

Hadoop 2.0 Fundamental

HDFS, MapReduce, Pig, Hive Fundamental

Hadoop Architecture (HDFS, MapReduce), Data Analysis (Pig, Hive)

Contents

- 01 Hadoop Overview
- 02 HDFS
- 03 MapReduce
- 04 Pig
- 05 Hive

Hadoop Overview

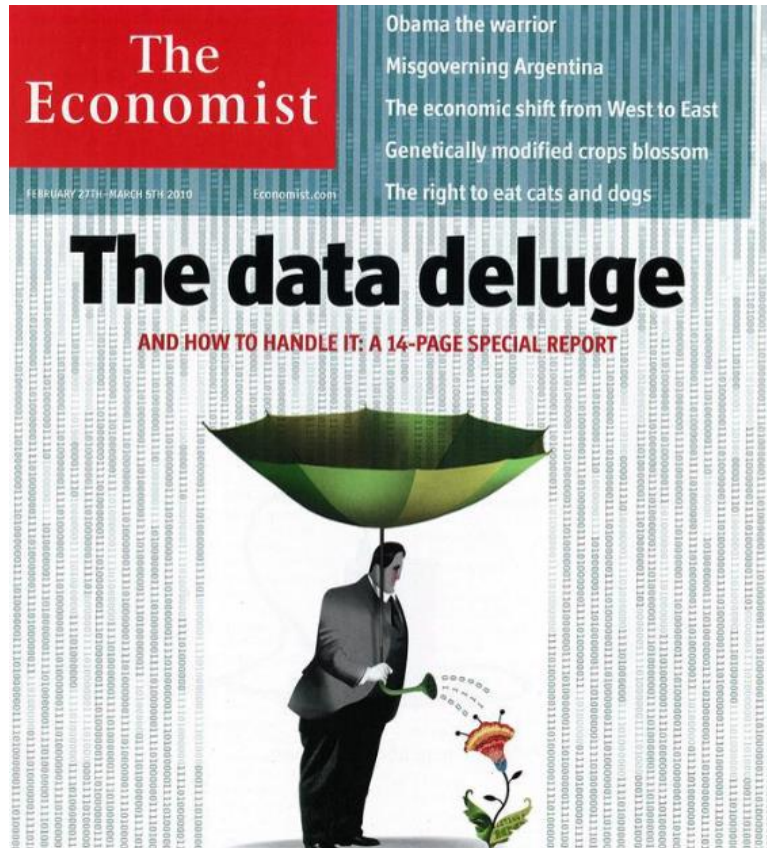
Hadoop History

Hadoop Architecture

Hadoop Installation

Hadoop Overview

Hadoop History : What is Bigdata?



The Economist 2010.2



<http://www.go-gulf.com/blog/60-seconds/>

Hadoop Overview

Hadoop History : Defining BigData

빅데이터 정의



1. 크기가 크고 빠르게 증가하는 데이터 파일
2. 일반적으로 TB 또는 PB
3. 비 정형 데이터
4. 관계형 모델이 적합하지 않음
5. 사용자, 어플리케이션, 시스템, 센서 등에서 파생된 데이터

Hadoop Overview

Hadoop History : Big Data Technology

분야	솔루션
NoSQL	HBase, Cassandra, MongoDB, CouchDB, Couchbase, Cloudata, Riak, Neo4j
Cache	Redis, Memcached
RPC	Thrift, Avro, Protocol Buffer
Collect	Scribe, Flume, Chukwa, Logstash, Fluentd
Query	Hive, Pig, Spark SQL, Hcatalog, Impala, Tajo, BigQuery
Streaming	Akka, Spark Streaming, Storm, Esper, S4
Search	Elastic Search, Solr, Katta
File System	Hadoop, Swift, GlusterFS, Ceph
ETC	Machine Learning(Mahout), Distributed Coordinator(Zookeeper) Queue(Kafka), Data Integration(Sqoop), Statistics(R), Workflow(Oozie)

Hadoop Overview

Hadoop History : Hadoop Motive

- Google
 - 400억 개의 웹 페이지 X 각 30KB = Petabyte
 - 오늘날 평균 디스크 읽기 성능 = 약 120MB/sec
 - 웹 페이지를 읽는 데 약 3개월 소요
 - 1000개의 드라이브에 저장하고 사용
- 시사점
 - 빠른 데이터 처리에 비해 매우 느린 I/O(Read/Write) 작업
 - 대용량 데이터를 위한 컴퓨팅은 이전과는 근본적으로 다른 접근 방법이 필요
 - 그렇다면 해결책은 → 데이터 중심의 병렬 reads 시스템
 - 1 HDD = 75MB/sec → 100 HDDs = 75 GB/sec

<http://www.internetlivestats.com/google-search-statistics/#trend>

Hadoop Overview

Hadoop History : RDBMS vs BigData

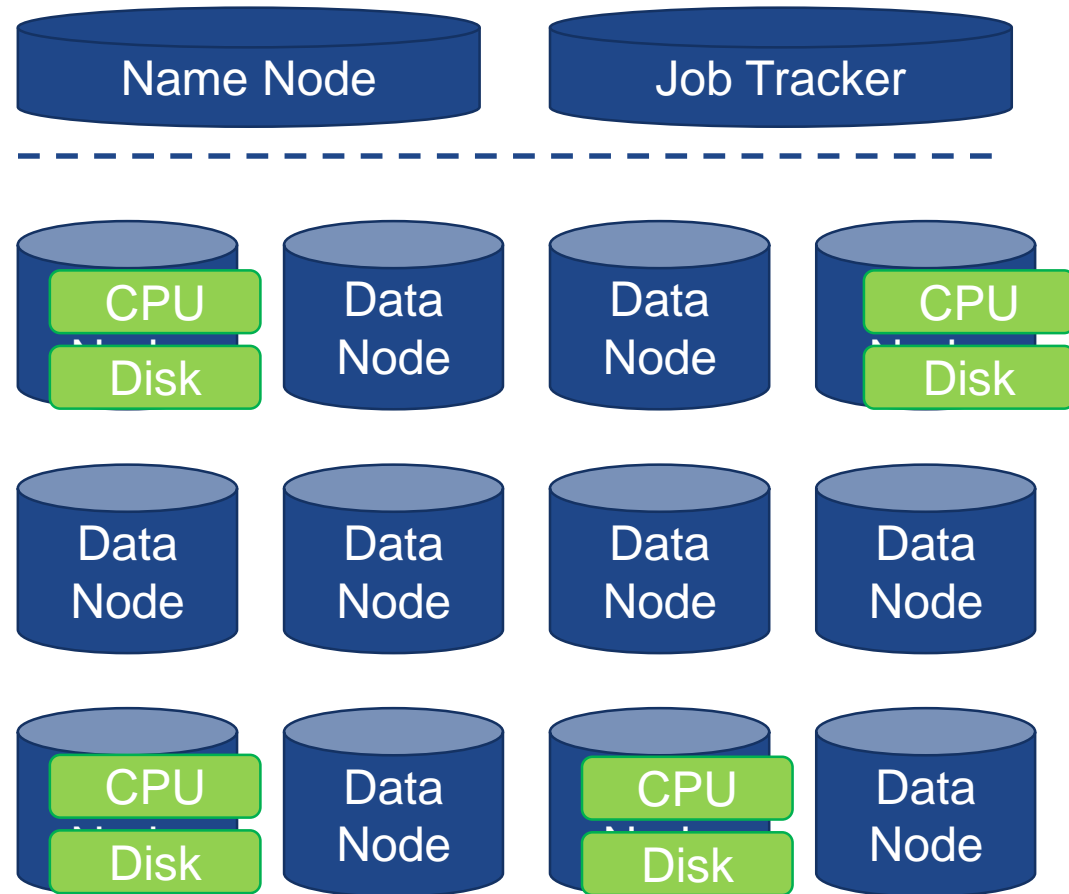
태그	관계형 데이터베이스	빅데이터
데이터	정규화를 만족하는 정형 데이터	비정형 데이터 문자, 동영상, 이미지 데이터
하드웨어	고가의 저장장치 관계형 데이터베이스 서버 Data Warehouse	저가의 Linux 또는 클라우드 환경
비정형 데이터	Scale Up	Scale Out
소프트웨어	관계형 데이터베이스 시스템 통계 패키지(SAS, SPSS) Data Mining	오픈소스 형태의 무료 패키지 Hadoop, NoSQL 무료 통계 솔루션 (R)
통계 법칙	파레토 법칙	롱테일 법칙

출처 : 이클립스 환경에서의 빅데이터 프로그래밍

Hadoop Overview

Hadoop History : Distributed Computing

- 신뢰성 요구사항
 - 선형적인 확장성
 - 장애에 대한 고립
 - 데이터 장애의 복구
- 병렬 처리 요구사항
 - 1PB 이상의 대용량 데이터 처리
 - 수 천대의 CPU를 동시에 최대 활용
 - 사용하기 쉬워야 함
- 급진적인 아키텍처
 - 비 공유 아키텍처
 - HDFS
 - MapReduce
 - 데이터가 있는 곳에서 컴퓨팅이 수행



Hadoop Overview

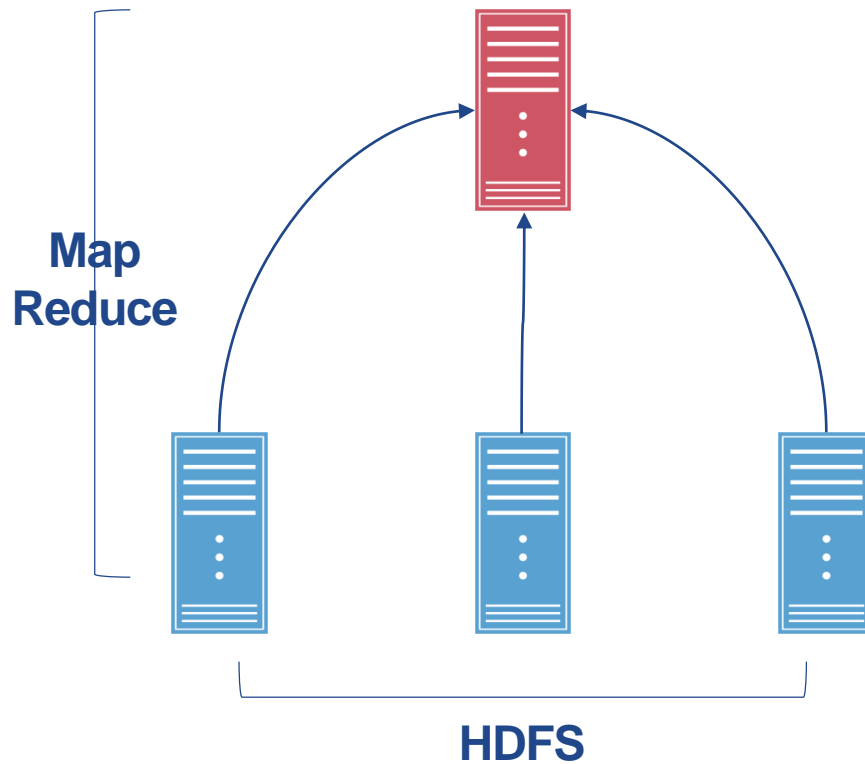
Hadoop History : What is Hadoop?

- 두 개의 구성요소
 - 오픈 소스 데이터 스토리지 : **HDFS**
 - 프로세싱 API : **MapReduce**
- 추가 프로젝트 / 라이브러리
 - HBase
 - Hive
 - Pig
 - etc...

Open Source	Commercial	Cloud
Apache Hadoop	Cloudera	AWS
	Hortonworks	Windows Azure HDInsight
	MapR	

Hadoop Overview

Hadoop History : What is Hadoop?



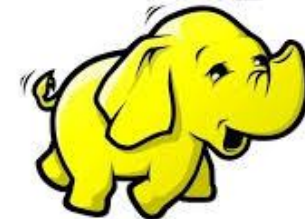
HDFS – 신뢰할 수 있는 공유 스토리지

+

MapReduce – 분산 컴퓨팅

=

hadoop



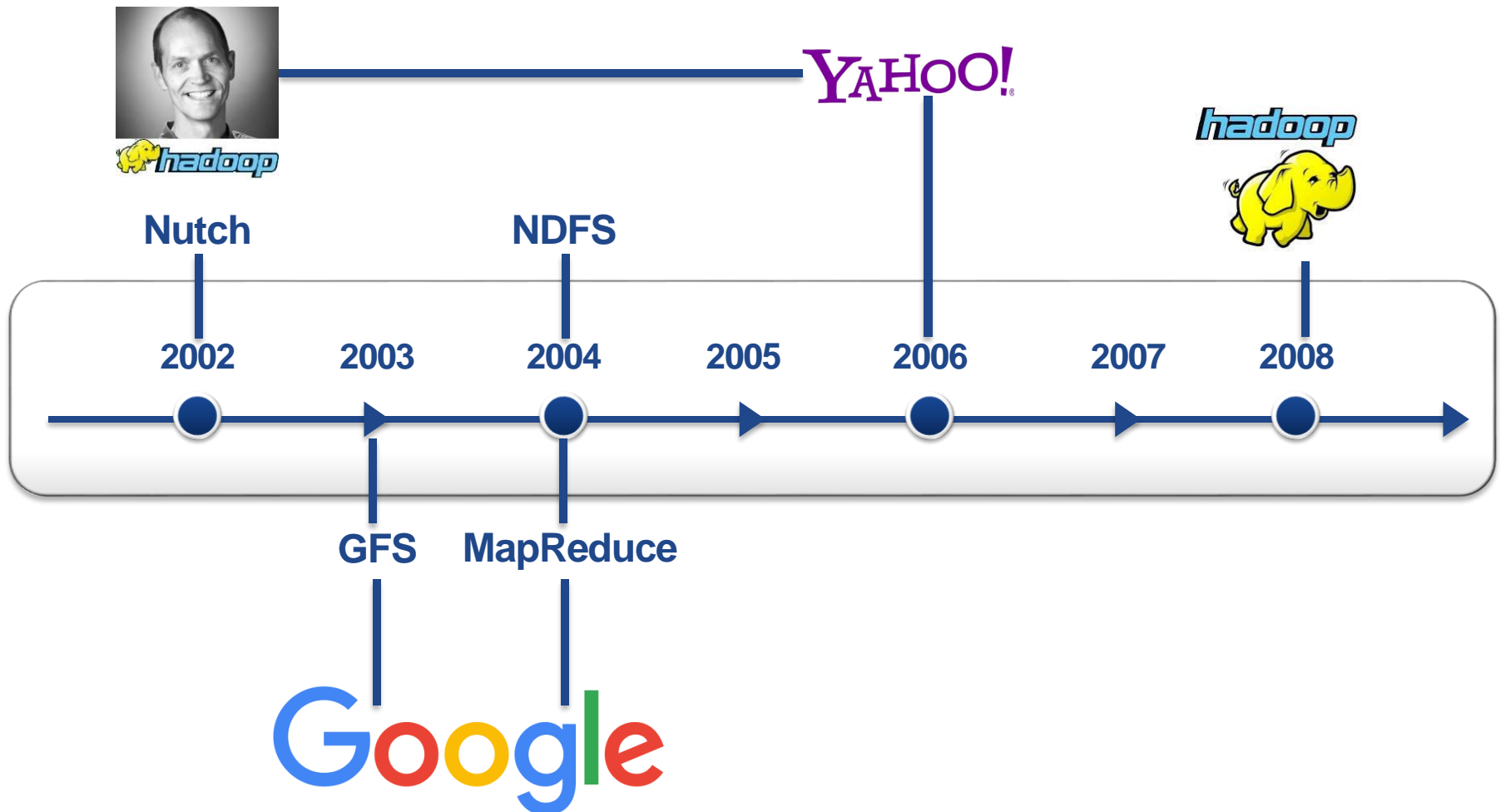
Hadoop Overview

Hadoop History : Hadoop Mode

동작모드	서버 대수	HDFS 프로세스	MapReduce 프로세스	이용목적
로컬모드	1대	HDFS 사용 안함 Name & Data node 프로세스 X	단일 자바 프로세스로 실행 Job & Task Tracker 프로세스 X	MapReduce 동작만 검증
유사 분산 모드	1대	Name & Data node 프로세스 한대의 서버 상에서 동작	Job & Task Tracker 프로세스 한대의 서버 상에서 동작	HDFS/MapReduce 동작검증 Hadoop Application 기능 검증
완전 분산 모드	다수	Name Node 프로세스는 마스터 Data Node 프로세스는 슬레이브 서버 상에서 동작	JobTracker 프로세스는 마스터 TaskTracker 프로세스는 슬레이브 서버 상에서 동작	상용환경 구축, 노드 간 통신을 포 함한 HDFS/MapReduce 기능 검증 애플리케이션 검증

