Techtonic 2018

-

Thu . Nov 15

-

SAMSUNG SDS Tower
West Campus B1F
Magellan Hall /Pascal Hall

SAMSUNG SDS
Realize your vision

Spark Cluster 어디까지 써 봤니?

클러스터 리소스 최적화를 위한 Spark 아키텍처

삼성SDS 노광선 프로 (분석플랫폼 Lab)

Techtonic 2018

Agenda

- Spark 특징 및 아키텍처
- SDS의 Spark 활용법
- Spark 리소스 관리
- Demo

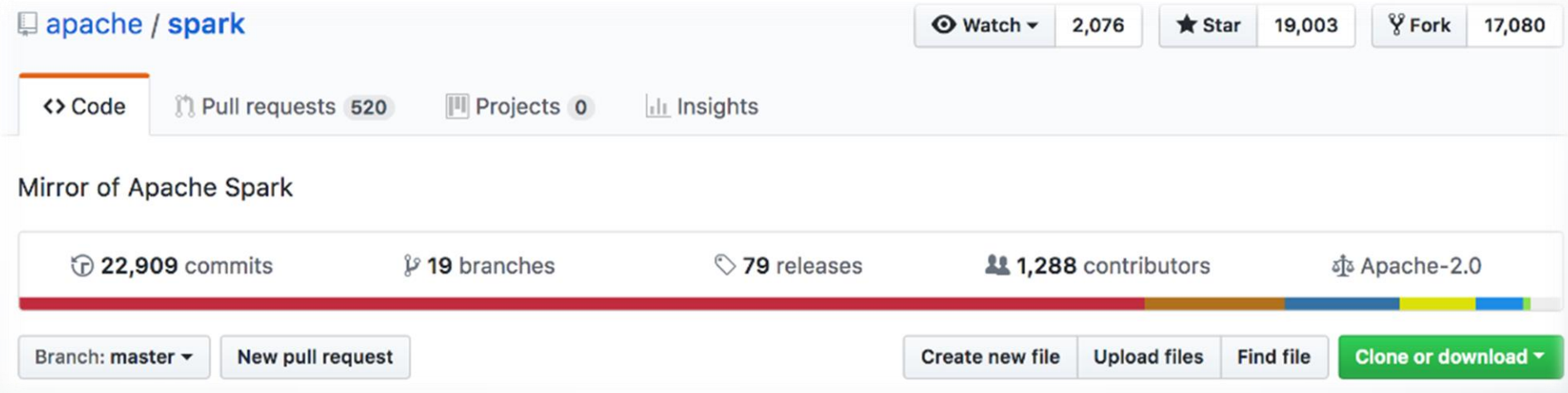
클러스터 리소스 최적화를 위한 Spark 아키텍처

Spark 특징 및 아키텍처

Spark

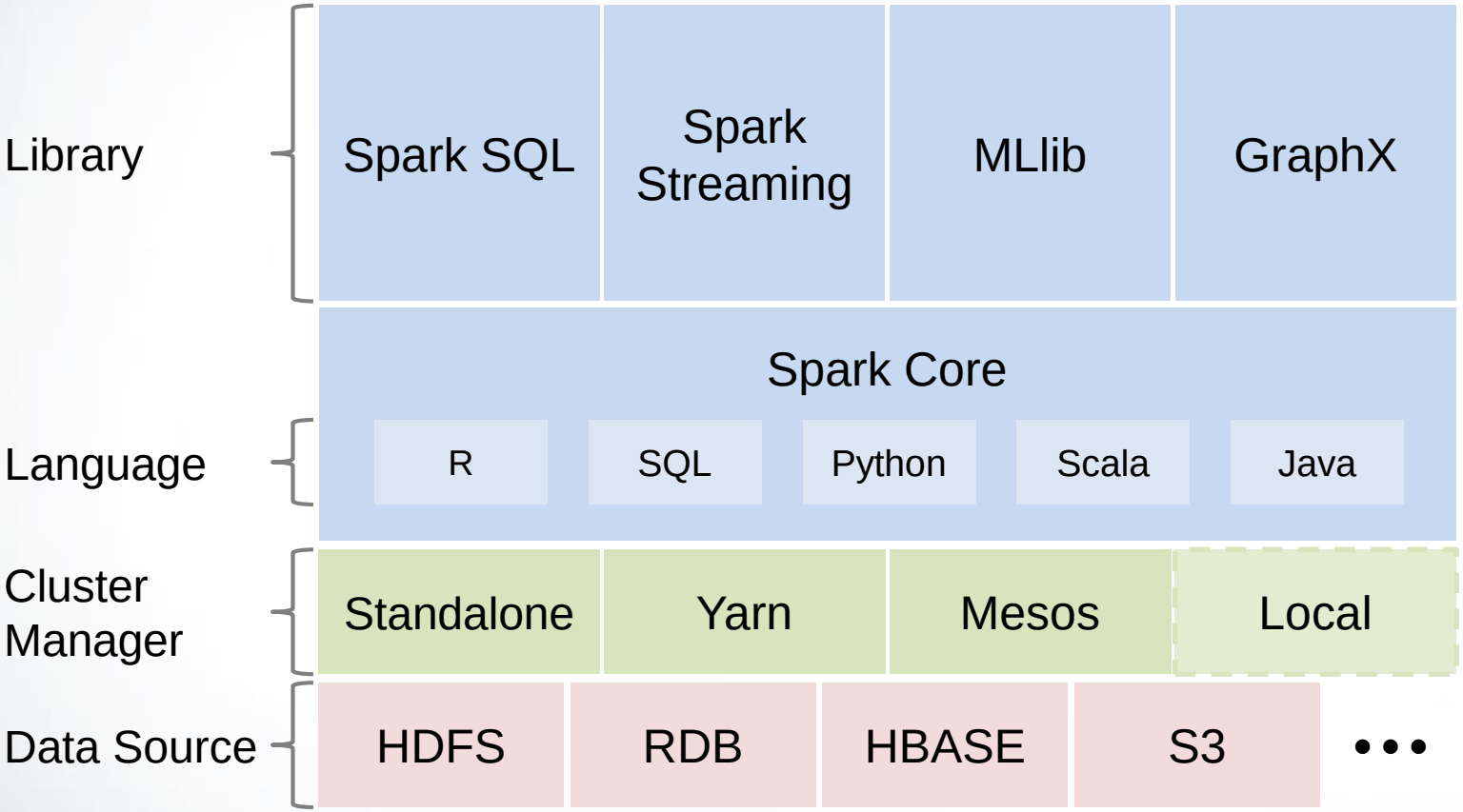
대용량 데이터 처리를 위한 통합 분석 엔진

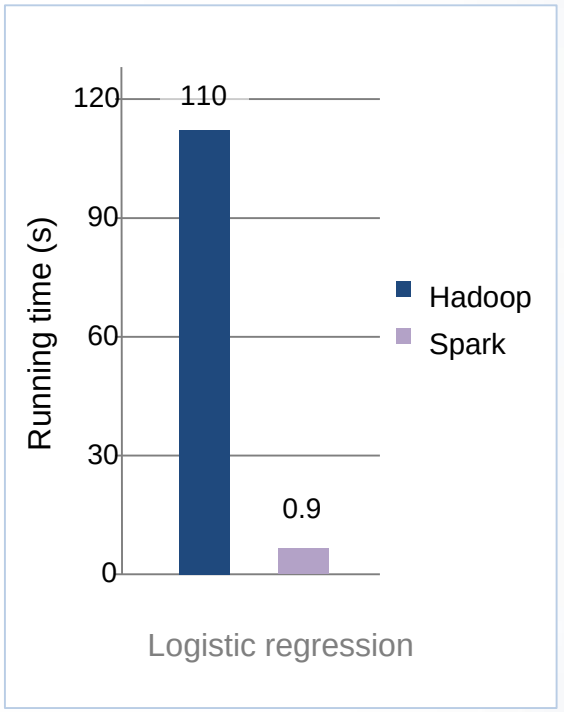
- **In-memory 클러스터 컴퓨팅 프레임워크**
- Hadoop MapReduce 대비 Machine Learning 등 반복 작업에 특화
- 2009년, UC Berkeley AMPLab에서 Mesos 어플리케이션으로 시작
- 2010년 Spark 논문 발표, 2012년 RDD 논문 발표
- 2013년에 Apache 프로젝트로 전환 후, **2014년 Apache Top-Level Project**
- **2018년 11/12 기준, v2.4.0 released**



https://github.com/apache/spark

Key Features - 유연하고 통합된 분석 환경, 빠른 속도





Key Features – 개발 편

Word Count 예제

Hadoop using Java

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Spark using Scala

```
val textFile = sc.textFile("hdfs://path/to/file")
val counts = textFile.flatMap(line => line.split("\\s+"))
                     .map(word => (word, 1))
                     .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://path/to/output")
```

