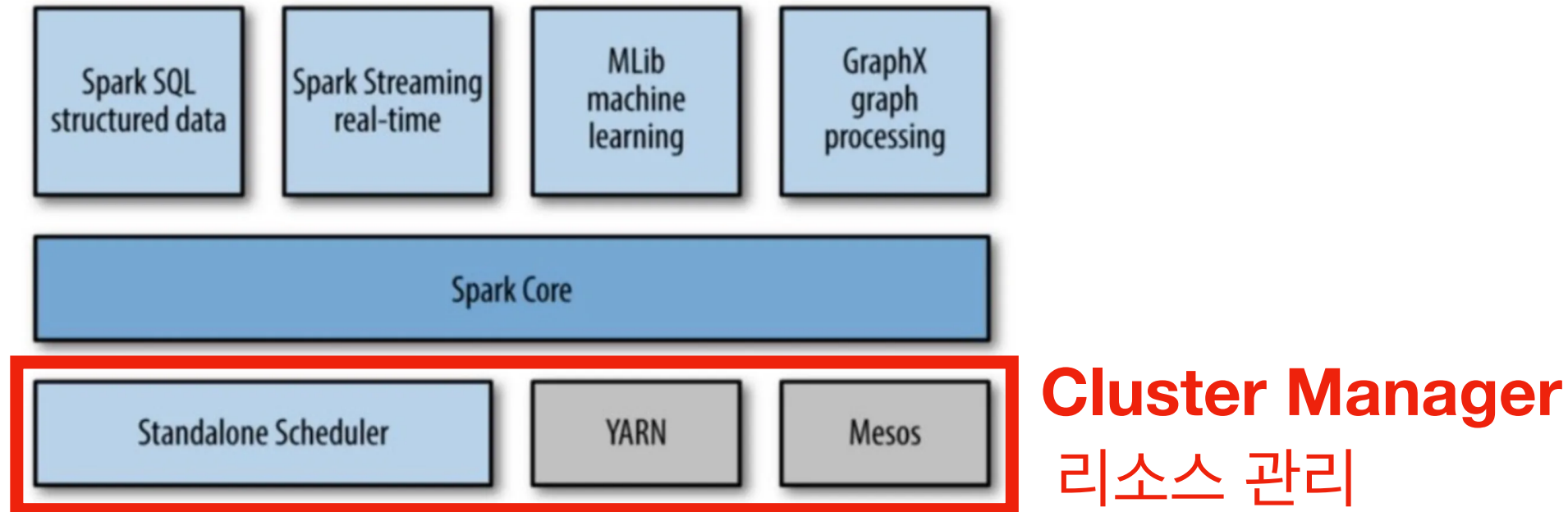


Spark Library
부가적 연산 담당

- 빅데이터 처리를 위한 다양한 기능을 제공하는 **Spark 자체 라이브러리**
 - Spark SQL: Spark 내에서 SQL 기능을 제공하는 라이브러리 (Query로 데이터 처리 가능)
 - Spark Streaming: 실시간 데이터 스트림을 처리하는 라이브러리 (Micro-batch)
 - MLib: Spark 기반의 머신러닝 기능을 제공하는 라이브러리
 - GraphX: 분산형 그래프 프로세싱 기능 제공하는 라이브러리

Spark의 구조

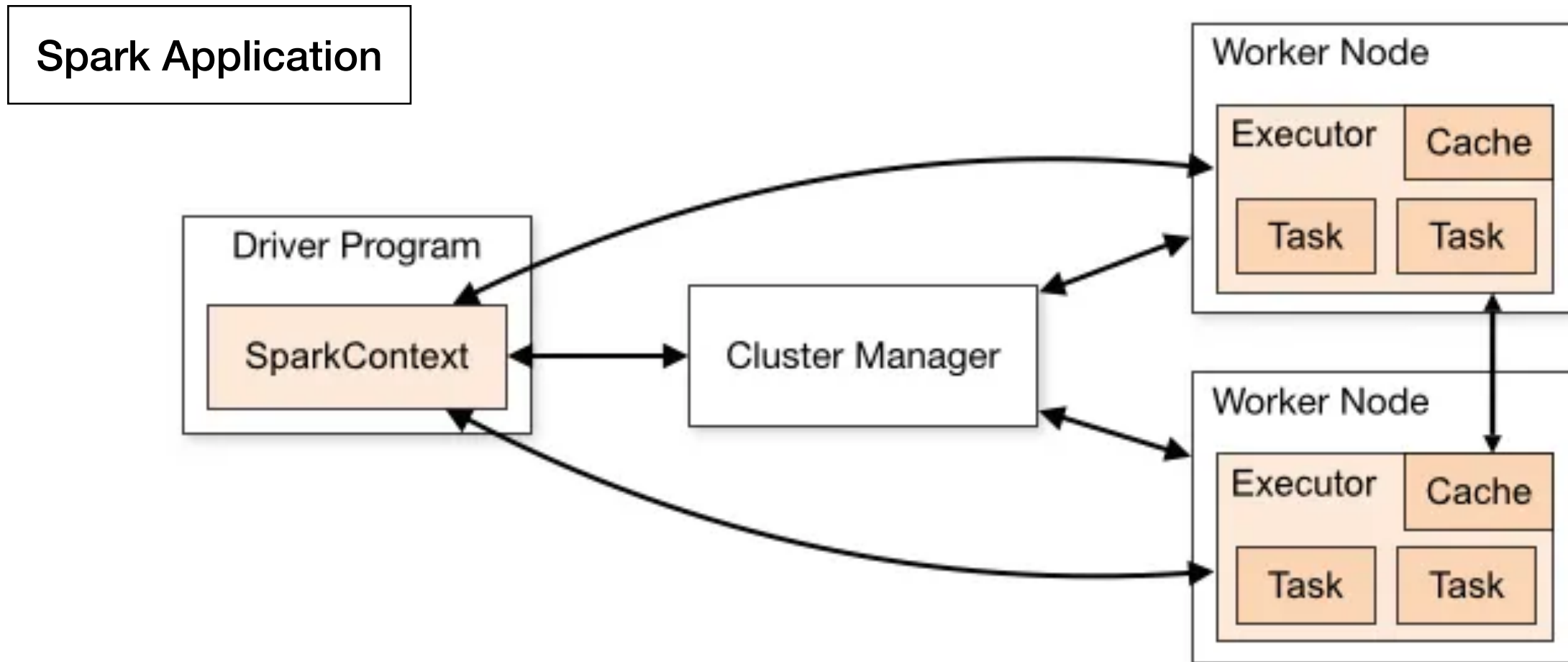
Spark Component



- 클러스터 환경에서 리소스를 할당하고 관리하는 역할
- Cluster Manager마다 스파크 애플리케이션을 실행하고, 클러스터 리소스를 할당하는 방식에 차이가 있음
- Standalone, Hadoop YARN, Apache Mesos 등 다양한 유형의 클러스터 매니저를 사용할 수 있음

