

이

회



강

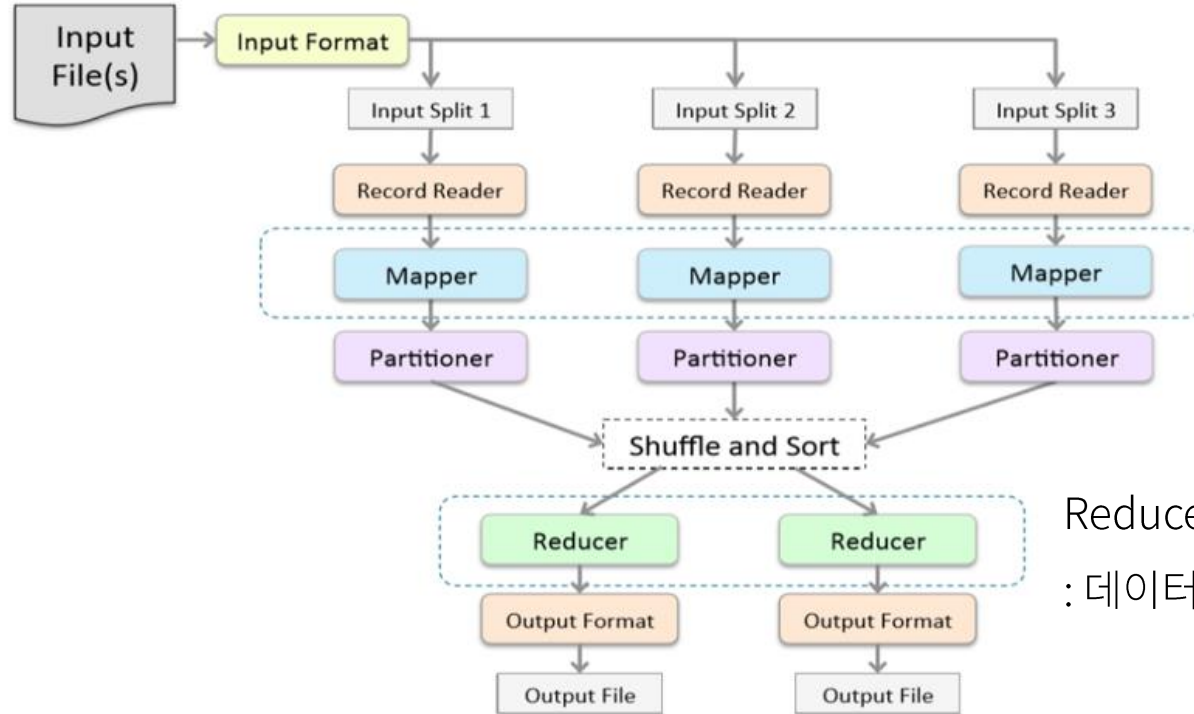
의



Data Engineering 180721 세션 - Hadoop MapReduce

MapReduce?

- 작업을 Map 단계와 Reduce로 나누어 간단하게 분산 처리를 실행할 수 있게 한 프로그래밍 모델!
- 한창 잘나가던 하둡 1.0 시절 JobTracker와 TaskTracker로 구성된 분산 처리 프레임워크의 의미로도 사용되었으나 YARN으로 대체됨



Mapper

: 데이터 변형 (transformation) 작업

Reducer

: 데이터 집계 (aggregation) 작업

Hadoop 1.0

Single Use System

Batch Apps

HADOOP 1.0

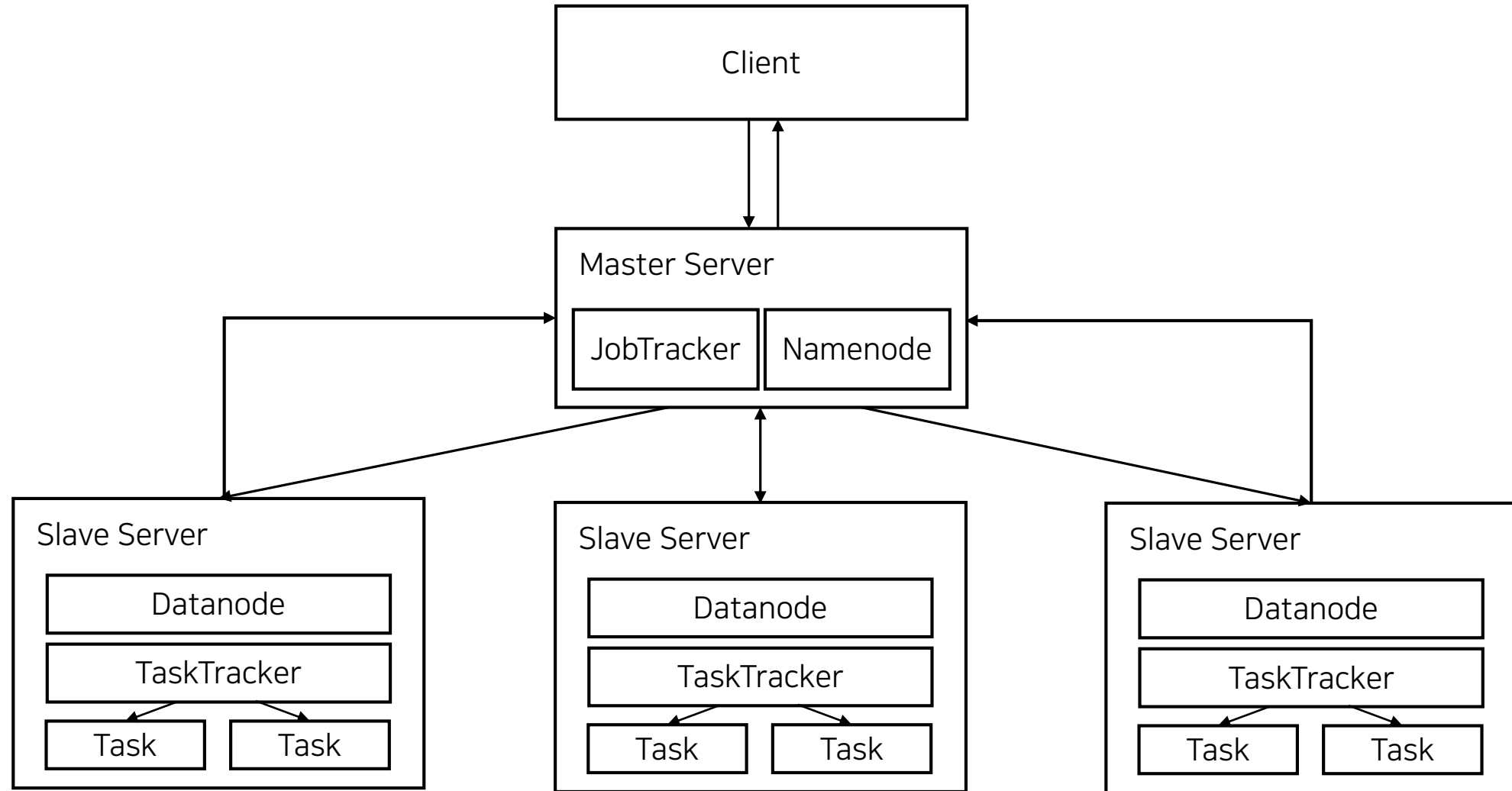
MapReduce

(cluster resource management
& data processing)

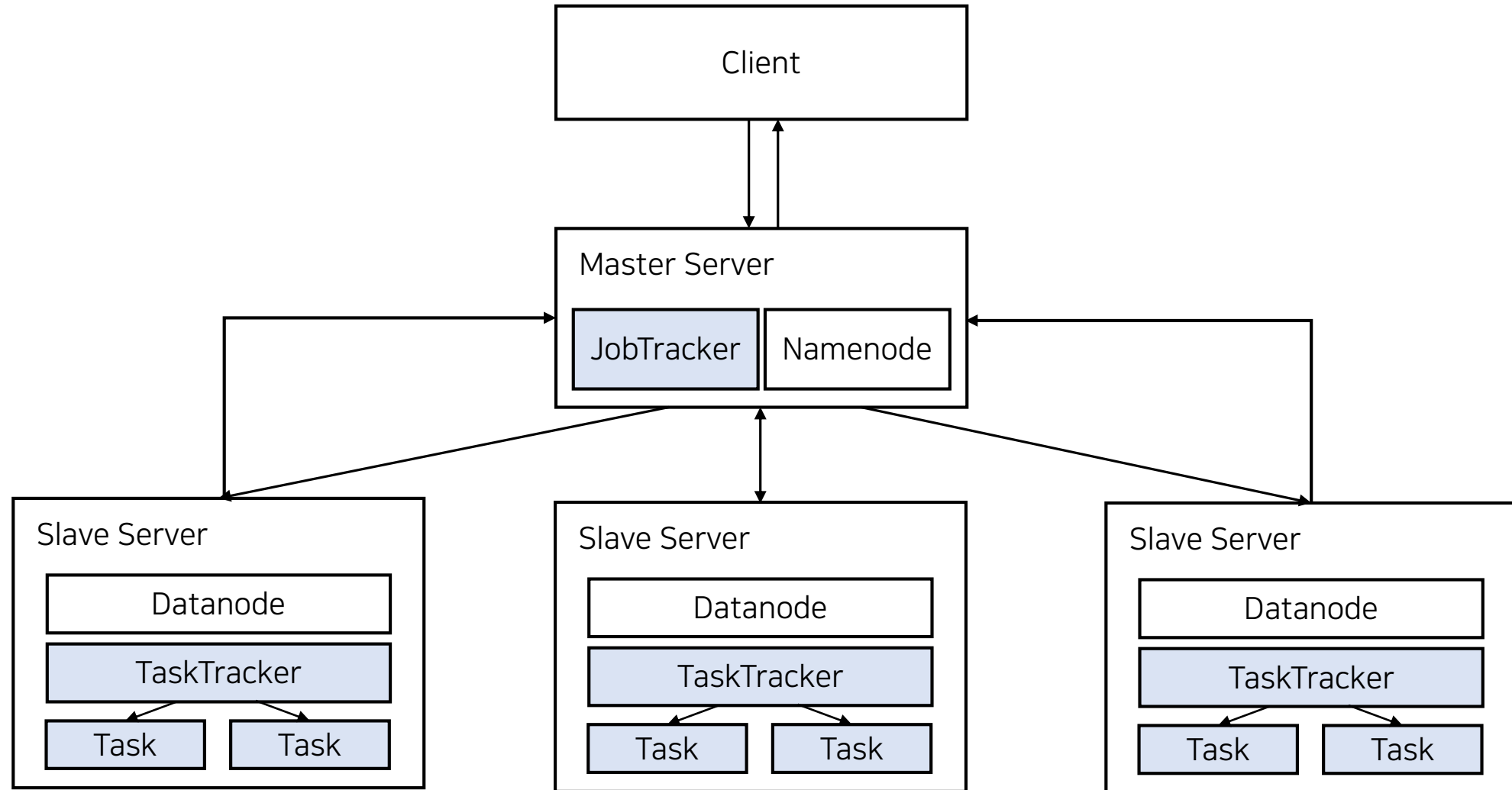
HDFS

(redundant, reliable storage)