I : {2}

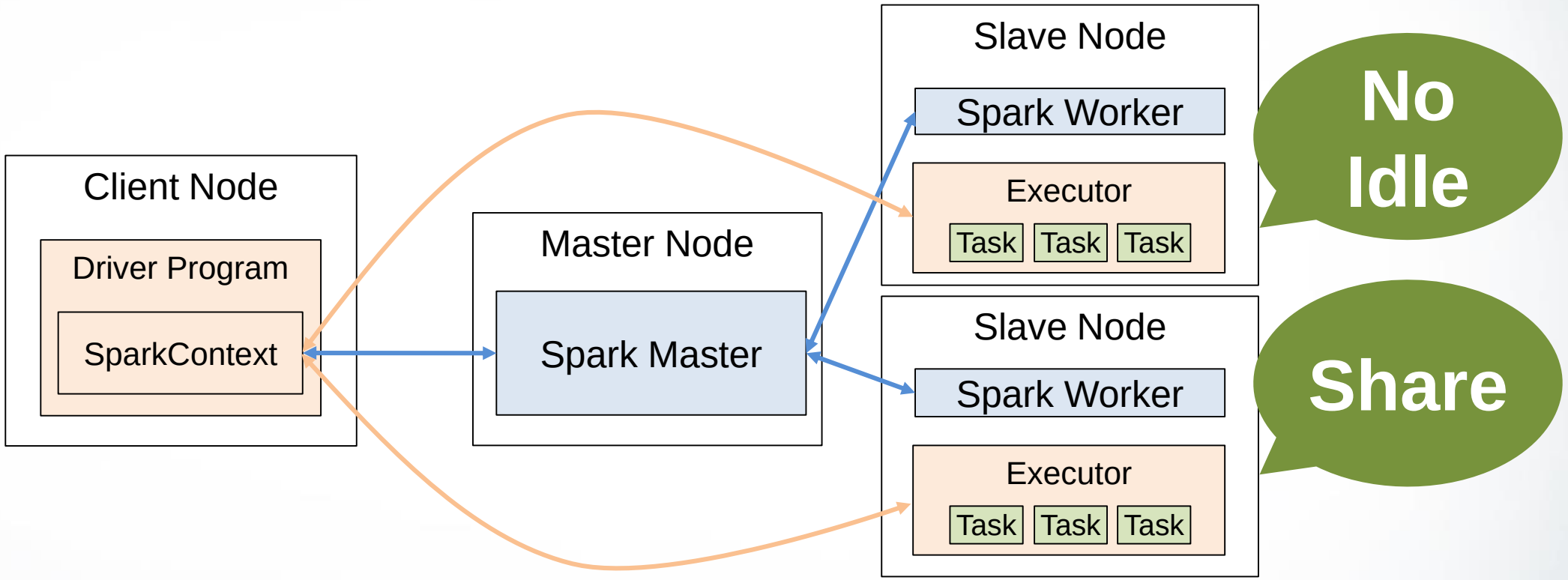
am : {2}

girl : {1}

happy: {2}

reduce

Architecture - Standalone mode

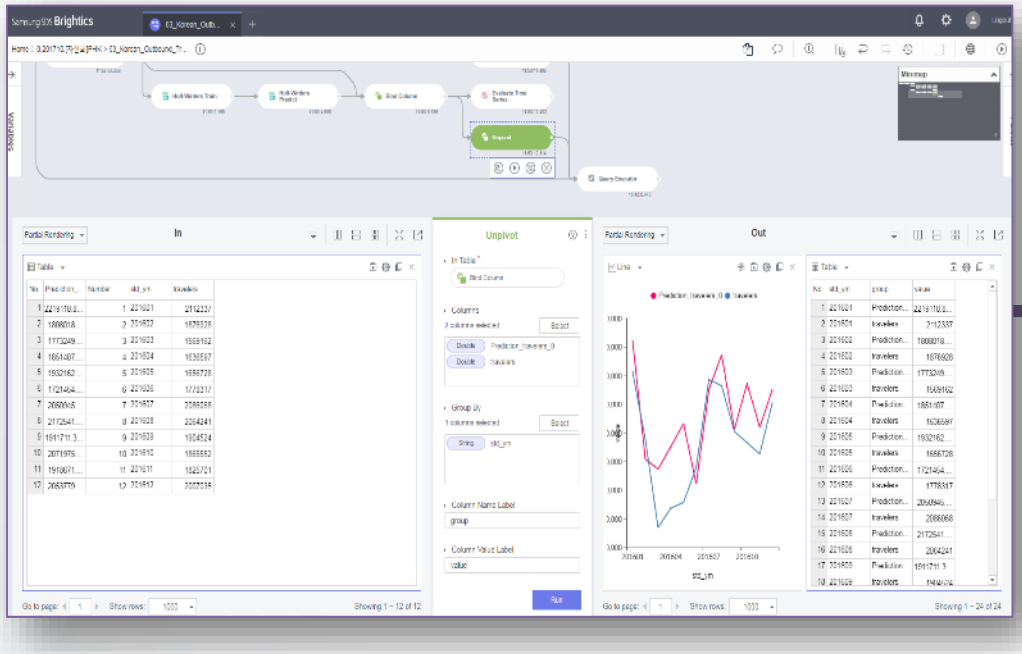


클러스터 리소스 최적화를 위한 Spark 아키텍처

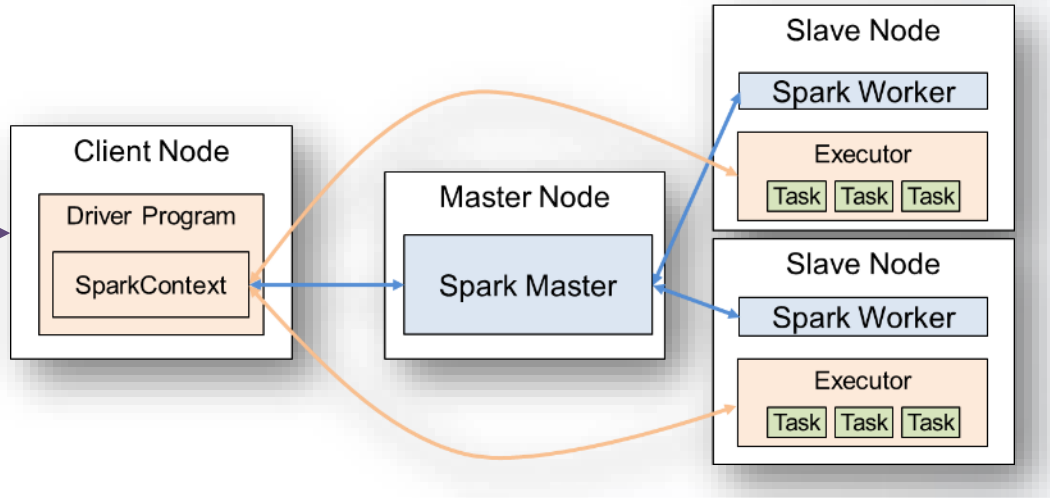
SDS의 Spark 활용법

Brightics AI with Spark

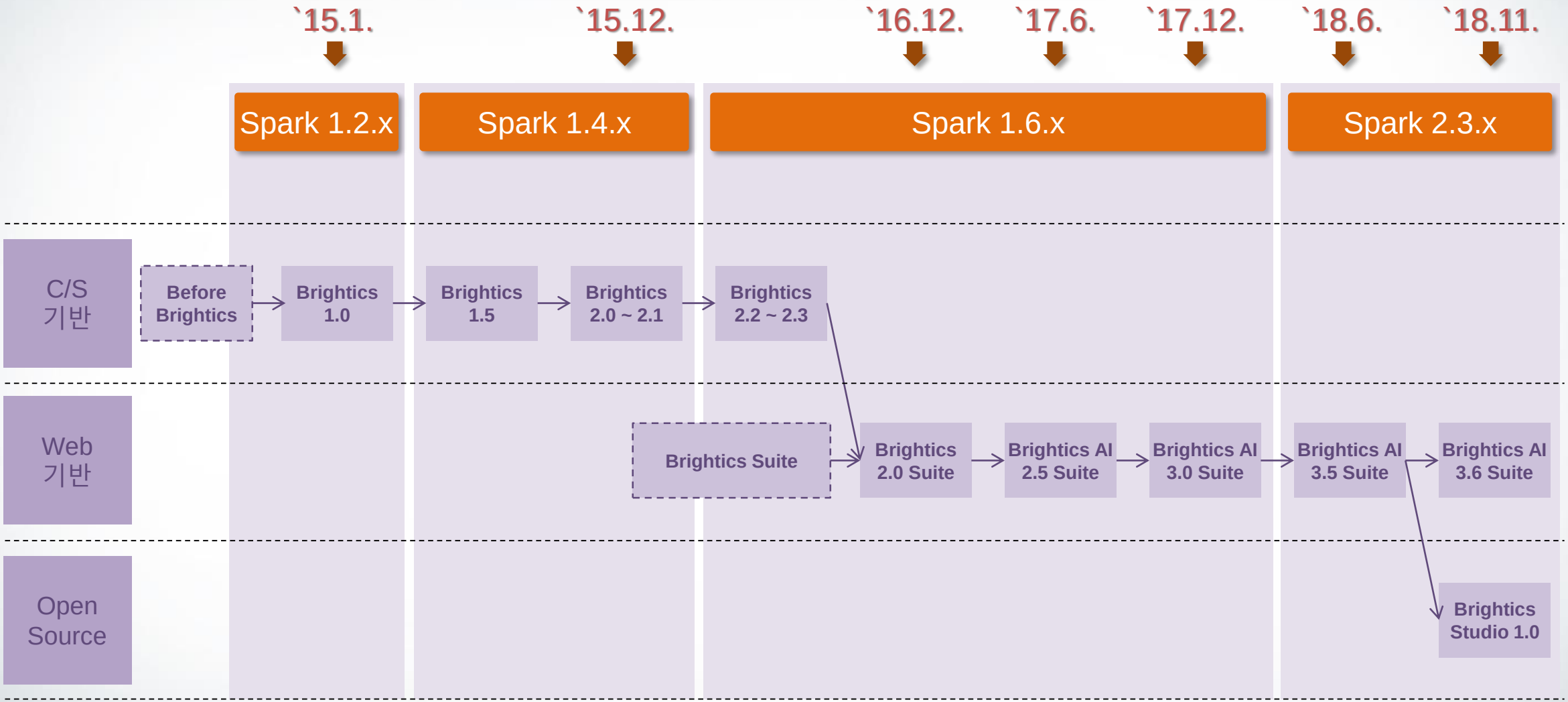
Brightics AI



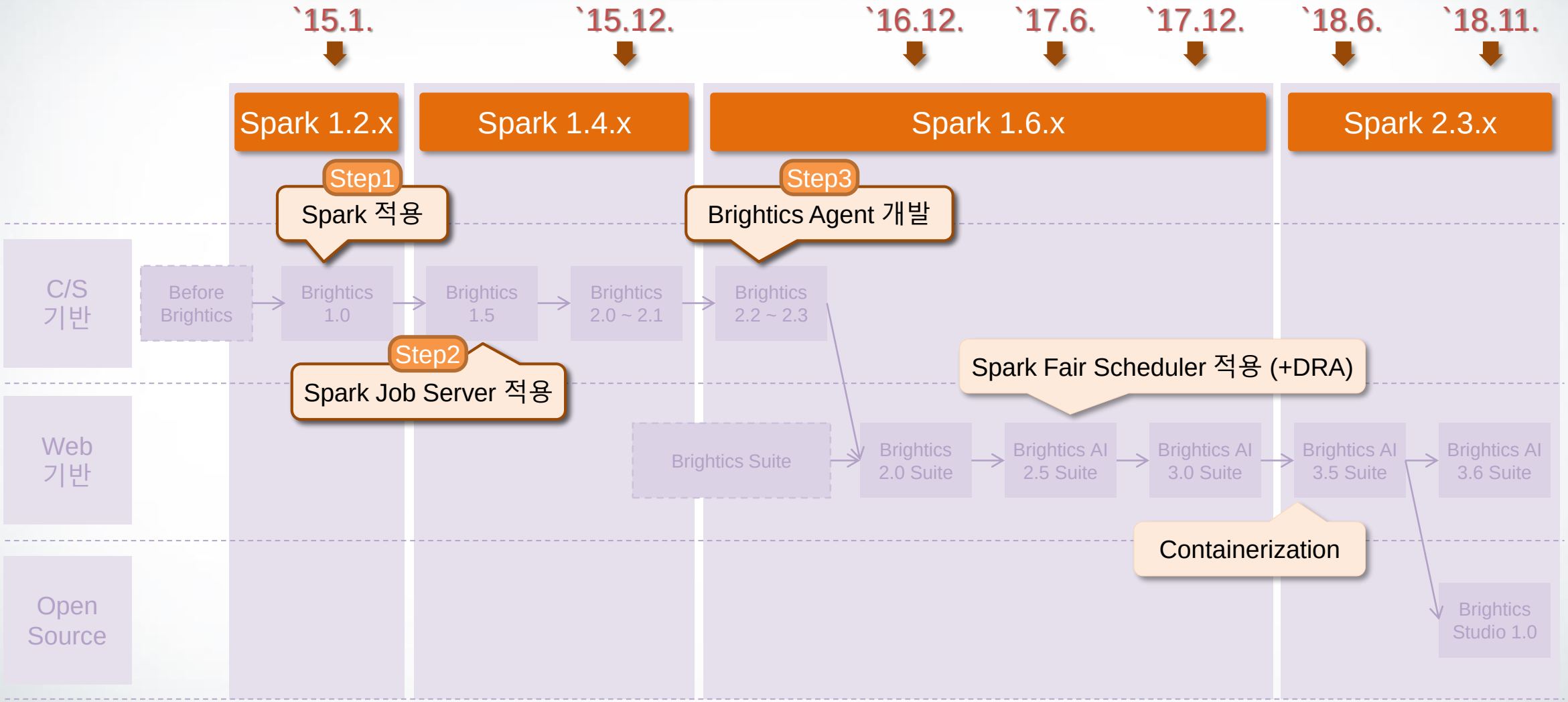
Spark Cluster



Brightics AI with Spark

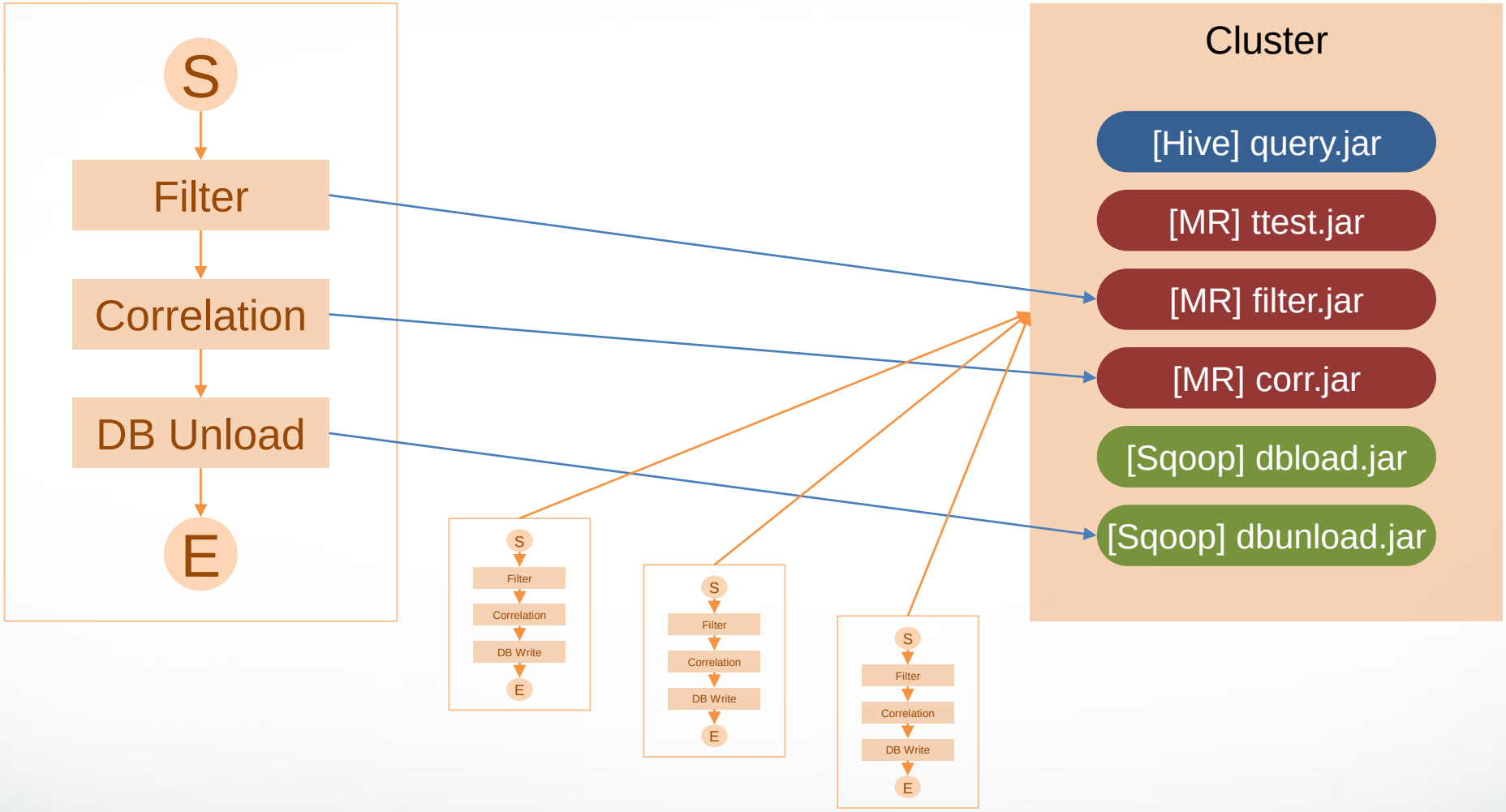


Brightics AI with Spark



Step1. Spark - 배경

Workflow 기반으로 모델링 수행. 각 함수 Application은 사전에 Jar 형태로 배포



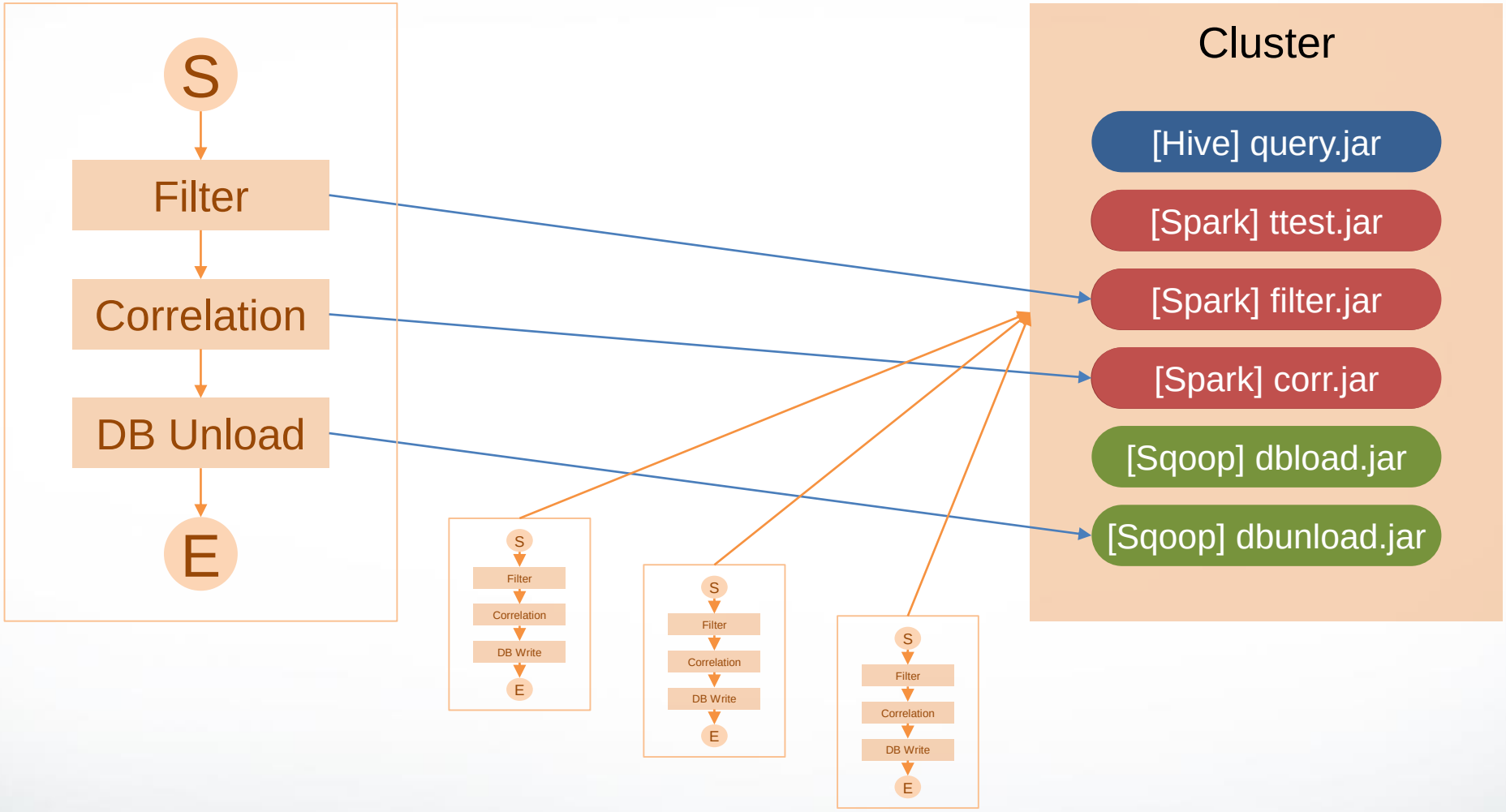
Step1. Spark – 적용 전 사용자 반응

"굉장히 많은 데이터를 분석할 수 있지만,

시간이 너무 오래 걸려요."