Hadoop Overview

Hadoop History : Hadoop birth

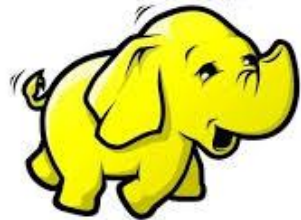


Hadoop Overview

Hadoop Architecture : Hadoop Platform

하둡
아키텍처

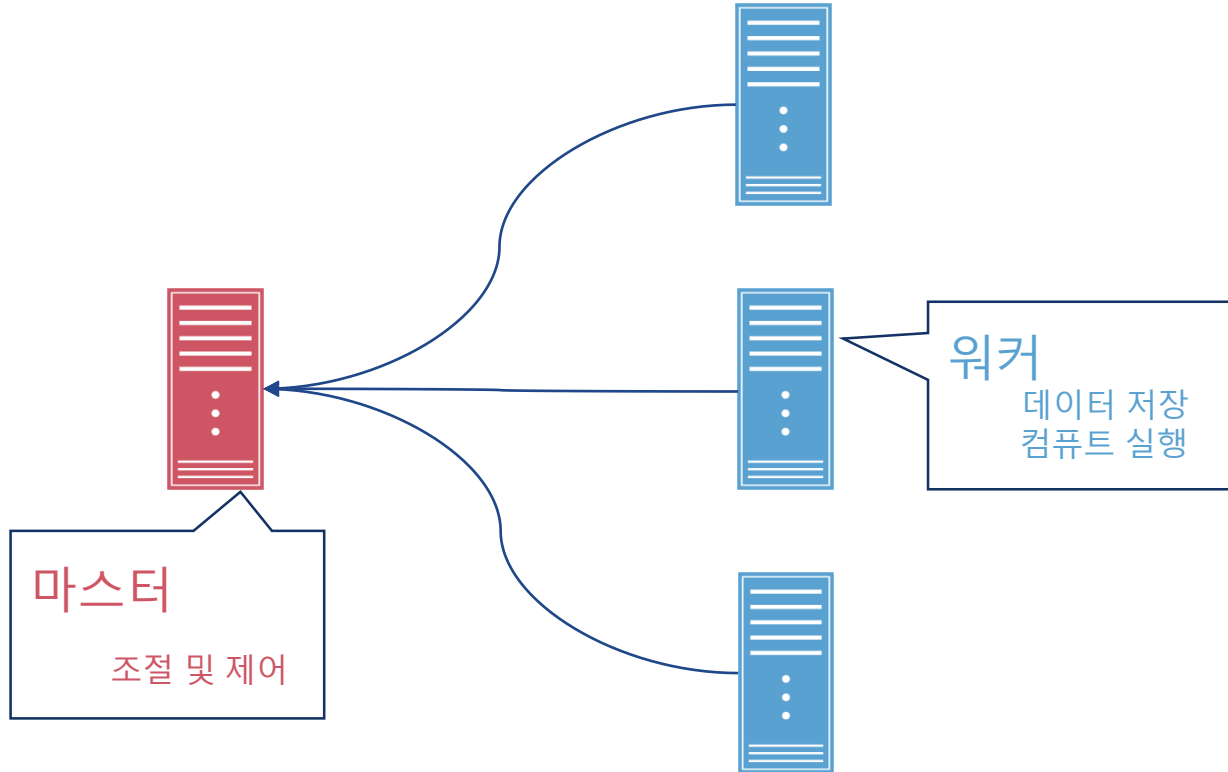
hadoop



1. 스토리지 플랫폼
 - Hadoop Distributed File System
 - HDFS
2. 컴퓨트 플랫폼
 - Yet Another Resource Negotiator
 - YARN

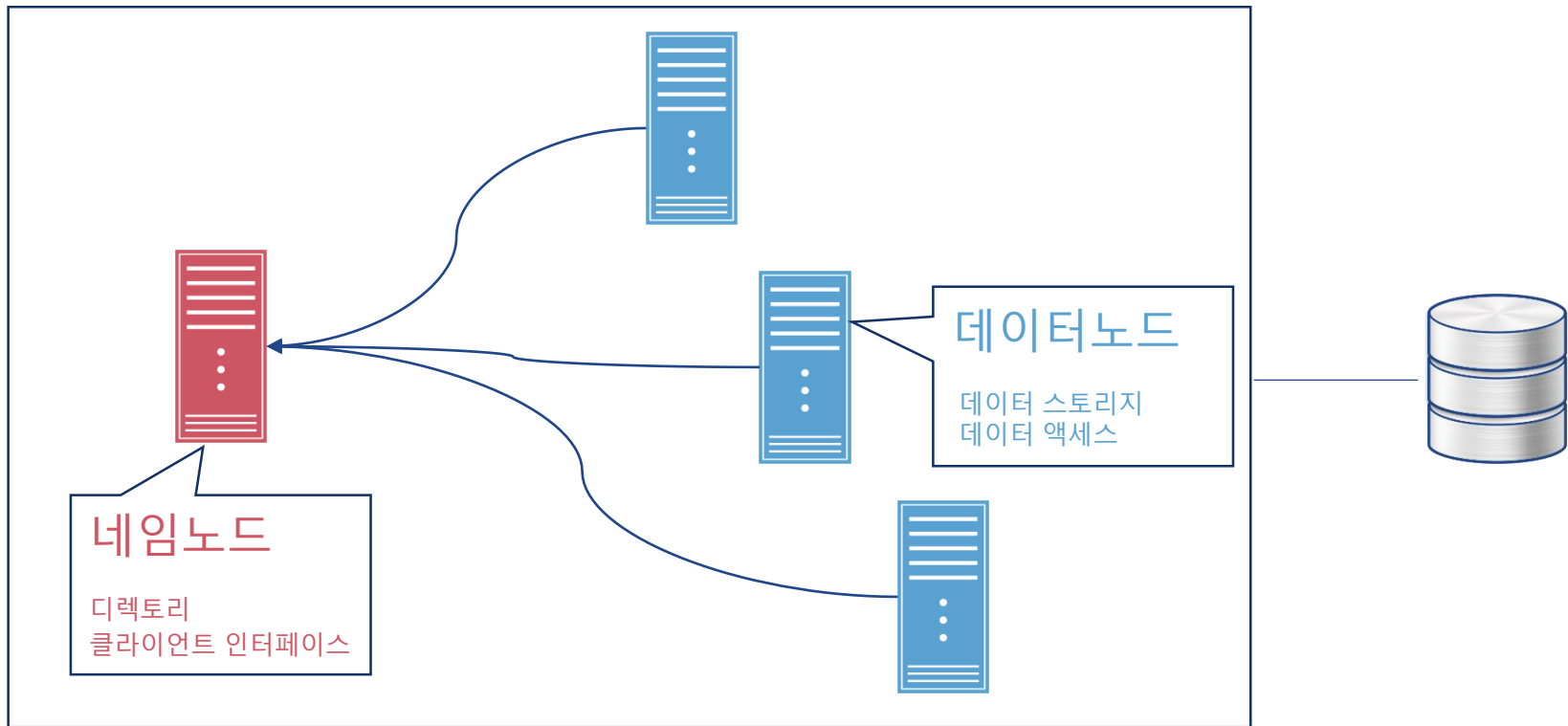
Hadoop Overview

Hadoop Architecture : Master & Worker



Hadoop Overview

Hadoop Architecture : HDFS Component



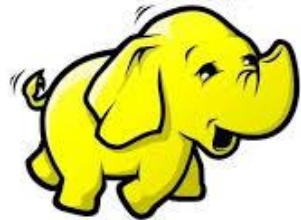
Hadoop Distributed File System

Hadoop Overview

Hadoop Design Principal

Hadoop Design Principal

hadoop



1. 데이터를 옮기는 것 보다
컴퓨팅을 옮기는 것이 더 저렴하다
2. 하드웨어는 고장 날 수 있고,
이것을 관리해야 한다.
3. 사용자에게 세부적인 실행을 숨겨야 한다.
4. 스트리밍을 사용하여 데이터에 접근한다.
5. 심플한 파일 시스템 일관성 모델을 사용한다.

Hadoop Overview

Hadoop Installation

Demo



1. Single Node Cluster Install
2. Multi Node Cluster Install

HDFS

HDFS Architecture

HDFS Demo

HDFS

HDFS Architecture : Functions of File System

파일시스템 기능

1. 데이터 저장 및 검색 제어
2. 파일과 폴더에 대한 메타데이터
3. 퍼미션 및 보안
4. 스토리지 공간의 효율적인 관리



HDFS

HDFS Architecture : Different File Systems



FAT32 - 4GB File 제한, 32GB 볼륨 제한
NTFS - 16EB 파일 제한, 16EB 볼륨 제한



HFS - 2GB File 제한, 2TB 볼륨 제한
HFS+ - 8EB 파일 제한, 8EB 볼륨 제한



ext3 - 2TB File 제한, 32TB 볼륨 제한
ext4 - 16TB 파일 제한, 1EB 볼륨 제한
XFS - 8EB 파일 제한, 8EB 볼륨 제한

HDFS

HDFS Architecture : Benefits of HDFS

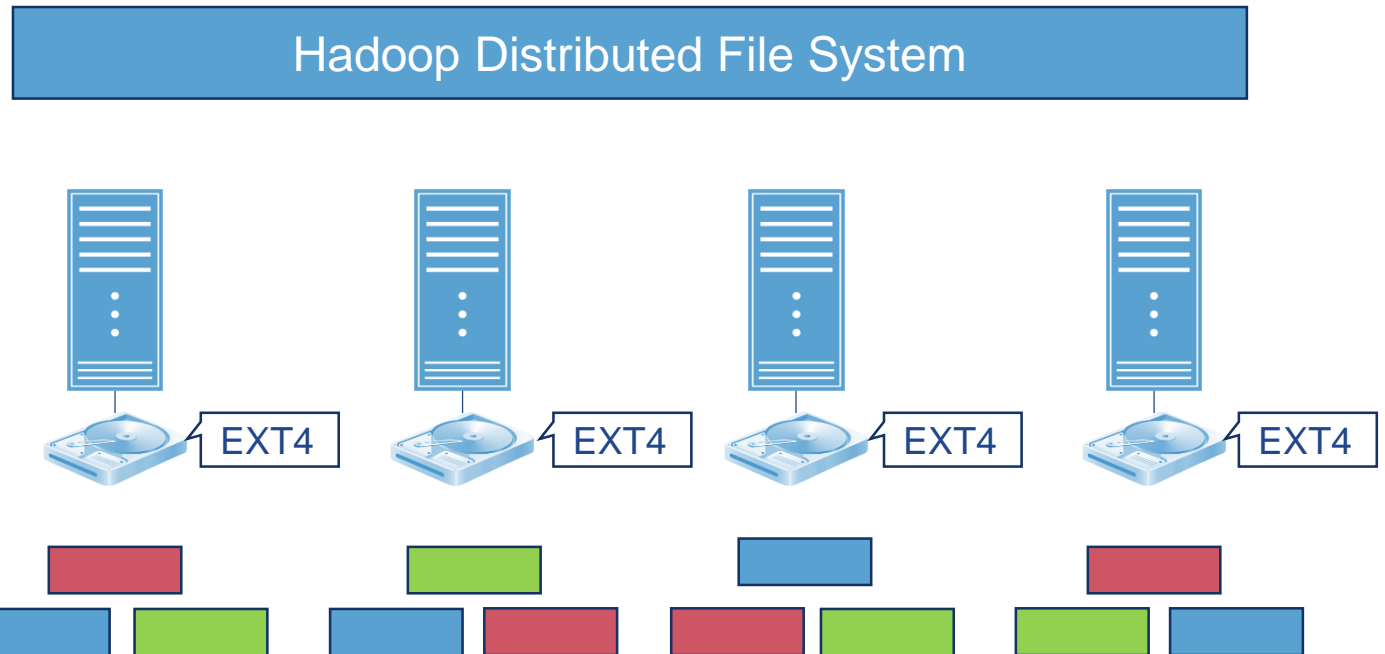
HDFS 2 장점

1. 분산 프로세싱 지원 (Blocks)
2. 실패 제어 (블록 복제)
3. 확장성 (확장 지원)
4. 비용효율 (일반적인 하드웨어)