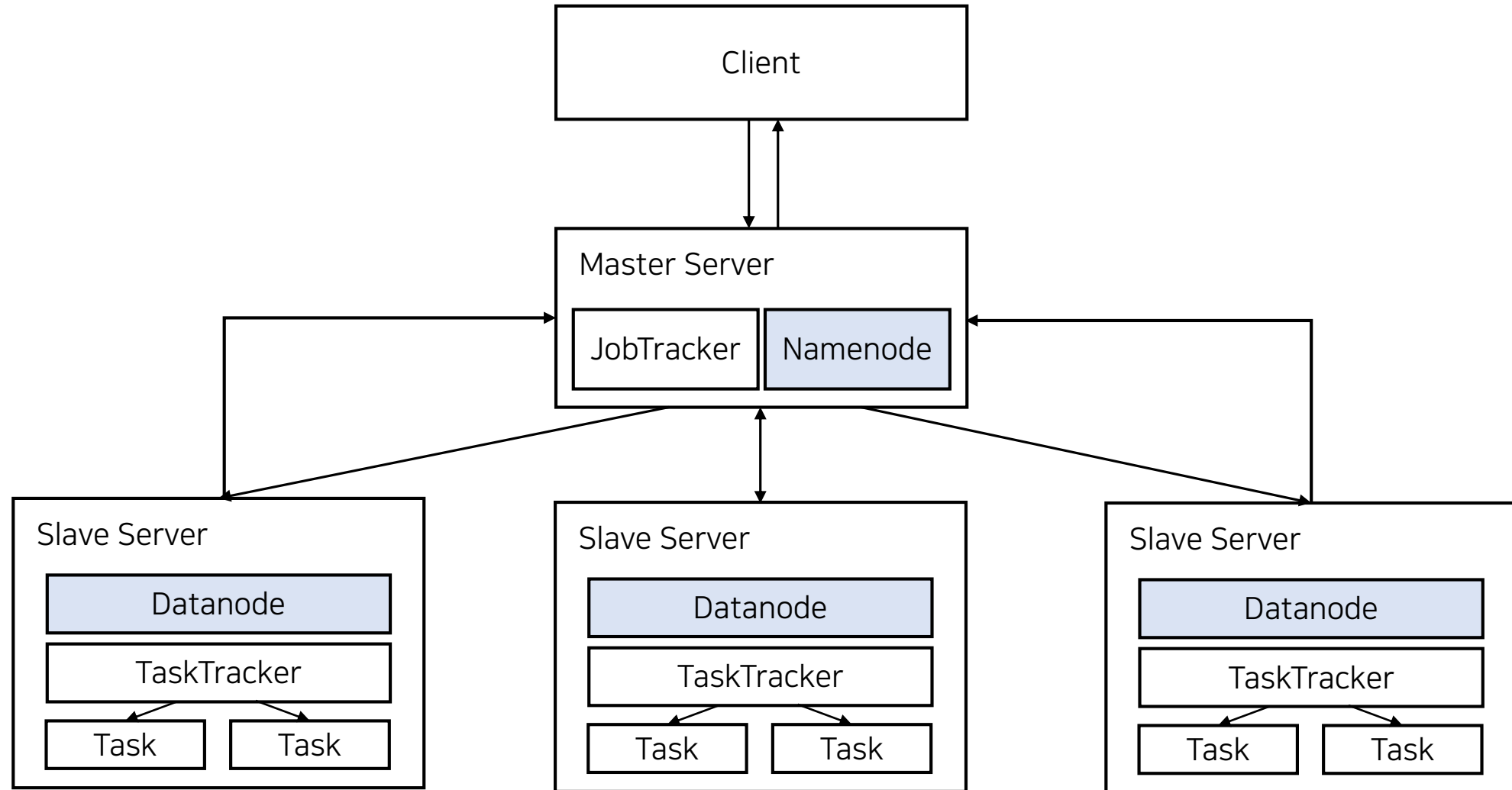
Hadoop 1.0 Architecture



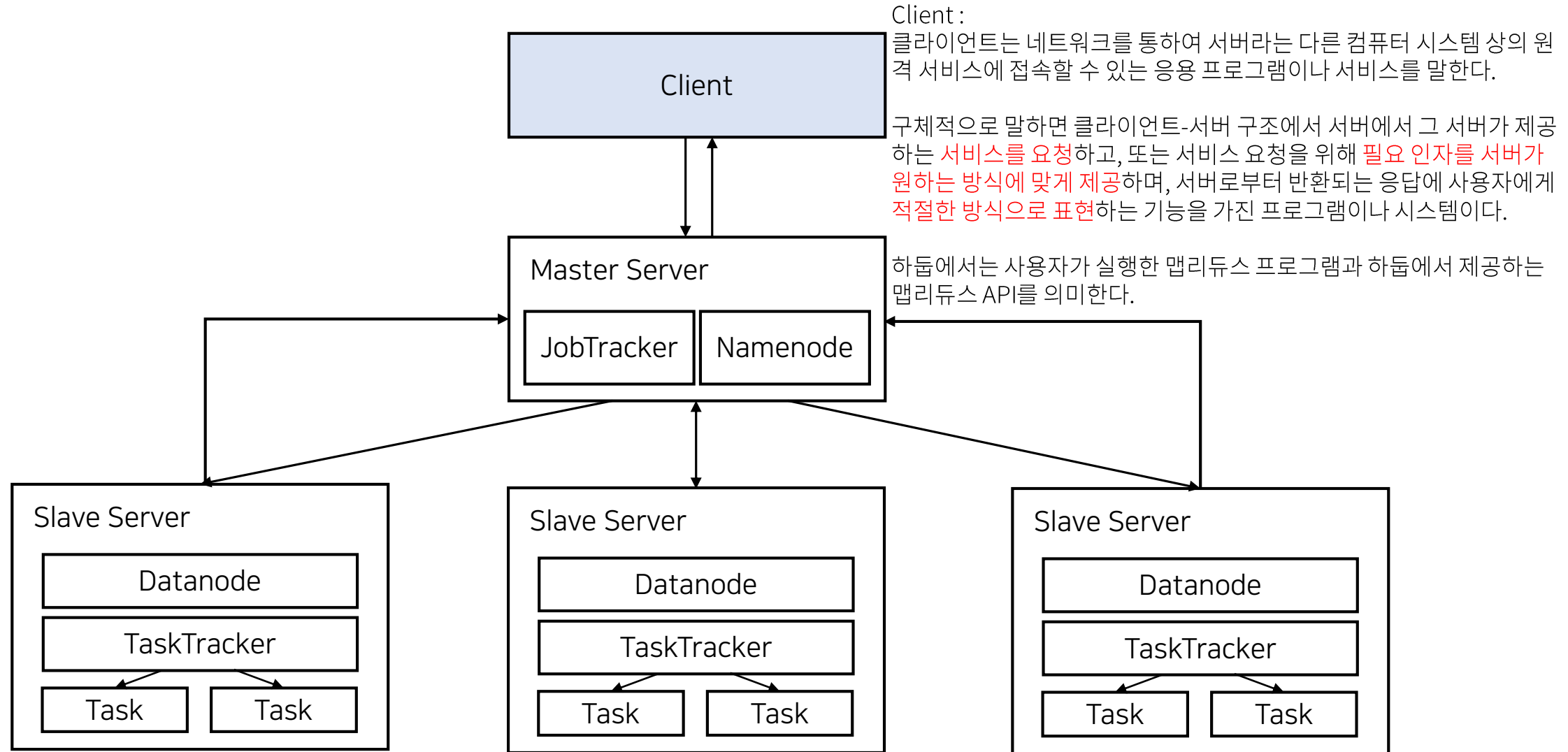
Hadoop 1.0 Architecture - MapReduce



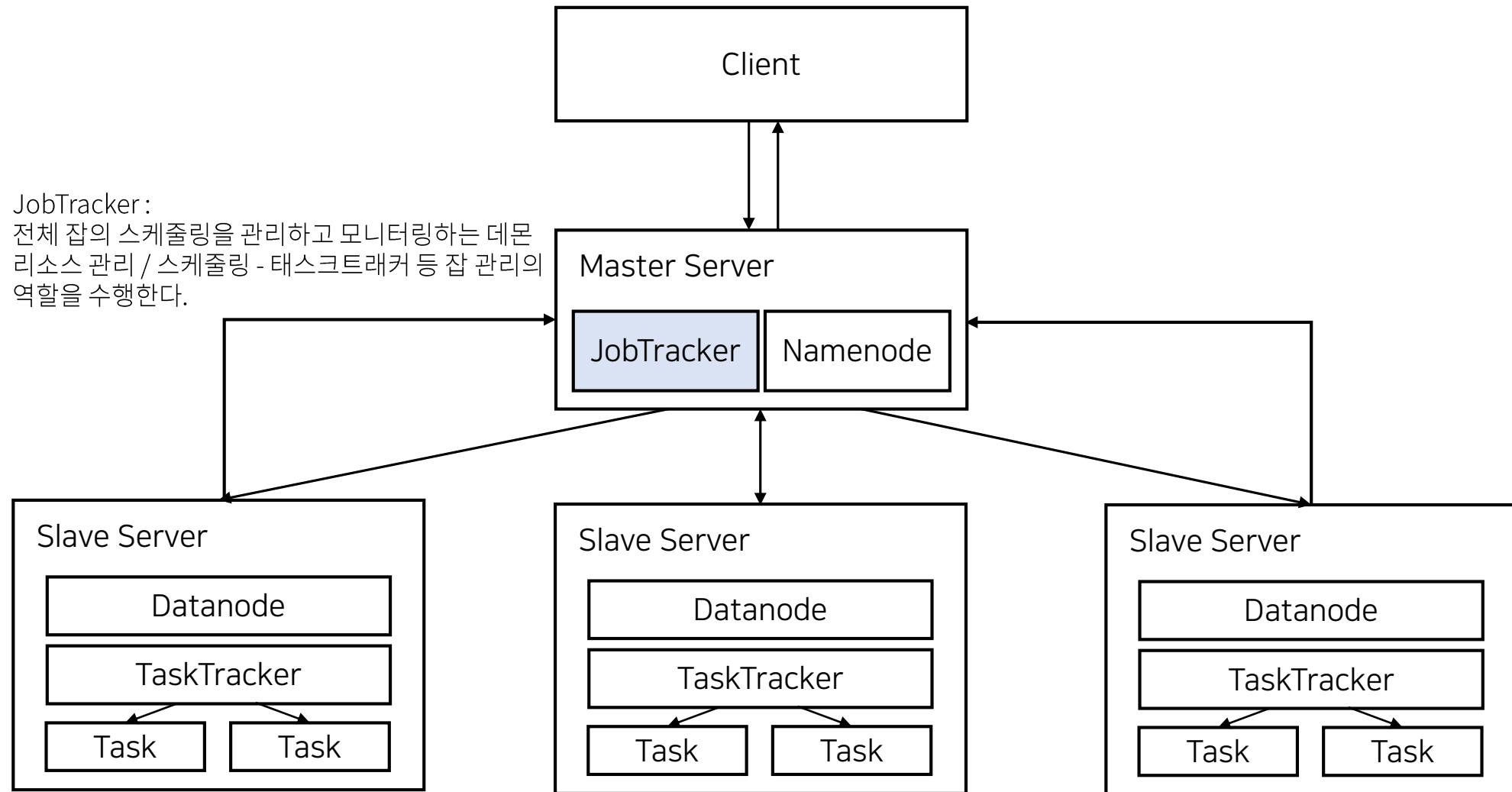
Hadoop 1.0 Architecture - HDFS



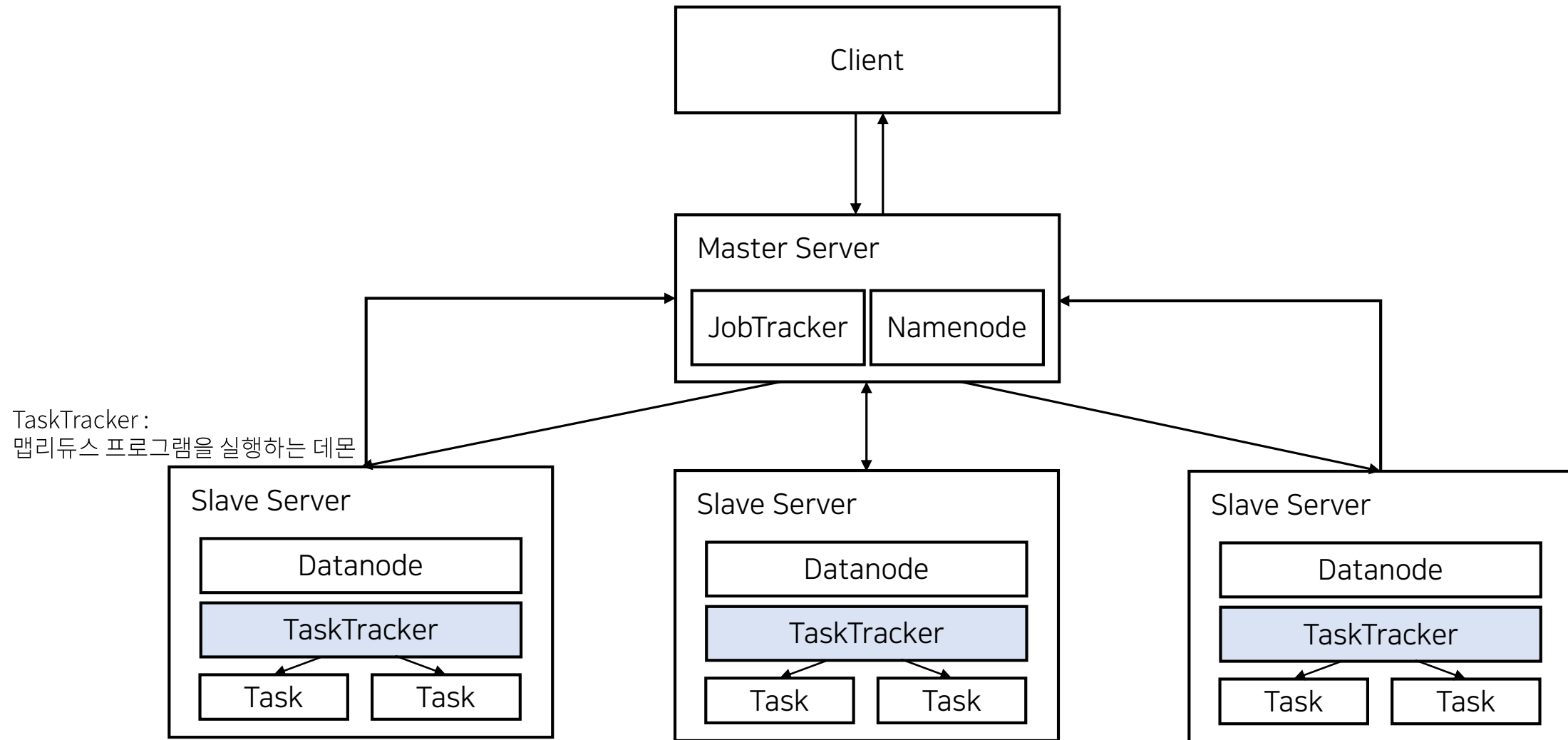
Hadoop 1.0 Architecture - MapReduce



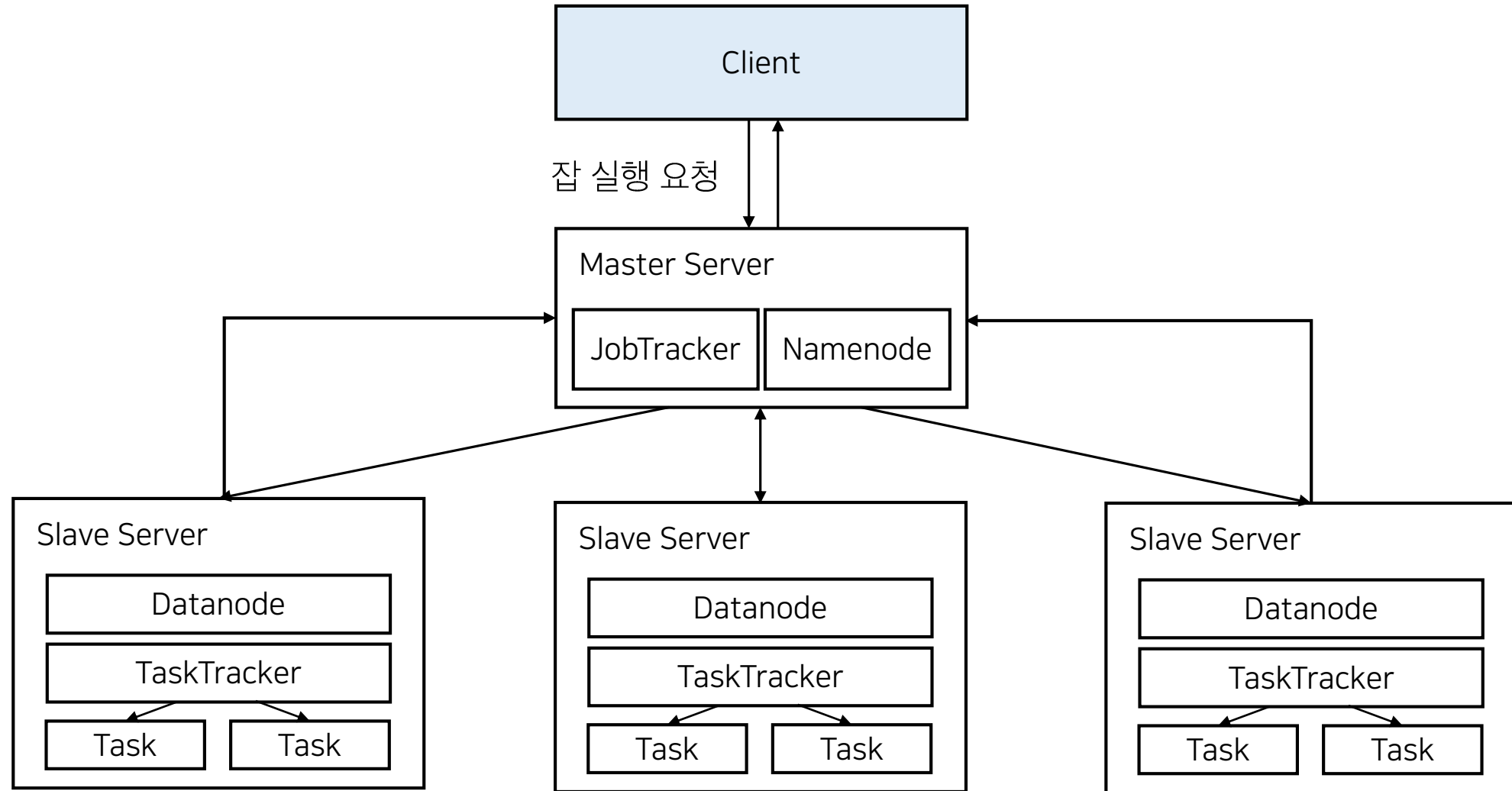
Hadoop 1.0 Architecture - MapReduce



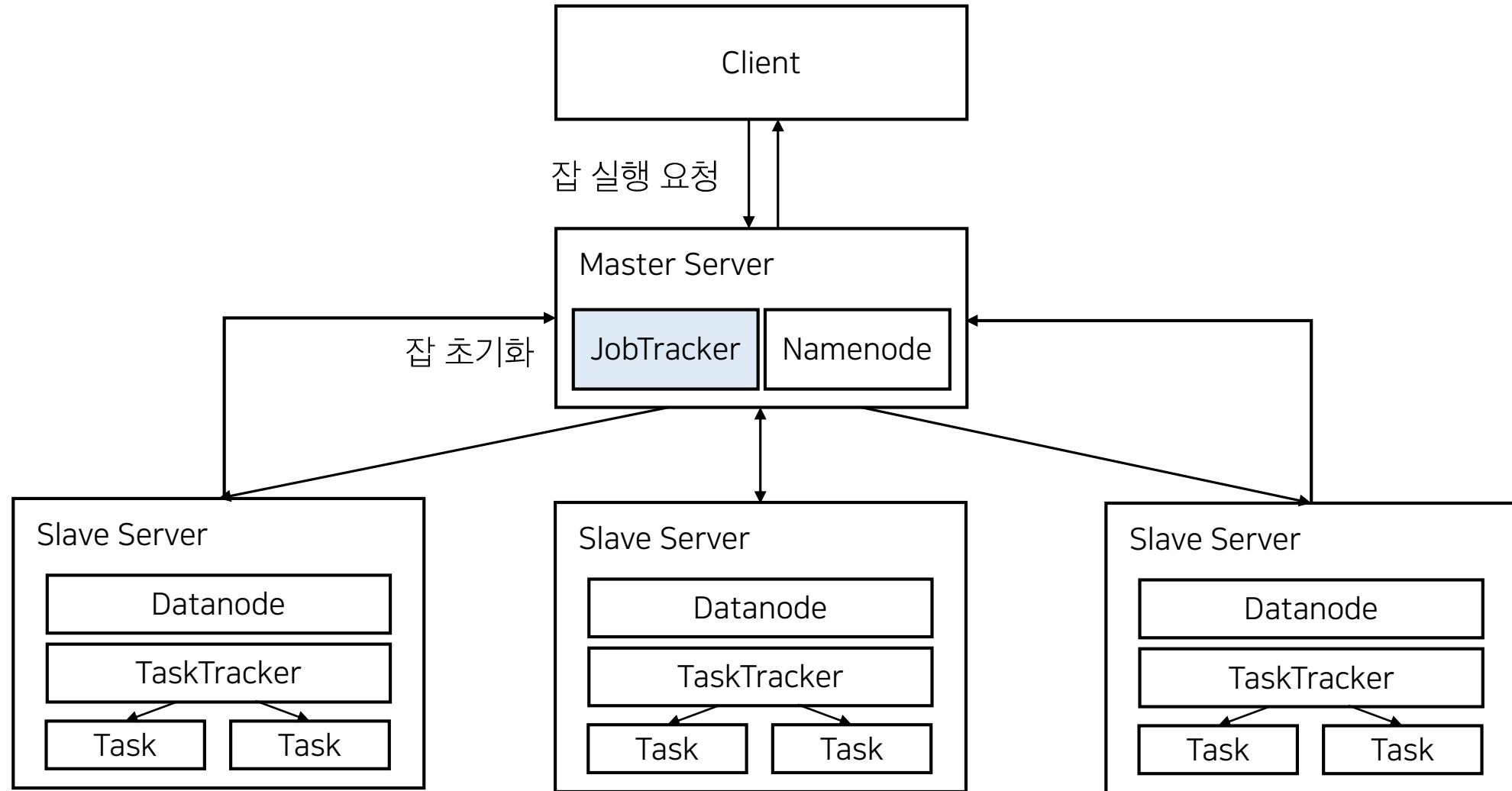
Hadoop 1.0 Architecture - MapReduce



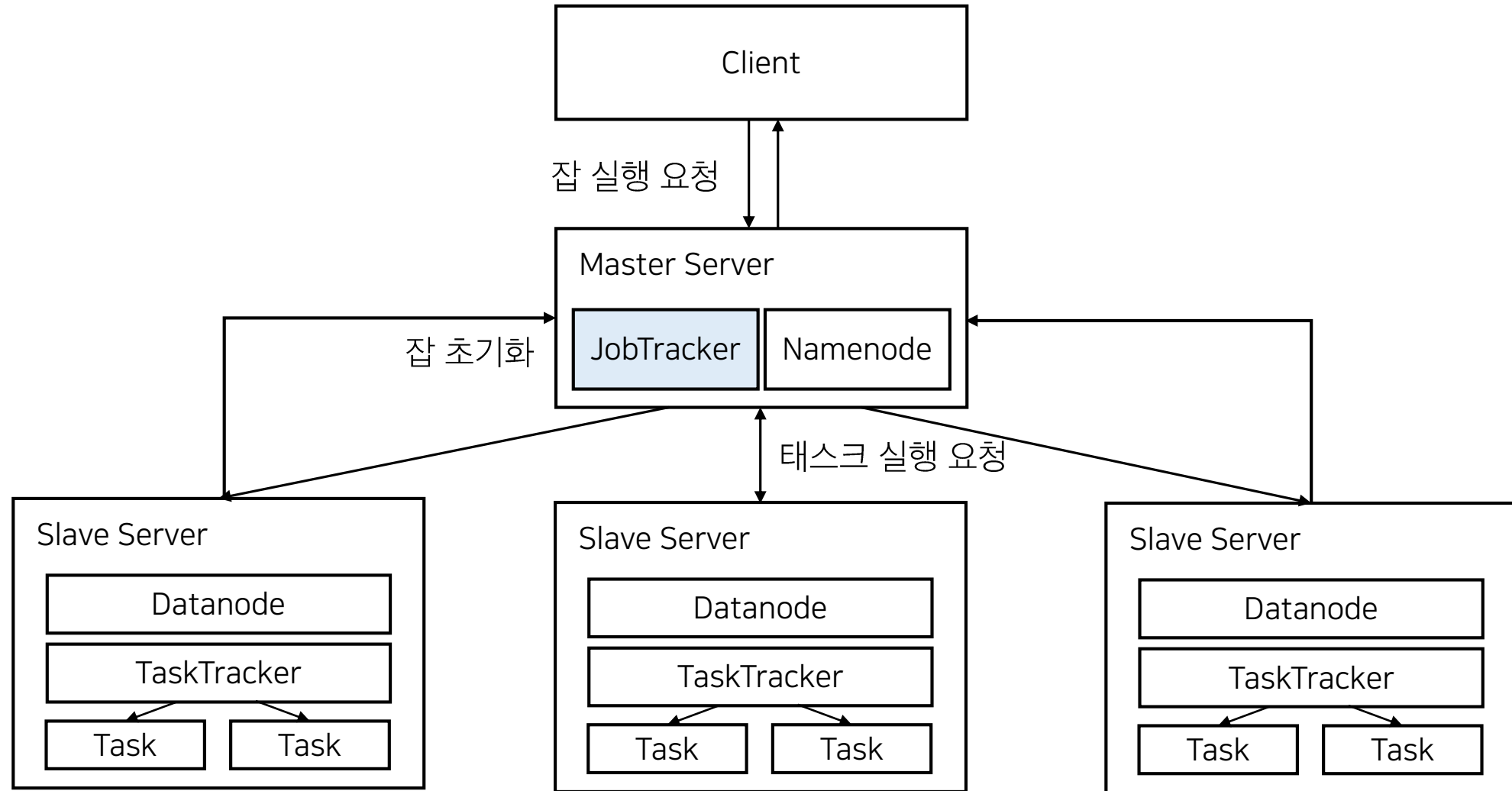
Hadoop 1.0 Architecture - MapReduce



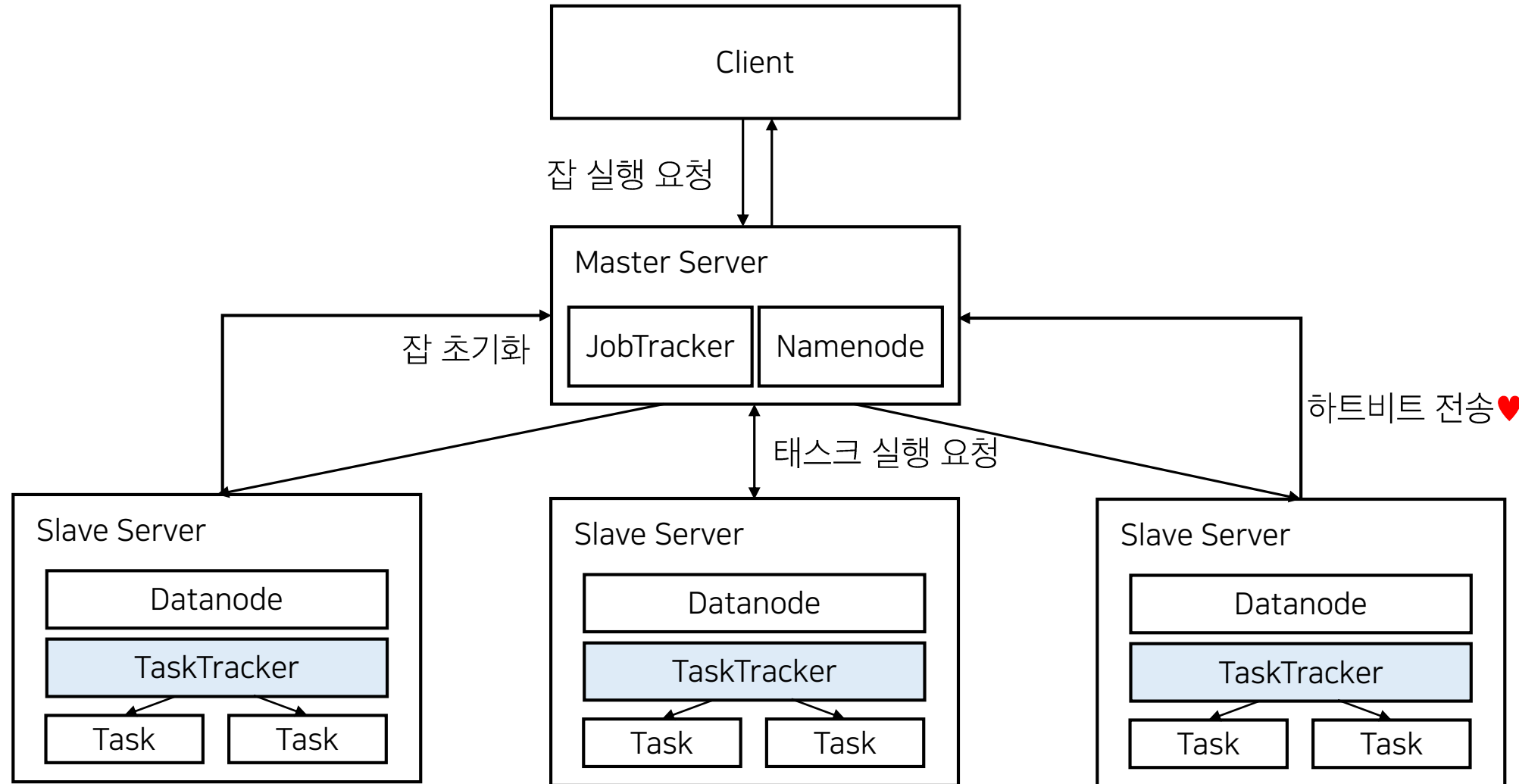
Hadoop 1.0 Architecture - MapReduce



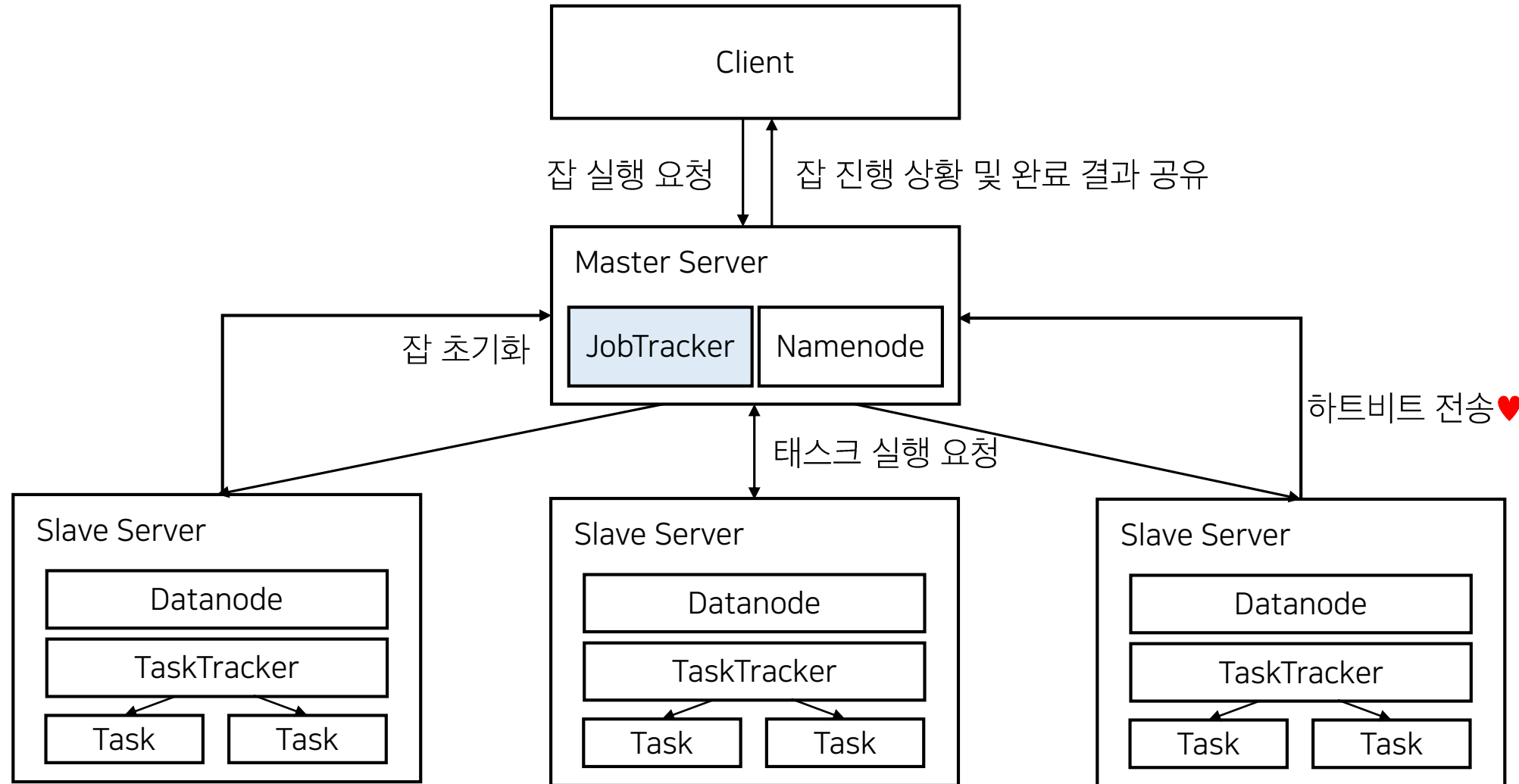