Spark의 구조

Spark의 동작 구조 (Spark 클러스터 아키텍처)

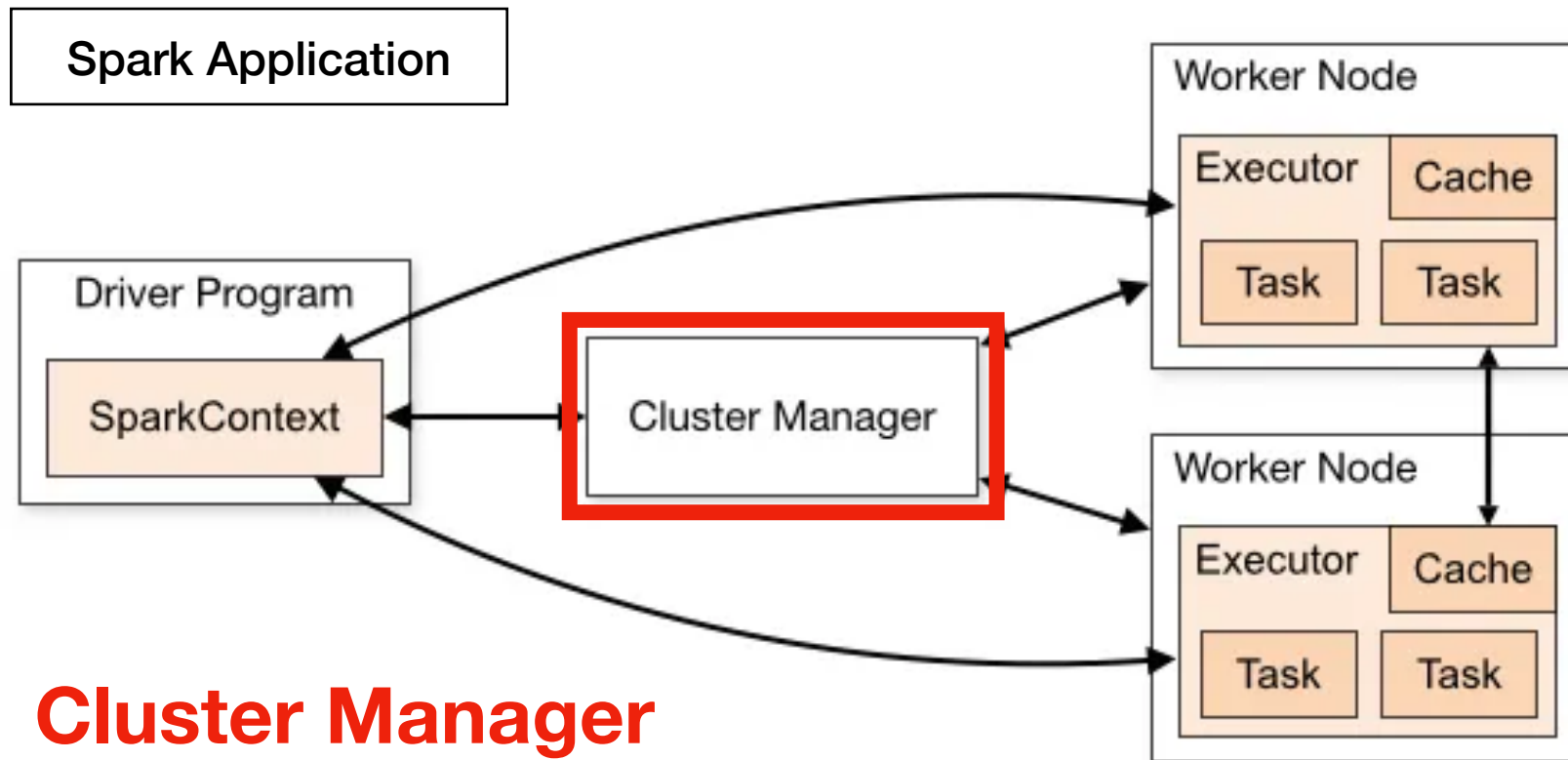


Spark Cluster

여러 대의 컴퓨터(노드)로 구성. Spark Application을 실행할 수 있는 환경.

Spark의 구조

Spark의 동작 구조 (Spark 클러스터 아키텍처)