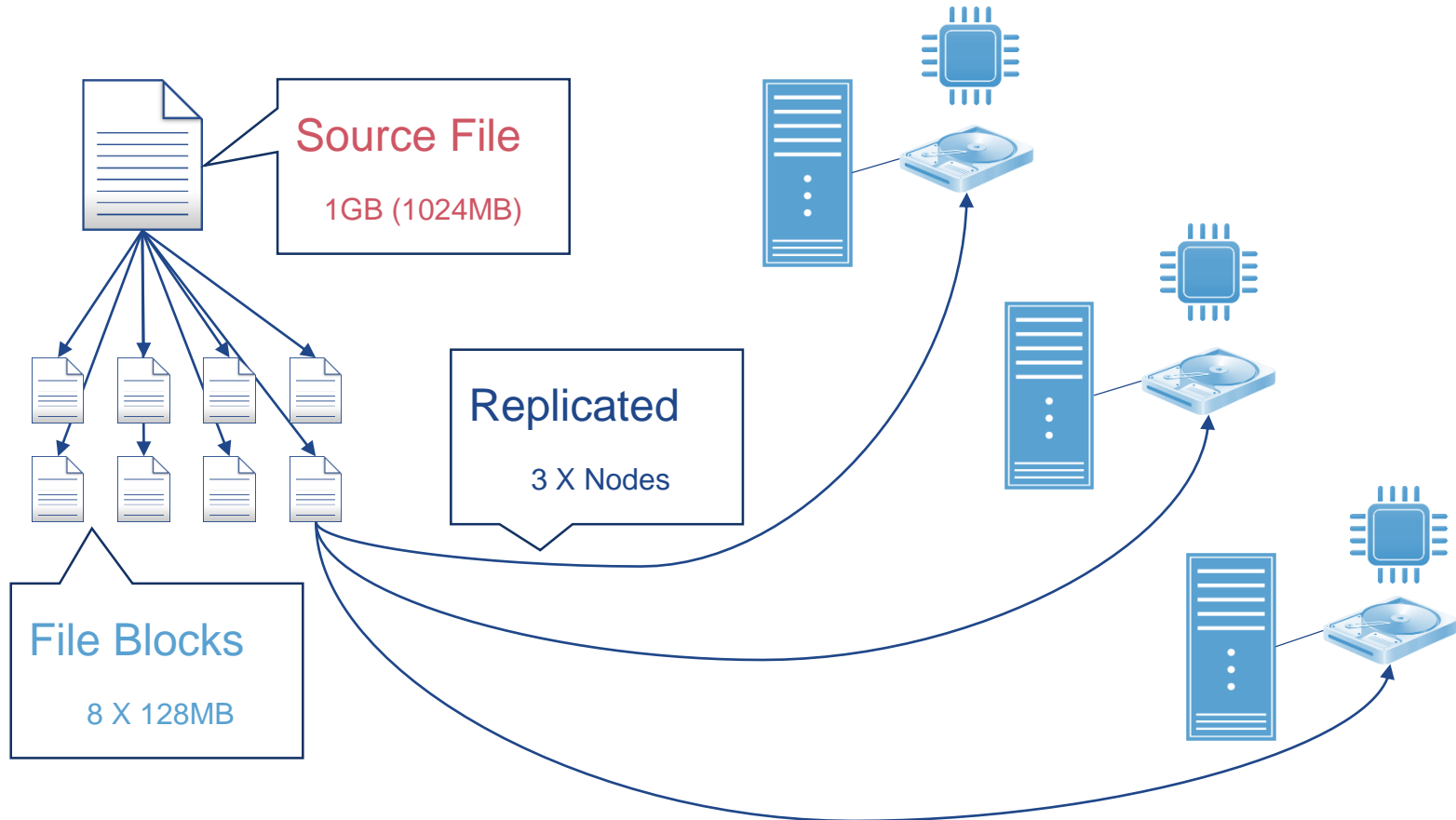
HDFS

HDFS Architecture : Local File System vs HDFS



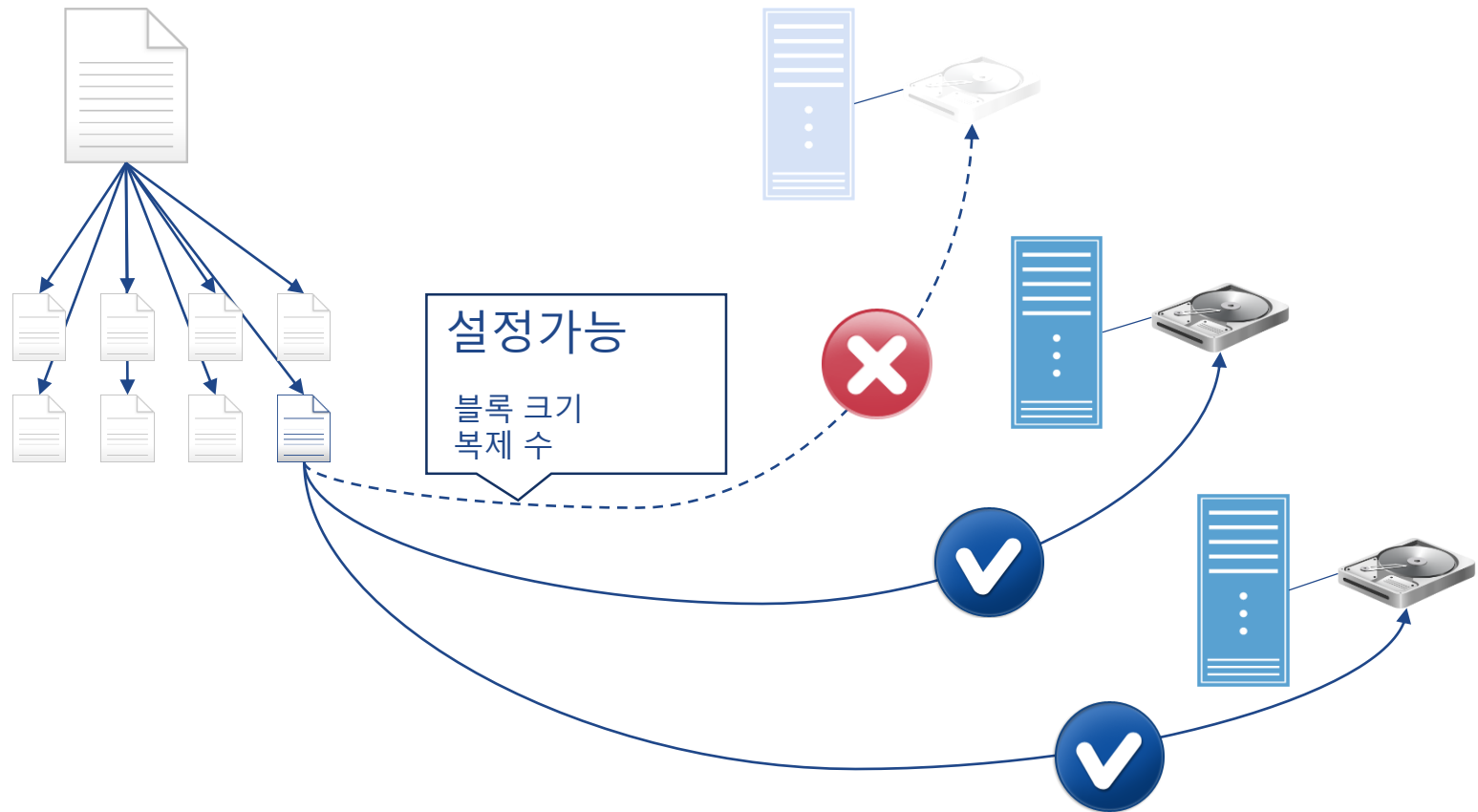
HDFS

HDFS Architecture



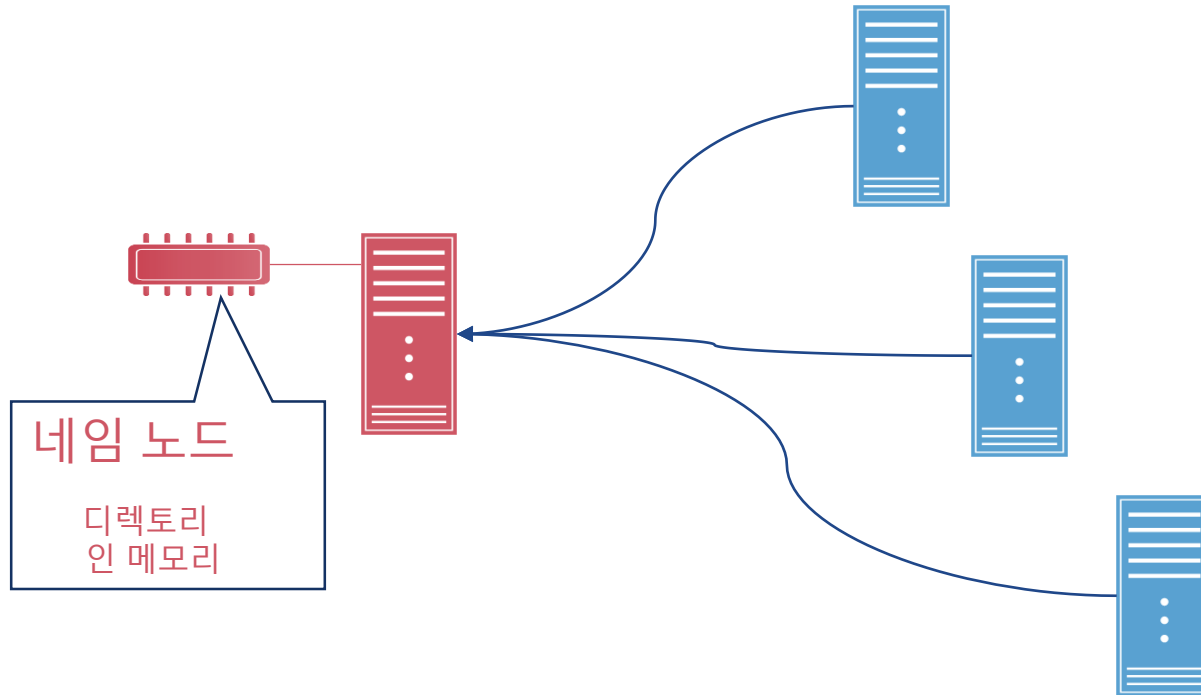
HDFS

HDFS Architecture



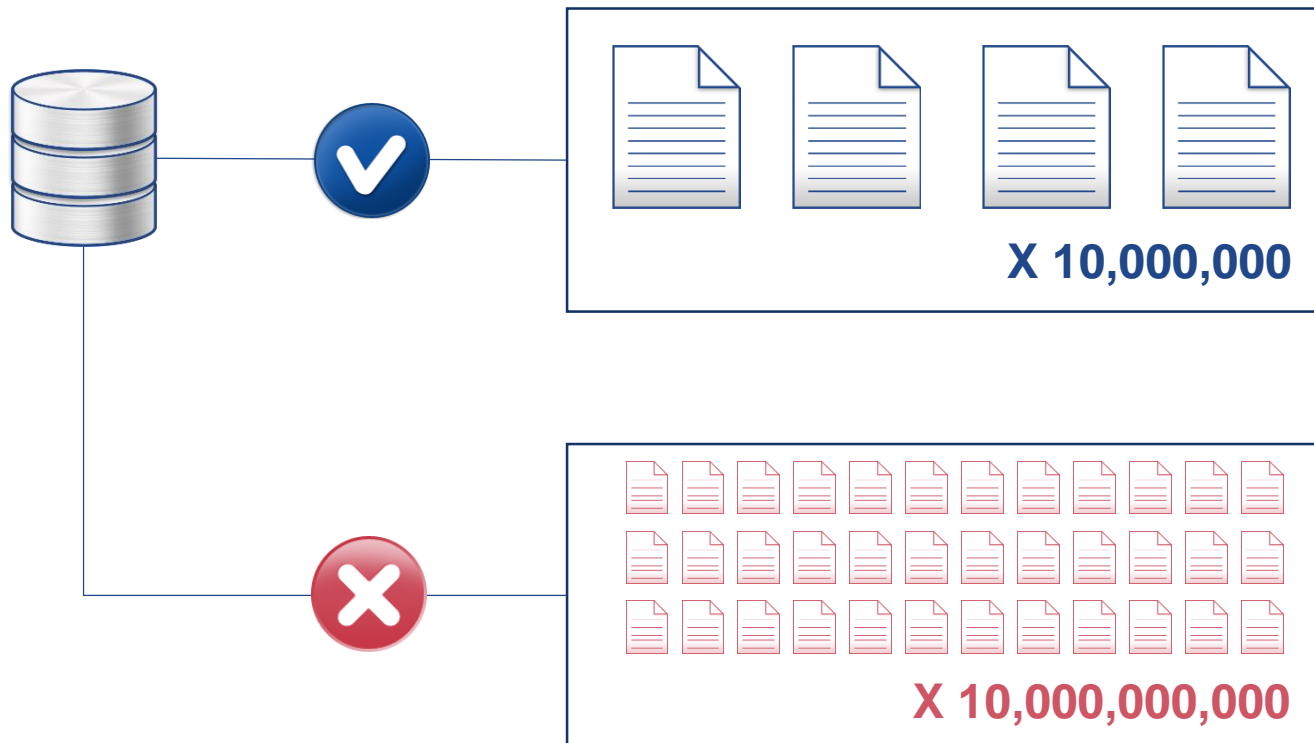
HDFS

HDFS Architecture



HDFS

HDFS Architecture



HDFS

HDFS Architecture : Benefits of HDFS

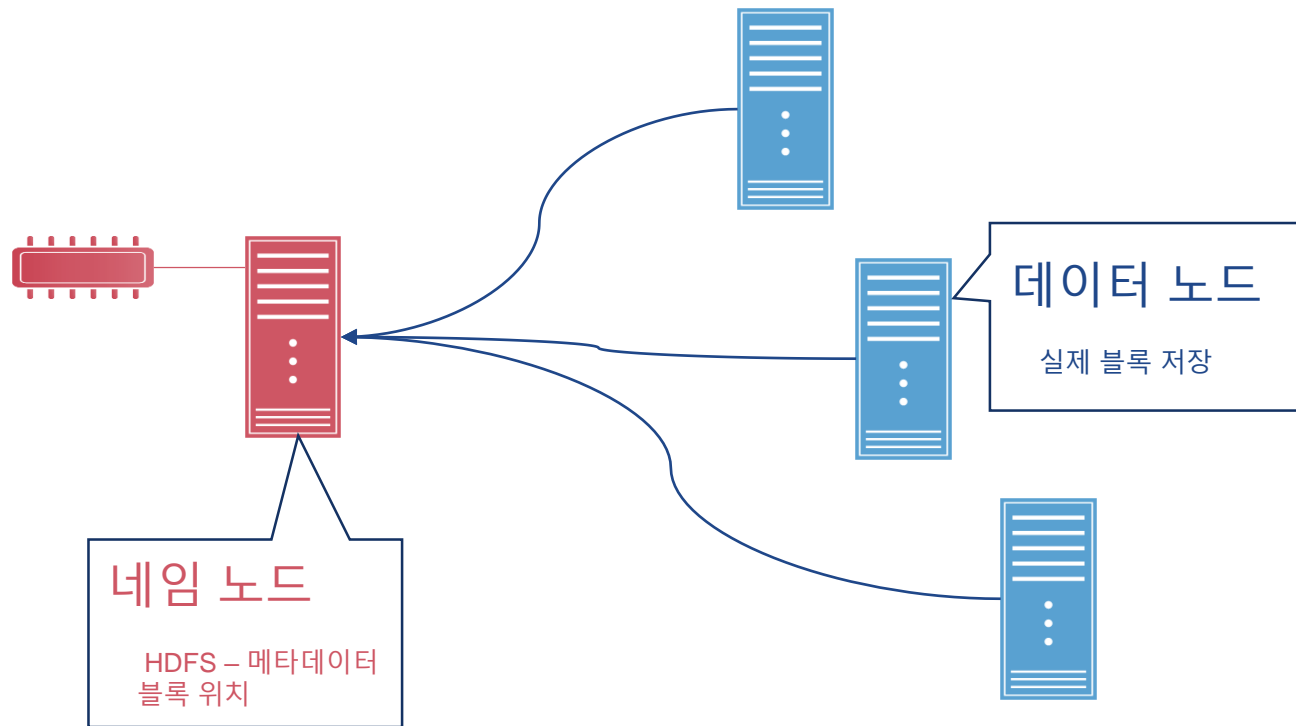
Hadoop 2
HDFS



1. Low Latency data processing
2. Automatic failover with Hot Standby
3. NFS Read/Write access to HDFS
4. Snapshots point in time recover
5. MapReduce Jobs are exclusive and Yarn HA
6. Heterogeneous Storage Media support (SSD, S3, SAN, NetApp Filers)
8. Coordinated caching of larger datasets in RAM for faster processing