Hadoop 1.0 Architecture - MapReduce



Hadoop 1.0 Architecture - MapReduce



Hadoop 1.0 Architecture - MapReduce



Hadoop 1.0 Architecture – Constraints

✓ JobTracker의 메모리 이슈

- JobTracker가 많은 정보를 메모리에 저장하고 있는 만큼, 메모리가 부족해질 확률이 높다!
그런데 JobTracker가 고장나면 맵리듀스 잡 실행이 불가능하다! (맵리듀스의 단일 고장점, SPOF)

✓ 맵리듀스의 리소스 관리 방식

- ‘슬롯’이라는 개념으로 클러스터에서 실행할 태스크 개수를 관리 (ex. Map 슬롯 6개, Reduce 슬롯 6개)
잡이 Map 작업이나 Reduce 작업만 실행할 경우 남은 슬롯은 잉여 자원이 되어 효율적 리소스 관리 X

✓ 버전 통일성

✓ 클러스터 확장성

- 단일 클러스터는 4,000대, 최대 동시 실행 태스크는 40,000개가 한계!

Hadoop 2.0

Single Use System

Batch Apps

HADOOP 1.0

MapReduce

(cluster resource management
& data processing)

HDFS

(redundant, reliable storage)



Multi Use Data Platform

Batch, Interactive, Online, Streaming, ...

HADOOP 2.0

MapReduce

(batch)

Tez

(interactive)

Others

(varied)

YARN

(operating system: cluster resource management)

HDFS2

(redundant, reliable storage)

Hadoop 2.0

Single Use System

Batch Apps

HADOOP 1.0

MapReduce

cluster resource management
& data processing

HDFS

(redundant, reliable storage)

Multi Use Data Platform

Batch, Interactive, Online, Streaming, ...

HADOOP 2.0

MapReduce

(batch)

Tez

(interactive)

Others

(varied)

YARN

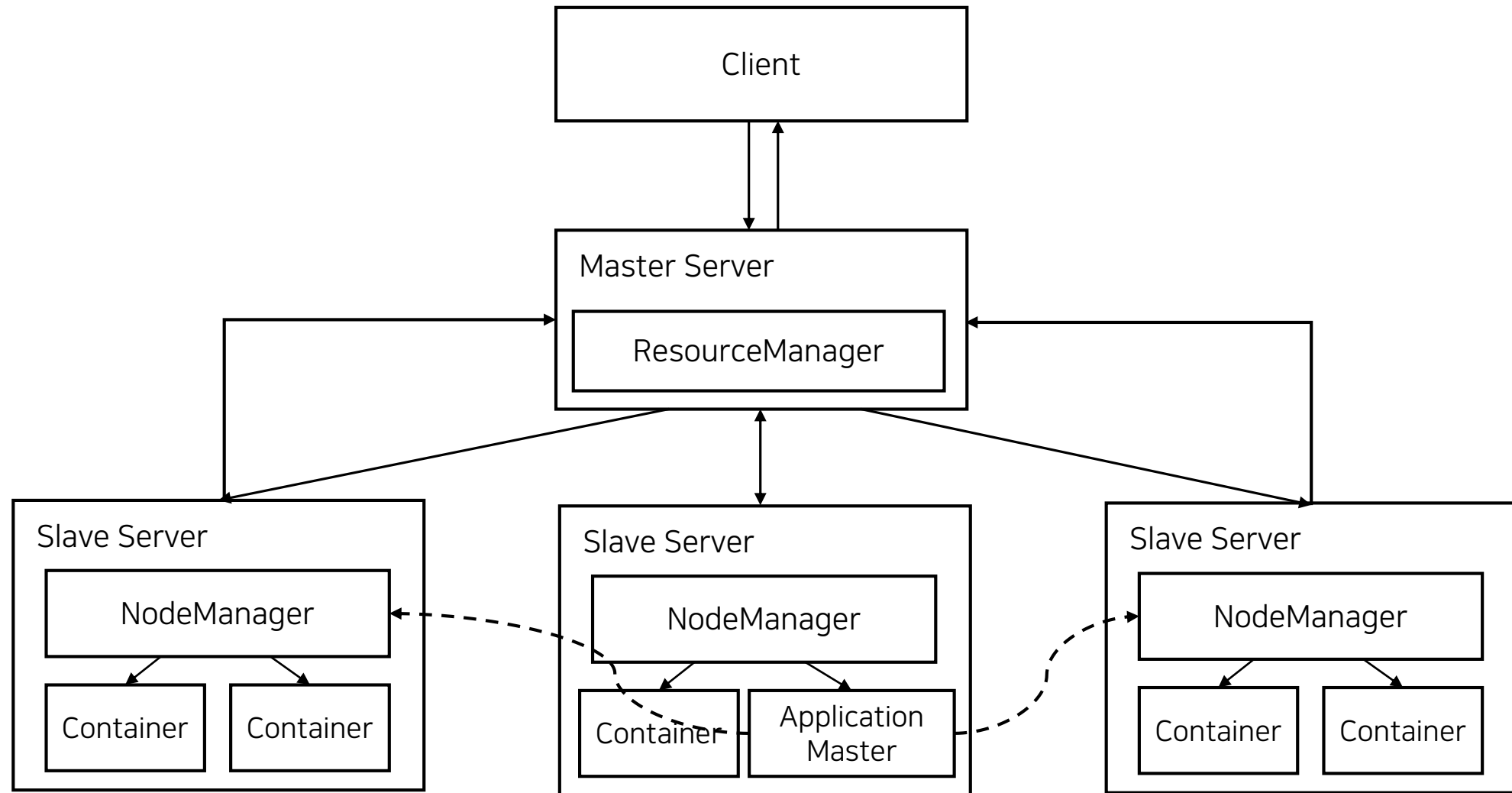
(operating system: cluster resource management)

HDFS2

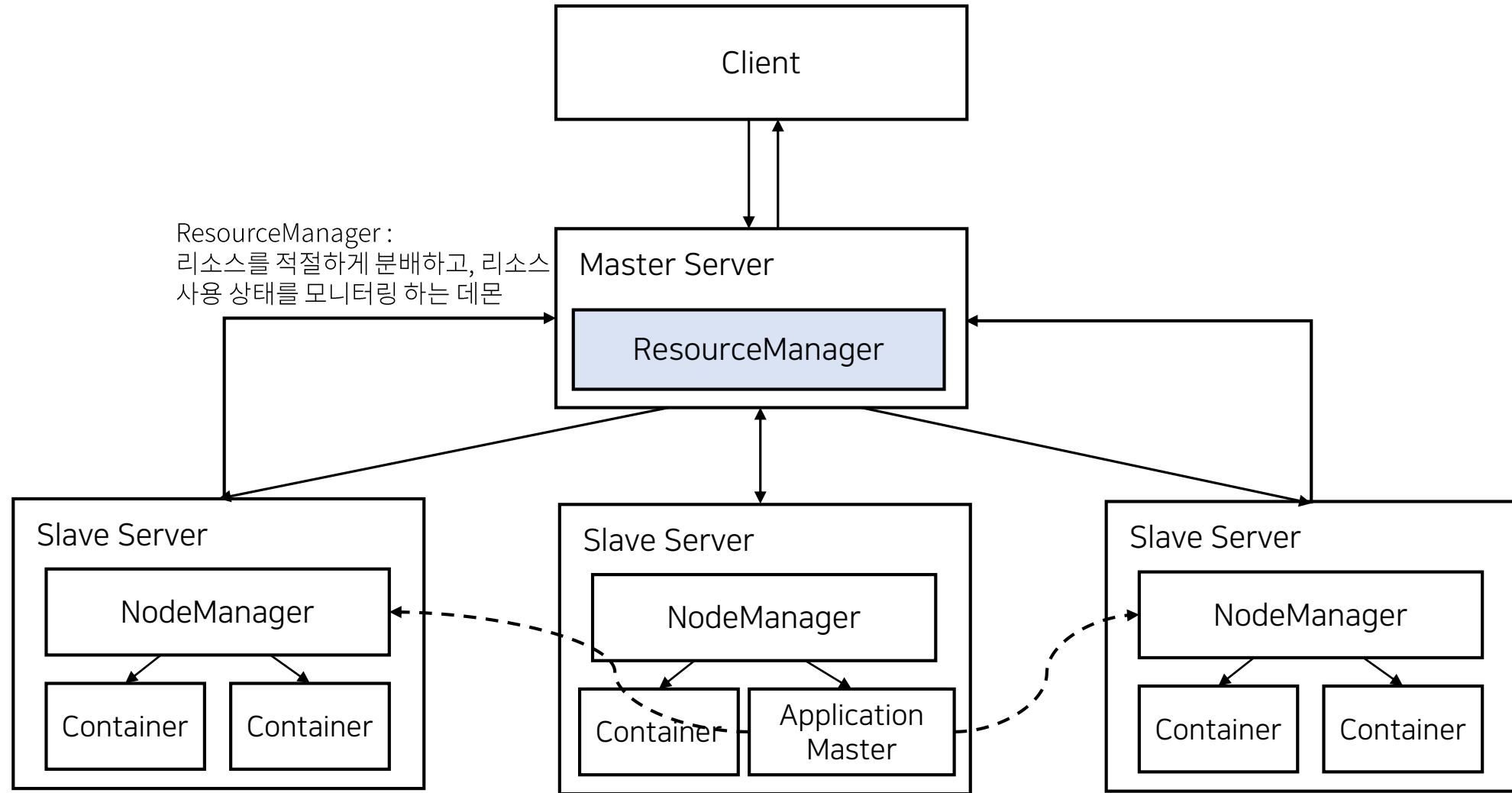
(redundant, reliable storage)