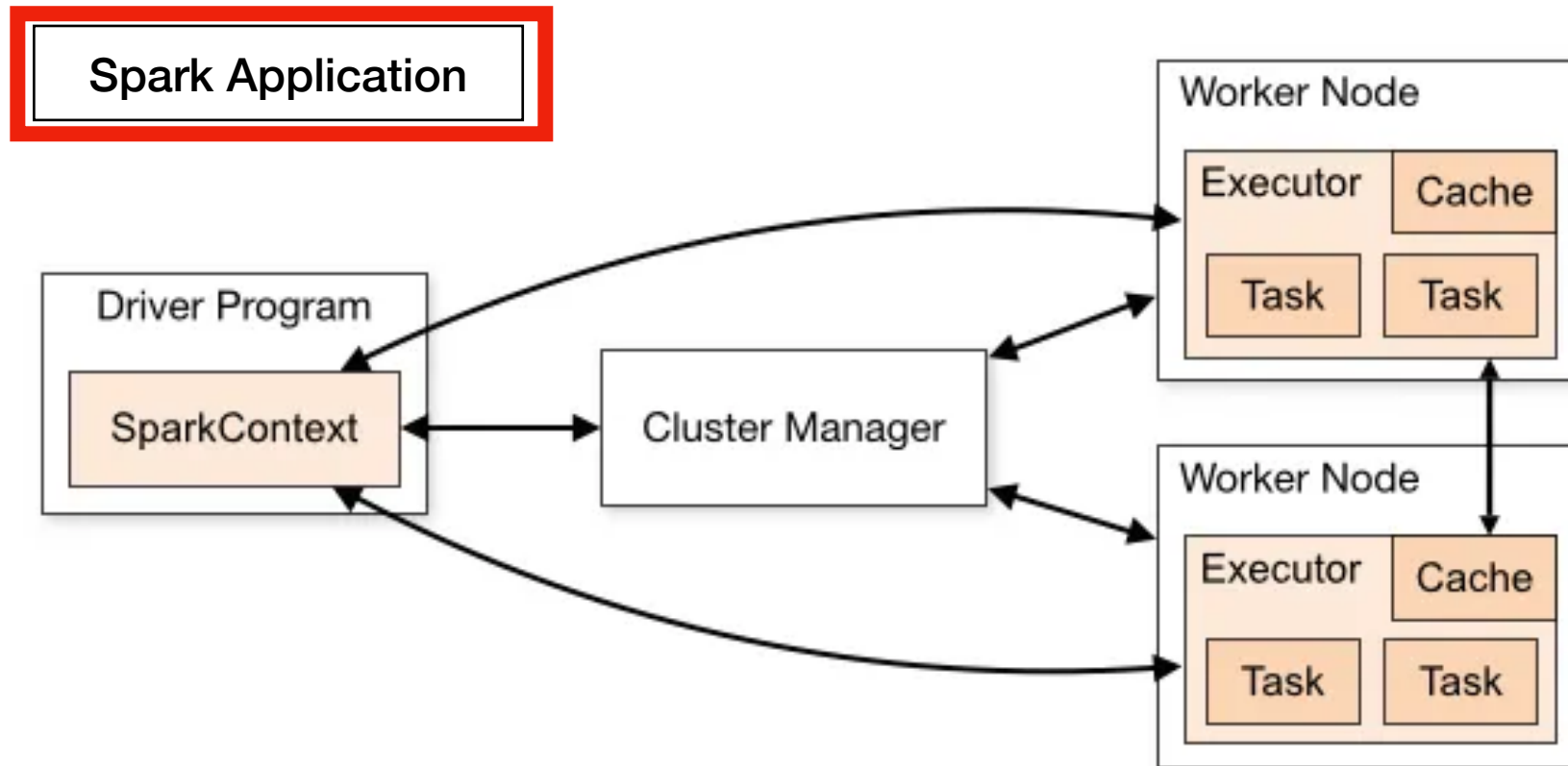


Cluster Manager

- 작업이 실행되는 노드를 관리하여, 스파크 어플리케이션의 리소스를 효율적으로 분배
 - StandAlone: 스파크에서 자체적으로 제공하는 클러스터 매니저. 각 노드에서 하나의 익스큐터만 실행 가능
 - Apache Mesos: 아파치 클러스터 매니저. 동적 리소스 공유 및 격리로 여러 소스의 워크로드를 처리
 - Hadoop YARN: 하둡 클러스터 매니저 (리소스 매니저, 노드 매니저)

Spark의 구조

Spark의 동작 구조 (Spark 클러스터 아키텍처)

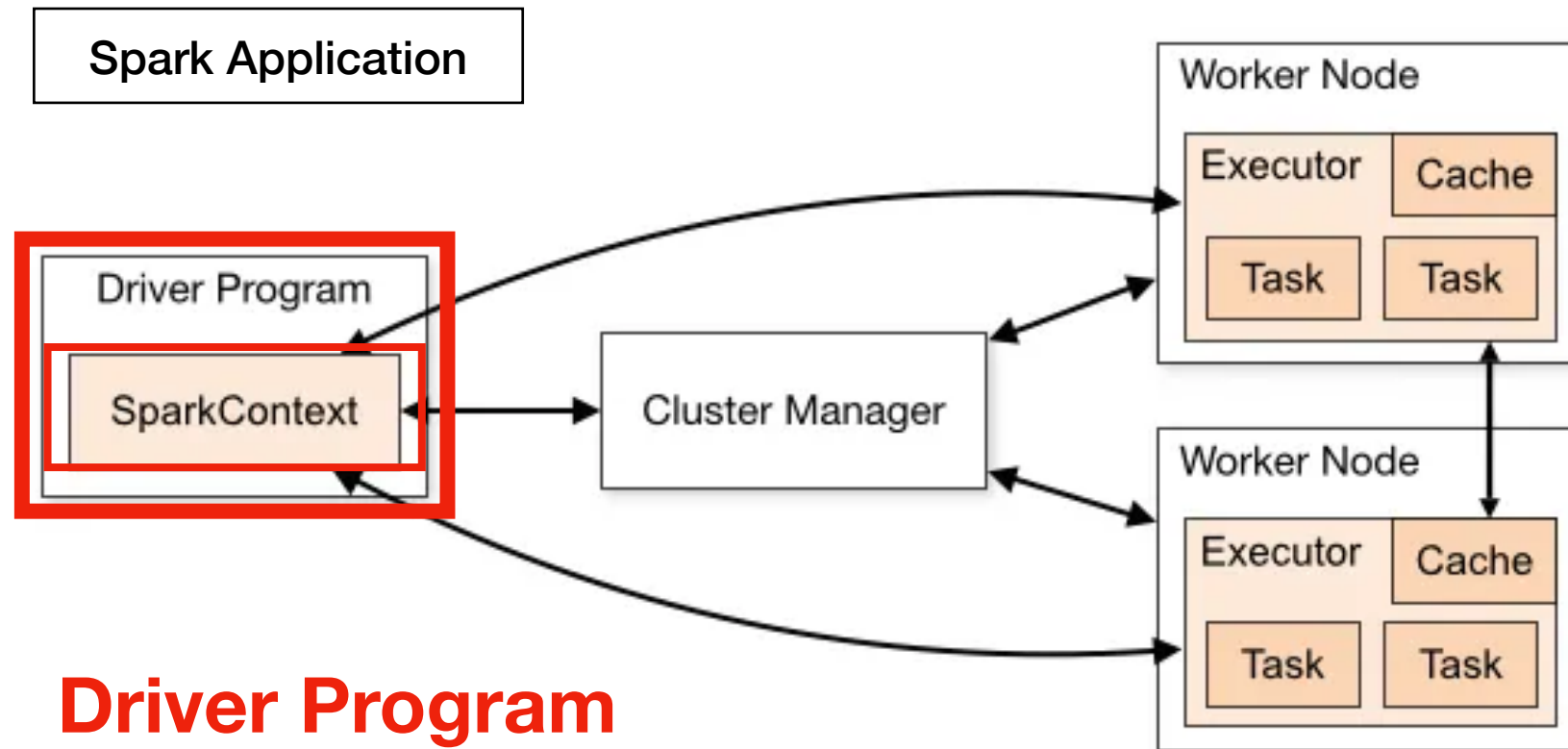


Spark Application

- Driver와 Executor Process로 실행되는 스파크 실행 프로그램
- 실제 일을 수행하는 역할

Spark의 구조

Spark의 동작 구조 (Spark 클러스터 아키텍처)