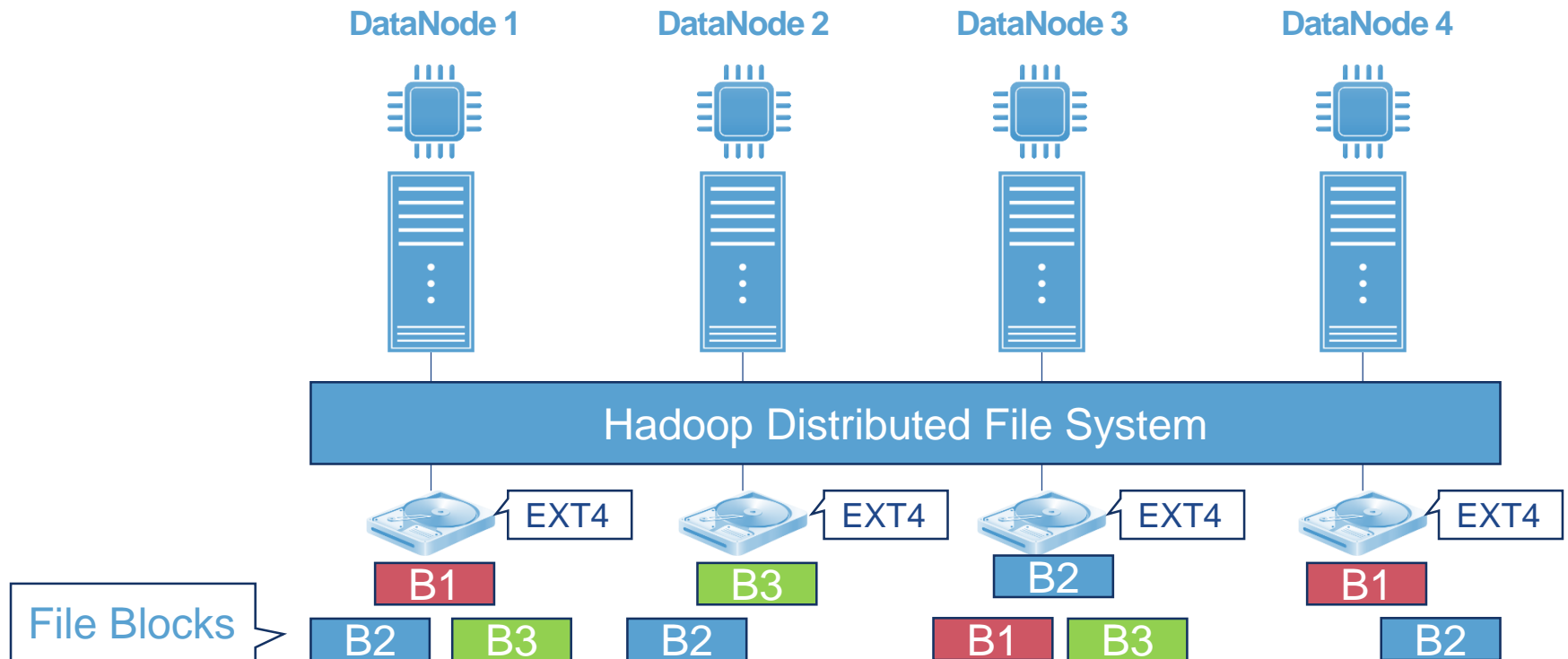
HDFS

HDFS Architecture



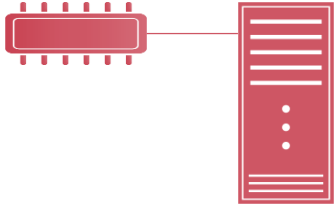
HDFS

HDFS Architecture : Datanode



HDFS

HDFS Architecture : NameNode



Name Node

File Name : MyFileinHDFS.log

File Size : 350MB

Replication Factor : 3

Blocks : BLK_0045732, BLK9610590,BLK_8851209

Block Locations

Permissions

Created By, Created On

Last Modified By, Last Modified On

HDFS

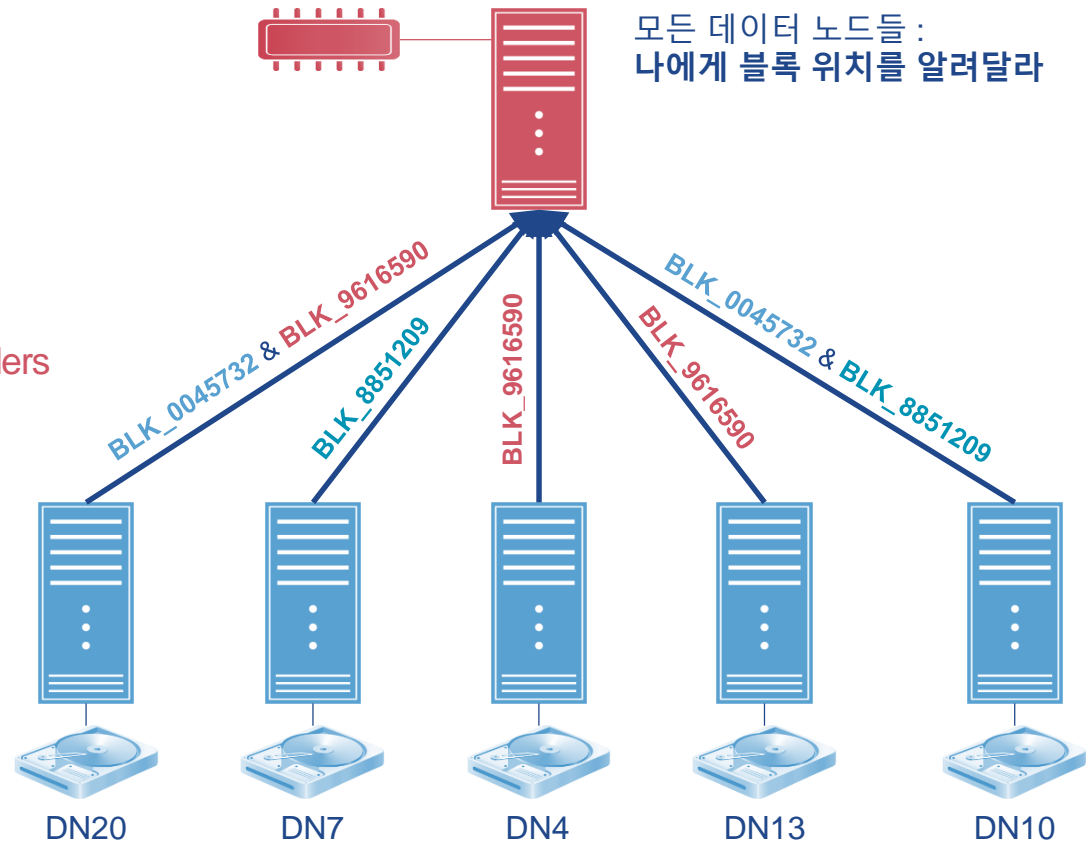
HDFS Architecture : NameNode

MyFileinHDFS.log

BLK_0045732	DN20	DN2	DN10
BLK_9616590	DN20	DN4	DN13
BLK_8851209	DN7	DN2	DN10

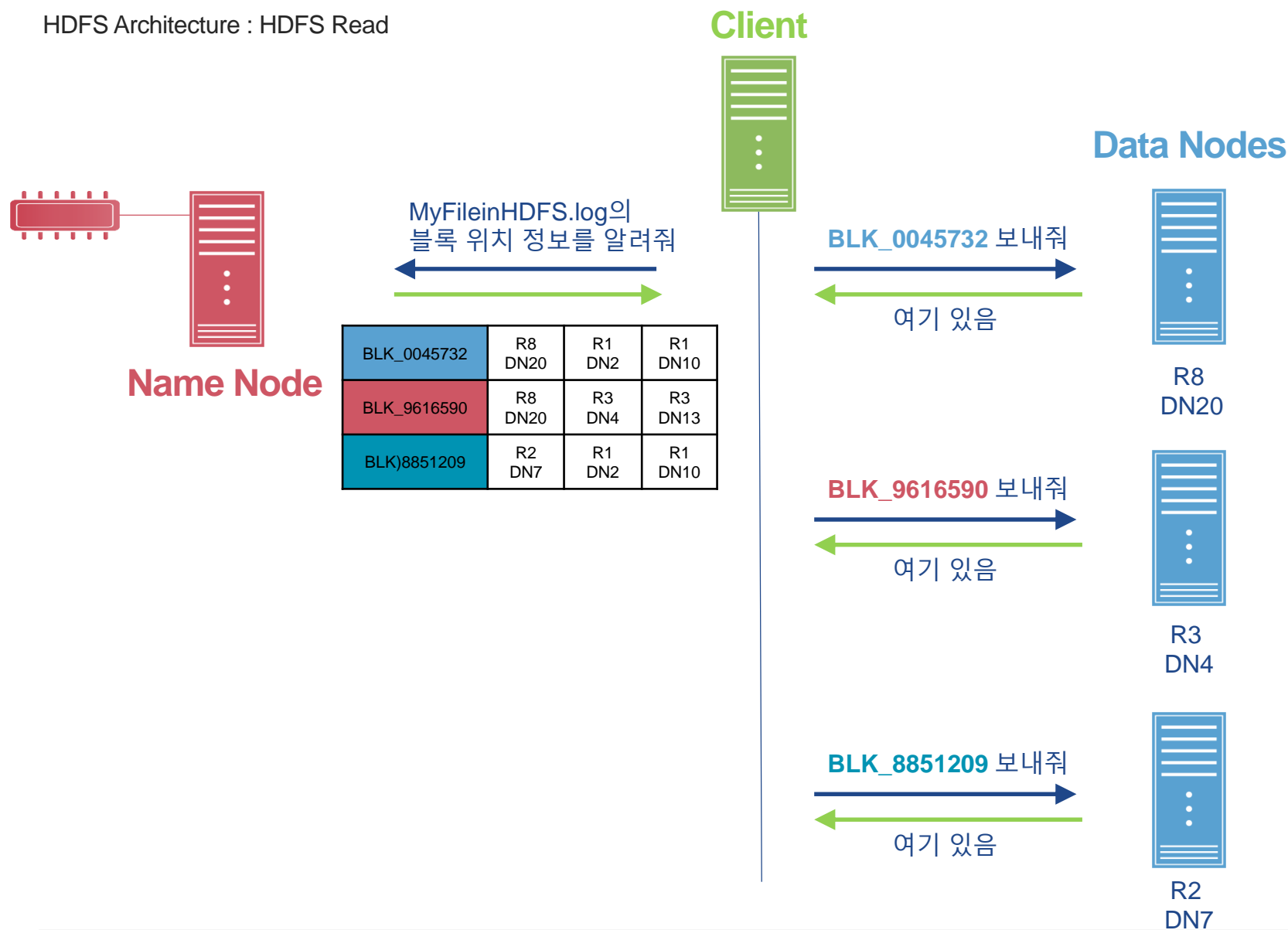
Name Node

In Disk – Metadata of Files & Folders
In Memory – Block locations



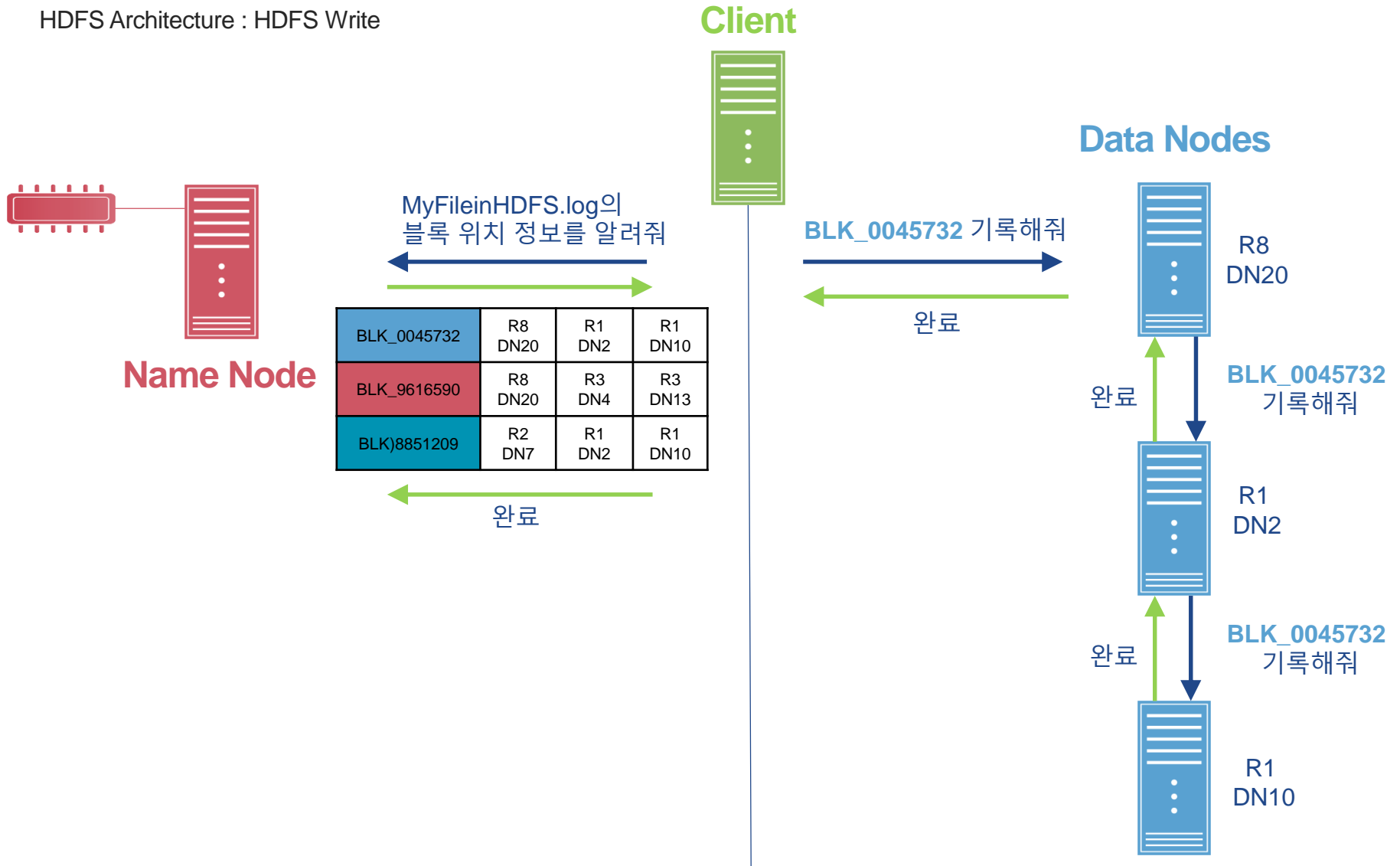
HDFS

HDFS Architecture : HDFS Read



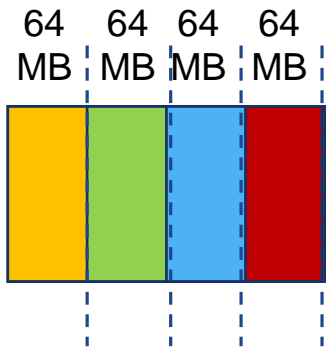
HDFS

HDFS Architecture : HDFS Write

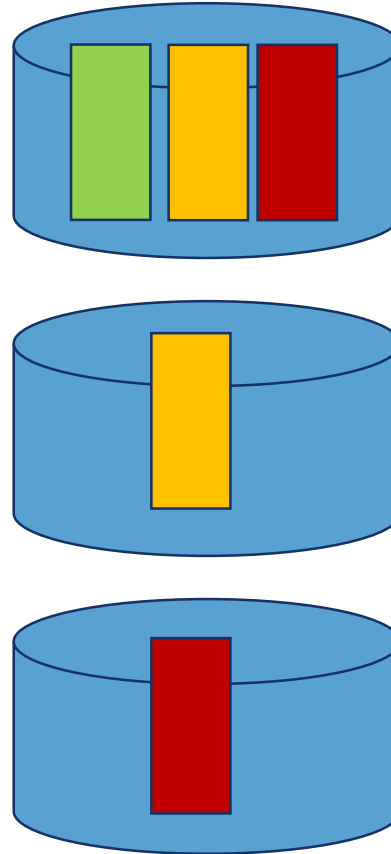


HDFS

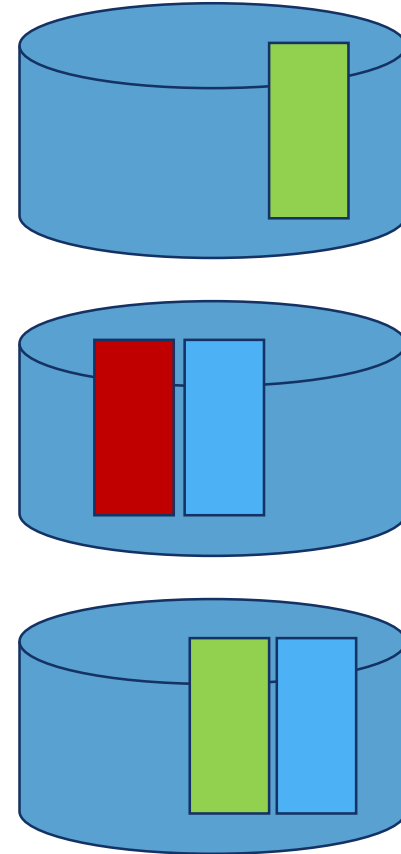
HDFS Architecture : Blocks



서버 랙 A

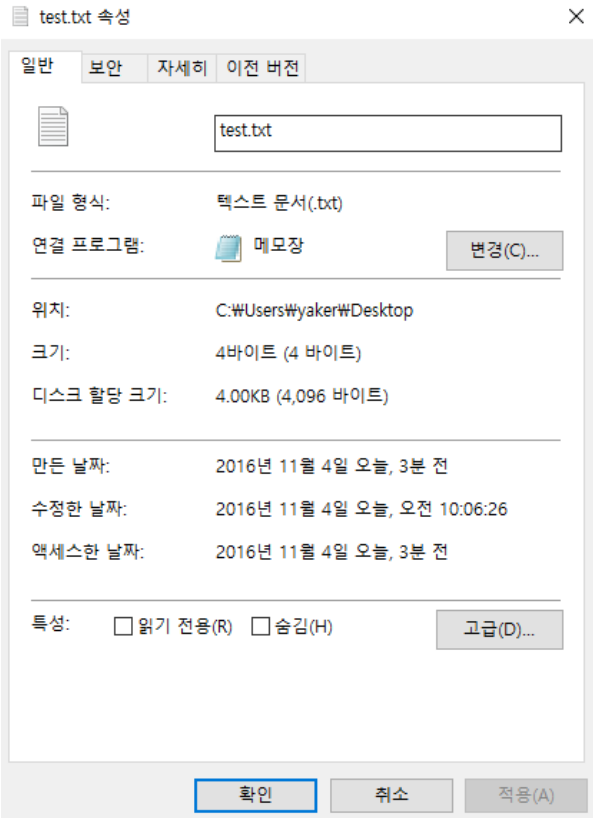


서버 랙 B



HDFS

HDFS Architecture : Blocks



NTFS

4KB Block Size	
File Size	Unused Space
2KB	2KB
8KB	0KB
13KB	3KB

HDFS

HDFS Architecture : Blocks

HDFS

256MB Block Size	
File Size	Unused Space
1MB	?

HDFS

HDFS Design Principal

HDFS Design Principal



1. HDFS is designed to :

- 큰 파일을 다루기 위해 (MB, GB, TB 그 이상)
- 데이터 액세스 (한번 쓰고 여러 번 읽음)
- 일반적인 하드웨어 (300만원~500만원)

2. HDFS is **not** designed to :

- 짧은 지연시간을 가진 데이터 액세스(OLTP)
- 작은 파일 (네임노드 메모리 증가)
- 임의적인 파일 수정 (추가만 지원)

HDFS

HDFS Design Principal

HDFS Design Principal



1. HDFS is designed to :

- 큰 파일을 다루기 위해 (MB, GB, TB 그 이상)
- 데이터 액세스 (한번 쓰고 여러 번 읽음)
- 일반적인 하드웨어 (300만원~500만원)

2. HDFS is **not** designed to :

- 짧은 지연시간을 가진 데이터 액세스(OLTP)
- 작은 파일 (네임노드 메모리 증가)
- 임의적인 파일 수정 (추가만 지원)

Summary

Points to remember

기억해야 할 것!!



1. HDFS Design Principal
2. Component of HDFS Architecture
3. File reads and writes in a cluster
4. Secondary Name Node & Standby Name Node
5. HDFS Block Replication

HDFS

HDFS Demo

Demo



1. Working with HDFS

- The Hadoop command
- Storing and reading files
- Listing file blocks

Map Reduce

MapReduce overview

MapReduce Architecture

MapReduce Demo

이들이 가지는 공통적인 **문제점**은 무엇일까?

A dark blue square containing the text "Facebook Social Network".

Facebook
Social
Network

A light green square containing the text "Google Search".

Google
Search

A teal square containing the text "LinkedIn Member Analytics".