Step1. Spark - 적용

Hadoop MapReduce를 Spark으로 전환



Step1. Spark - 사용자 반응

"**분석 시간**을 굉장히 많이 **단축**했어요."

"이제 전 공정을 **한번에 분석**할 수 있어요."

Step1. Spark - 사용자 반응

성능

"단순한 함수도 **10초 이상 기다려야** 해요."

"누군가 작업 중에, **Spark를 사용할 수 없어요**."

리소스

Step1. Spark - 사용자 반응

성능

"단순한 함수도 **10초 이상 기다려야** 해요."

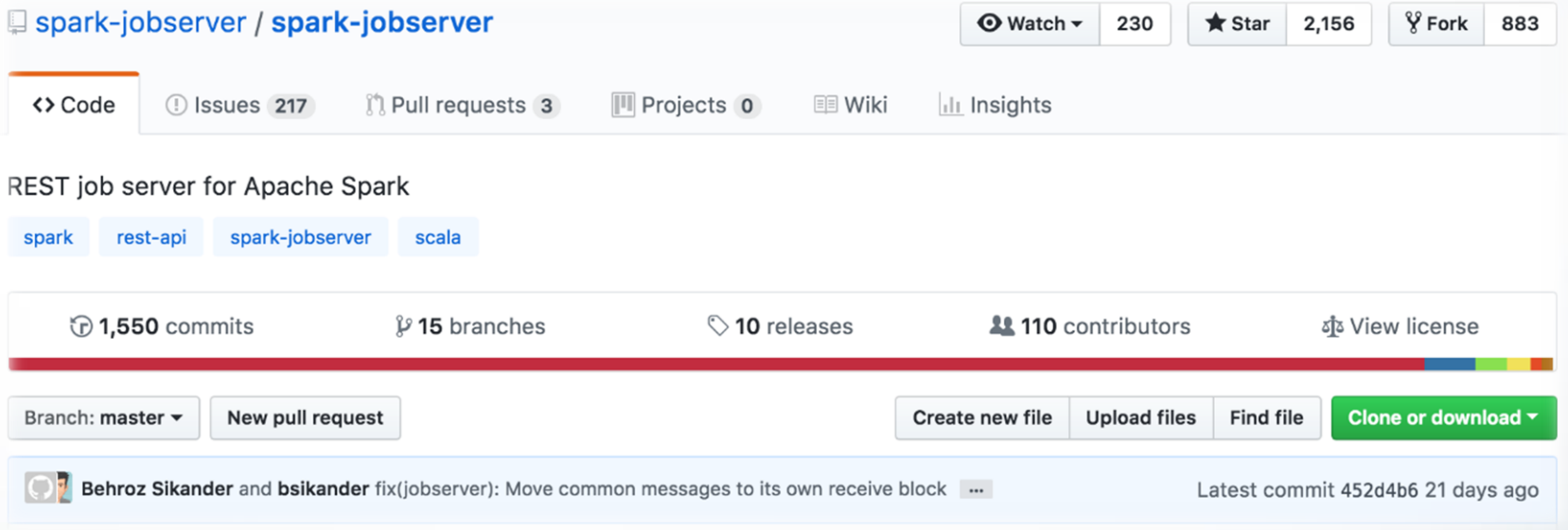
새로운 Spark Job 실행 환경 필요

리소스

Step2. Spark Job Server

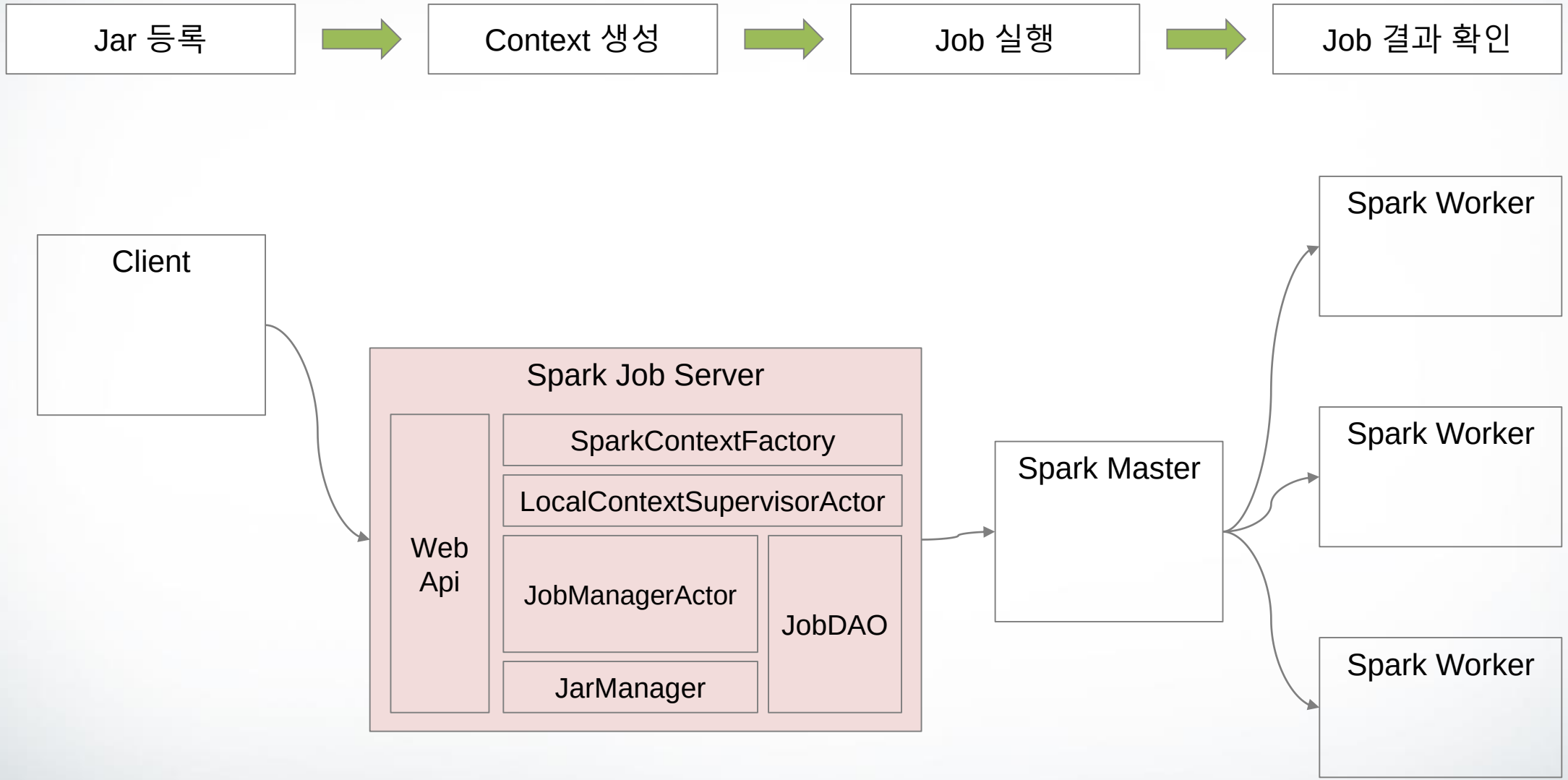
Spark Job 실행을 위한 Rest API 서버

- 미디어 플랫폼 회사 Ooyala에서 공개한 오픈소스 프로젝트
- **Spark Job, Jar, Context를 관리할 수 있는 Rest API 제공**
- Spark Context 유지시키면서, 여러 Job이 공유할 수 있게 함
- Scala, Spray, Slick 등의 기술로 개발
- 2018년 10/31 기준, v0.8.0 released

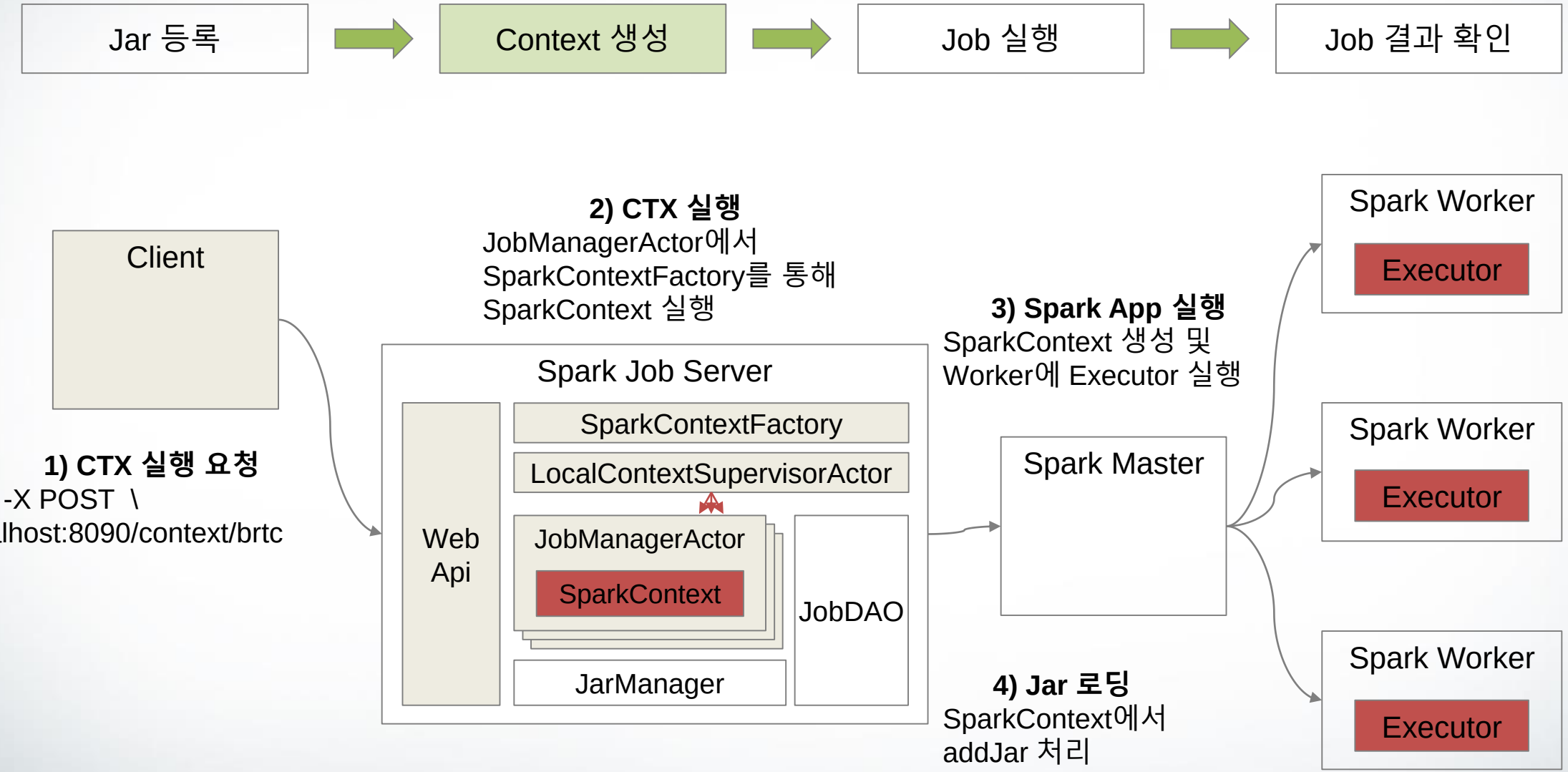


https://github.com/spark-jobserver/spark-jobserver

Step2. Spark Job Server - 동작 방식



Step2. Spark Job Server - 동작 방식



Step2. Spark Job Server - 효과

Spark Application 실행에 있어서 Low Latency 확보

GOOD

- 초기 실행 시간 단축 : 약 8s → 약 1s
- App간 Data 유지 가능하여,
  분석 시간 단축
- Workflow 및 Script 방식으로
  Interactive한 분석 환경 제공 가능

BAD

- 제공 함수보다 Script 기반으로
  분석 시도
- 추가된 관리 포인트

Step2. Spark Job Server - 사용자 반응

"모델 실행이 **안돼요**."

"Spark Job Server가 **멈췄어요**."

"Spark가 **죽어요**."