Driver Program

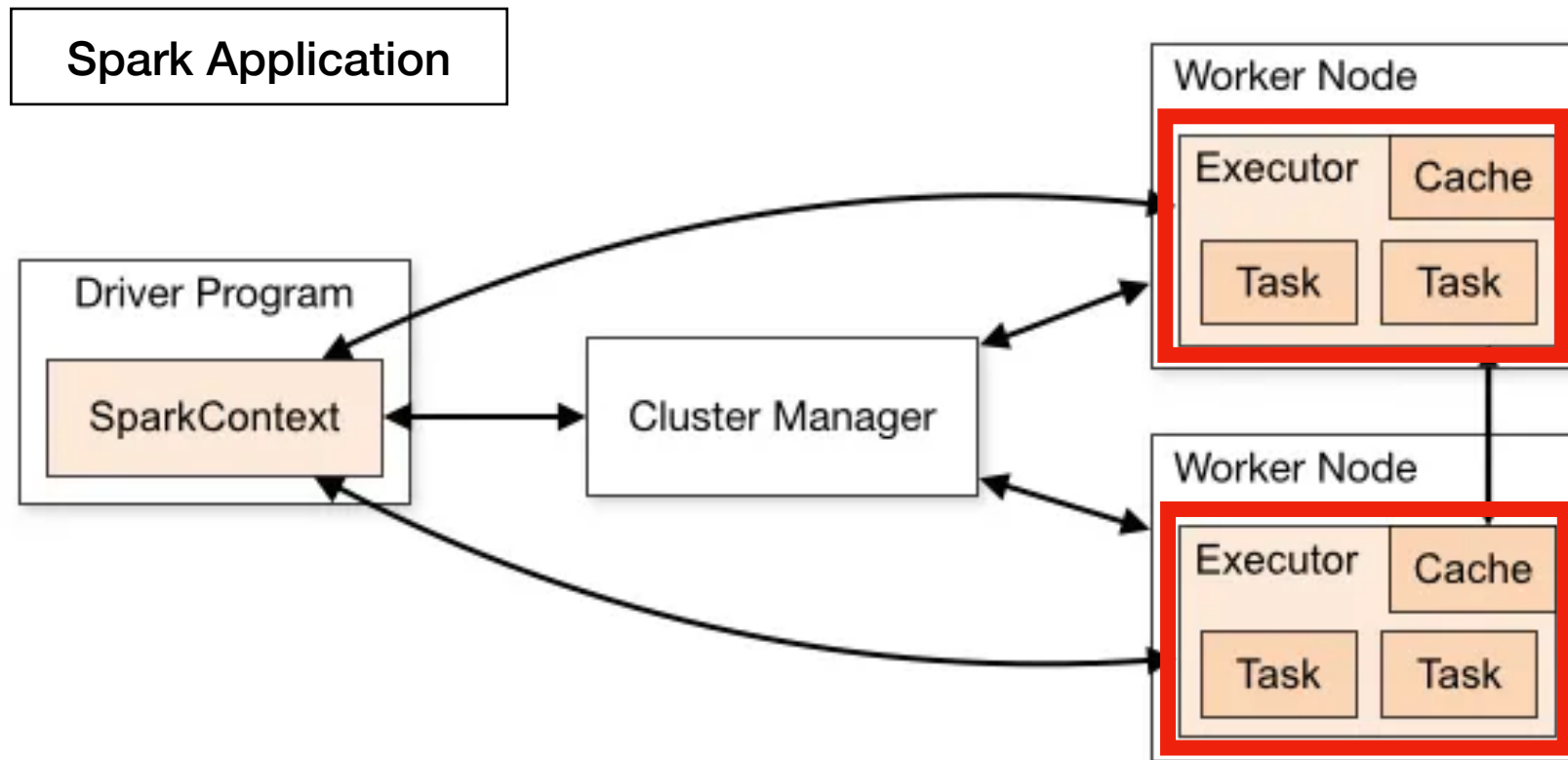
- 하나의 노드 안에서 실행되며, 스파크 전체의 main() 함수를 실행
- SparkContext 객체를 생성
- job(=Spark Application에서 전달한 작업)을 task 단위로 변환하여 Executor로 전달

Spark Context (→ Spark Session)

- 스파크 어플리케이션과 스파크 클러스터 상호작용에 사용되는 인터페이스
- Spark 2.0 이상에서는 SparkSession을 사용

Spark의 구조

Spark의 동작 구조 (Spark 클러스터 아키텍처)