LinkedIn
Member
Analytics

MapReduce

MapReduce overview

Facebook
Social
Network

사용자 로그

매달 10억 개 이상

각 사용자에게 공통 친구들 수를
ms단위로 보여줌

함께 아는 친구 10명

MapReduce

MapReduce overview



Google
Search

400억개 이상의 웹페이지

1초 내로 검색 결과를 돌려줌

3,730,000 검색 결과 (0.34초)

MapReduce

MapReduce overview

LinkedIn
Member
Analytics

4억 5천 정도의 멤버

복잡한 측정치를
사용자 요구에 맞게 보여줌

최근 90일동 프로필 조회자(명)

MapReduce

MapReduce overview

대량의 데이터 셋



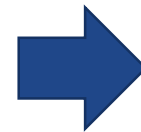
한 개의 명확한 인사이트

함께 아는 친구 10명

4 최근 90일동 프로필 조회자(명)

원시 데이터의 수십억 개 Row

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha



4 최근 90일동안
프로필 조회자(명)

원시 데이터의 수십억 개 Row

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha

요약 및
정렬



From Member	Profile Views
Janani	50
Swetha	15
Vitthal	22
Shreya	23
Jitu	32
Pradeep	10

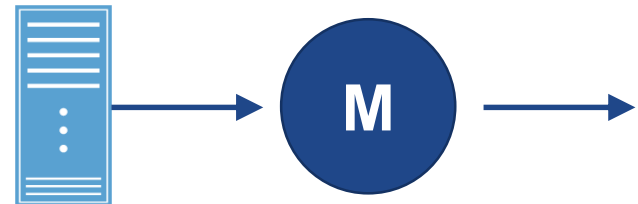
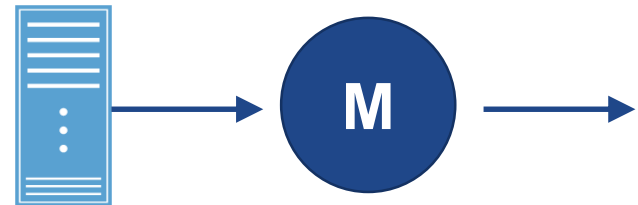
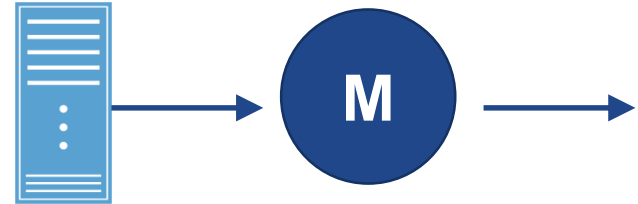
MapReduce

MapReduce Map Flow

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani

View ID	From Member	To Member
3	Shreya	Pradeep
4	Jitu	Vitthal

View ID	From Member	To Member
5	Shreya	Janani
6	Jitu	Swetha

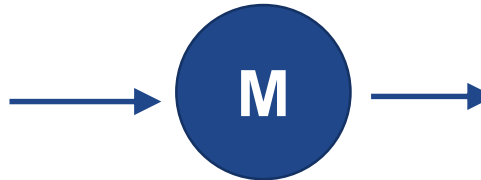


해당 작업은 각 Mapper에서 독립적으로 발생됨

MapReduce

MapReduce overview : Map Flow

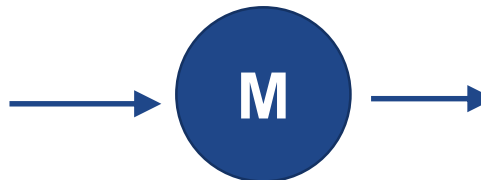
View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani



From Member	Profile Views
-------------	---------------

{ Jitu, 1 }
{ Janani, 1 }

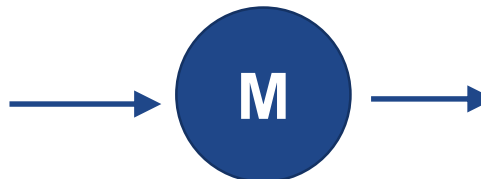
View ID	From Member	To Member
3	Shreya	Pradeep
4	Jitu	Vitthal



From Member	Profile Views
-------------	---------------

{ Pradeep, 1 }
{ Vitthal, 1 }

View ID	From Member	To Member
5	Shreya	Janani
6	Jitu	Swetha

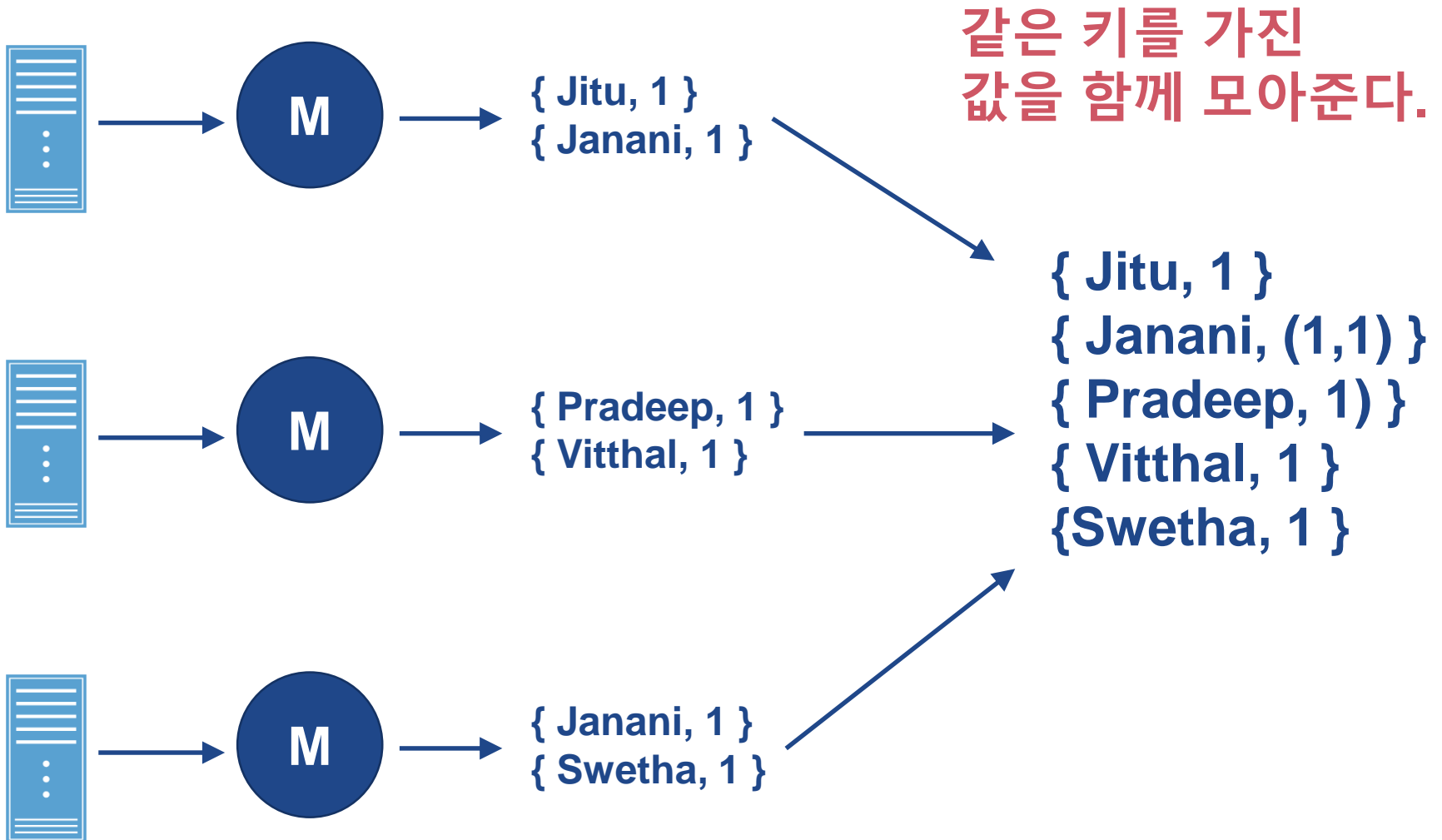


From Member	Profile Views
-------------	---------------

{ Janani, 1 }
{ Swetha, 1 }

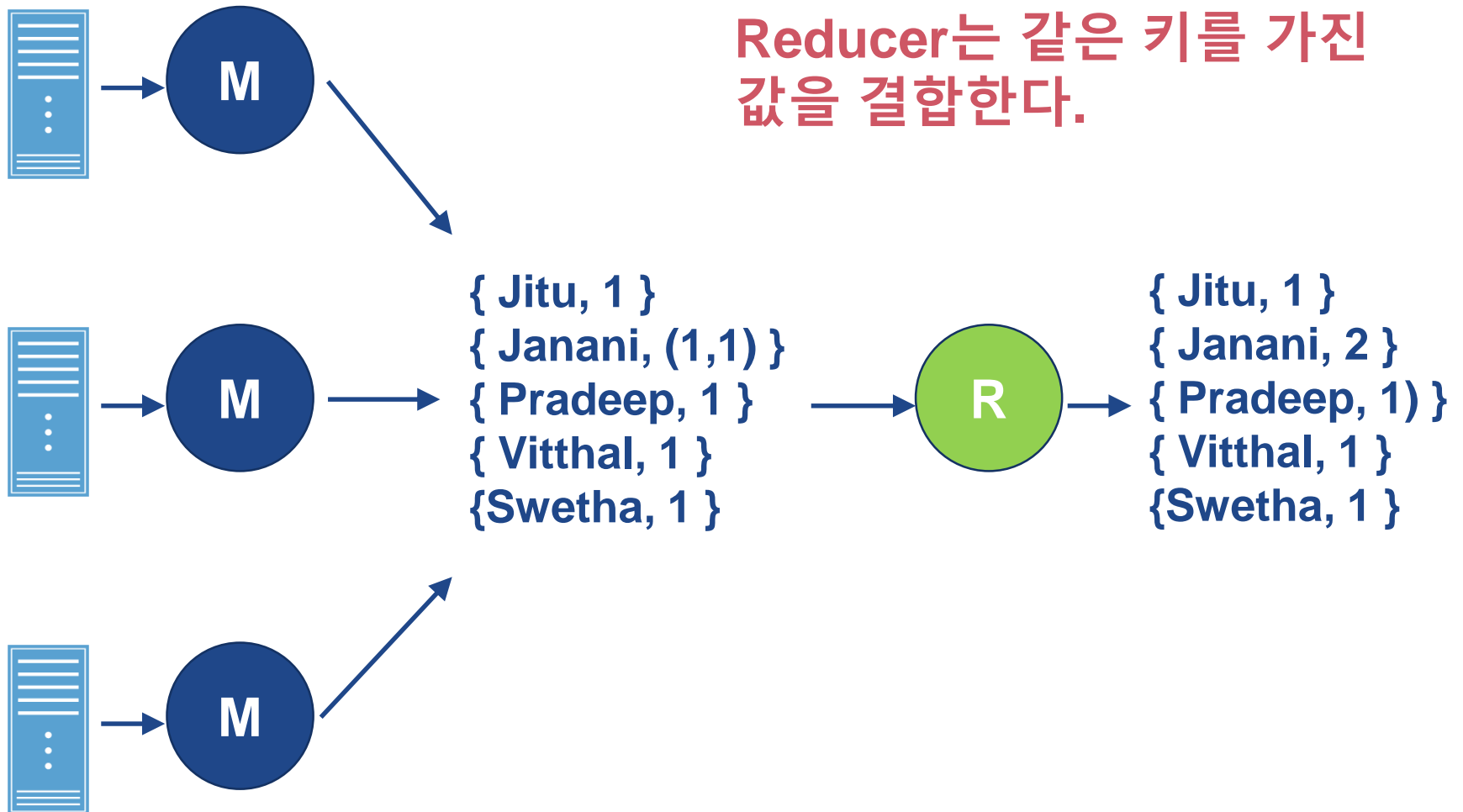
MapReduce

MapReduce overview : Reduce Flow



MapReduce

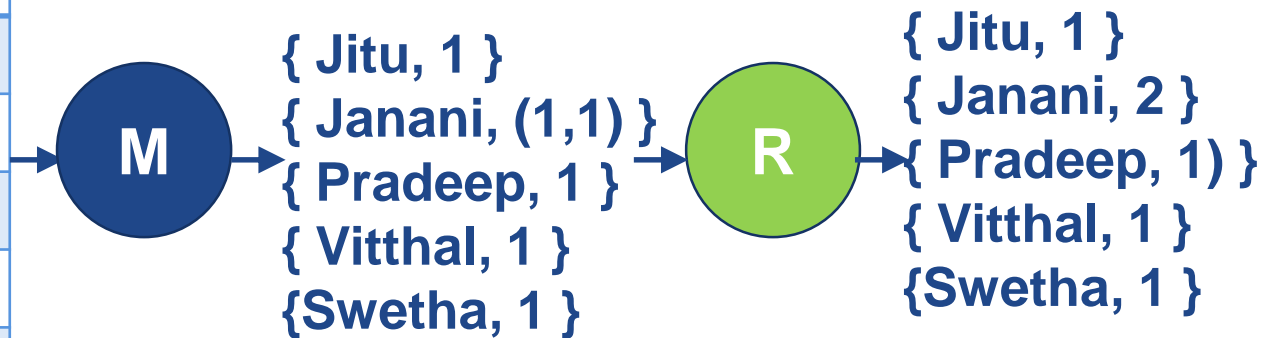
MapReduce overview : Reduce Flow



MapReduce

MapReduce overview : MapReduce Basic Flow

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha



모든 MapReduce 작업의 기본 폼

MapReduce

MapReduce overview

MapReduce 잡 구현



Map

Map 로직이 있는
클래스 구현

Reduce

Reduce 로직이 있는
클래스 구현

Main

잡을 셋업하기 위한
Driver Program

MapReduce

MapReduce overview : Map Step

Map Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

Map 로직은 Mapper 클래스를
상속받은 클래스 내에 구현

4가지 타입의 파라미터를 가짐

MapReduce

MapReduce overview : Map Step

**<input key type,
input value type>**

<K,V>



Map

Input Key, Value 쌍은

디스크 내에 있는

Input File을 어떻게 읽어 들일지

결정하는 것이다.

MapReduce

MapReduce overview : Map Step

**<input key type,
input value type>**

<K,V>



Map

Input Text File

**first line in a file
second line in a file
third in a file**

Input Key, Value Pairs

**<1, first line in a file>
<2, second line in a file>
<3, third in a file>**

MapReduce

MapReduce overview : Map Step

**<output key type,
output value type>**

<K,V>



Map

Output Key, Value 쌍은

Map단계에서

내가 수행하는 처리에 달려있다.

MapReduce

MapReduce overview : Map Step

**<output key type,
output value type>**

<K,V>



Map

Input Key, Value Pairs

<1, first line in a file>
<2, second line in a file>
<3, third in a file>

Output Key, Value Pairs

<1, first>
<1, line>
<1, in>
<1, a>
<1, file>

MapReduce

MapReduce overview : Reduce Step

Reduce Class

<input key type,
input value type,
output key type,
output value type>

Reducer Class

**Reduce 로직은 Reducer 클래스를
상속받은 클래스 내에 구현**

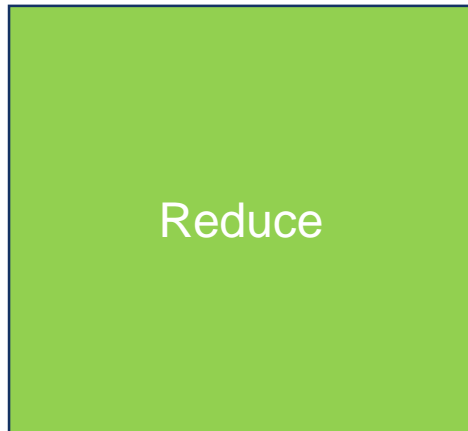
4가지 타입의 파라미터를 가짐

MapReduce

MapReduce overview : Reduce Step

**<input key type,
input value type>**

<K,V>



**Input Key, Value 쌍은
Map 단계의 output과
동일한 Key/Value 쌍이어야 한다.**

MapReduce

MapReduce overview : Reduce Step

**<input key type,
input value type>**

<K,V>



Reduce

**Map Output Key, Value Pairs
= Reduce Input Key, Value Pairs**

<1, first>

<1, line>

<1, in>

<1, a>

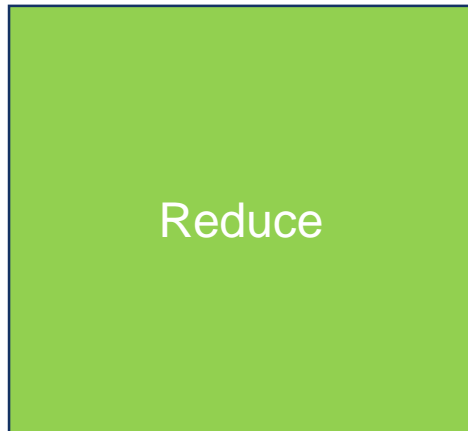
<1, file>

MapReduce

MapReduce overview : Reduce Step

**<output key type,
output value type>**

<K,V>



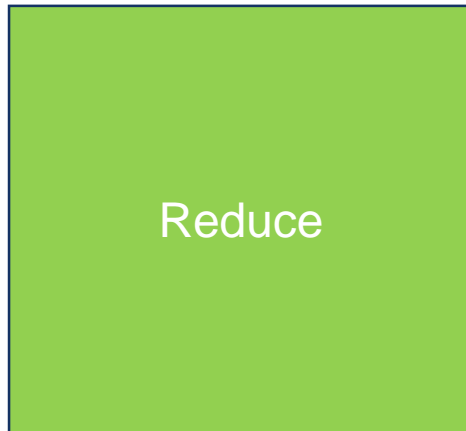
Output Key, Value 쌍은
Reduce 단계에서
내가 수행하는 처리에 달려있다.