Step2. Spark Job Server - Long Term Context의 한계

Spark Job Server 에서 지속적으로 OOM 발생 이슈 발견

제공되는 분석 함수보다는 직접 Script 실행을 원함

▼

Script 기능 개발 이후, **많은 사용자들이 Script 를 요청**

▼

Spark Job Server 를 재기동하기 전까지는 **memory 사용량이 증가**

▼

Spark Job Server (Driver) 에서 **OOM 발생**

Step2. Spark Job Server - Long Term Context의 한계

Spark Job Server 에서 지속적으로 OOM 발생 이슈 발견

제공되는 분석 함수보다는 직접 Script 실행을 원함

▼

Spark Job Server 제거 ➔ Job실행 Agent 직접 구현

▼

Spark Job Server (Driver) 에서 **OOM 발생**

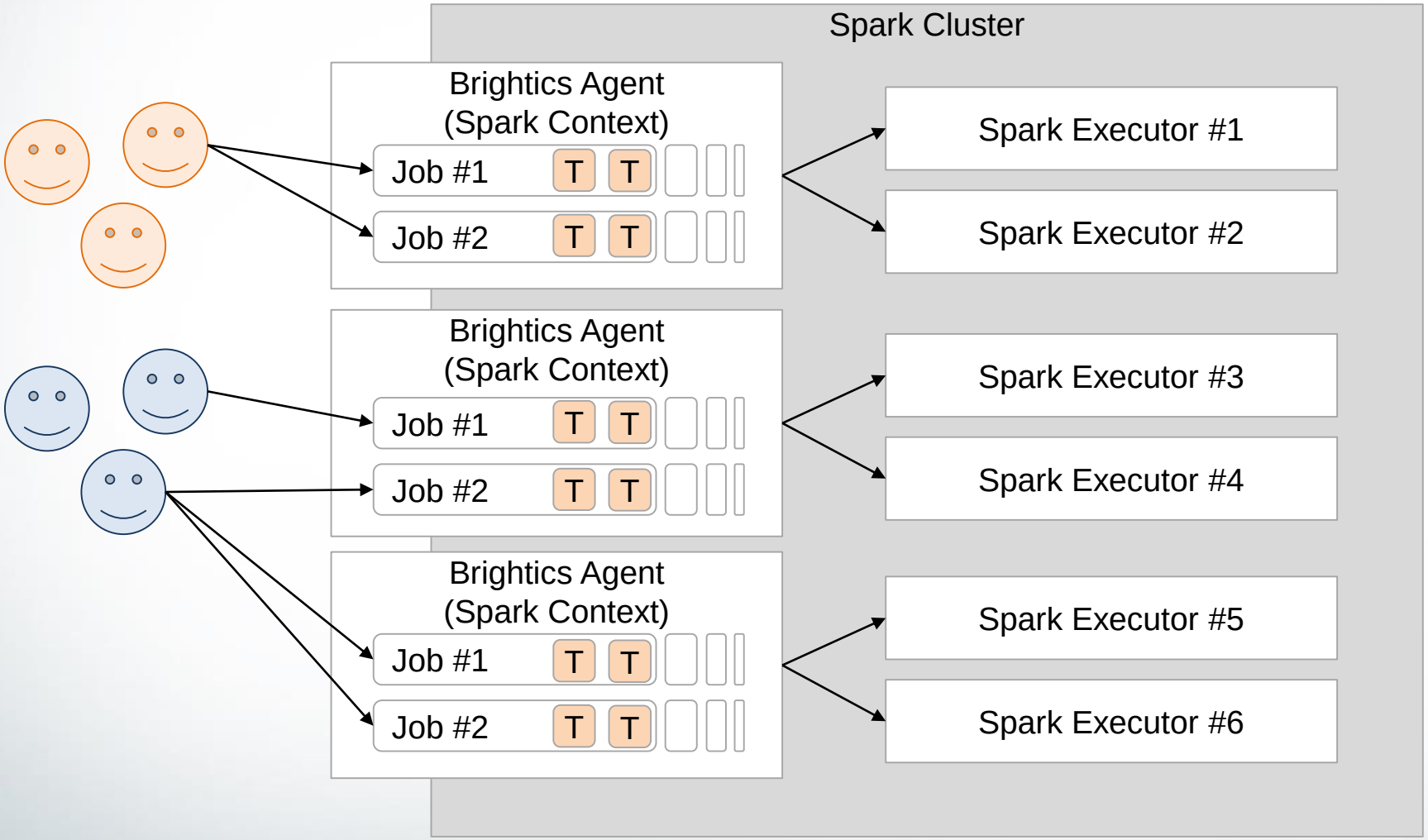
Step3. Brightics Agent

Spark Job Server 대체를 위해 Brightics Agent 개발

- Script 해석 : SparkILoop 인터프리터 방식 ➔ **FSC 컴파일** 방식
- Script 실행 : Spark Job으로 구현 ➔ Agent 자체 로직으로 변경
- Jar등록 방식 : 사용자가 Jar Upload 방식 ➔ Agent 자체에서 addJar
- SparkContext 관리 : 다중 SparkContext ➔ **단일 SparkContext**
- 경량화 : 구조 단순화. 불필요 로직 제거. **18,160K ➔ 146K**

Step3. Brightics Agent - 다중 Context를 활용한 아키텍처 구성

Spark Job Server와 역할은 동일하면서, **경량화되고 안정적**인 Brightics Agent 적용
Brightics Agent를 Resource Pool 처럼 활용하여, 사용자/그룹별, 용도별 Resource 관리 가능



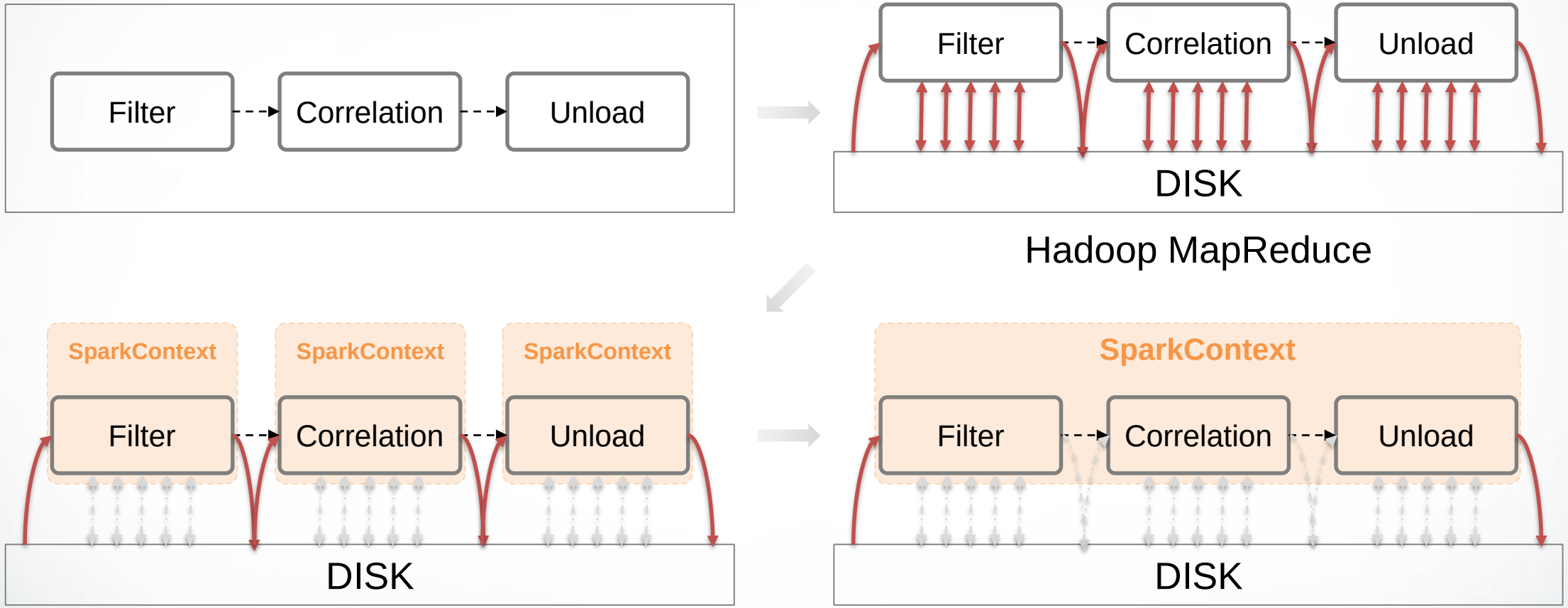
1) 하나의 실행 로직이 병렬 Job 으로 구성. ex) Group By