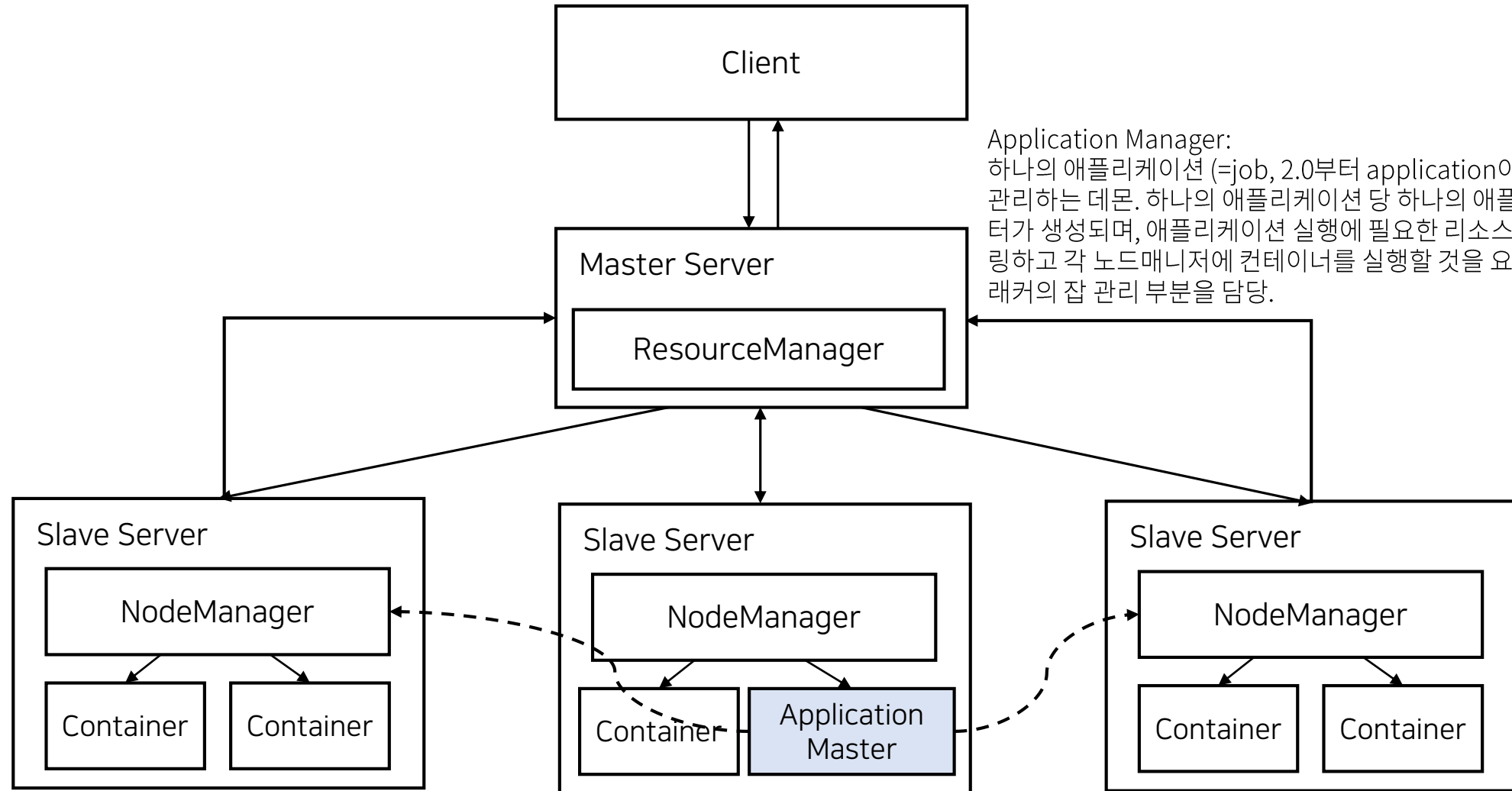
YARN – Yet Another **Resource** Negotiator



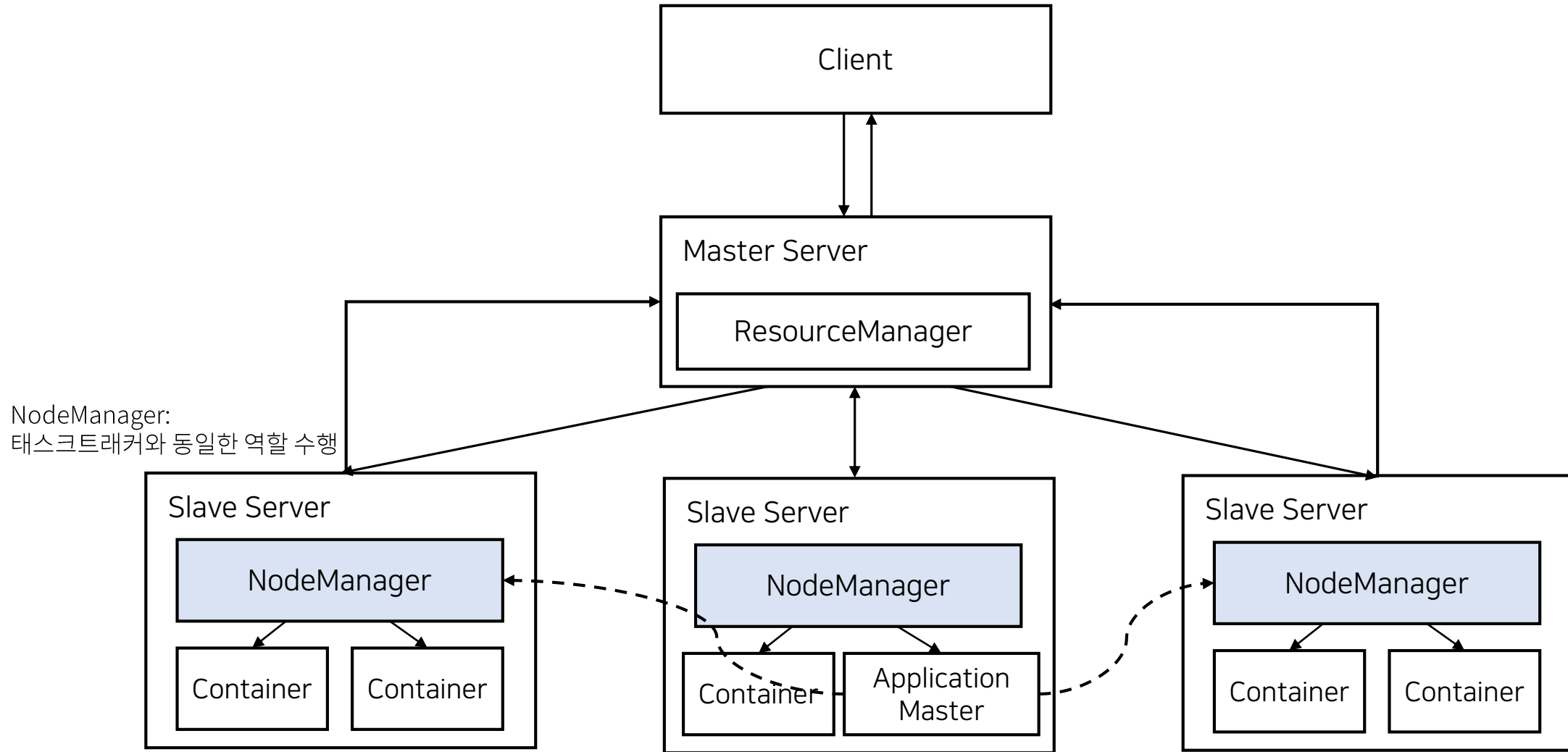
YARN – Yet Another **Resource** Negotiator



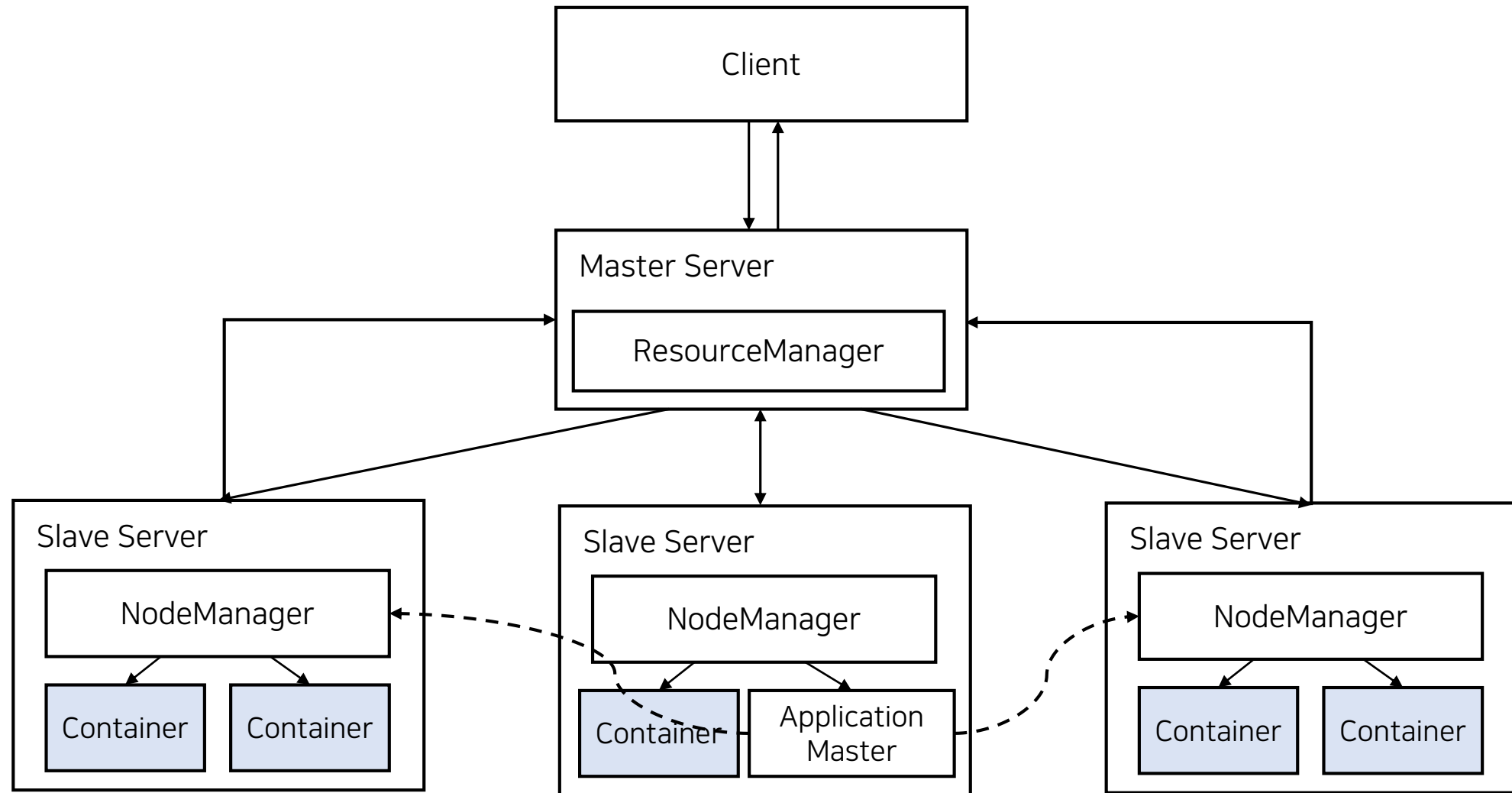
YARN – Yet Another **Resource** Negotiator



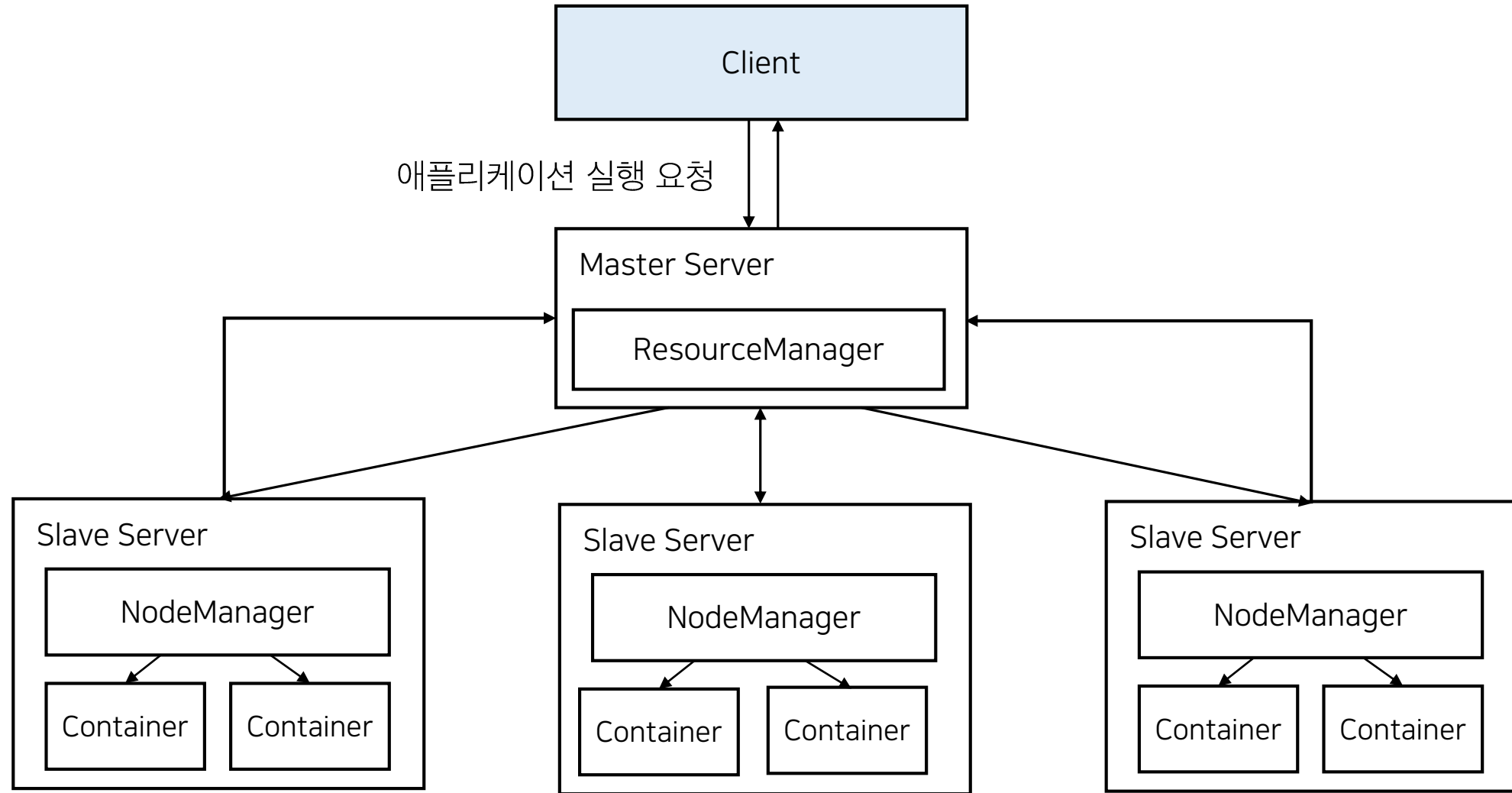
YARN – Yet Another **Resource** Negotiator