MapReduce

MapReduce overview : Reduce Step

**<output key type,
output value type>**

<K,V>

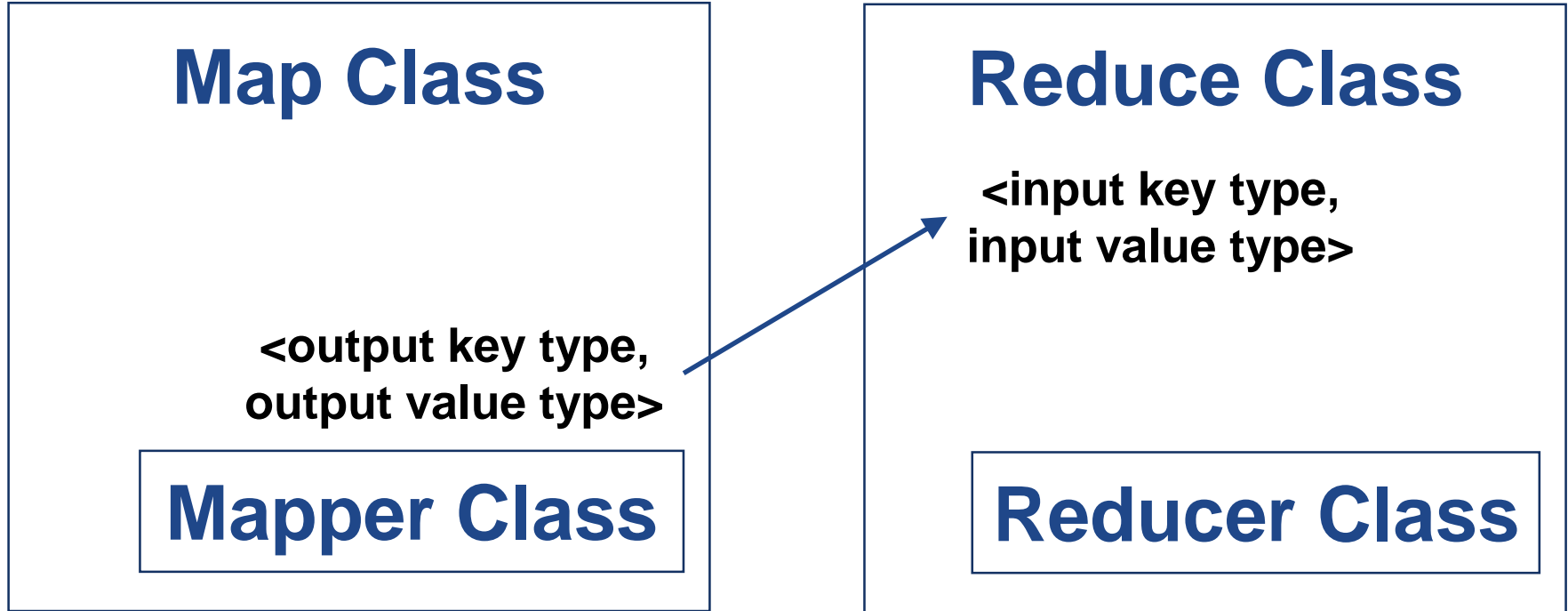


Reduce Output Key, Value Pairs

<first, 1>
<Second, 1>
<Third, 1>
<line, 3>
<in, 3>
<a, 3>
<file, 3>

MapReduce

MapReduce overview : Data Type Matching



Mapper의 Output Type은 Reduce의 Input Type과 일치해야 한다.

MapReduce

MapReduce overview : Driver Program

Main Class

Job Object

Input File Path
Output File Path
Mapper Class
Reducer Class
Output Data Types

Mapper 클래스와 Reducer 클래스는

Main Class 내에 설정된

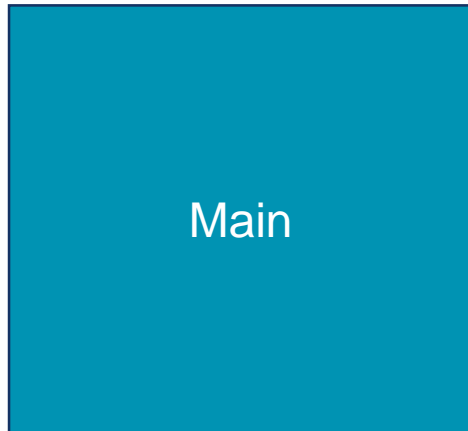
Job에 의해서 사용되어진다.

Job은 설정이 필요한

속성들의 집합이다.

MapReduce

MapReduce overview : Driver Program



Input File Path
MapReduce에 공급될
Input 데이터의 위치

Output File Path
결과 파일이 쓰여질
아직 존재하지 않는 디렉터리

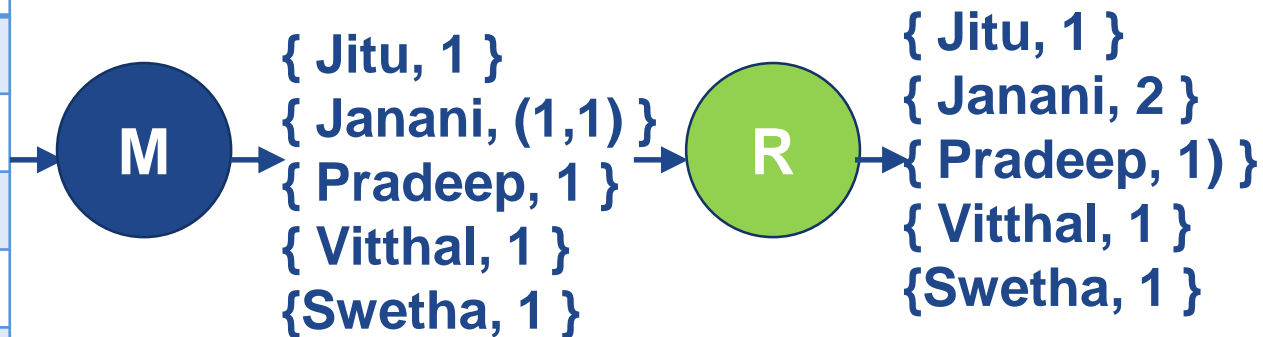
Mapper & Reducer 클래스
구현된 자바 클래스 코드

Output Data Types
최종 결과의 데이터 타입

MapReduce

MapReduce overview : MapReduce Basic Flow

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha

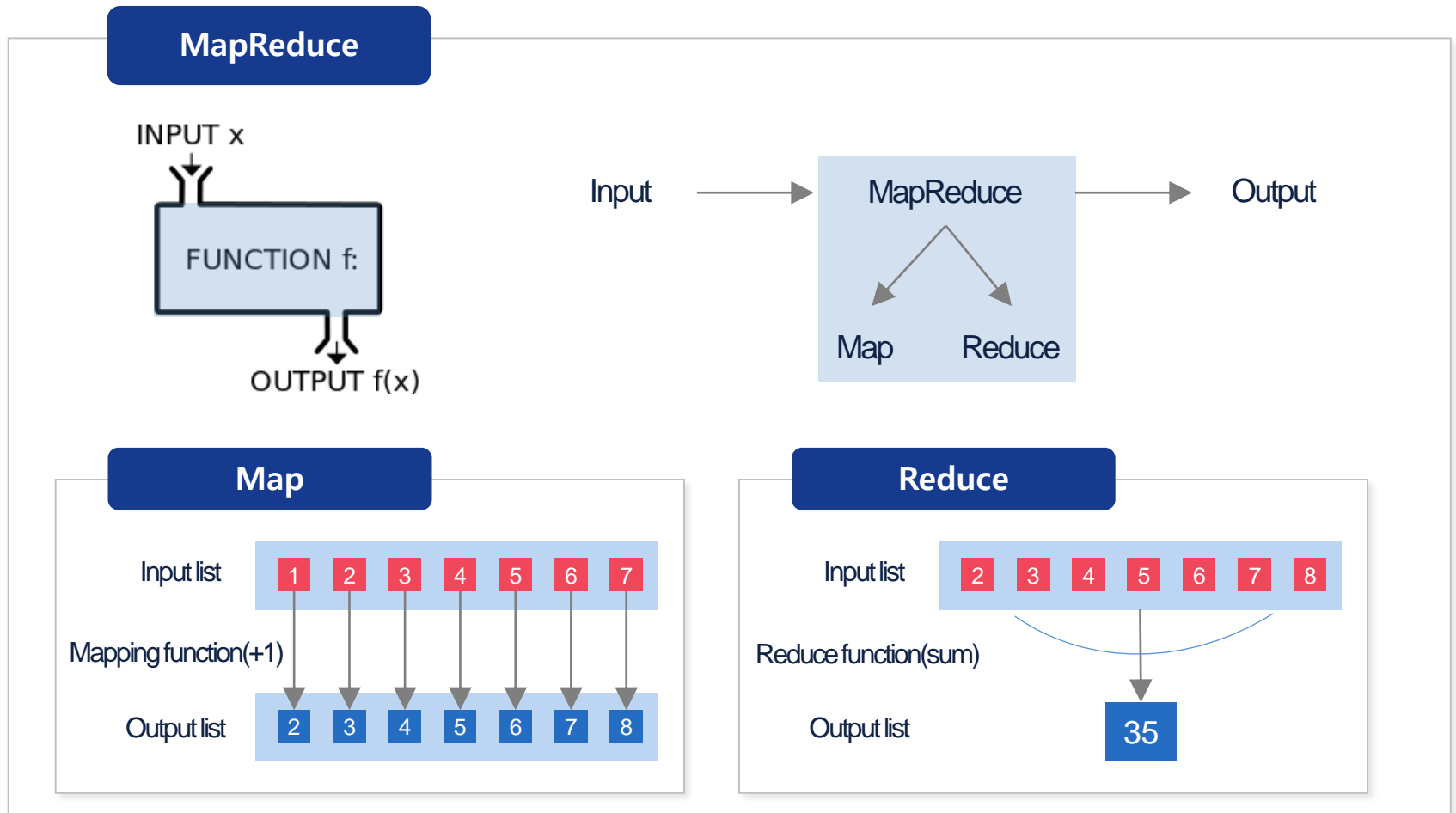


모든 MapReduce 작업의 기본 폼

MapReduce

MapReduce Architecture

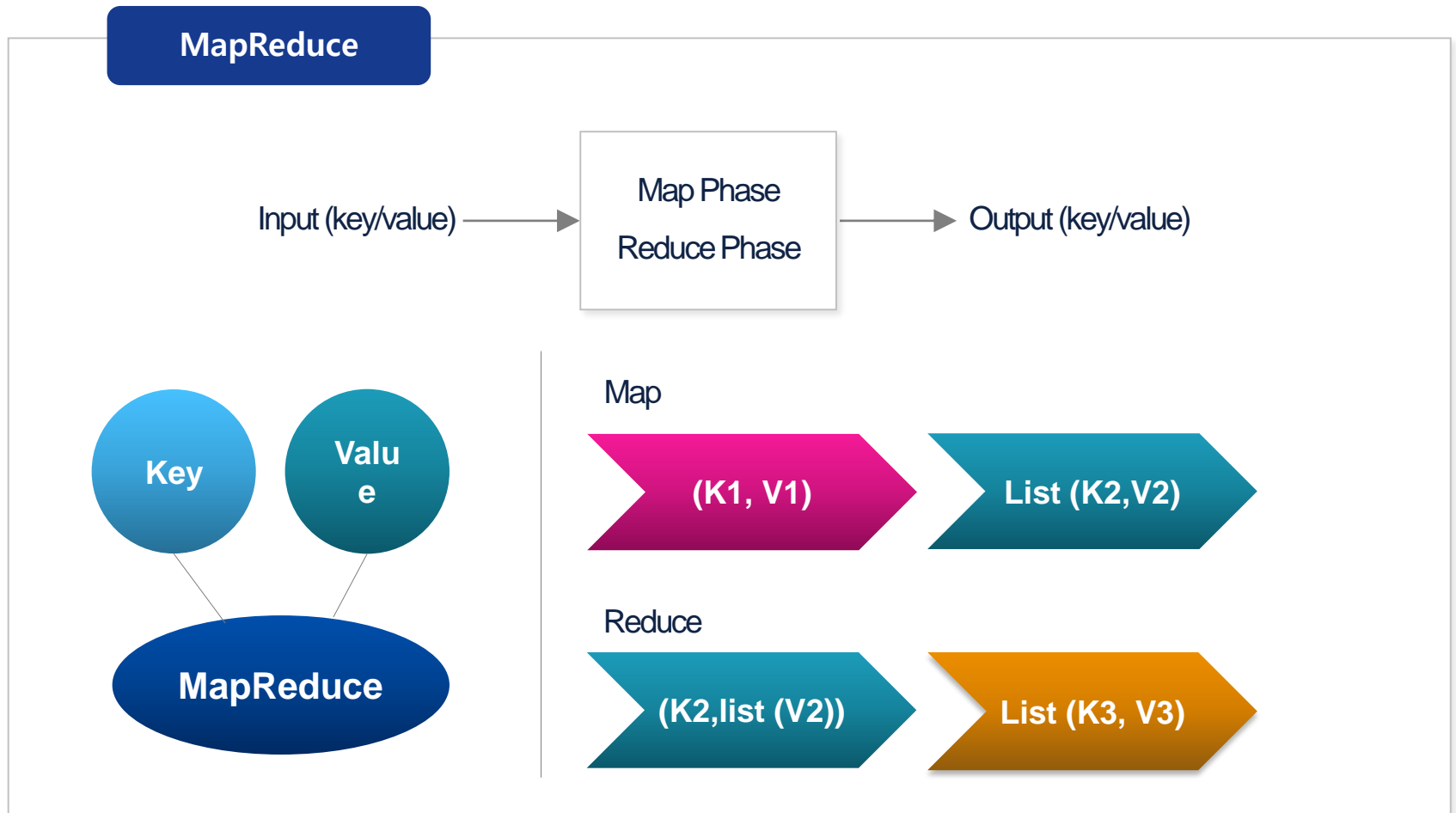
- MapReduce = Functional Programming Paradigm