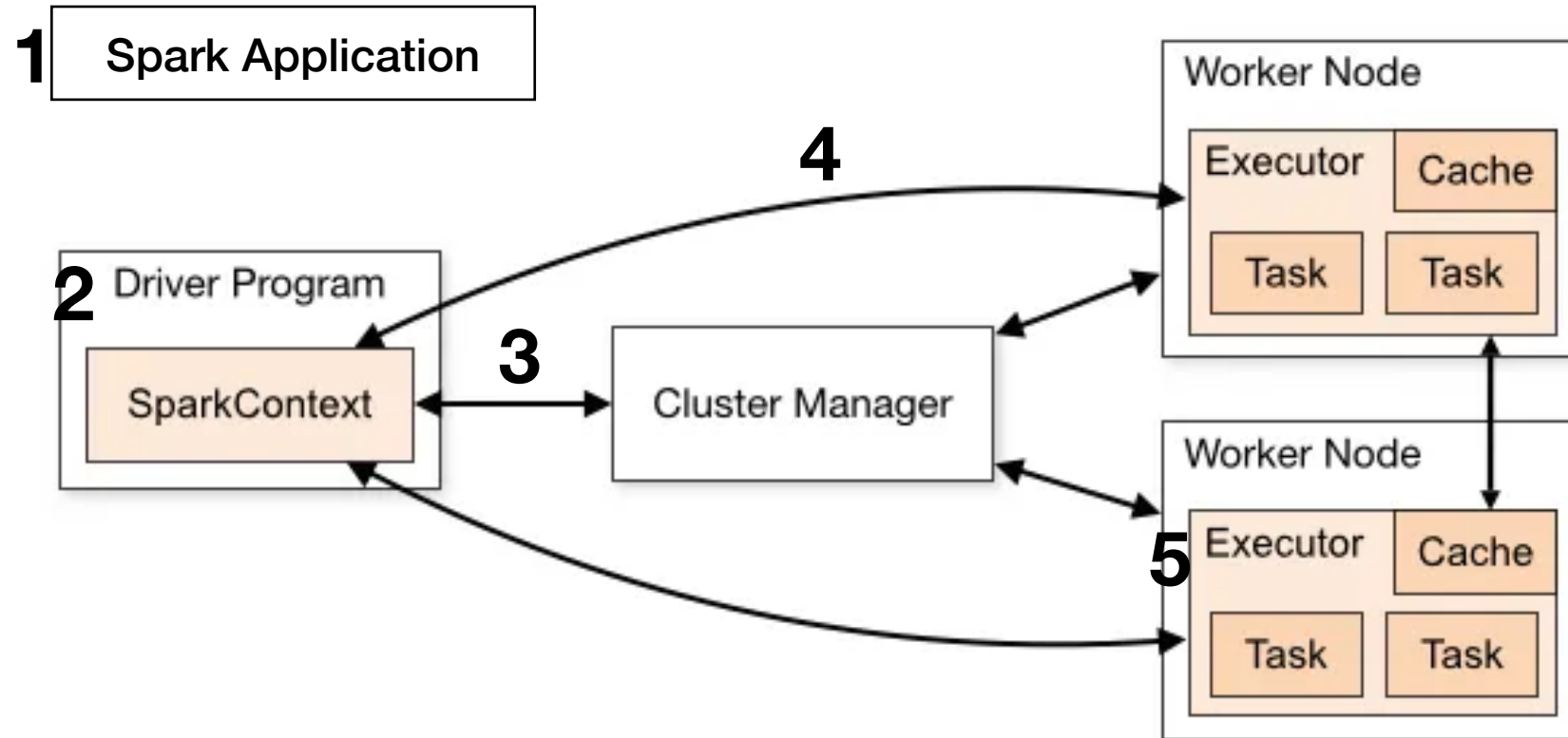


Spark Executor

- 다수의 워커 노드에서 실행
- 드라이버가 할당한 task를 수행하여 결과를 반환
- 블록매니저를 통해 캐시하는 RDD를 저장

Spark의 구조

Spark의 동작 구조 (Spark 클러스터 아키텍처)



1. 사용자가 **Application** 제출 → **Driver program**이 실행됨
2. Driver가 main()을 실행하여 **SparkContext** 생성
3. Driver가 **Cluster Manager**에게 Executor 실행을 위한 리소스 요청
4. SparkContext는 작업 내용(=job)을 task 단위로 분할하여 **Executor**에 전달
5. 각 Executor는 task를 수행하고 결과 저장

3. Spark RDD

Spark RDD

Apache Spark의 3가지 자료구조

- **RDD(Resilient Distributed Dataset)**
 - Apache Spark **1.x** 아키텍처의 기반
 - low-level transformation and control의 경우/schema를 포함하지 않아도 무관한 경우에 사용
- **Data Frame**
 - Spark **1.3.x** 부터는 named column으로 구성된 데이터의 분산 집합인 DataFrame이 등장
 - Spark 내부에서 최적화를 할 수 있는 기능들이 추가. 기존 RDD에 스키마를 부여하고 질의나 API를 통해 데이터를 쉽게 처리할 수 있음
- **DataSet**
 - Spark **2.0**부터는 DataFrame과 DataSet가 Dataset으로 병합되어 데이터 처리를 통합
 - 내부 동작 방식에는 Catalyst Optimizer를 통해 실행 시점에 최적화된 코드를 제공하여, 언어에 무관하게 동일한 성능을 보장
 - 개념적으로 DataFrame은 DataSet[Row]로 간주되며, DataSet의 부분집합으로 볼 수 있음

Spark RDD

Spark RDD란?

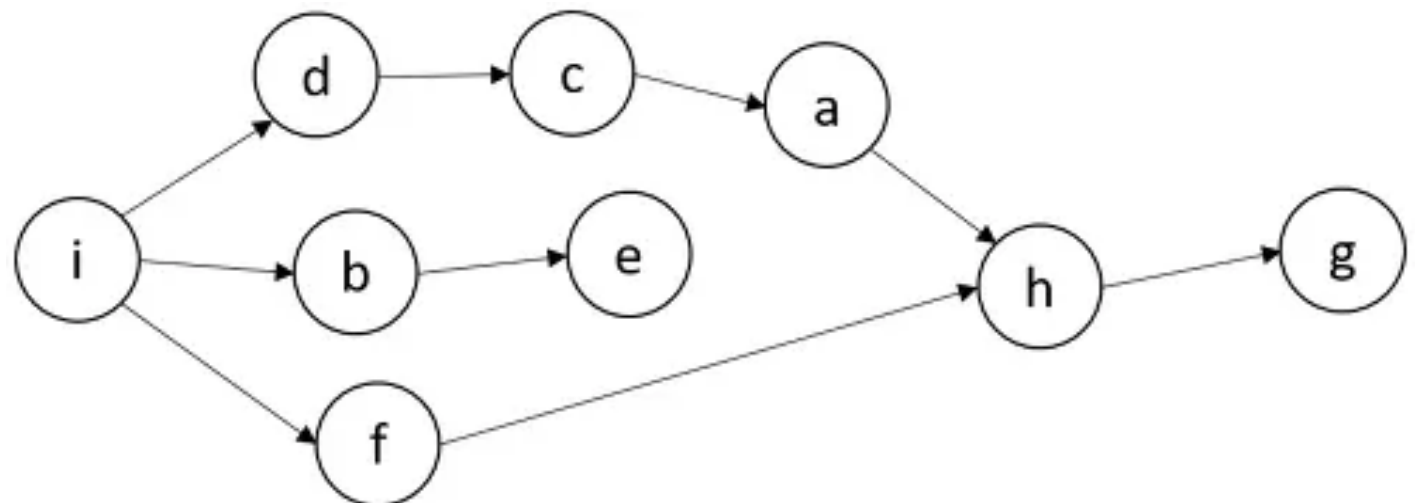
- 스파크에서 쓰는 자료구조 중 하나, 기본이 되는 자료구조
- 복원력 있는(**Resilient**) 분산(**Distributed**) 데이터셋
 - RDD: Resilient Distributed Dataset
- 분산 환경에서 데이터를 처리하는 불변(**immutable**) 컬렉션

Spark RDD

RDD의 특징

Resilient (탄력성)

- 노드 장애 발생 시, 데이터가 손실되지 않도록 복구 가능 (리니지(Lineage) 기반 복구); **Fault-tolerant**
 - 리니지: Spark는 모든 연산 기록(Transformation)을 DAG(Directed Acyclic Graph) 형태로 유지
→ 중간 데이터가 저장되지 않더라도, 원본 데이터와 연산 기록을 이용해 자동으로 다시 계산 가능.

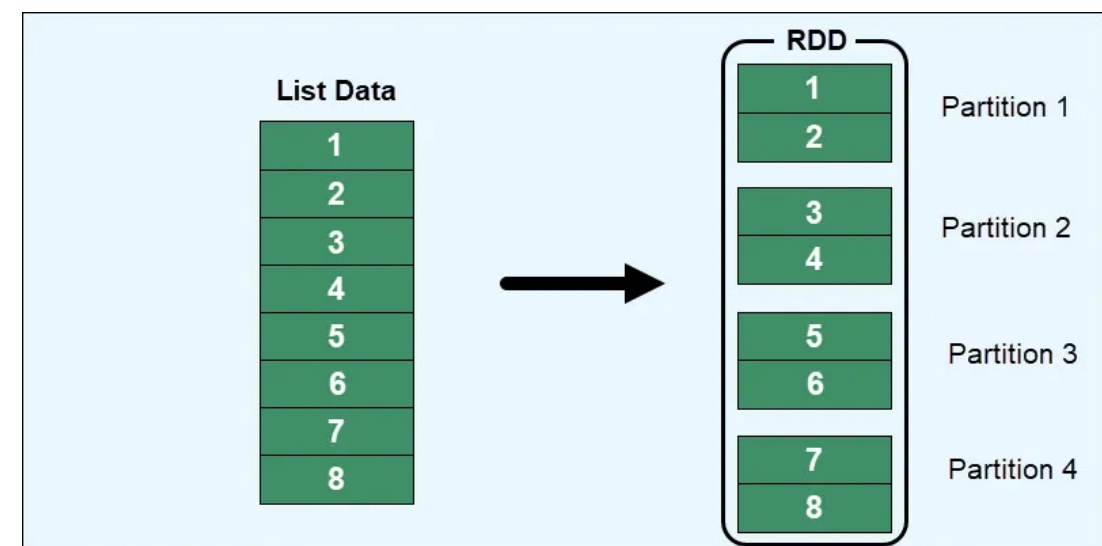


Spark RDD

RDD의 특징

Distributed (분산 처리)