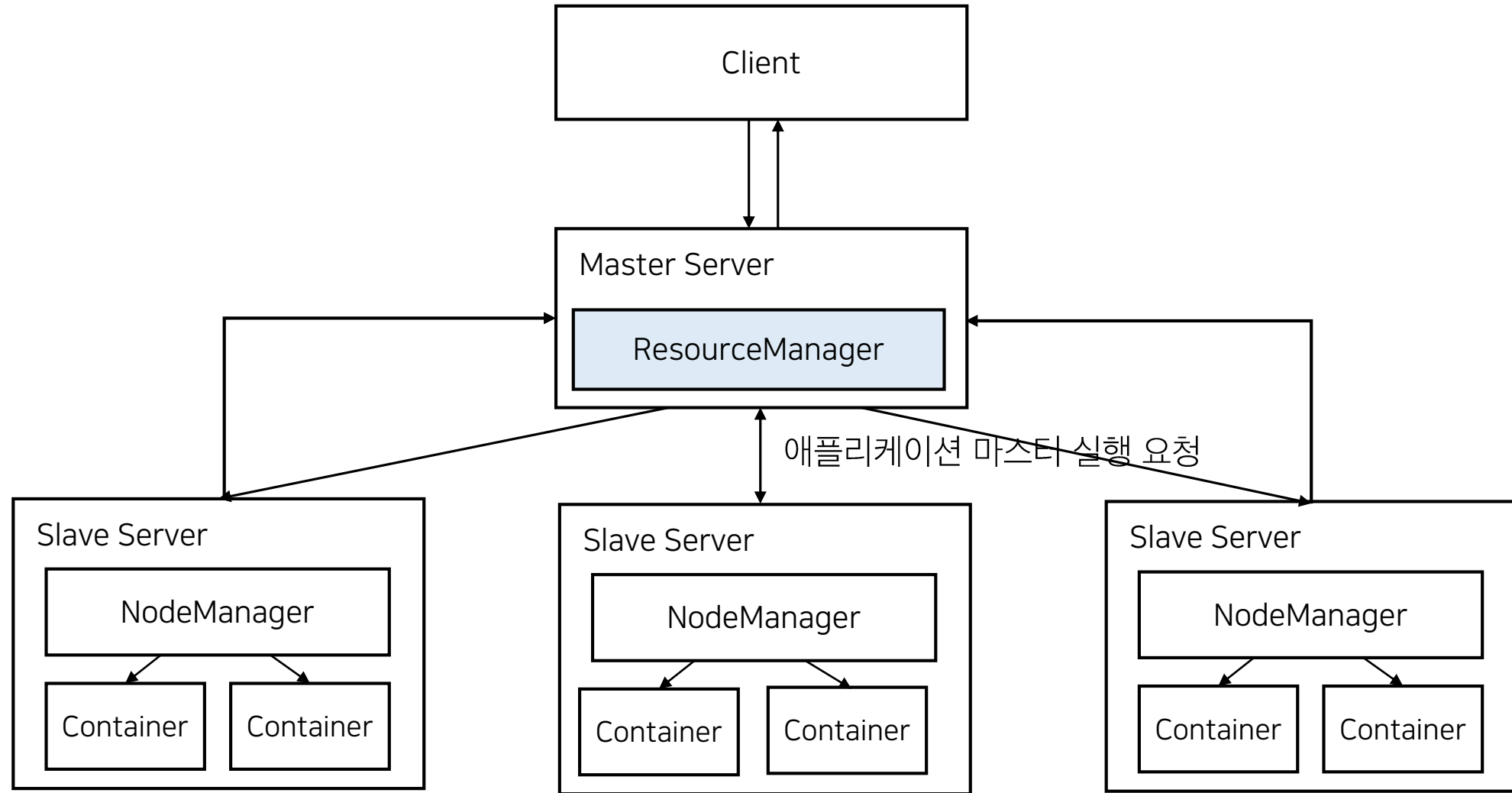
YARN – Yet Another **Resource** Negotiator



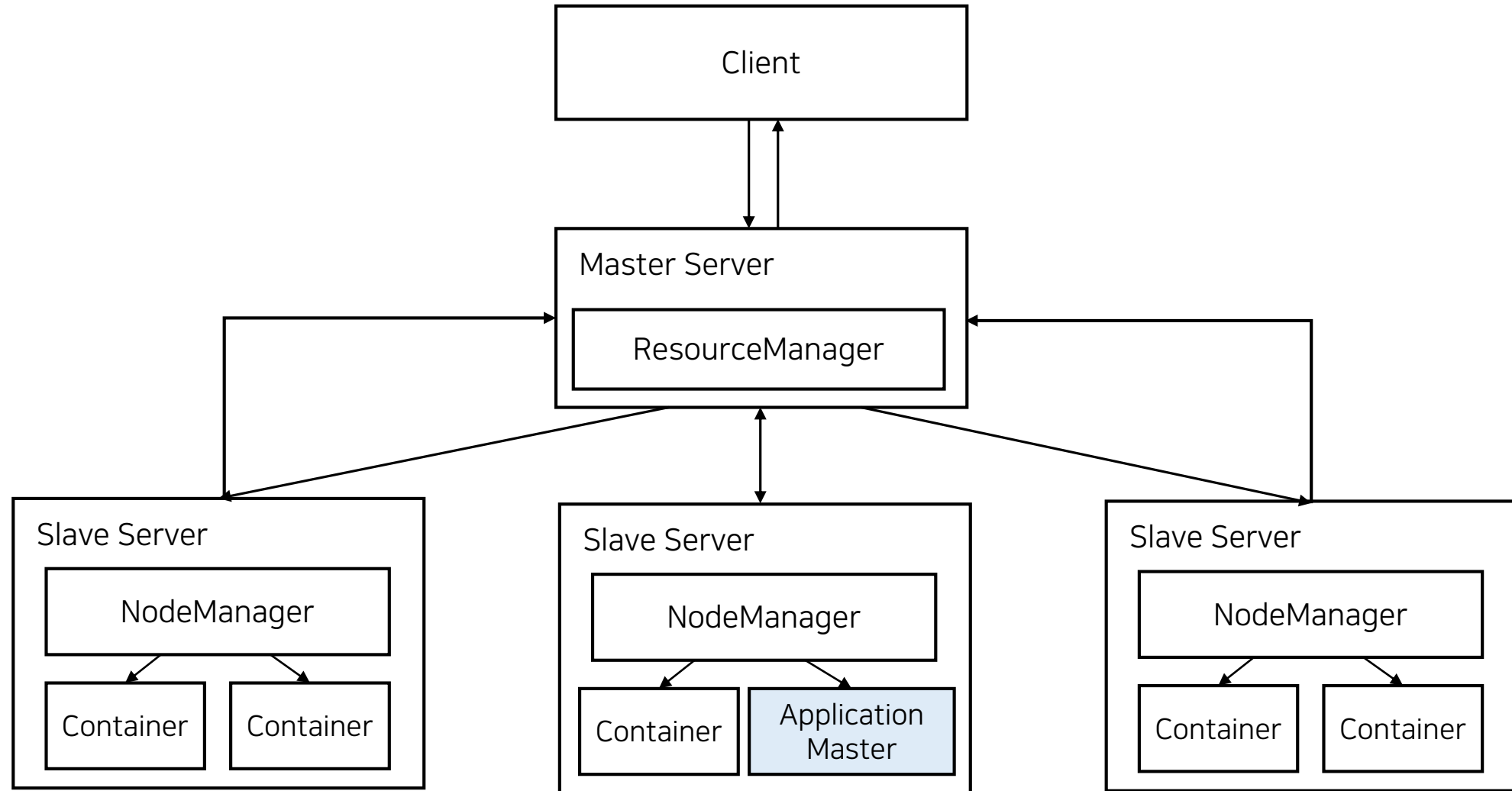
YARN – Yet Another **Resource** Negotiator