MapReduce

MapReduce Architecture

- MapReduce Works

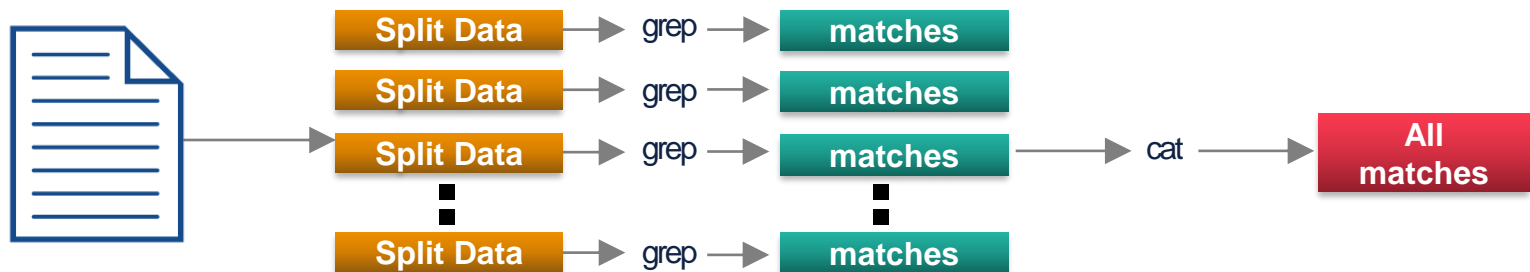


MapReduce

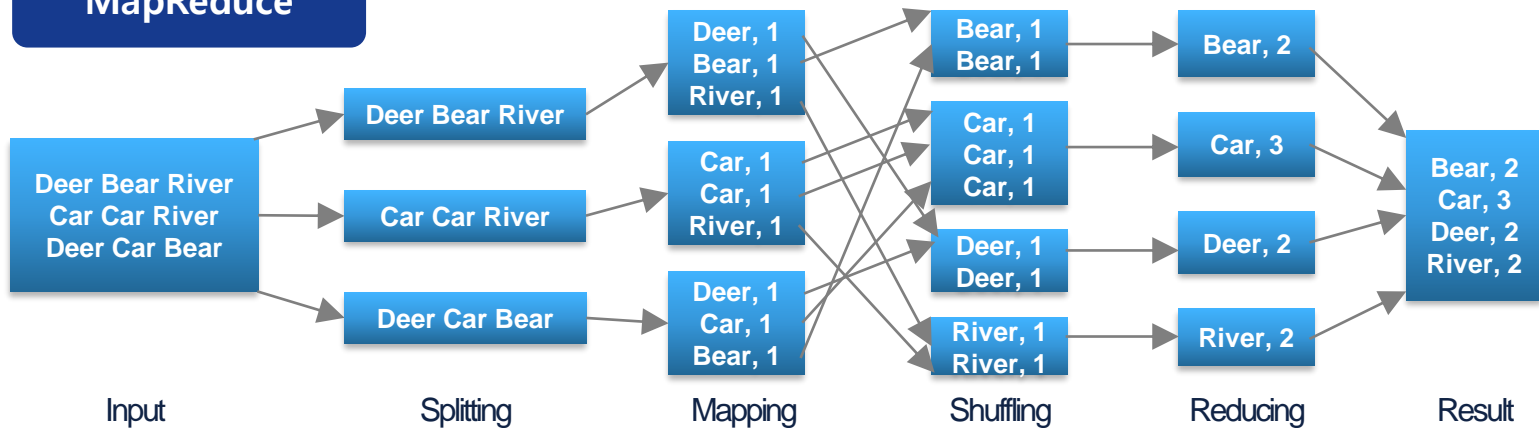
MapReduce Architecture

- Traditional way **vs** MapReduce

Traditional Way



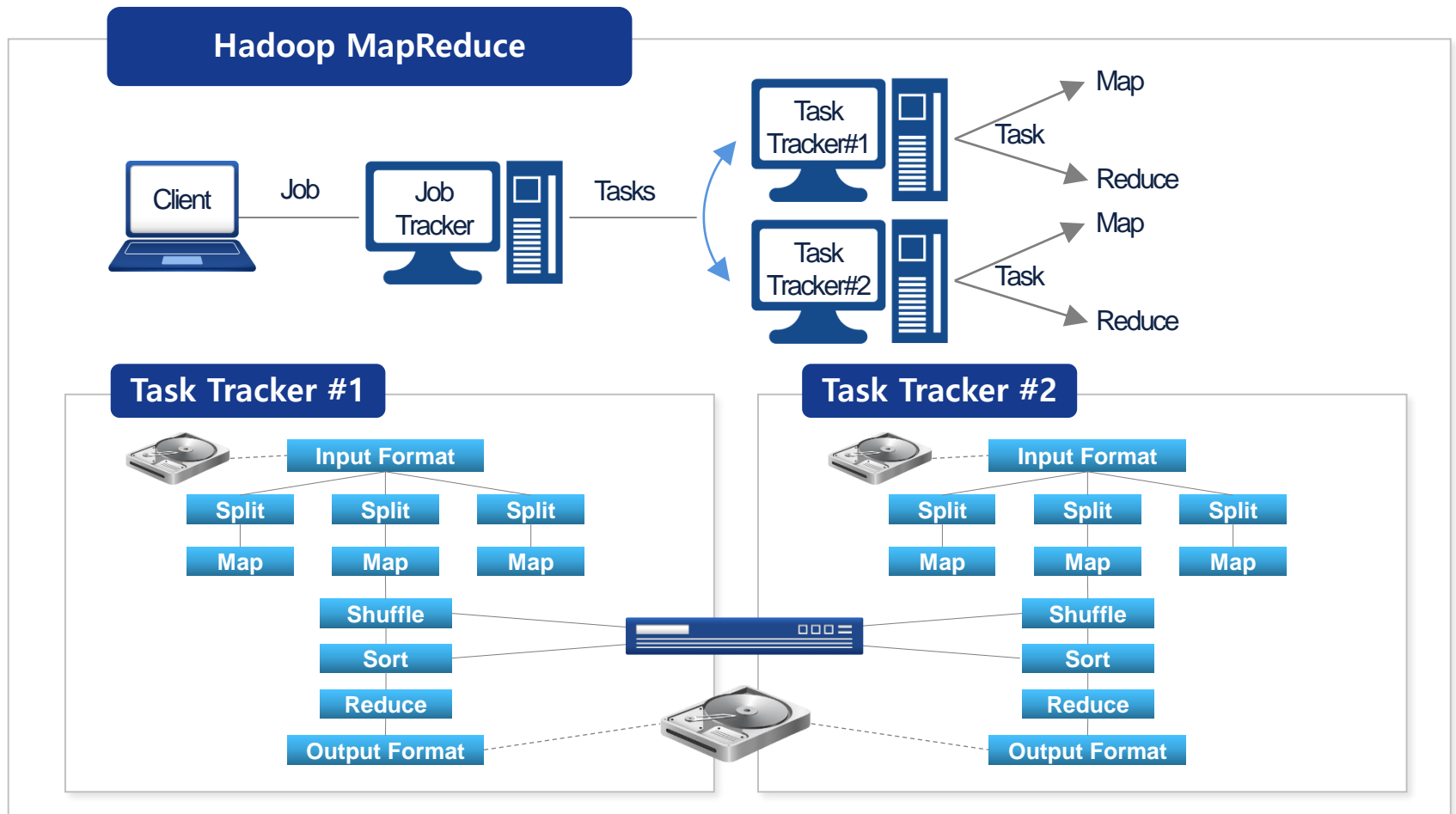
MapReduce



MapReduce

MapReduce Architecture

- Hadoop MapReduce Component



MapReduce

MapReduce Architecture

- Hadoop MapReduce

Map & Reduce 단계

Map 단계

1. Data 지역성
2. 최적화된 split size = block size
3. Map의 결과는 local disk에 저장
4. Reducer 0 => Map의 결과는 HDFS 저장

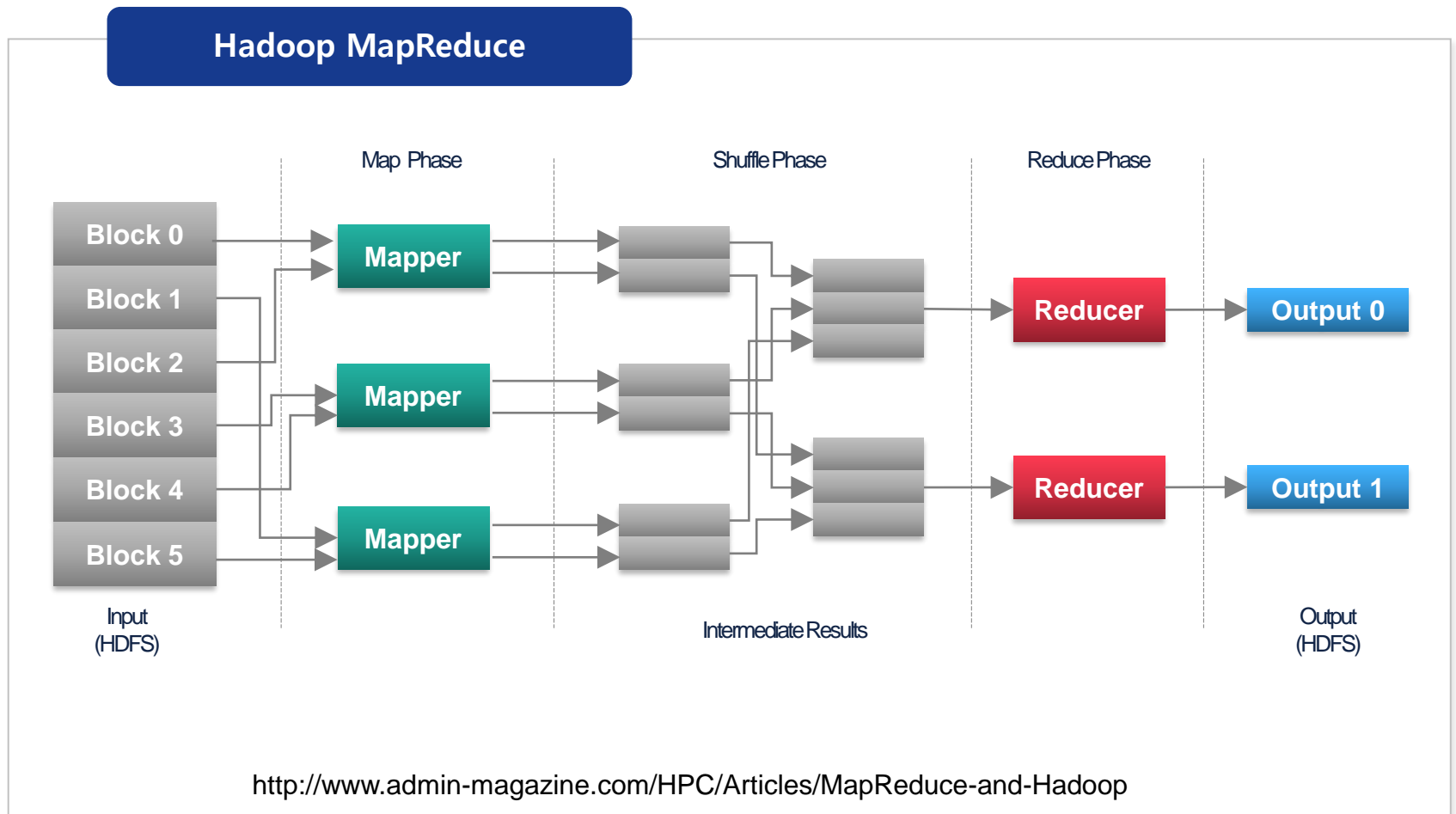
Reduce 단계

1. No data locality
2. Reducer의 숫자는 독립적으로 결정
3. Reducer의 결과는 HDFS에 쓰여짐

MapReduce

MapReduce Architecture

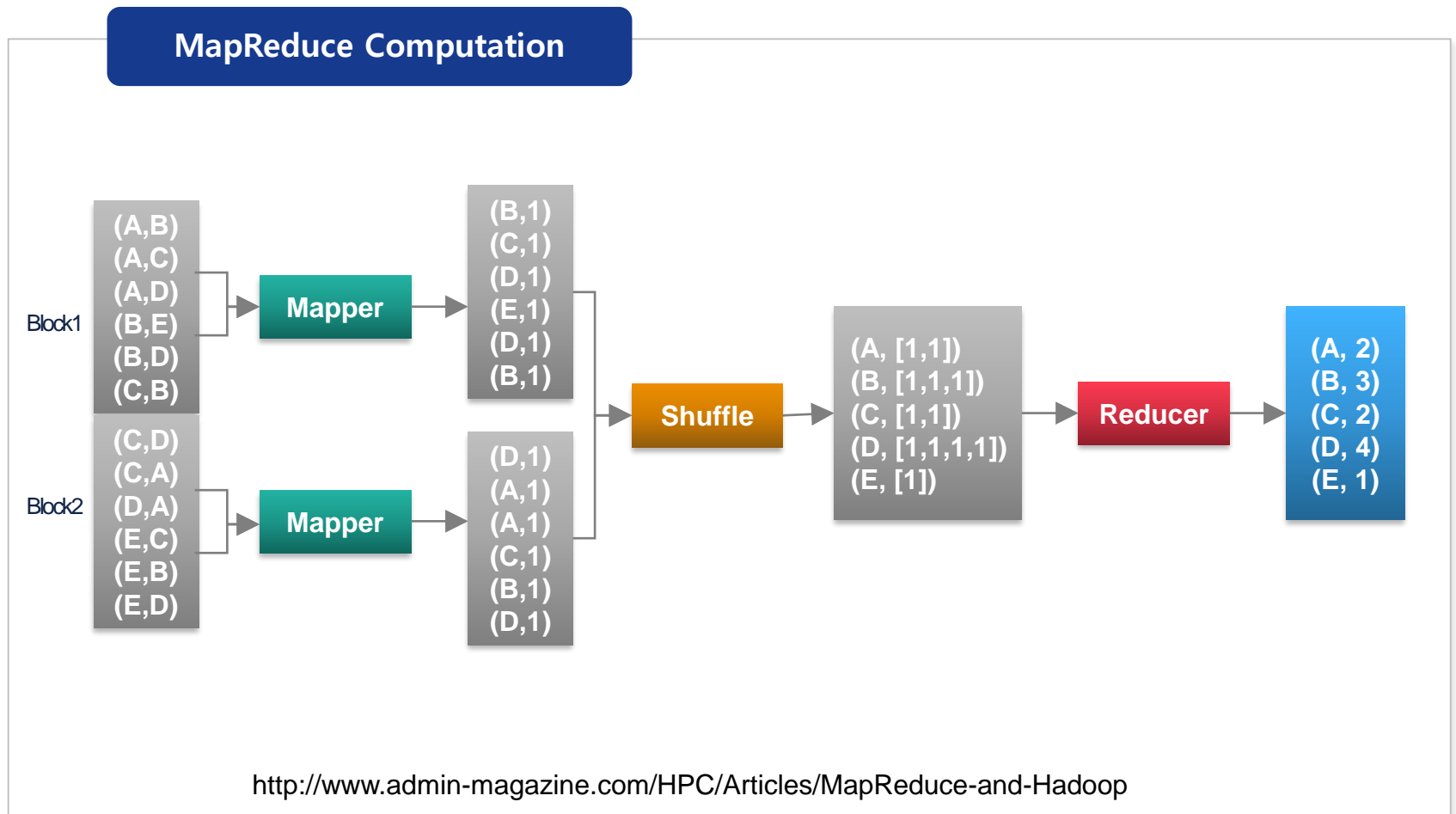
- Hadoop MapReduce Framework



MapReduce

MapReduce Architecture

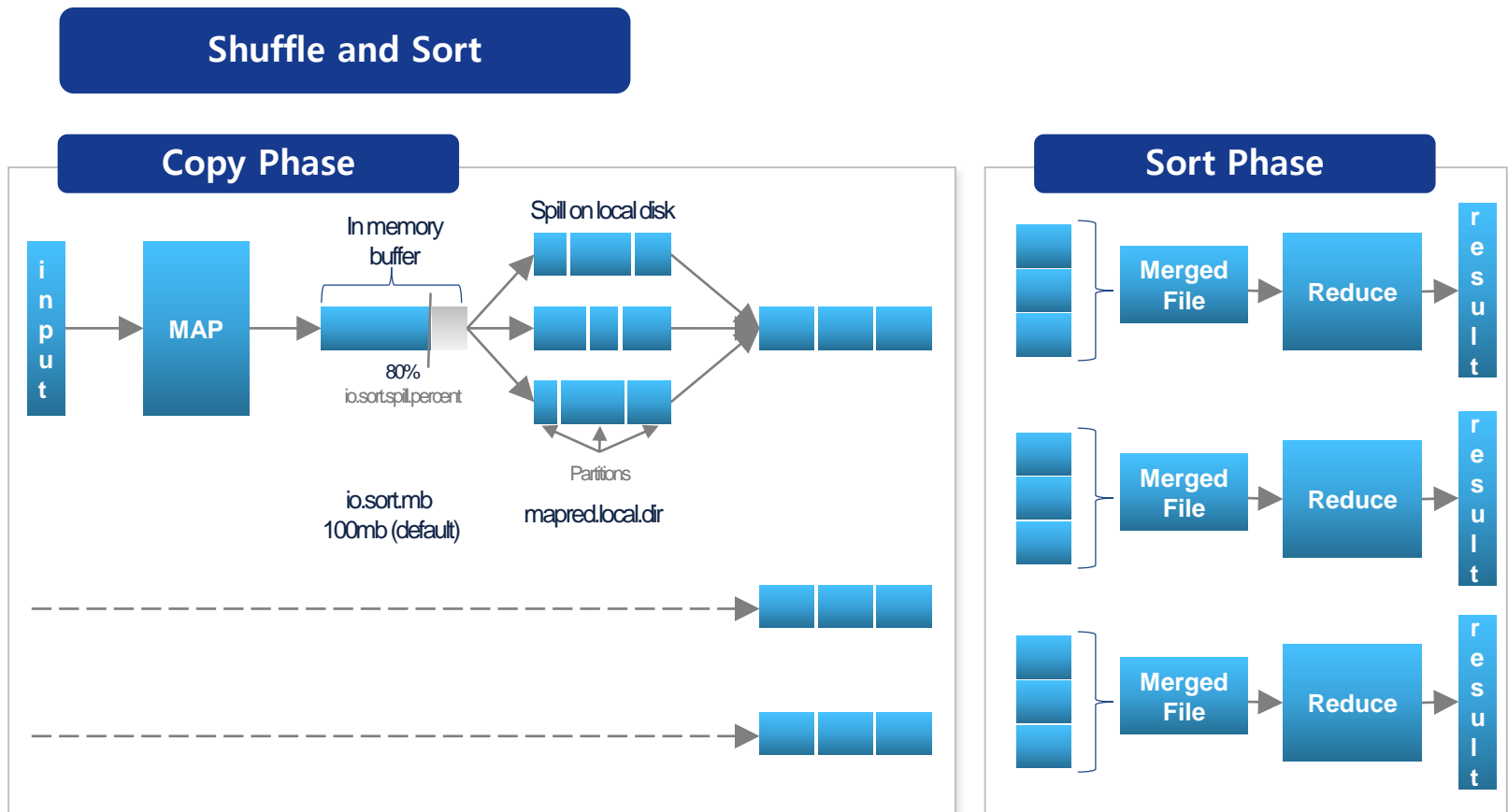
- Hadoop MapReduce Framework



MapReduce

MapReduce Architecture

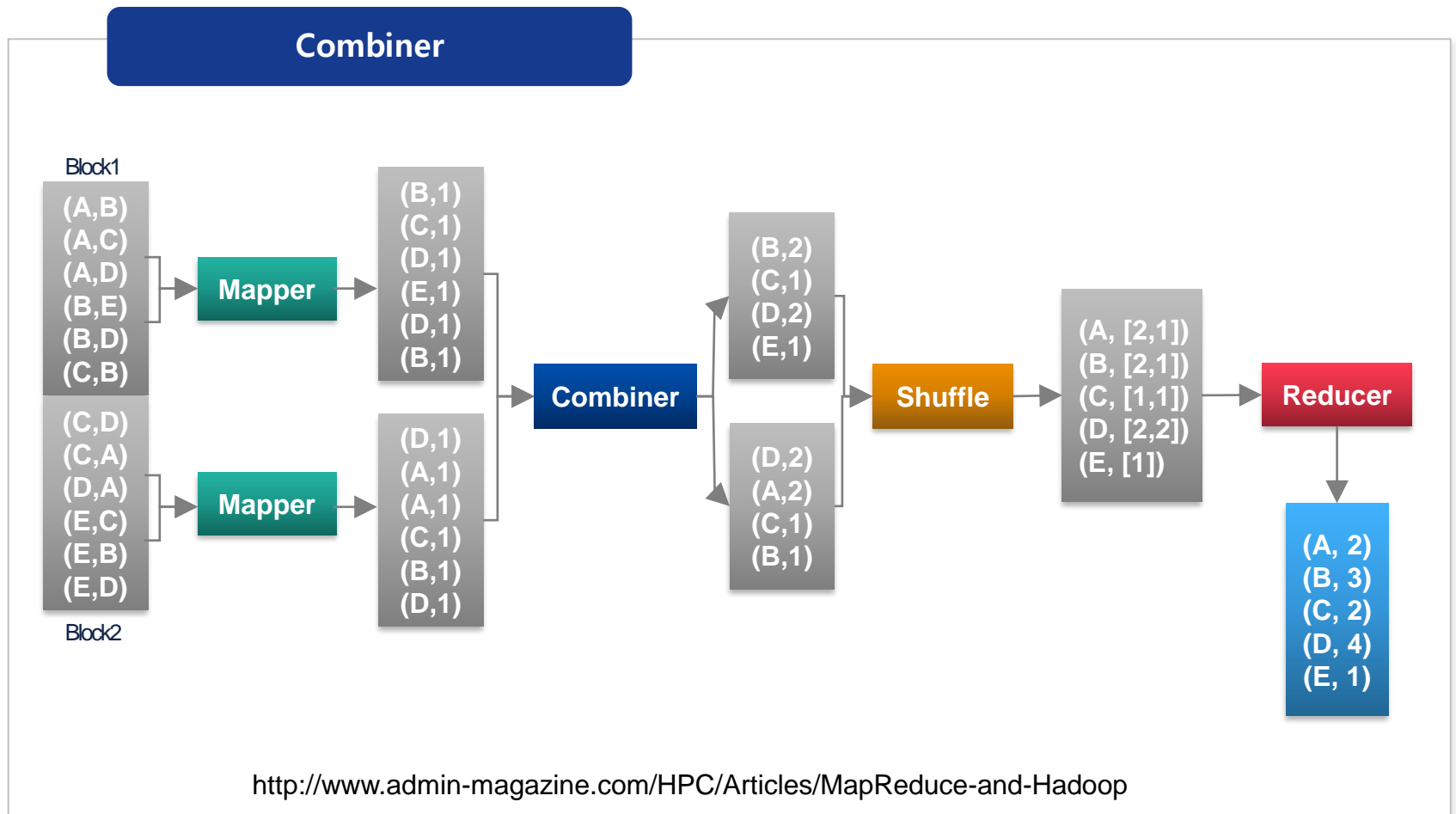
- Hadoop MapReduce Framework



MapReduce

MapReduce Architecture

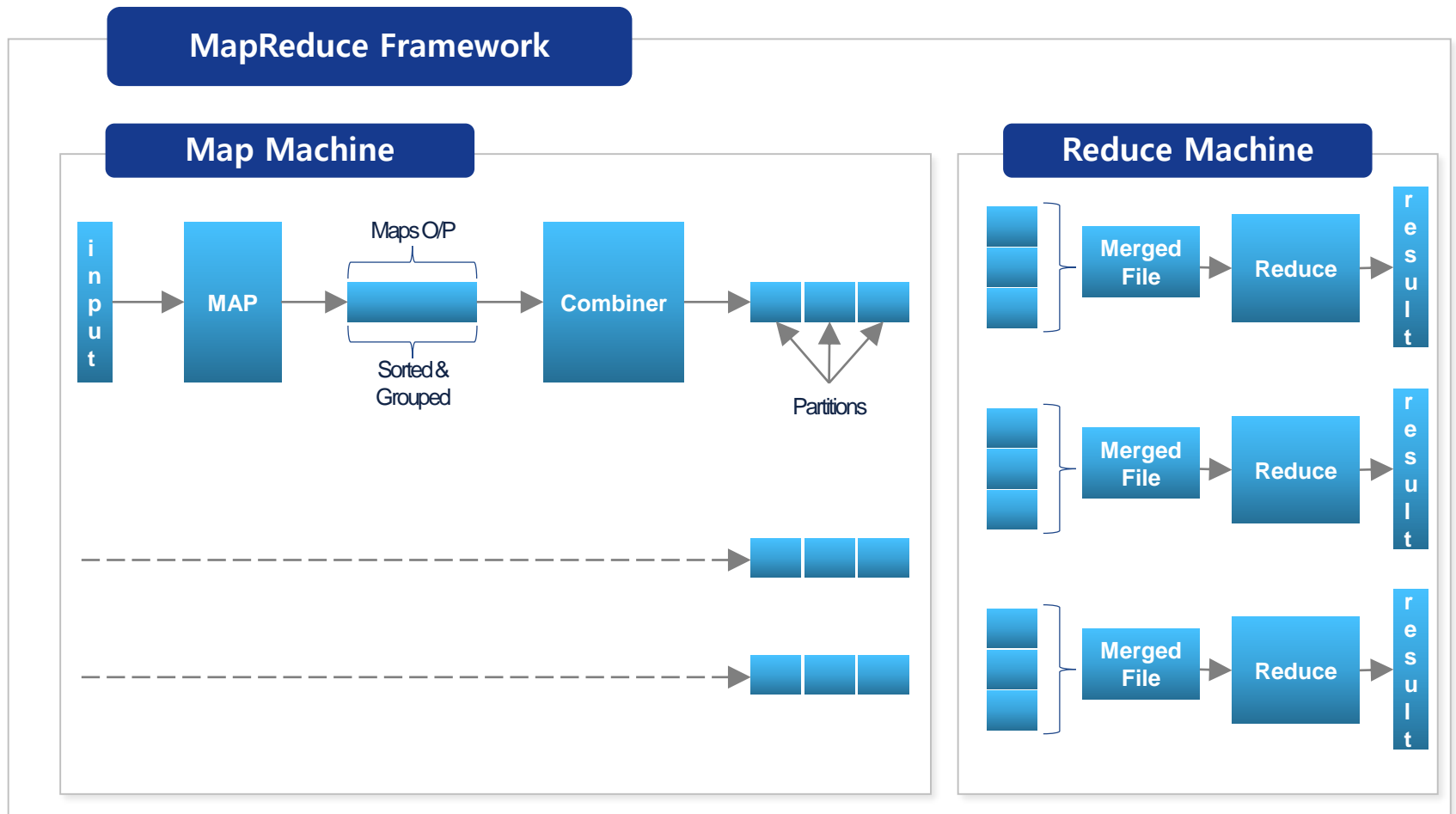
- Hadoop MapReduce Framework



MapReduce

MapReduce Architecture

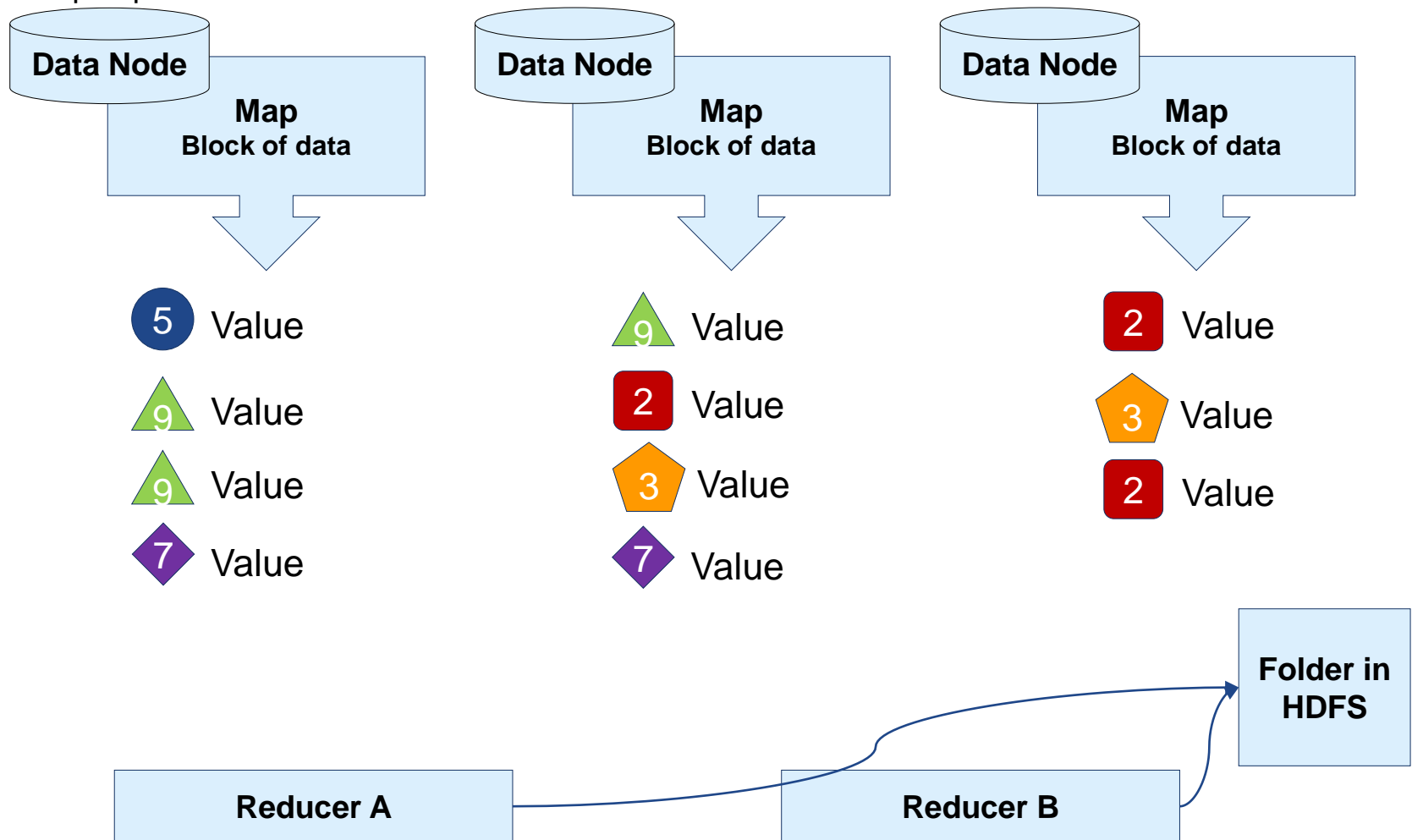
- Hadoop MapReduce Framework



MapReduce

MapReduce Architecture

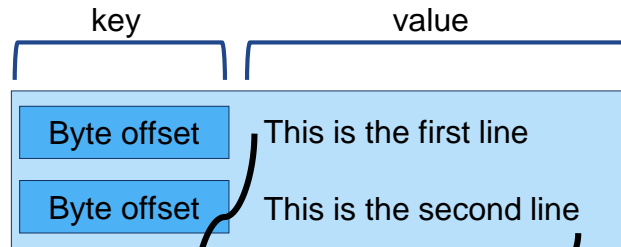
- Hadoop MapReduce Data Flow



MapReduce

MapReduce Architecture

- MapReduce Word Count Example



Key	Value	Key	Value