2) 하나의 실행이 단일 Job의 연속으로 구성. Ex) wordcount

3) 한 사용자가 여러 Agent를 사용. Ex) 관리자, Batch Job

MapReduce에서 Brightics Agent까지

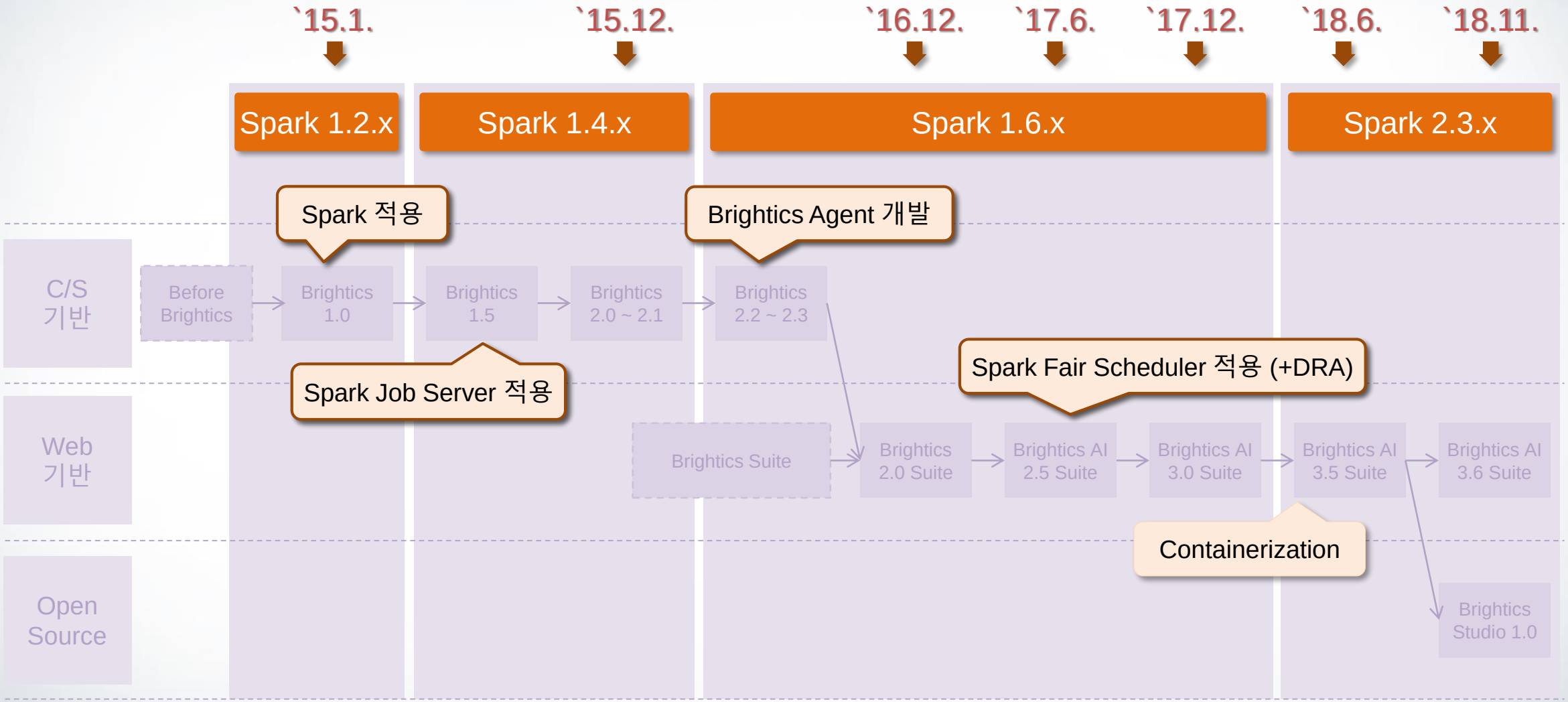
빠른 데이터 처리를 위한 시도



클러스터 리소스 최적화를 위한 Spark 아키텍처

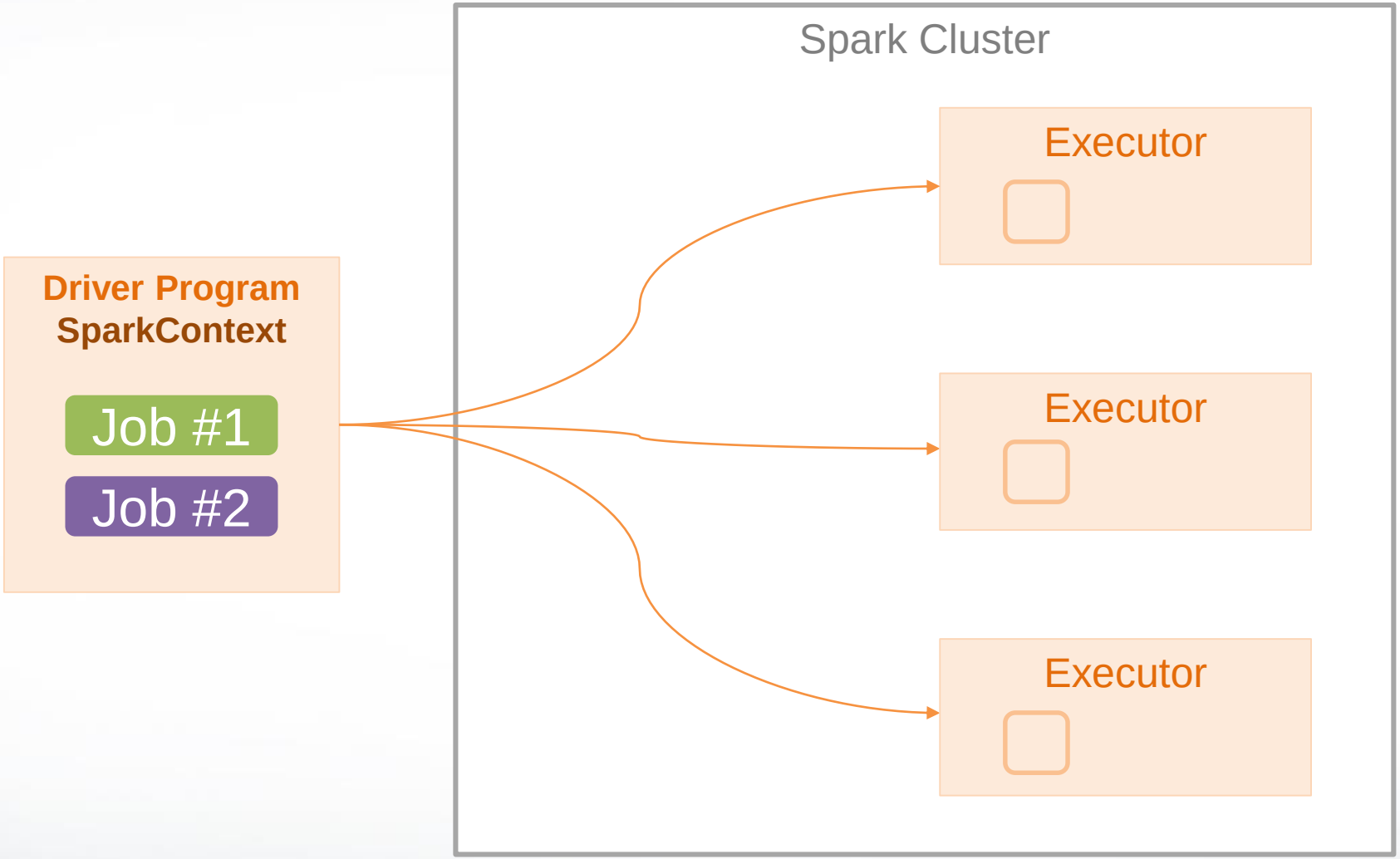
Spark 리소스 관리

Brightics AI with Spark



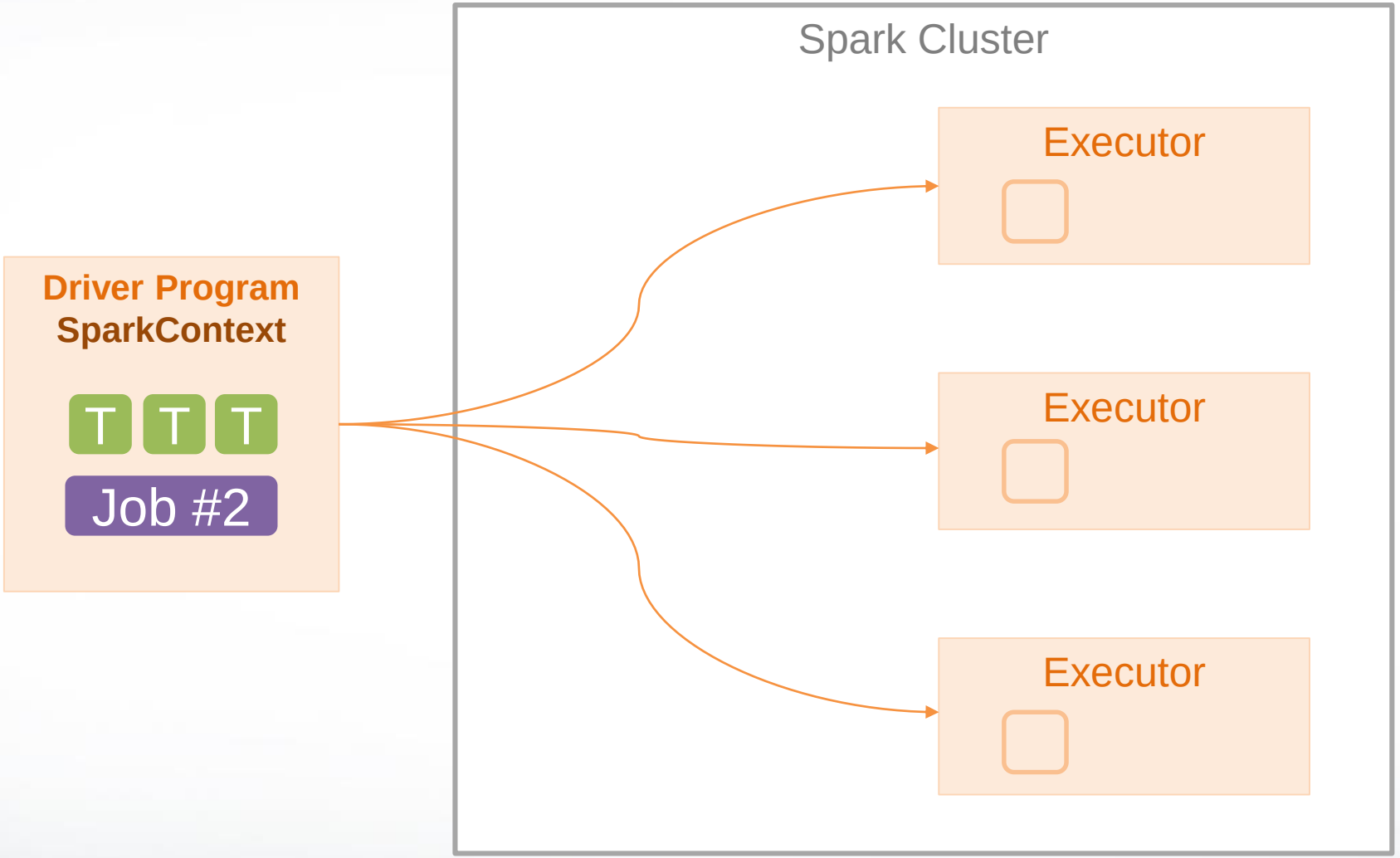
Spark Job Execution

Spark Job은 Task 단위로 분할



Spark Job Execution

Task는 여러 Executor에서 병렬로 수행



Job vs Job

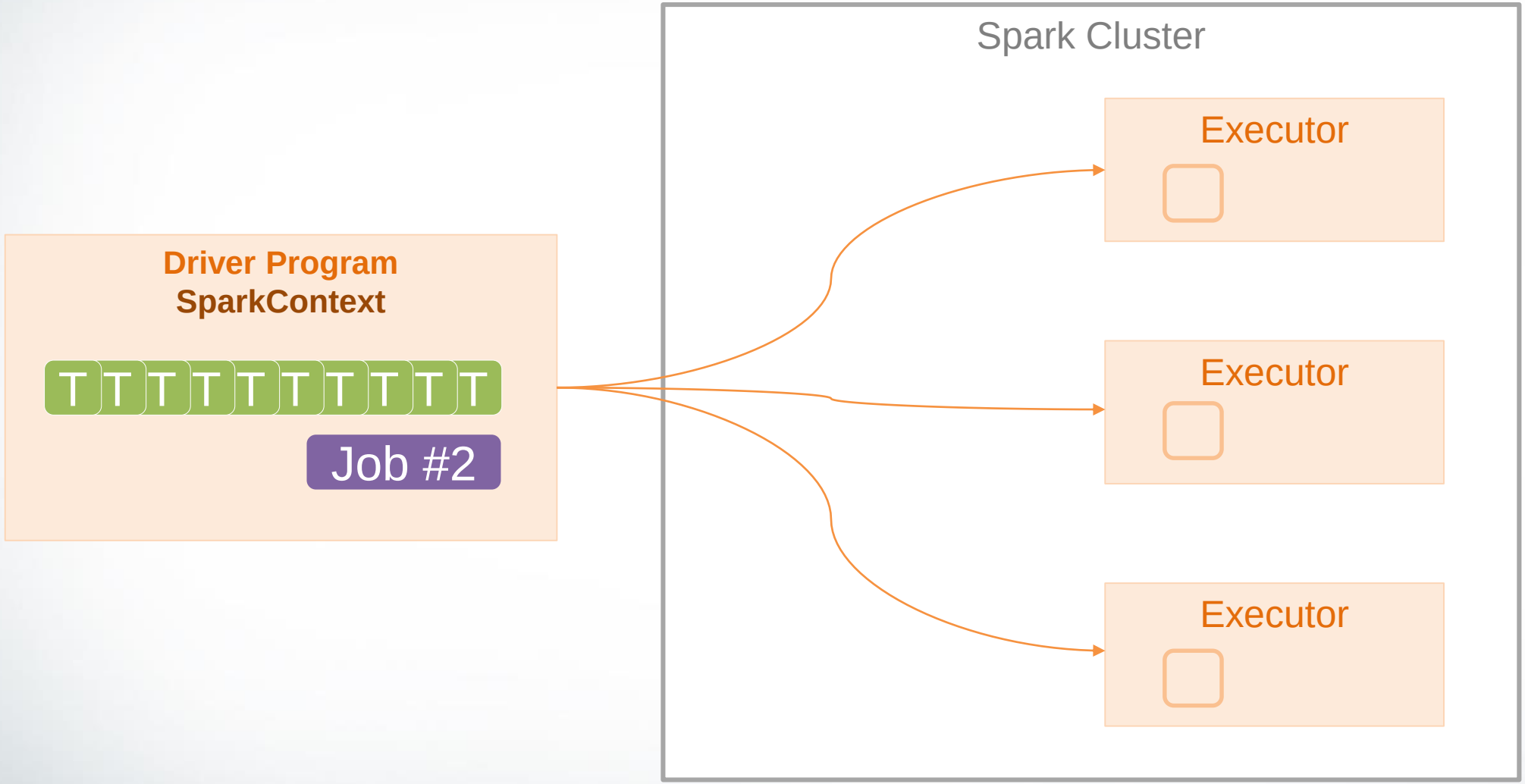
Spark Fair Scheduler

Spark Fair Scheduler

Q. 만약, 큰 데이터를 오래 돌리는 Job이 실행된다면?

Spark Fair Scheduler

먼저 실행된 Job의 Task부터 실행 (FIFO)



A. Spark Fair Scheduler를 활용해 Job 간의 경합 해결

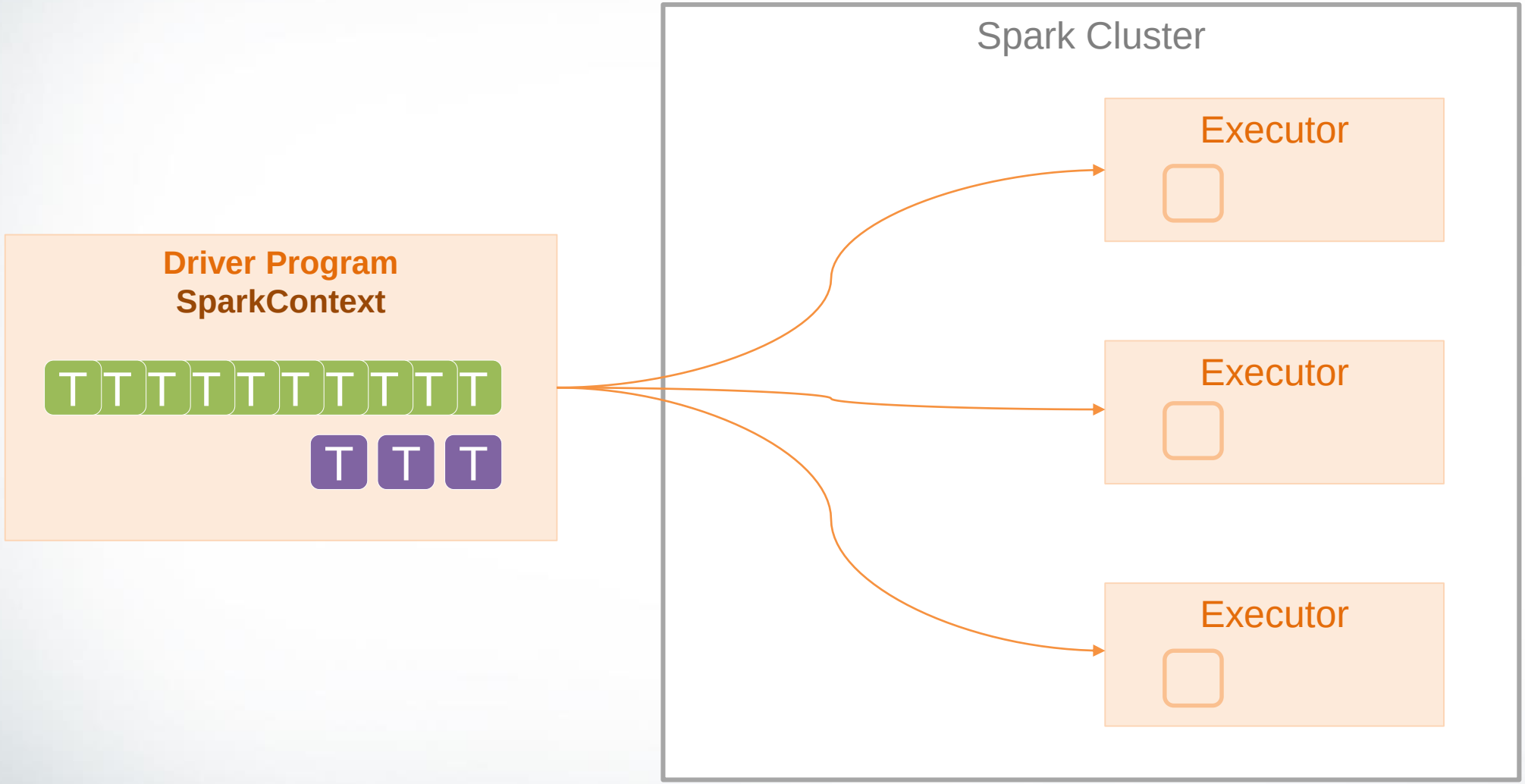
- Spark Job Scheduling 방식에는 FIFO (default)와 FAIR가 있음
  FIFO(default)는 Job을 한 개씩 실행. **FAIR**는 Job을 동시에 동일한 리소스(tasks)로 실행.

  scala> conf.set("spark.scheduler.mode", "FAIR")