- 여러 클러스터 노드(Worker Node)에 데이터를 분산 저장하여 병렬 연산 수행 → 처리 속도 향상
- RDD는 자동으로 여러 개의 파티션(Partition)으로 나누어 분산 저장됨 (Spark는 파티션 단위로 병렬 연산을 수행하여 성능을 극대화)
 - Partition은 저장의 기본 단위, Task는 실행의 최소 단위. 하나의 Task가 하나의 파티션을 처리



Spark RDD

RDD의 특징

Immutable (불변성)

- 한 번 생성된 RDD는 변경할 수 없음 (Read-only)
- RDD를 변경하려면 새로운 **RDD**를 생성해야 함.



Lazy Evaluation (지연 연산)

- **RDD 연산(Transformation)**은 즉시 실행되지 않고, **Action**이 호출 될 때 실행됨.
- 불필요한 연산을 줄이고 최적화된 실행 계획을 수립 가능.

Spark RDD

RDD의 연산

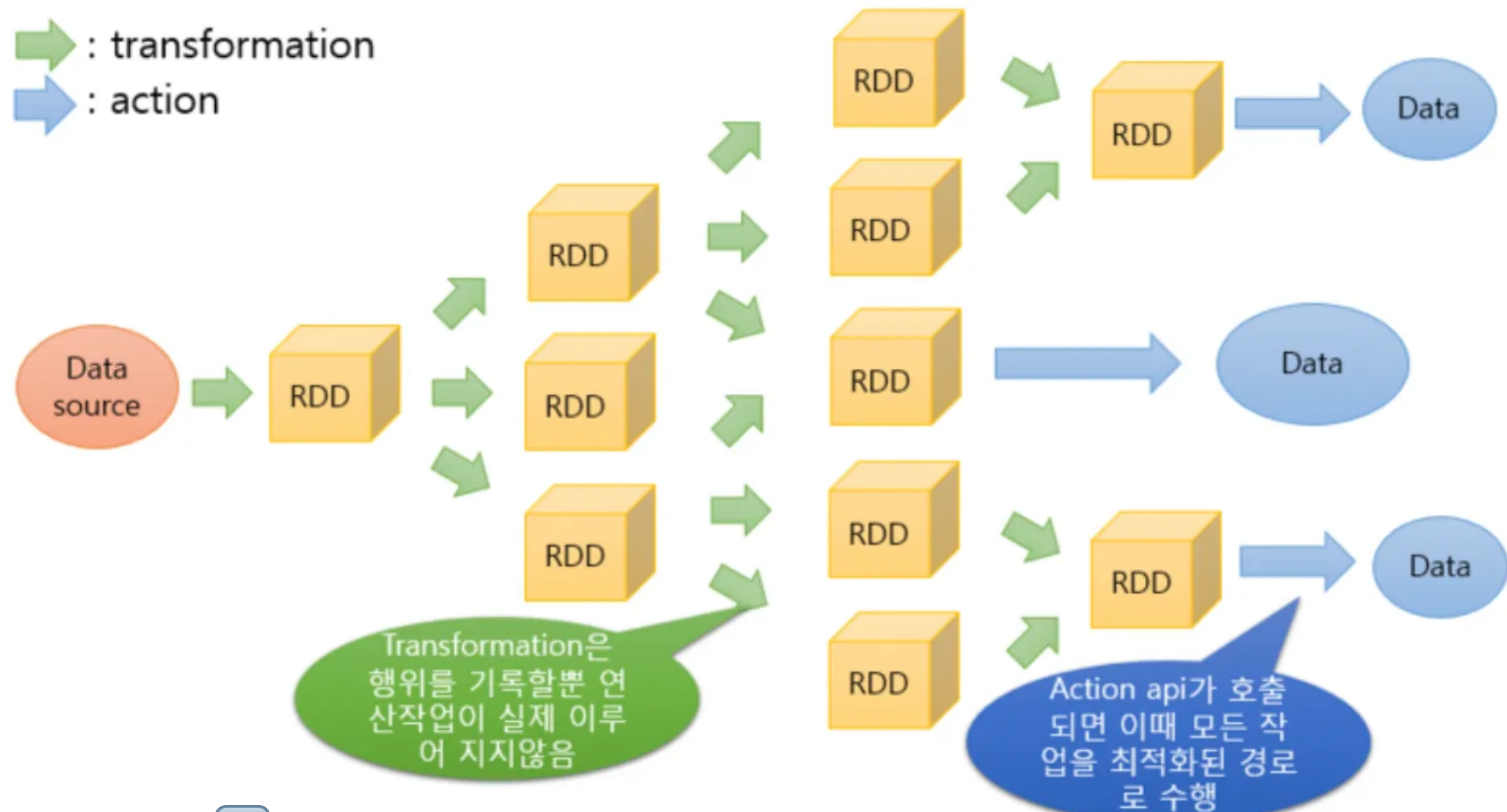
RDD도 다른 자료구조처럼 다양한 기능의 연산용 함수를 지원

-  **Transformation Type**: 기존의 RDD를 입력으로 받아서, 새로운 RDD로 변형하는 연산
 - Action 연산이 호출될 때까지, 연산의 실제 처리는 수행되지 않음
(**Lazy Evaluation**)
-  **Action Type**: RDD에서 최종 결과값을 반환하거나, 외부 저장소에 값을 기록하는 연산