YARN – Yet Another **Resource** Negotiator

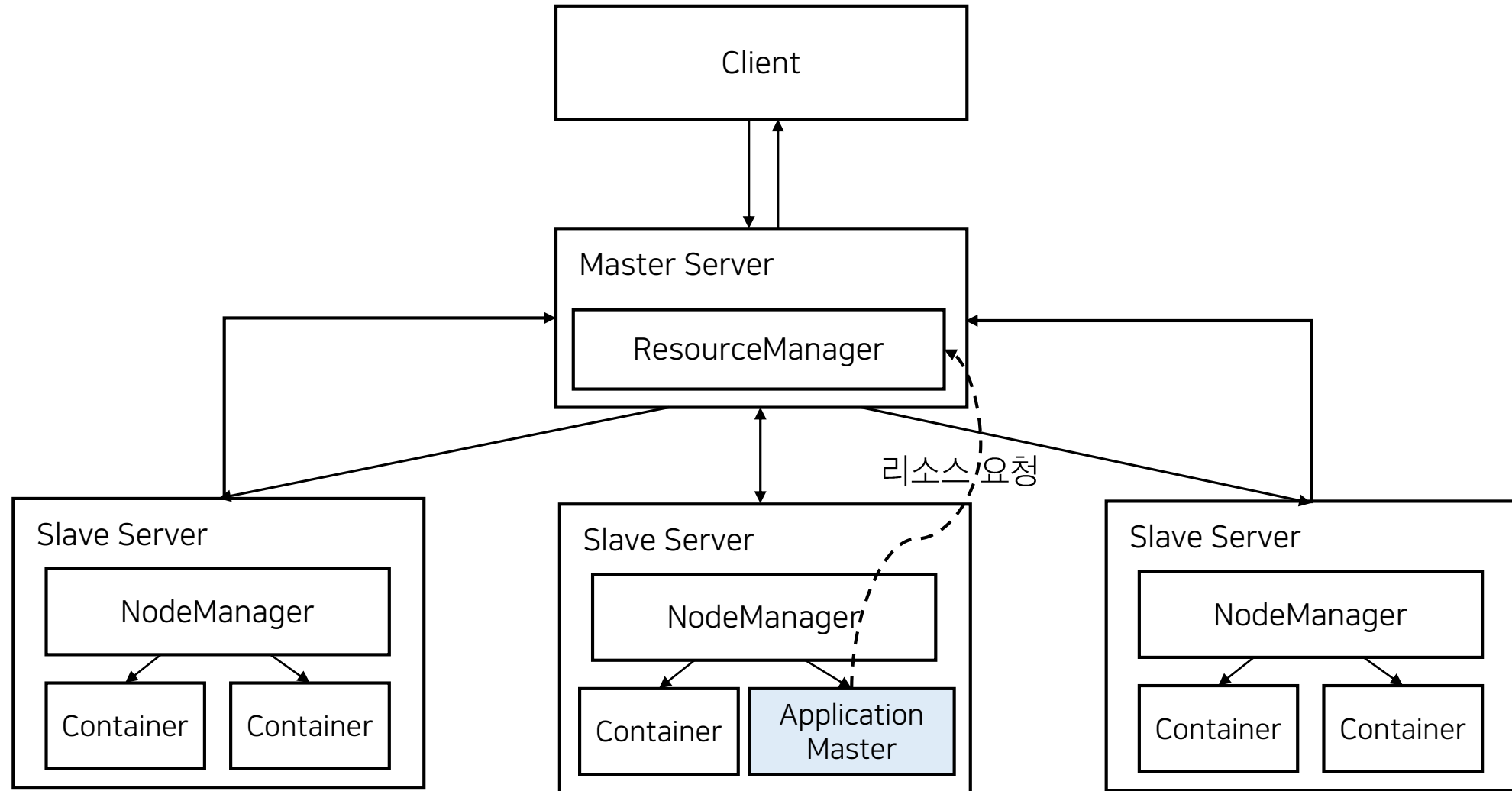


YARN – Yet Another **Resource** Negotiator

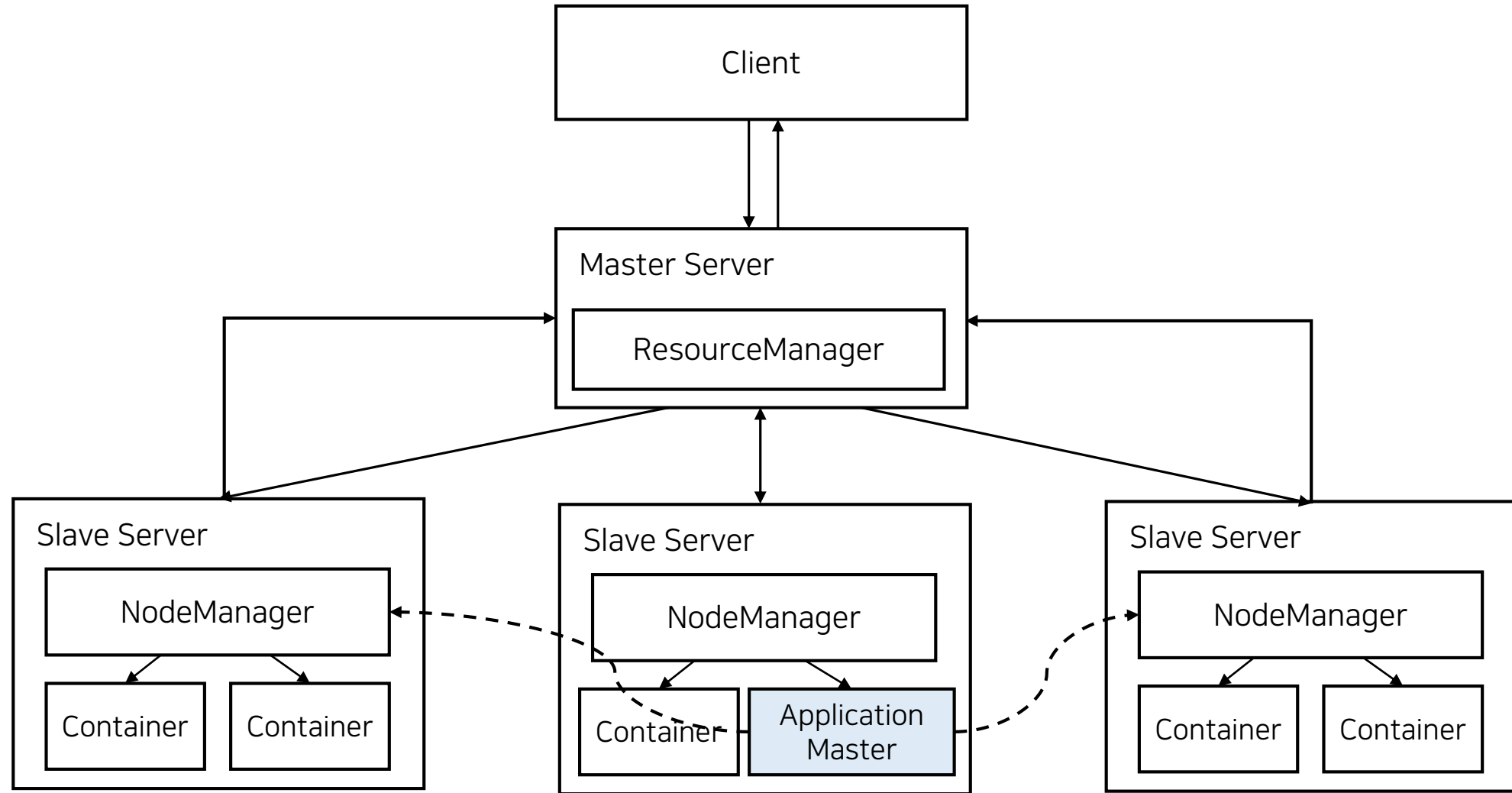


컨테이너에서 애플리케이션 마스터 실행

YARN – Yet Another **Resource** Negotiator



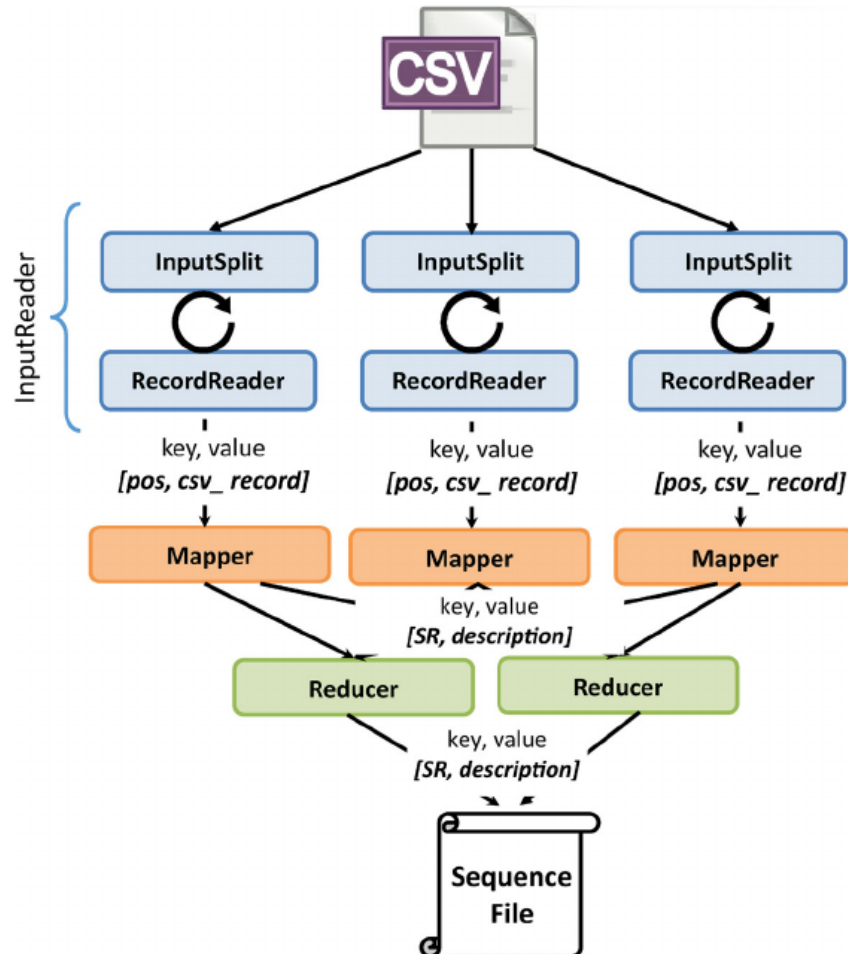
YARN – Yet Another **Resource** Negotiator



컨테이너 실행 요청

MapReduce Programming

- 각 단계에서 모든 Input, Output은 (Key , Value) 의 페어 (쌍) 형태이다.



파일 입력도 (Key, Value) 단위로 읽는다.

➔ java filewriter가 아닌 하둡에서 독자적으로 만든 InputFormat을 사용한다!

MapReduce Programming – InputFormat

InputFormat	기능
TextInputFormat	텍스트 파일을 분석할 때 사용하며, 개행 문자를 기준으로 레코드를 분류한다. Key는 라인 번호 값을 가지는 LongWritable 객체이며, Value에는 라인의 내용이 Text 형태로 담겨있다.
KeyValueTextInputFormat	TextInputFormat과 유사한 key에 임의의 값이 들어간다는 것에서 차이가 있다.
NLineInputFormat	Map 태스크가 입력 받을 텍스트 파일의 라인 수를 제한하고 싶을 때 사용한다.
DelegatingInputFormat	여러 개의 서로 다른 입력 포맷을 사용하는 경우에 각 경로에 대한 작업을 위임한다.

MapReduce Programming – InputFormat

InputFormat	기능
CombineFileInputFormat	여러 개의 파일을 스플릿으로 묶어 사용한다.
SequenceFileInputFormat	SequenceFile을 입력데이터로 쓸 때 사용한다.
SequenceFileAsBinaryInputFormat	SequenceFile의 키와 값을 임의의 바이너리 객체로 변환하여 사용한다.
SequenceFileAsTextInputFormat	SequenceFile의 키와 값을 Text 객체로 변환하여 사용한다.

MapReduce Programming – InputFormat

InputFormat	기능
CombineFileInputFormat	여러 개의 파일을 스플릿으로 묶어 사용한다.

SequenceFile?

바이너리 형태의 키와 값의 목록으로 구성된 텍스트 파일!
압축과 직렬화 프레임워크를 이용해 다양한 유형을 저장할 수 있다!

SequenceFile File Layout

Data

Key	Value	Key	Value	Key	Value	Key	Value
-----	-------	-----	-------	-----	-------	-----	-------

MapReduce Programming – InputFormat

InputFormat	기능
CombineFileInputFormat	여러 개의 파일을 스플릿으로 묶어 사용한다.
SequenceFileInputFormat	SequenceFile을 입력데이터로 쓸 때 사용한다.
SequenceFileAsBinaryInputFormat	SequenceFile의 키와 값을 임의의 바이너리 객체로 변환하여 사용한다.
SequenceFileAsTextInputFormat	SequenceFile의 키와 값을 Text 객체로 변환하여 사용한다.

MapReduce Programming – WritableComparable

데이터 타입 또한 Java에서 제공하는 기본 데이터 타입을 사용하지 않고,
Hadoop에서 만든 **MapReduce에 최적화된 데이터 타입**을 사용한다!

WritableComparable

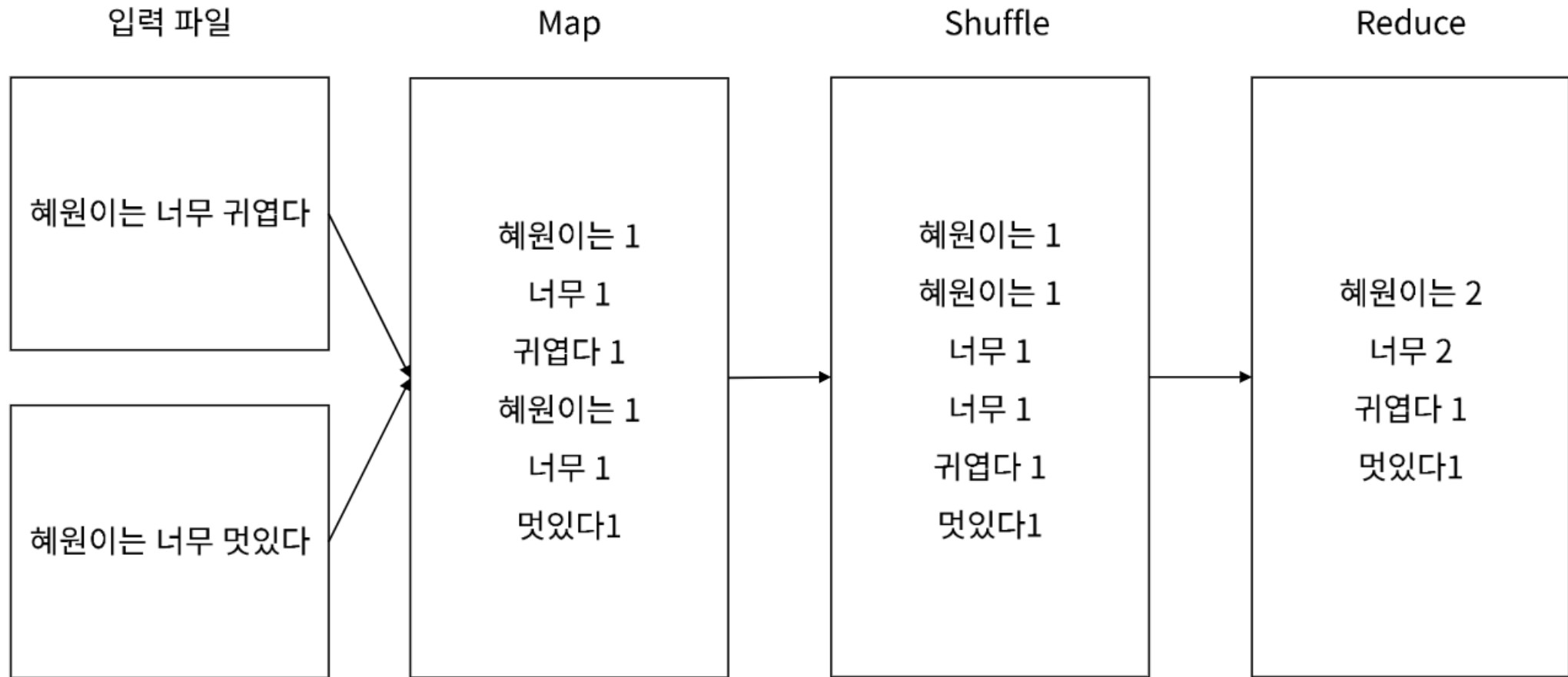
Writable : 데이터의 값을 직렬화하는 메소드를 갖고 있는 인터페이스

Comparable : 정렬을 처리하는 메소드를 제공하는 인터페이스

MapReduce Programming – WritableComparable

클래스명	대상 데이터 타입
BooleanWritable	Boolean
ByteWritable	단일 Byte
DoubleWritable	Double
FloatWritable	Float
IntWritable	Integer
LongWritable	Long
TextWrapper	UTF8 형식의 문자열
NullWritable	데이터 값이 필요 없을 경우에 사용

WordCount



Driver Class

```
package com.ybigta.example.wordcount;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCountDriver extends Configured implements Tool {
    public static void main(String[] args) throws Exception {
        Int result = ToolRunner.run(new Configuration(), new WordCountDriver(), args);
        System.exit(result);
    }
}
```

Driver Class

```
@Override
public int run(String[] args) throws Exception {
    String[] otherArgs = new GenericOptionsParser(getConf(), args). getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: WordCountDriver<input_path> <output_path>");
        System.exit(2);
    }
    Job job = Job.getInstance(getConf(), "word count");
    job.setJarByClass(WordCountDriver.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    job.waitForCompletion(true);
    return 0;
}
```

Driver Class

Job의 환경 설정 및 클러스터로의 전송을 담당하는 class!

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

Driver Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

필요한 라이브러리를 추가

Driver Class

```
public class WordCountDriver extends Configured implements Tool {
```

Configured : 실행 시점의 설정 정보를 가져오기 위하여 상속 (실행 시 커맨드 창에 입력하는 Output Directory 등)
Tool : 무정언어가 설명해줄 예정 😊

```
@Override  
public int run(String[] args) throws Exception {  
    String[] otherArgs = new GenericOptionsParser(getConf(), args). getRemainingArgs();  
    if (otherArgs.length != 2) {  
        System.err.println("Usage: WordCountDriver<input_path> <output_path>");  
        System.exit(2);  
    }  
}
```

Driver Class

```
public class WordCountDriver extends Configured implements Tool {  
  
    public static void main(String[] args) throws Exception {  
        Int result = ToolRunner.run(new Configuration(), new WordCountDriver(), args);  
        System.exit(result);  
    }  
}
```

run : job을 실행하는 함수. 0을 리턴하면 정상적으로 종료된 것이며, 0이 아닌 수를 반환할 경우 비정상 종료를 의미한다.
args는 command에서 실행할 때 명령문에 넣은 인자를 담은 array이다.

```
if (otherArgs.length != 2) {  
    System.err.println("Usage: WordCountDriver<input_path> <output_path>");  
    System.exit(2);  
}
```

Driver Class

```
public class WordCountDriver extends Configured implements Tool {  
  
    public static void main(String[] args) throws Exception {  
        Int result = ToolRunner.run(new Configuration(), new WordCountDriver(), args);  
        System.exit(result);  
    }  
  
    @Override  
    public int run(String[] args) throws Exception {  
        String[] otherArgs = new GenericOptionsParser(getConf(), args). getRemainingArgs();  
    }  
}
```

otherArgs에 설정 정보를 저장한다.