- SparkContext의 Scheduler 설정을 위해서는 **conf/fairscheduler.xml** 에서 Pool을 정의

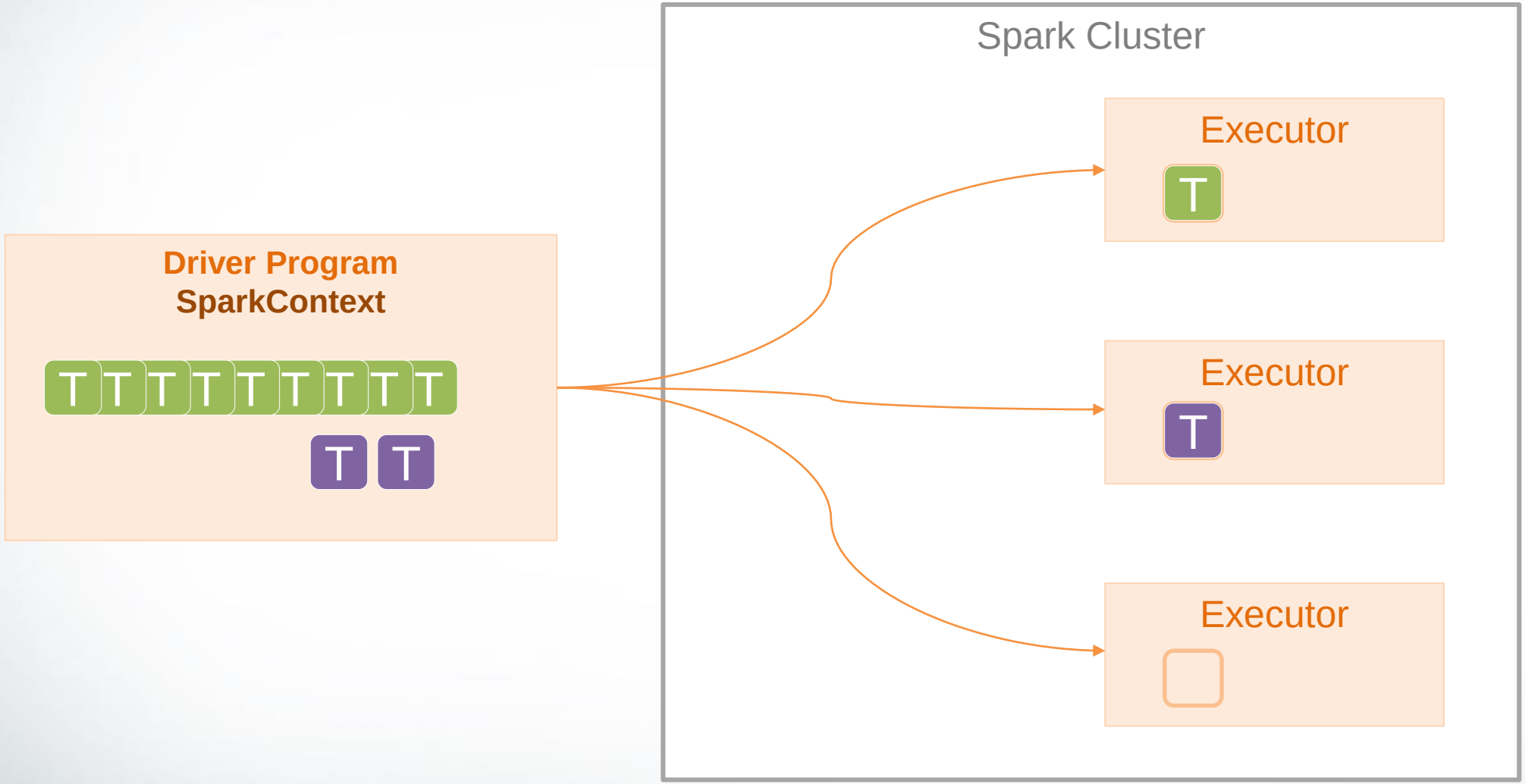
Spark Fair Scheduler

Fair Scheduler에서는 여러 Job이 동시에 실행



Spark Fair Scheduler

Fair Scheduler에서는 여러 Job이 동시에 실행



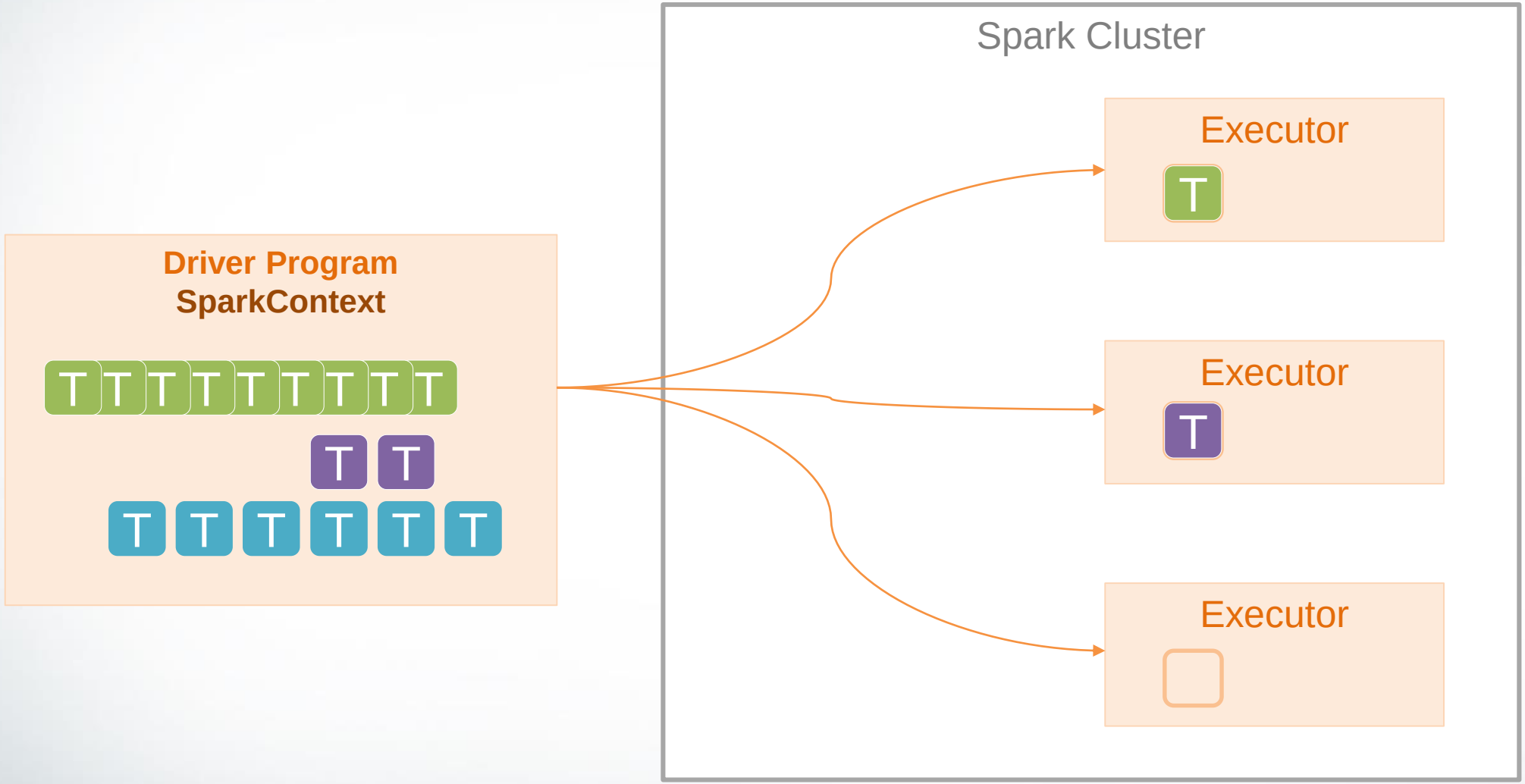
Spark Fair Scheduler

Q. 모든 Resource가 다 점유된 상태에서

긴급하게 다른 Job을 실행시켜야 한다면?

Spark Fair Scheduler

Fair Scheduler에서는 여러 Job이 동시에 실행



Spark Fair Scheduler with Pool

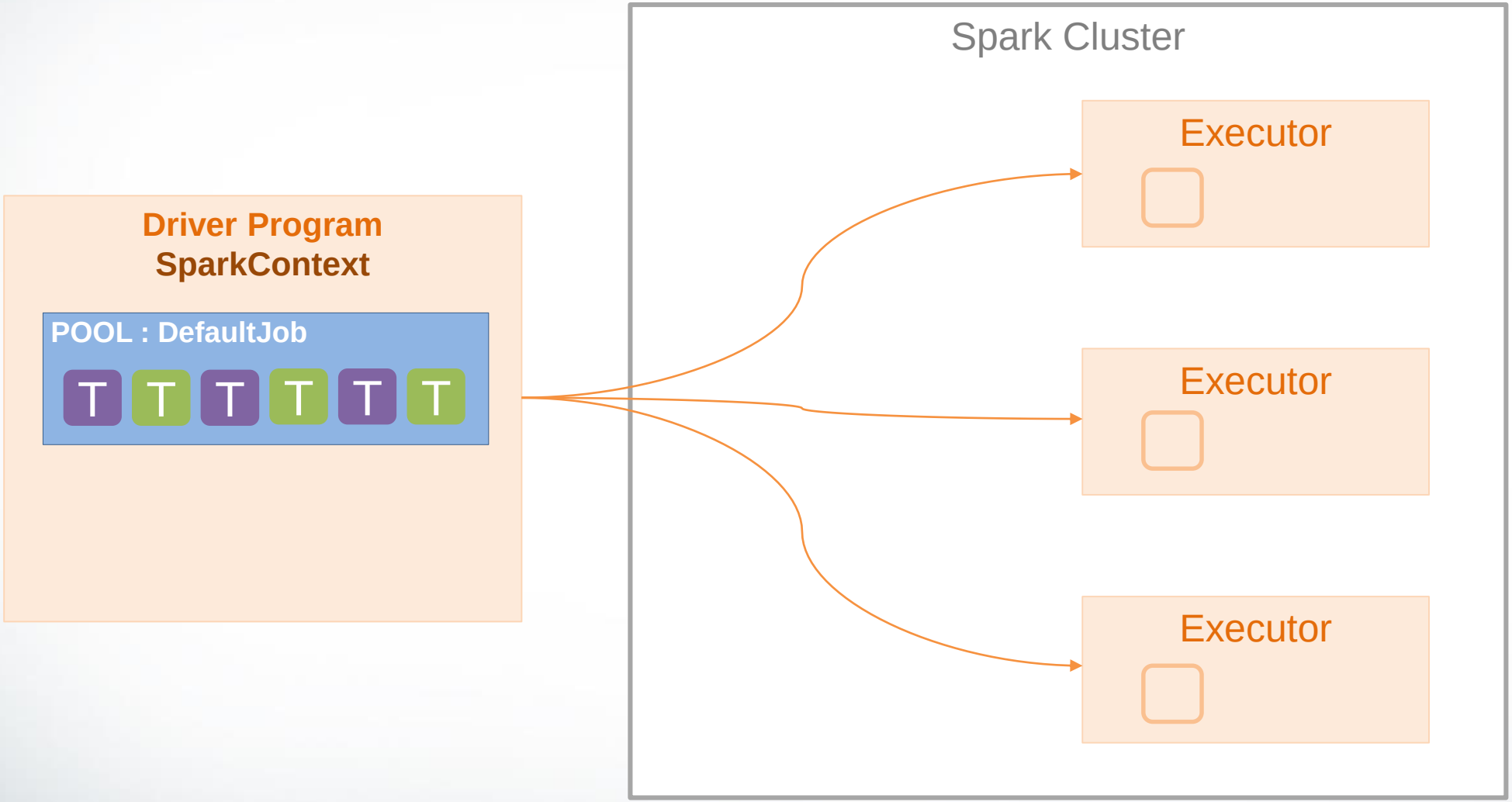
A. Pool의 옵션을 통해 Job의 우선순위 제어 가능

- Job 실행 전에 pool을 지정하여 원하는 Pool에서 실행 가능

  scala> sc.setLocalProperty("spark.scheduler.pool", “FasterJob")