Spark RDD

RDD의 연산

➡ : transformation
➡ : action

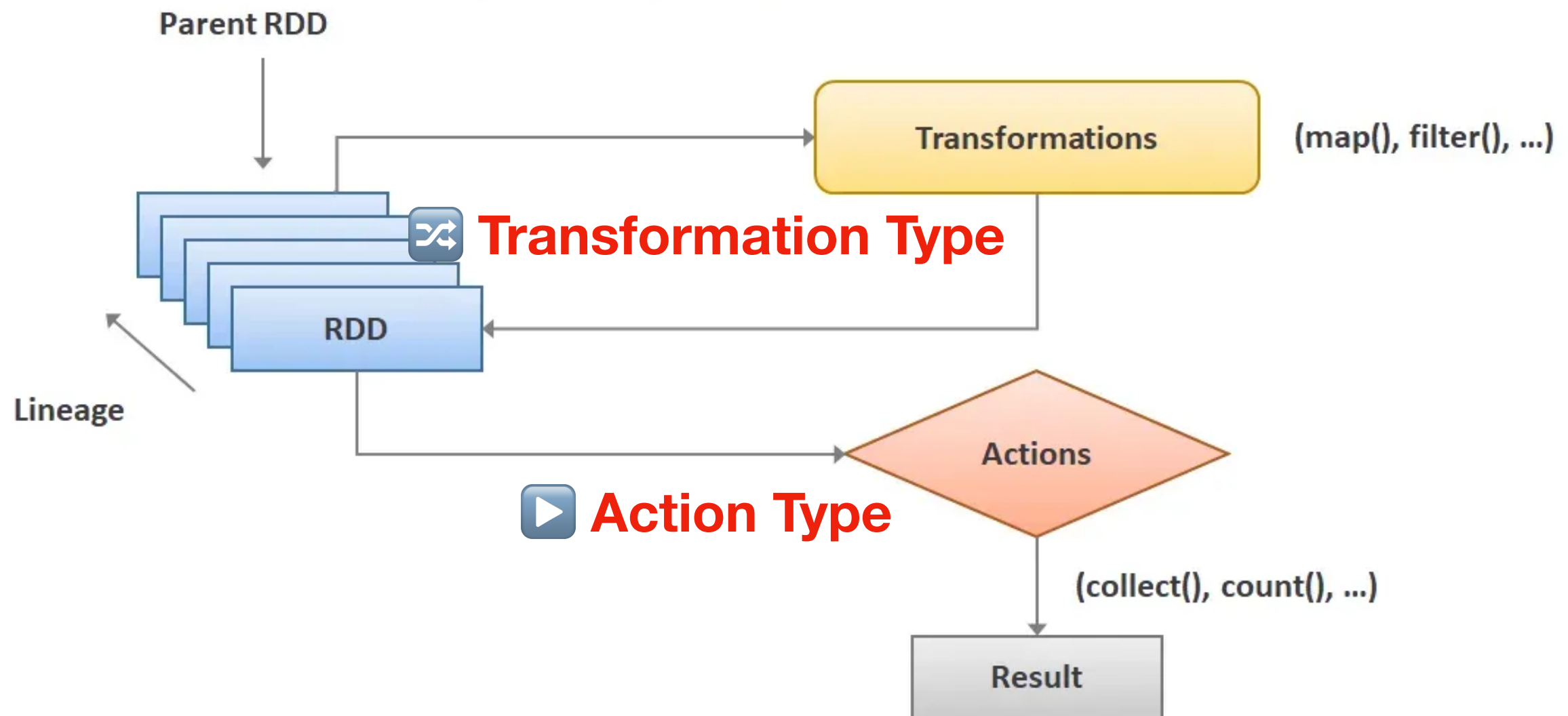


Transformation Type

Action Type


Spark RDD

RDD의 연산



Spark RDD


RDD의 연산

-  **Transformation Type:** 기존의 RDD를 입력으로 받아서, 새로운 RDD로 변형하는 연산

연산자	설명	예제
map(f)	각 요소에 함수 f를 적용하여 새로운 RDD 생성	<code>rdd.map(lambda x: x * 2)</code>
filter(f)	조건 f를 만족하는 요소만 남김	<code>rdd.filter(lambda x: x > 5)</code>
flatMap(f)	각 요소를 여러 개의 요소로 변환	<code>rdd.flatMap(lambda x: x.split(" "))</code>
distinct()	중복 제거된 새로운 RDD 생성	<code>rdd.distinct()</code>
union()	두 RDD를 합침	<code>rdd1.union(rdd2)</code>
intersection()	두 RDD의 교집합 반환	<code>rdd1.intersection(rdd2)</code>
groupByKey()	키를 기준으로 그룹화	<code>rdd.groupByKey()</code>
reduceByKey(f)	같은 키를 가진 요소를 병합	<code>rdd.reduceByKey(lambda a, b: a + b)</code>
sortBy(f)	함수 f에 따라 요소 정렬	<code>rdd.sortBy(lambda x: x)</code>
join()	두 RDD를 키를 기준으로 조인	<code>rdd1.join(rdd2)</code>

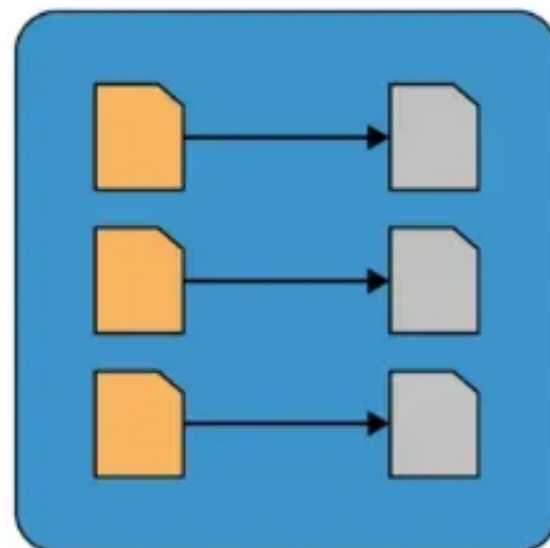
Spark RDD

RDD의 연산

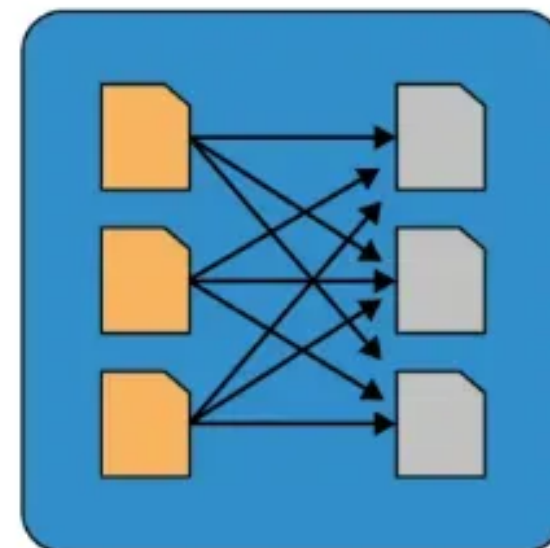
-  **Transformation Type**: 기존의 RDD를 입력으로 받아서, 새로운 RDD로 변형하는 연산