GenericOptionsParser는 Command Line을 parsing해주며, getConf()는 현재 설정정보를 반환한다.

Driver Class

```
public class WordCountDriver extends Configured implements Tool {

    public static void main(String[] args) throws Exception {
        Int result = ToolRunner.run(new Configuration(), new WordCountDriver(), args);
        System.exit(result);
    }

    @Override
    public int run(String[] args) throws Exception {
        String[] otherArgs = new GenericOptionsParser(getConf(), args). getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: WordCountDriver<input_path> <output_path>");
            System.exit(2);
        }
    }
}
```

설정 정보의 배열 길이가 2가 아닐 경우 (input, output 디렉토리 외 추가 인자 입력 혹은 디렉토리 누락)
에러 메시지를 띄우며 비정상 종료한다.

Driver Class

```
Job job = Job.getInstance(getConf(), "word count");
```

Job 객체 생성

```
job.setReducerClass(IntSumReducer.class);
```

```
job.setInputFormatClass(TextInputFormat.class);
```

```
job.setOutputFormatClass(TextOutputFormat.class);
```

```
job.setOutputKeyClass(Text.class);
```

```
job.setOutputValueClass(IntWritable.class);
```

```
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
```

```
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

```
job.waitForCompletion(true);
```

```
return 0;
```

```
}
```

```
}
```

Driver Class

```
Job job = Job.getInstance(getConf(), "word count");
```

```
job.setJarByClass(WordCountDriver.class);  
job.setMapperClass(TokenizerMapper.class);  
job.setCombinerClass(IntSumReducer.class);  
job.setReducerClass(IntSumReducer.class);
```

setJarByClass()를 해당 driver class가 포함된 jar를 Job의 jar 파일로 설정
어떤 클래스를 Mapper, Reducer를 사용할 것인지 지정

```
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

```
job.waitForCompletion(true);  
return 0;
```

```
}
```

```
}
```

Driver Class

```
Job job = Job.getInstance(getConf(), "word count");  
  
job.setJarByClass(WordCountDriver.class);  
job.setMapperClass(TokenizerMapper.class);  
job.setCombinerClass(IntSumReducer.class);  
job.setReducerClass(IntSumReducer.class);
```

setJarByClass()를 해당 driver class가 포함된 jar를 Job.
어떤 클래스를 Mapper, Reducer를 사용할 것인가

```
job.setOutputValueClass(IntWritable.class);  
  
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
  
job.waitForCompletion(true);  
return 0;  
}  
}
```



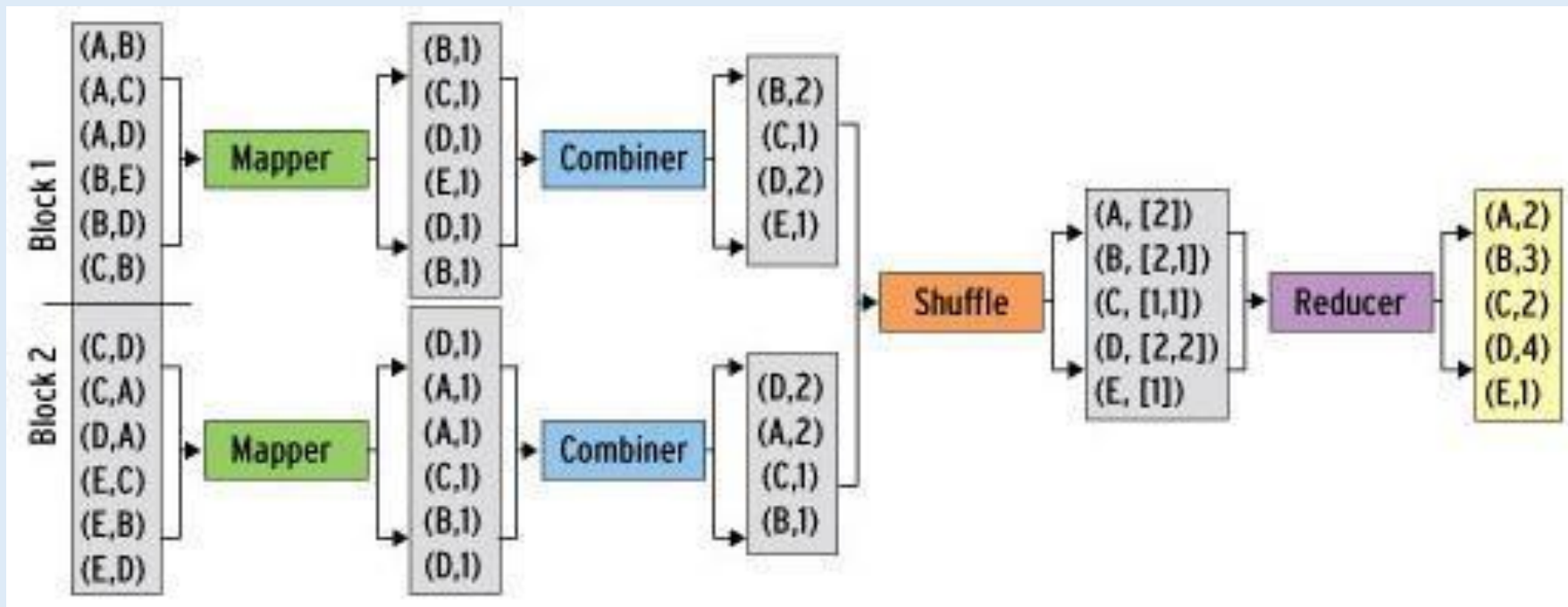
Driver Class

```
Job job = Job.getInstance(getConf(), "word count");
```

```
job.setJarByClass(WordCountDriver.class);
```

```
job.setMapperClass(TokenizerMapper.class);
```

```
job.setCombinerClass(IntSumReducer.class);
```



Driver Class

```
Job job = Job.getInstance(getConf(), "word count");

job.setJarByClass(WordCountDriver.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

각 단계마다 key - value 데이터 타입 설정

```
return 0;
```

```
}
```

```
}
```

Driver Class

```
Job job = Job.getInstance(getConf(), "word count");

job.setJarByClass(WordCountDriver.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

Input / Output Directory ^{설정}
addInputPath() vs. setInputPaths()

Driver Class

```
Job job = Job.getInstance(getConf(), "word count");

job.setJarByClass(WordCountDriver.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
```

Job 실행

* Job을 실행하는 두 가지 메소드 : `waitForCompletion()` vs. `submit()`

`waitForCompletion()`은 Job을 실행시키고 끝날 때까지 기다렸다가 종료하는 데에 비해, `submit()`은 바로 종료한다는 차이가 있다!

Job이 다 끝나면 정상 종료를 의미하는 0을 리턴한다

```
job.waitForCompletion(true);
return 0;
```

```
}
```

```
}
```

Driver Class

```
public class WordCountDriver extends Configured implements Tool {

    public static void main(String[] args) throws Exception {
        Int result = ToolRunner.run(new Configuration(), new WordCountDriver(), args);
        System.exit(result);
    }

    @Override
    public int run(String[] args) throws Exception {
        String[] otherArgs = new GenericOptionsParser(getConf(), args). getRemainingArgs();
        if (otherArgs.length != 2) {
            System.err.println("Usage: WordCountDriver<input_path> <output_path>");
            System.exit(2);
        }
    }
}
```

Driver Class

```
Job job = Job.getInstance(getConf(), "word count");

job.setJarByClass(WordCountDriver.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

job.waitForCompletion(true);
return 0;
}
}
```

Driver Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCountDriver extends Configured implements Tool {
    public static void main(String[] args) throws Exception {
        Int result = ToolRunner.run(new Configuration(), new WordCountDriver(), args);
        System.exit(result);
    }
}
```

Driver Class

```
@Override
public int run(String[] args) throws Exception {
    String[] otherArgs = new GenericOptionsParser(getConf(), args). getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: WordCountDriver<input_path> <output_path>");
        System.exit(2);
    }
    Job job = Job.getInstance(getConf(), "word count");
    job.setJarByClass(WordCountDriver.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    job.waitForCompletion(true);
    return 0;
}
```

Mapper Class

```
package com.ybigta.example.wordcount;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import java.util.StringTokenizer;

public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable ONE = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException{

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, ONE);
        }
    }
}
```


Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
```

Mapper 생성

```
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException{

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, ONE);
        }
    }
}
```

Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
```

네 개의 매개변수 중 앞의 두개는 Input key - Value의 데이터 타입을 의미

```
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException{

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, ONE);
        }
    }
}
```

Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
```

네 개의 매개변수 중 뒤의 두개는 output의 key - value의 데이터 타입을 의미

```
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException{

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, ONE);
        }
    }
}
```

Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {  
    private final static IntWritable ONE = new IntWritable(1);  
    private Text word = new Text();
```

Output key와 value가 될 객체를 미리 선언!

WordCount의 경우 Map 과정이 끝나면 value 값은 모두 1이 되기 때문에 상수 객체를 만들어주었다.

```
    StringTokenizer itr = new StringTokenizer(value.toString());  
    while (itr.hasMoreTokens()) {  
        word.set(itr.nextToken());  
        context.write(word, ONE);  
    }  
}
```

Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {  
    private final static IntWritable ONE = new IntWritable(1);  
    private Text word = new Text();
```

@Override

```
public void map(LongWritable key, Text value, Context context)  
    throws IOException, InterruptedException{
```

Map 함수에 Mapper가 해야 할 일을 구현한다.
Context는 job에 대한 설정 정보이다.

```
}
```

```
}
```

```
}
```

Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {  
    private final static IntWritable ONE = new IntWritable(1);  
    private Text word = new Text();  
  
    @Override  
    public void map(LongWritable key, Text value, Context context)  
        throws IOException, InterruptedException{  
  
        StringTokenizer itr = new StringTokenizer(value.toString());  
  
    }  
}
```

한 줄을 단어 단위로 쪼갬다.

Mapper Class

(전략)

```
public class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable ONE = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException{

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, ONE);
        }
    }
}
```

각 단어들에 대해 (단어, 1)로 구성된 쌍을 output으로 작성한다.

Reducer Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    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);
    }
}
```


Reducer Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
```

```
public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

Reducer 생성 및 Input / Output의 Key - Value 데이터 타입 설정

```
    throws IOException, InterruptedException{
        Int sum = 0;
        for (IntWritable val: values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Reducer Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();
```

Output의 value가 될 객체를 생성한다. (reducer에서는 key가 변하지 않기 때문에 따로 객체 생성하지 않음)

```
    int sum = 0;
    for (IntWritable val: values) {
        sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
}
```

Reducer Class

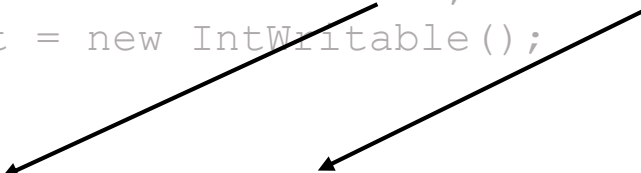
```
package com.ybigta.example.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException{

        result.set(sum);
        context.write(key, result);
    }
}
```



Reducer의 input 데이터는 (Word, 1)인데, 계산을 편리하기 위해 같은 키값을 가진 쌍들을 묶어 (Word, [1,1,1,1,1])의 형태로 Input을 변형한다. (Shuffle)

Reducer Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

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

단어의 개수를 담은 int 변수를 설정하고, iterable 안의 수를 모두 더한다.

Reducer Class

```
package com.ybigta.example.wordcount;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    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);
    }
}
```

result에 sum의 값을 할당한 후, (word, 개수)로 output을 작성한다.

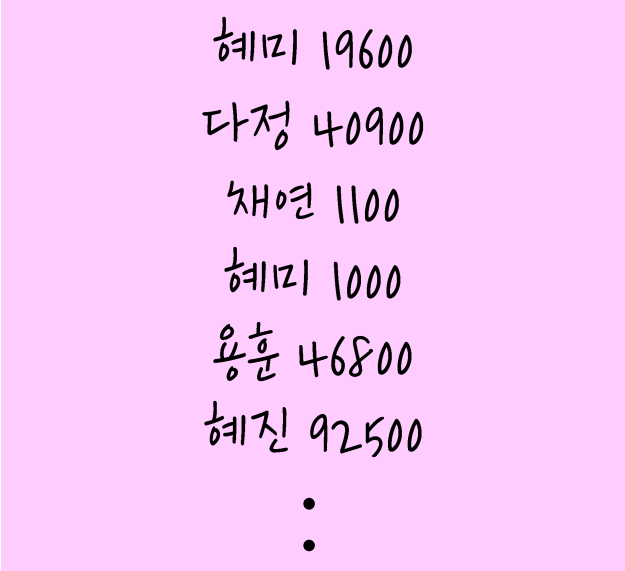
Practice(1) - Wordcount

WordCount를 직접 돌려봅시다!

Practice(2)

당신은 쇼핑몰 '데엔'의 오너이다.

8월에 있을 고객 감사 이벤트를 위해 상반기 가장 구매 금액이 가장 높은 고객을 찾으려고 한다.



혜미 19600
다정 40900
재연 1100
혜미 1000
용훈 46800
혜진 92500
•
•

<Input 예시>

오늘 배운 MapReduce 모델을 이용하여 이 달의 우수 고객을 찾으시오!