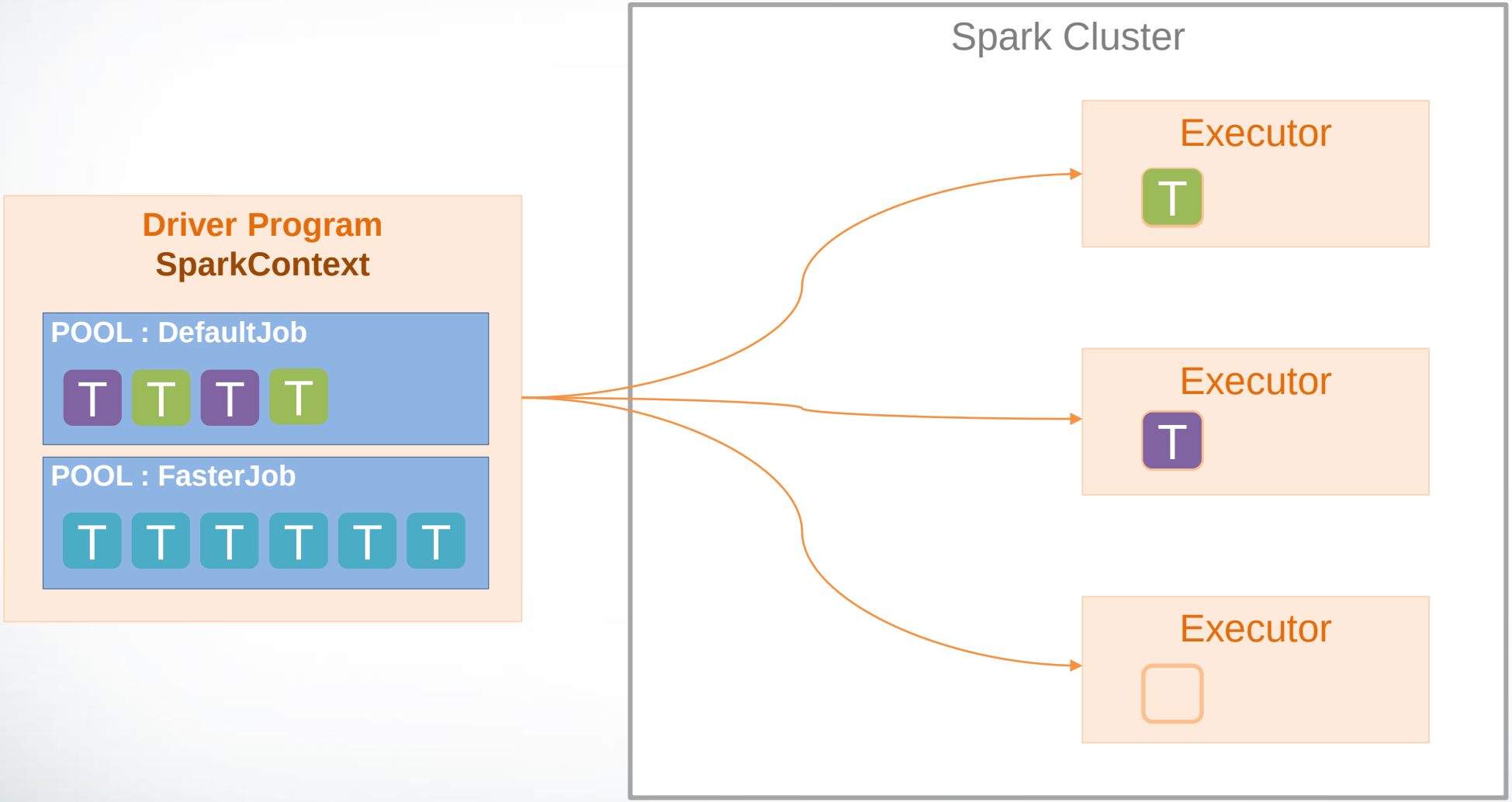
Spark Fair Scheduler with Pool

Pool의 역할을 나누어, 효과적으로 Job 관리 가능



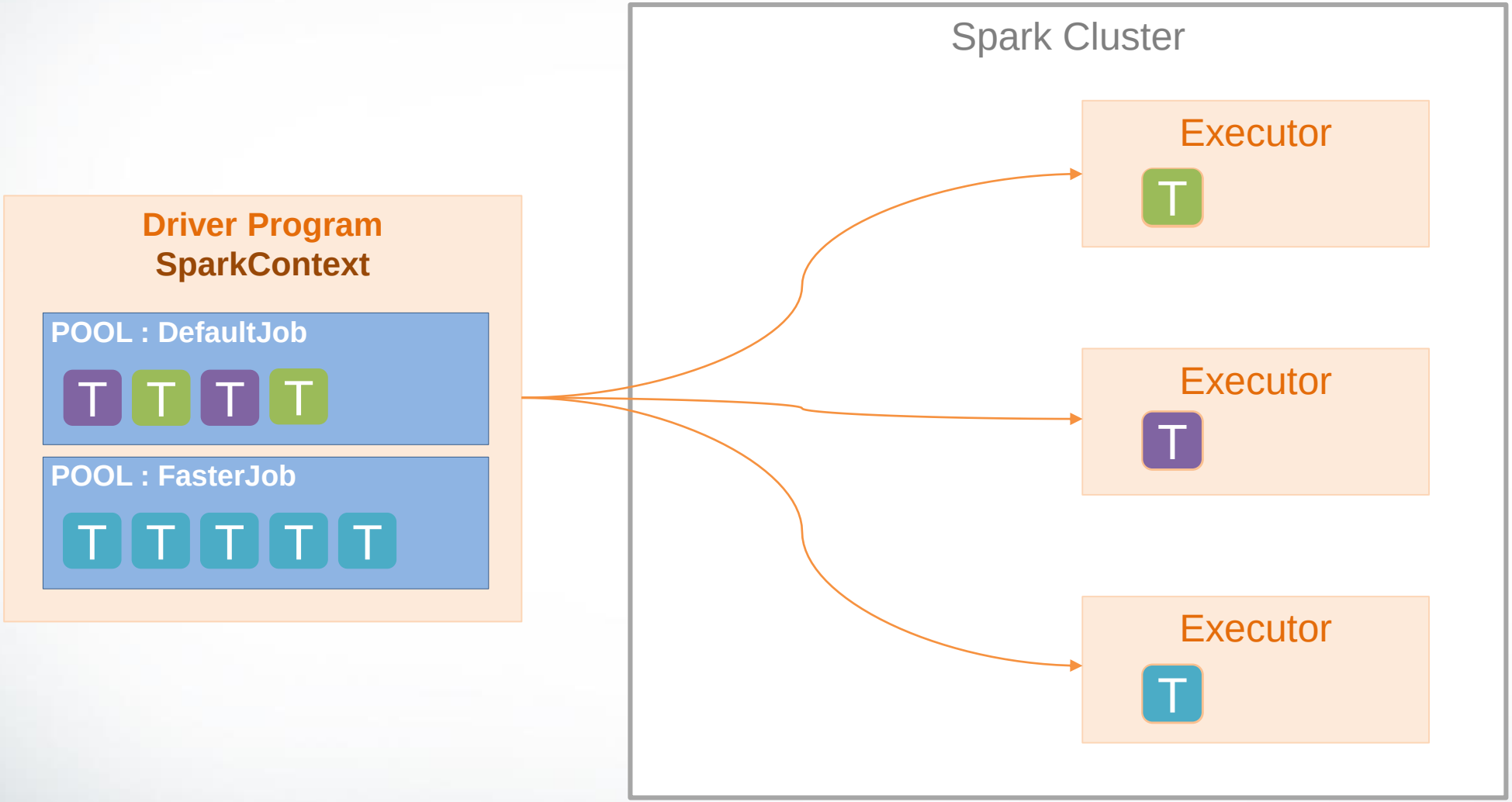
Spark Fair Scheduler with Pool

Pool의 역할을 나누어, 효과적으로 Job 관리 가능



Spark Fair Scheduler with Pool

Pool의 역할을 나누어, 효과적으로 Job 관리 가능



Spark Fair Scheduler with Pool

Pool의 역할을 나누어, 효과적으로 Job 관리 가능

```
<allocations>
  <pool name="default">
    <schedulingMode>FAIR</schedulingMode>
    <weight>1</weight>
    <minShare>2</minShare>
  </pool>
  <pool name="brightics1">
    <schedulingMode>FAIR</schedulingMode>
    <weight>1000</weight>
    <minShare>0</minShare>
  </pool>
  <pool name="brightics2">
    <schedulingMode>FAIR</schedulingMode>
    <weight>1000</weight>
    <minShare>0</minShare>
  </pool>
  <pool name="brightics3">
    <schedulingMode>FAIR</schedulingMode>
    <weight>1000</weight>
    <minShare>0</minShare>
  </pool>
</allocations>
```

- “default” Pool 에서는
  상태 및 데이터 확인 작업 수행
- 다른 Pool 에서는
  일반적인 Spark Job 수행

Spark Fair Scheduler with Pool

Pool의 옵션을 통해 Job의 우선순위 제어 가능

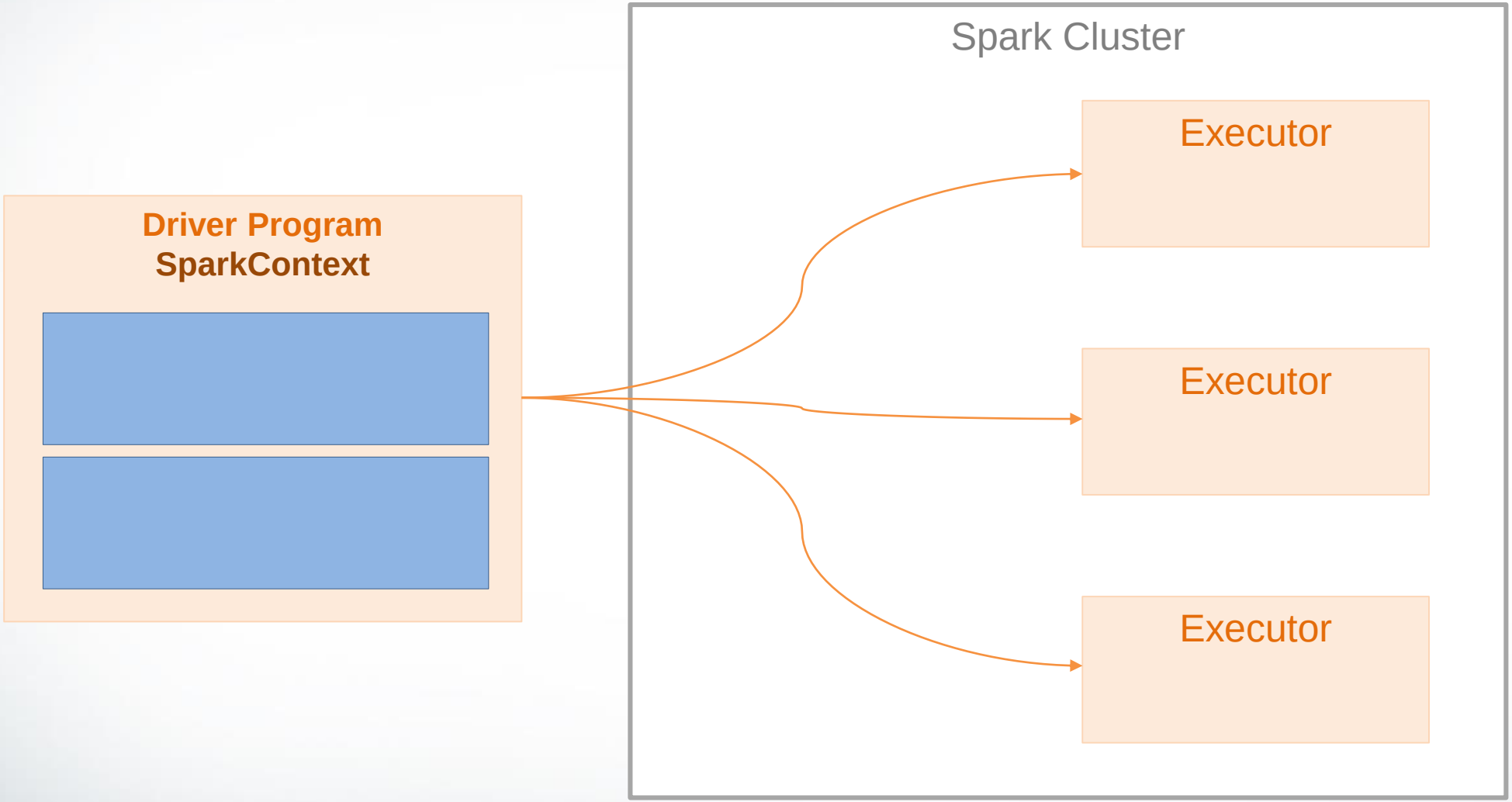
- 여러 Pool 에서 동시에 여러 Job이 실행되면,
  높은 “weight”의 Pool이 더 많은 리소스를 사용