	Narrow Transformation	Wide Transformation
데이터 이동 여부	파티션 내부에서만 처리	파티션 간 데이터 이동 (Shuffle 발생)
예시 연산	<code>map()</code> , <code>filter()</code> , <code>flatMap()</code> , <code>union()</code>	<code>reduceByKey()</code> , <code>groupByKey()</code> , <code>join()</code> , <code>intersection()</code>

Narrow Dependencies




Wide Dependencies



Spark RDD

RDD의 연산

-  **Transformation Type**: 기존의 RDD를 입력으로 받아서, 새로운 RDD로 변형하는 연산

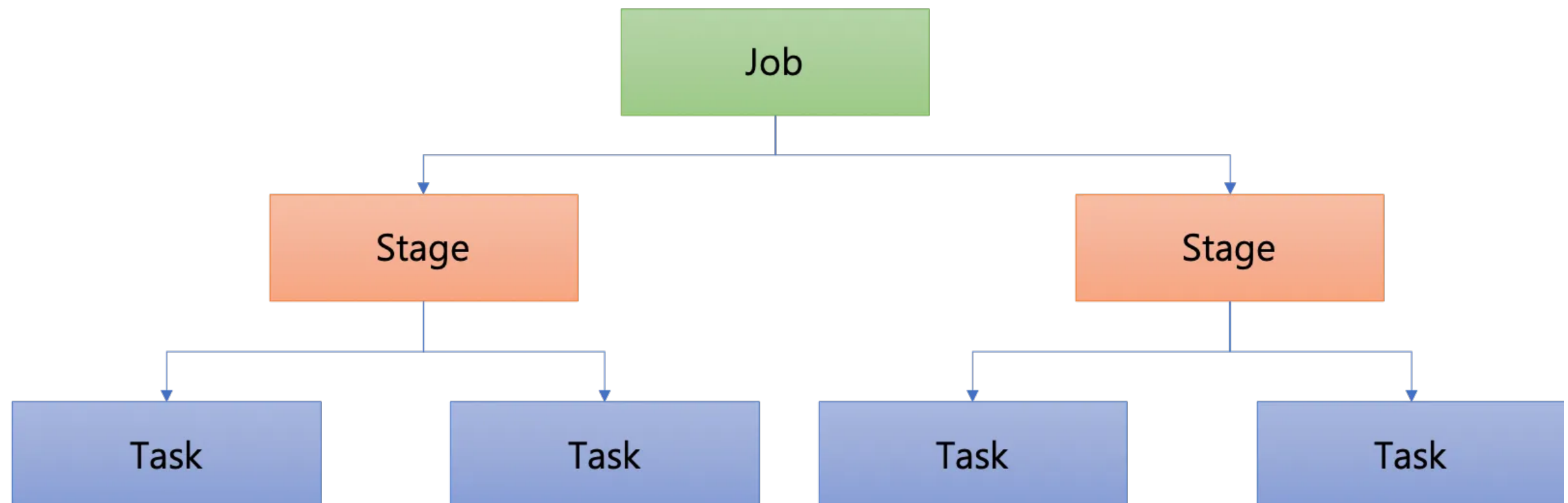
	Narrow Transformation	Wide Transformation
데이터 이동 여부	파티션 내부에서만 처리	파티션 간 데이터 이동 (Shuffle 발생)
예시 연산	<code>map()</code> , <code>filter()</code> , <code>flatMap()</code> , <code>union()</code>	<code>reduceByKey()</code> , <code>groupByKey()</code> , <code>join()</code> , <code>intersection()</code>
속도	빠름	상대적으로 느림
Stage 구분	동일 Stage에서 처리 가능	새로운 Stage로 나뉨

- Shuffle(파티션 간 데이터 이동)이 발생하면 비용이 증가되므로,
Wide Transformation의 사용을 최소화하는 것이 좋음
- **Stage**: Job을 Wide Transformation(Shuffle 발생)을 기준으로 나눈 실행 단위

Spark RDD

RDD의 연산

- Job: Action(collect(), count())이 실행될 때 생성되는 작업 단위
- **Stage**: Job을 Wide Transformation(Shuffle 발생)을 기준으로 나눈 실행 단위
- Task: Stage가 여러 개의 파티션(Task)으로 나뉘어 실행됨



Spark RDD


RDD의 연산

- ▶ **Action Type:** RDD에서 최종 결과값을 반환하거나, 외부 저장소에 값을 기록하는 연산

연산자	설명	예제
collect()	모든 데이터를 리스트로 반환	<code>rdd.collect()</code>
count()	RDD의 요소 개수 반환	<code>rdd.count()</code>
first()	첫 번째 요소 반환	<code>rdd.first()</code>
take(n)	처음 n개의 요소 반환	<code>rdd.take(3)</code>
reduce(f)	모든 요소를 하나로 병합	<code>rdd.reduce(lambda a, b: a + b)</code>
takeOrdered(n)	정렬 후 처음 n개의 요소 반환	<code>rdd.takeOrdered(3)</code>
saveAsTextFile(path)	데이터를 텍스트 파일로 저장	<code>rdd.saveAsTextFile("/path")</code>
foreach(f)	각 요소에 함수 f를 적용	<code>rdd.foreach(print)</code>

Spark RDD

RDD의 연산

-  **Action Type**: RDD에서 최종 결과값을 반환하거나, 외부 저장소에 값을 기록하는 연산
 - Action 연산이 호출되면 앞선 모든 Transformation 연산들이 차례대로 수행됨
 - **캐싱을 통해** 반복적인 연산에 대해 데이터를 다시 계산하지 않고, 캐시된 결과를 사용하여 빠르게 처리

실습! & 과제

실습 상세안내

[https://
yonseibigdata.notion
.site/Spark-RDD-
bc975f1643dd4ca0b
b6f58996f7a48dd?
pvs=4](https://yonseibigdata.notion.site/Spark-RDD-bc975f1643dd4ca0bb6f58996f7a48dd?pvs=4)

DE 구글 드라이브

[https://
drive.google.com/
drive/folders/
1bEmxuQj0lYfdeqh2
ptOyNoVvGvP84qsN
?usp=sharing](https://drive.google.com/drive/folders/1bEmxuQj0lYfdeqh2ptOyNoVvGvP84qsN?usp=sharing)

Reference

DE팀 22기 김진형 선배님, 23기 오재현 선배님 발제자료

<https://velog.io/@dddwsd/Hadoop-ecosystem0|란>

<https://wikidocs.net/26630>

<https://bomwo.cc/posts/spark-rdd/>

<https://mangkyu.tistory.com/128>

<https://medium.com/analytics-vidhya/spark-rdd-low-level-api-basics-using-pyspark-a9a322b58f6>

<https://phoenixnap.com/kb/resilient-distributed-datasets>

<https://artist-developer.tistory.com/17?category=962892>

끝