- Pool 에서 Job이 실행되면,
  최소한 **“minShare”만큼의 cores를 얻도록 시도**

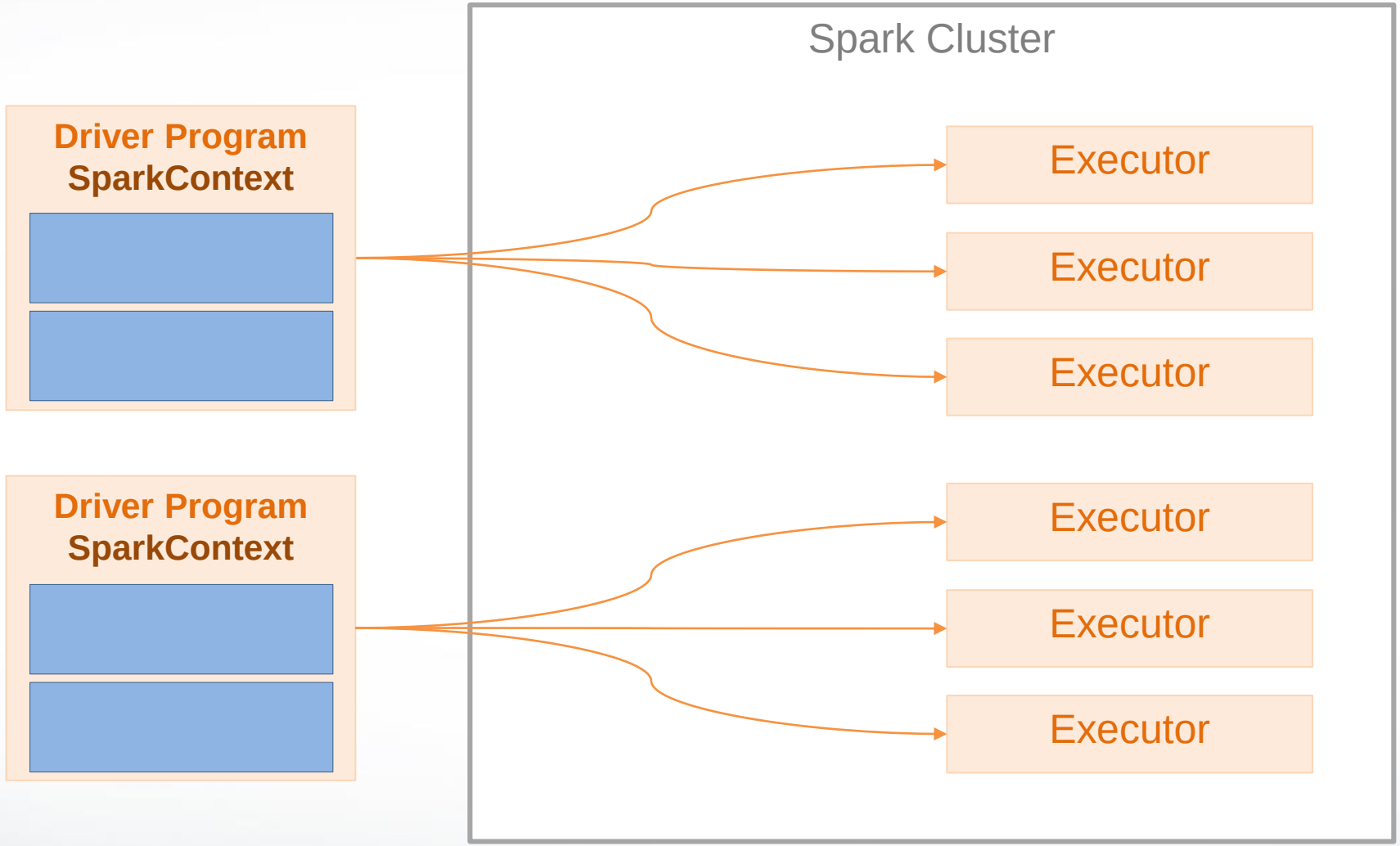
Application vs Application

Dynamic Resource Allocation

Dynamic Resource Allocation



Dynamic Resource Allocation

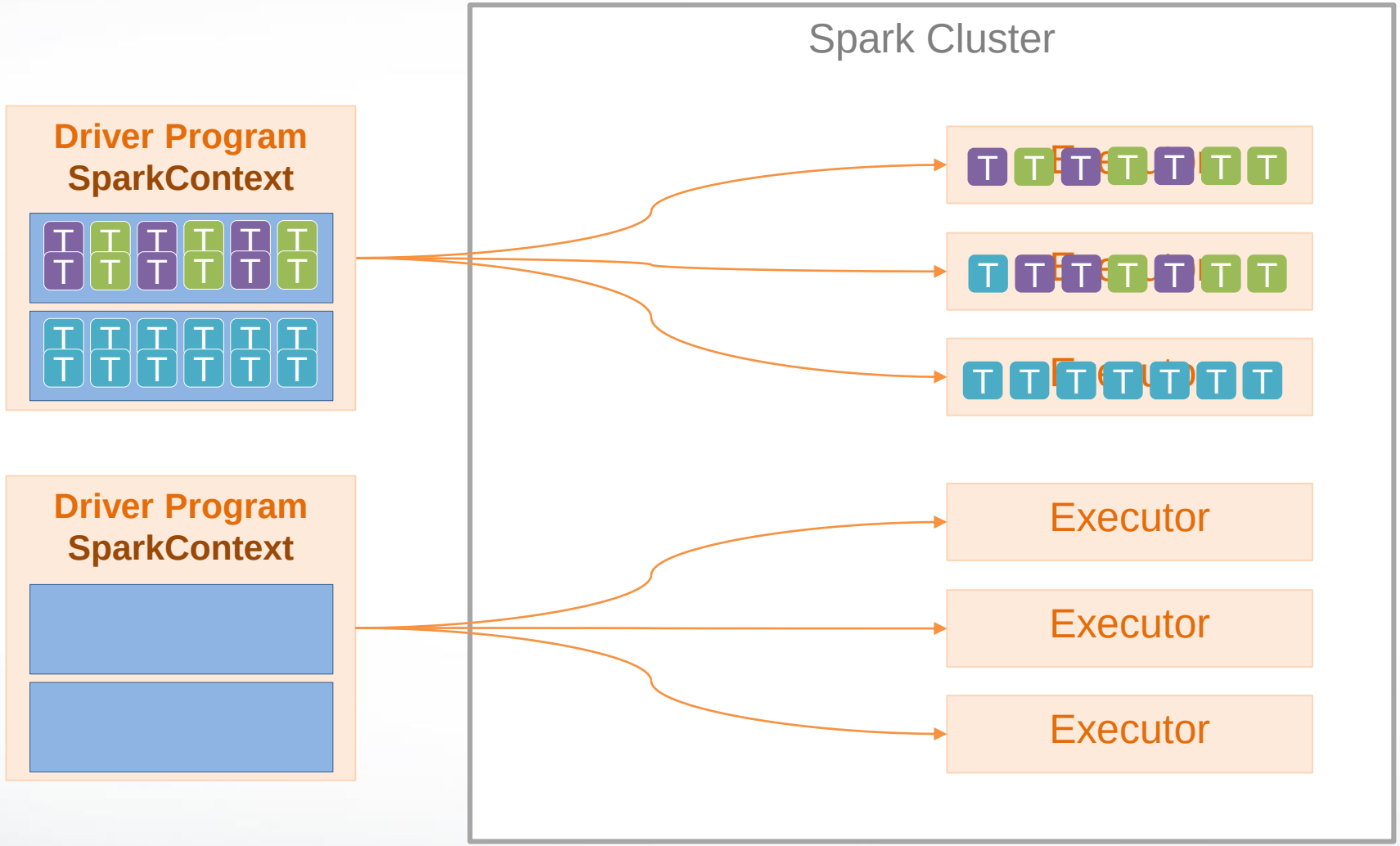


Dynamic Resource Allocation

Q. 한 Spark Context에만 너무 많은 Job이 있다면?

Dynamic Resource Allocation

아무 작업도 수행하지 않는 Application에서 비효율적으로 Resource를 점유



Dynamic Resource Allocation

A. Dynamic Resource Allocation을 활용하여,

Application 간 효율적 리소스 사용 가능

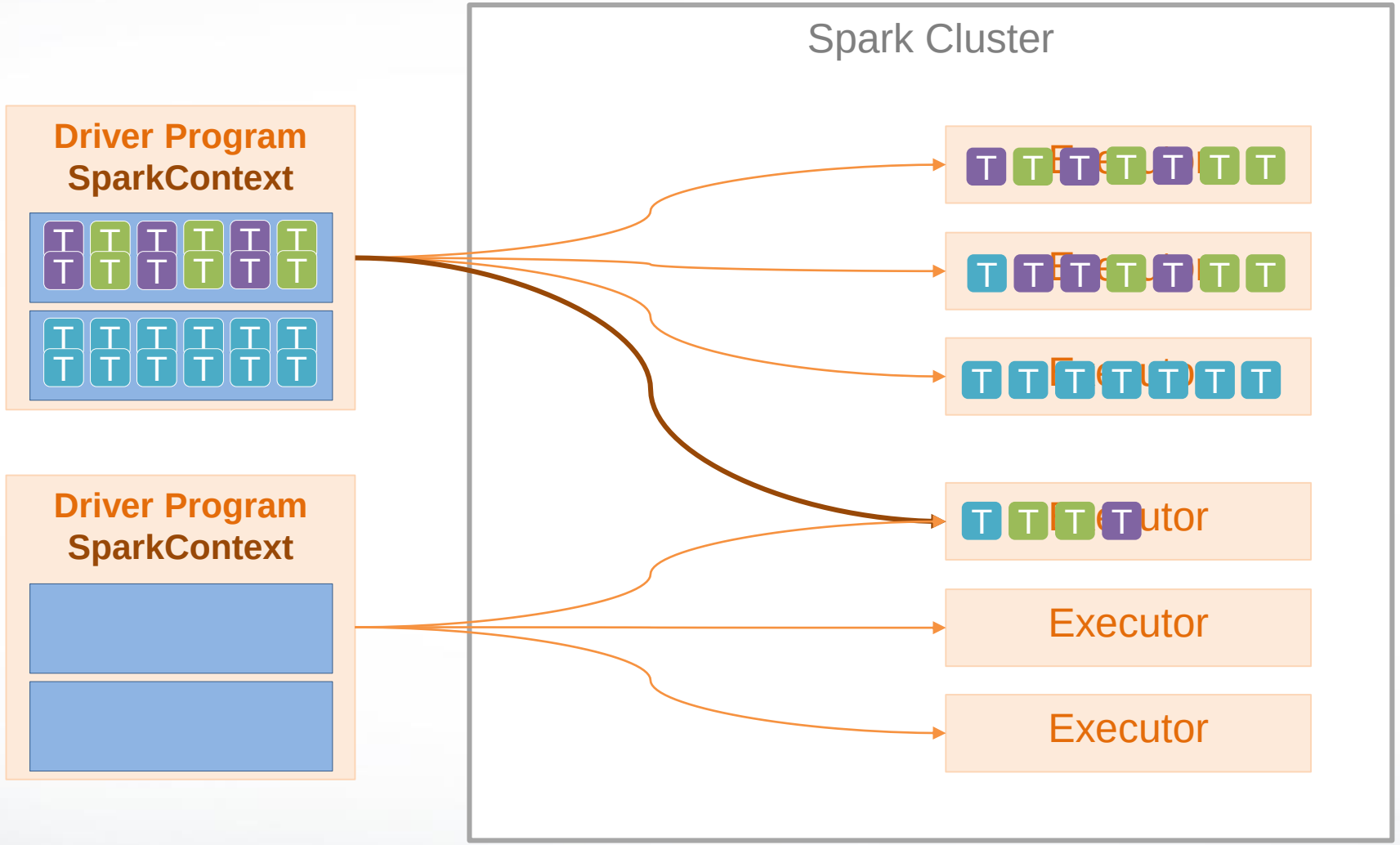
Dynamic Resource Allocation

```
# $SPARK_HOME/conf/spark-defaults.conf
spark.dynamicAllocation.enabled true
spark.shuffle.service.enabled true
spark.dynamicAllocation.minExecutors 0
spark.dynamicAllocation.maxExecutors 100
spark.dynamicAllocation.initialExecutors 2
spark.dynamicAllocation.executorIdleTimeout 30s
```

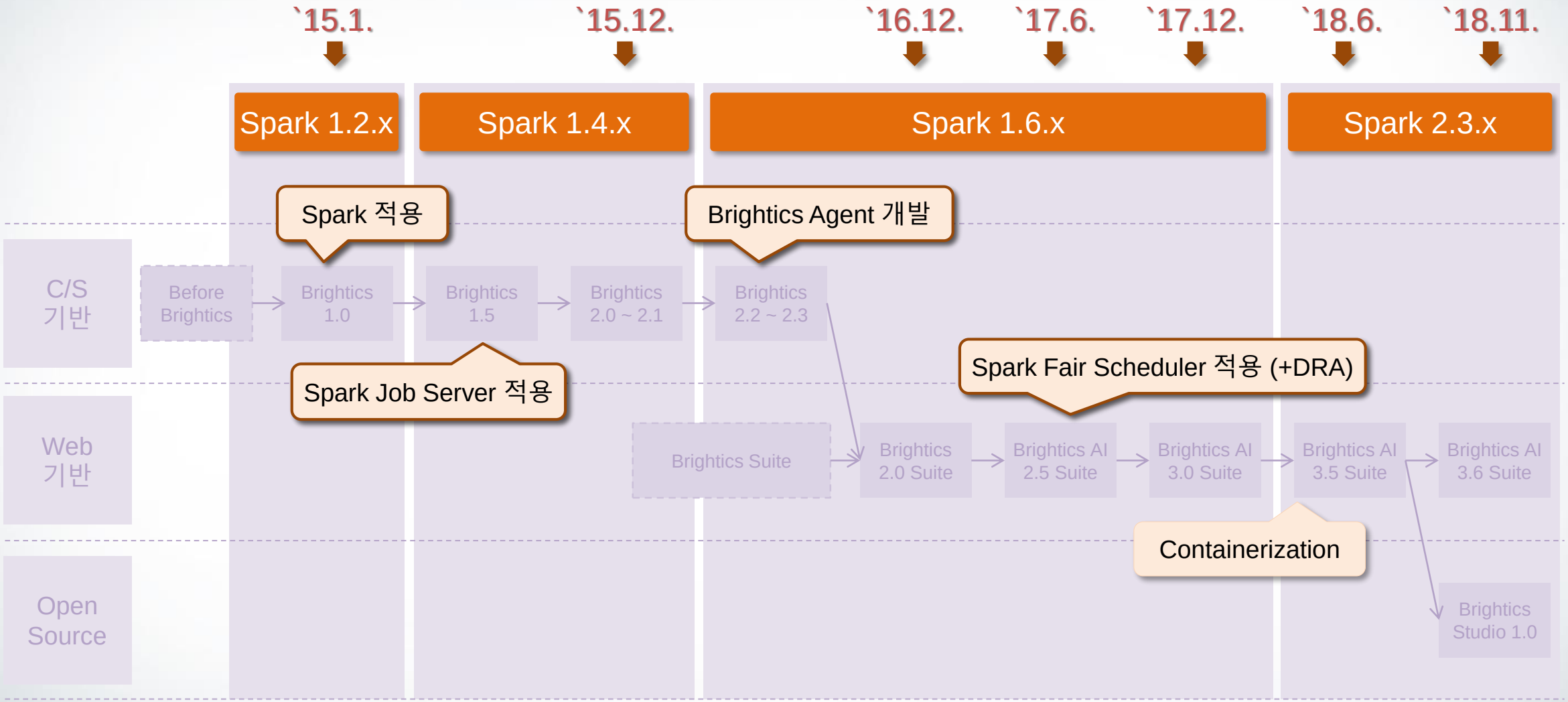
- Spark Standalone 에서의 Dynamic Resource Allocation **단위는 Executor 수**
  : 총 cores 수 (--total-executor-cores, spark.cores.max) ,
  Executor당 cores 수 (--executor-cores) 에 의존
- 아무 작업도 없을 때 ➔ 최초 리소스할당 (spark.dynamicAllocation.initialExecutors)
  최소 리소스로 반환 (spark.dynamicAllocation.minExecutors)
- 작업 실행할 때 ➔ 가용 리소스가 있다면 최대한 사용하면서 작업 시작
  spark.dynamicAllocation.maxExecutors 등으로 상한선 제어 가능
- 작업이 완료될 때 ➔ 최소 리소스로 반환
  다른 App이 있다면 그 App이 남는 리소스 사용

Dynamic Resource Allocation

동적으로 Executor를 배분하여 효과적으로 Resource 사용 가능



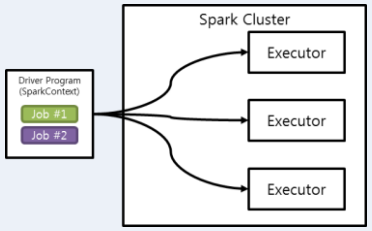
Brightics AI with Spark



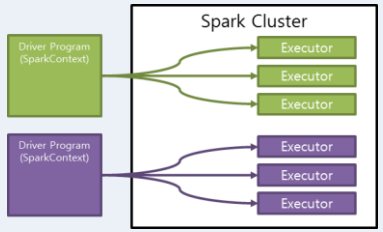
Scalable Spark 아키텍처

개인 사용자부터 Cloud 서비스까지, 모든 Scale 형태의 Spark Cluster 서비스 가능

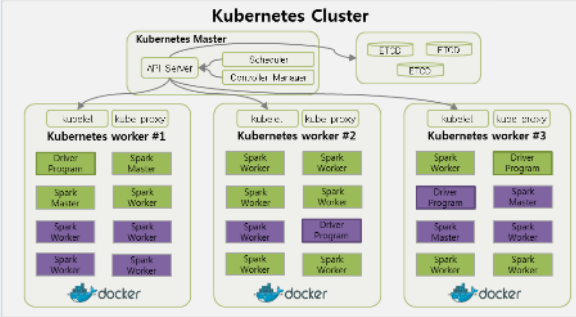
Local Mode
for
Single User



Single Context
for
a Group



Multiple Contexts
for
a Company



Multiple Clusters
for
a Cloud

Demo

Demo

1. Spark Job Server를 사용할 때와 안 할 때의 실행시간 비교

2. FIFO scheduler와 FAIR scheduler 각각에서 Job 실행

Q & A

Thank you