This	1	This	1
is	1	is	1
the	1	the	1
first	1	second	1
line	1	line	1


Reducer A

first 1
second 1
the 1
This 2

Reducer B

is 2
line 2

```
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens())
{
    word.set(tokenizer.nextToken());
    context.write(word, one);
}
```



Key	Value
This	1
This	1
the	1
the	1
first	1
second	1

Key	Value
line	1
line	1
is	1
is	1

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

MapReduce

MapReduce Architecture

- MapReduce Word Count Example

Word Count Example

1. Hadoop Cluster Login

- 1) `ssh ryan@hdp.pvm.centos`

2. Make data directory in HDFS

- 1) `sudo -u hdfs hdfs dfs -mkdir /user/ryan`
- 2) `sudo -u hdfs hdfs dfs -chown ryan:hdfs /user/ryan`
- 3) `sudo -u hdfs hdfs dfs -mkdir /user/ryan/data`
- 4) `sudo -u hdfs hdfs dfs -lsr /user/ryan`

3. Upload data

- 1) `hadoop fs -put ./Data/book1.txt /user/ryan/data/`
- 2) `hadoop fs -lsr /user/ryan`

4. MapReduce example execution

- 1) `hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/ryan/data /user/ryan/output/book`

5. Result

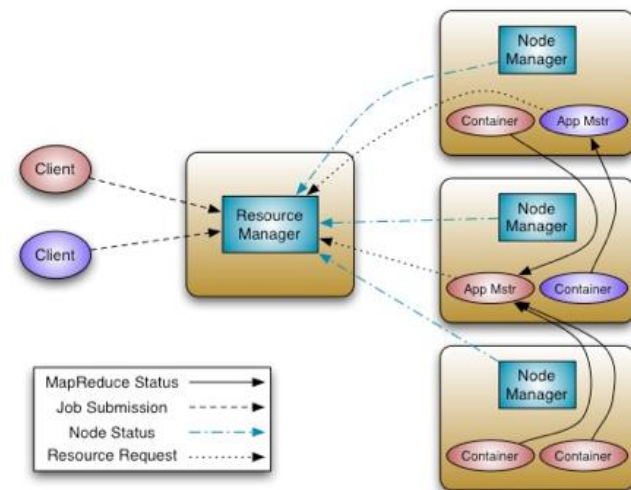
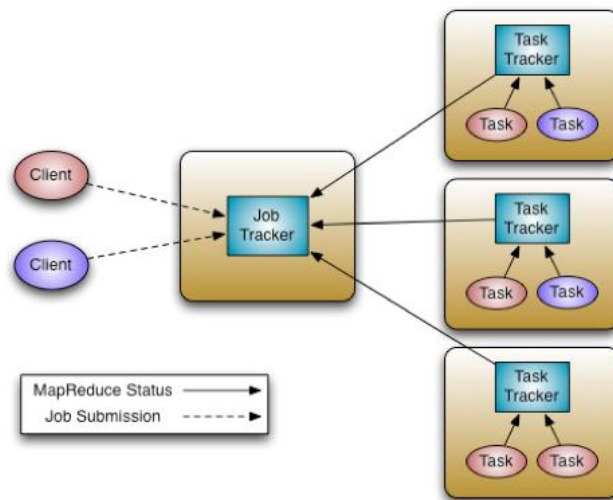
- 1) `hadoop fs -cat /user/ryan/output/book/part-r-0000 | more`

MapReduce

MapReduce Architecture

- Hadoop YARN

YARN

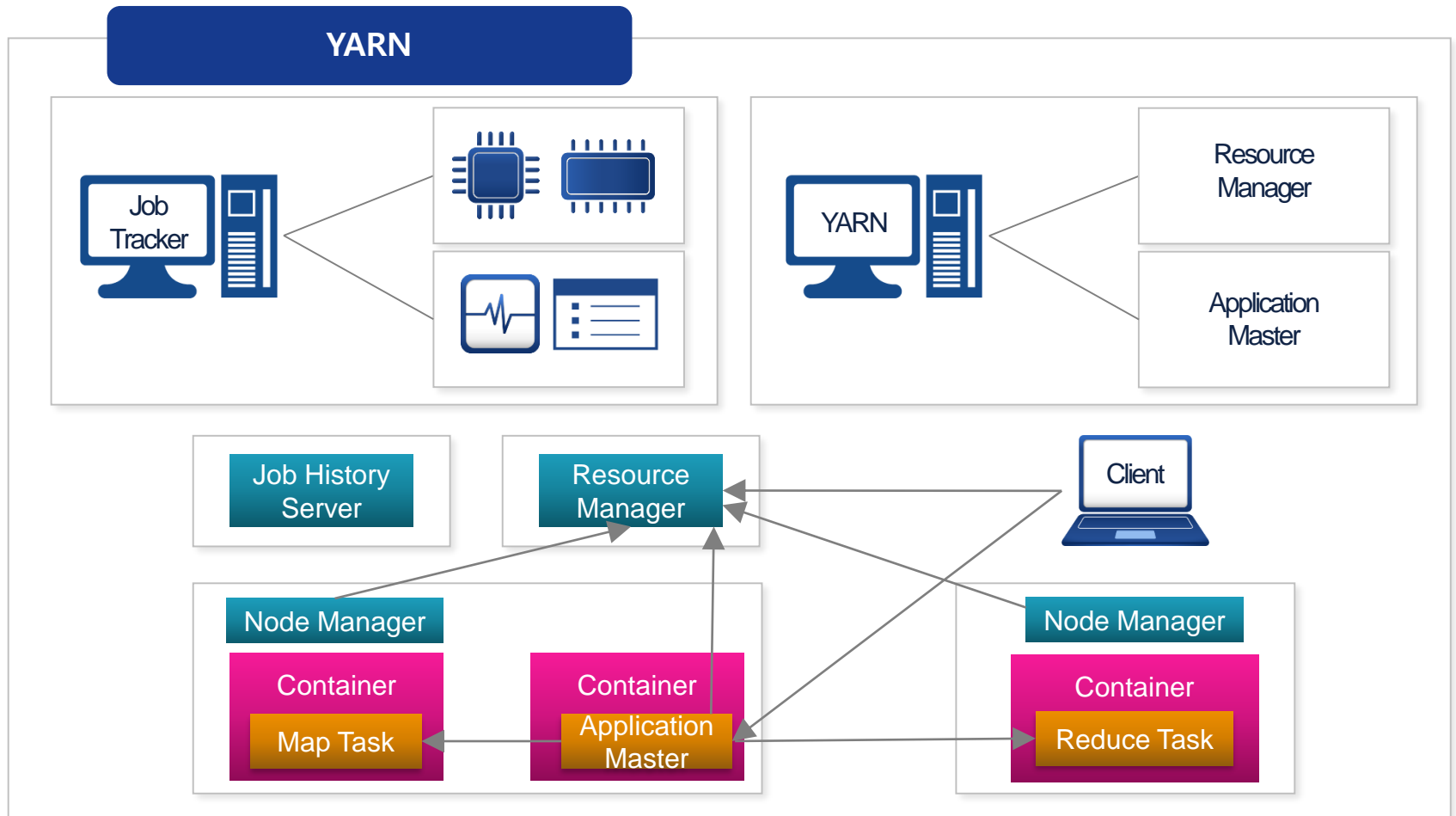


출처: <http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/>

MapReduce

MapReduce Architecture

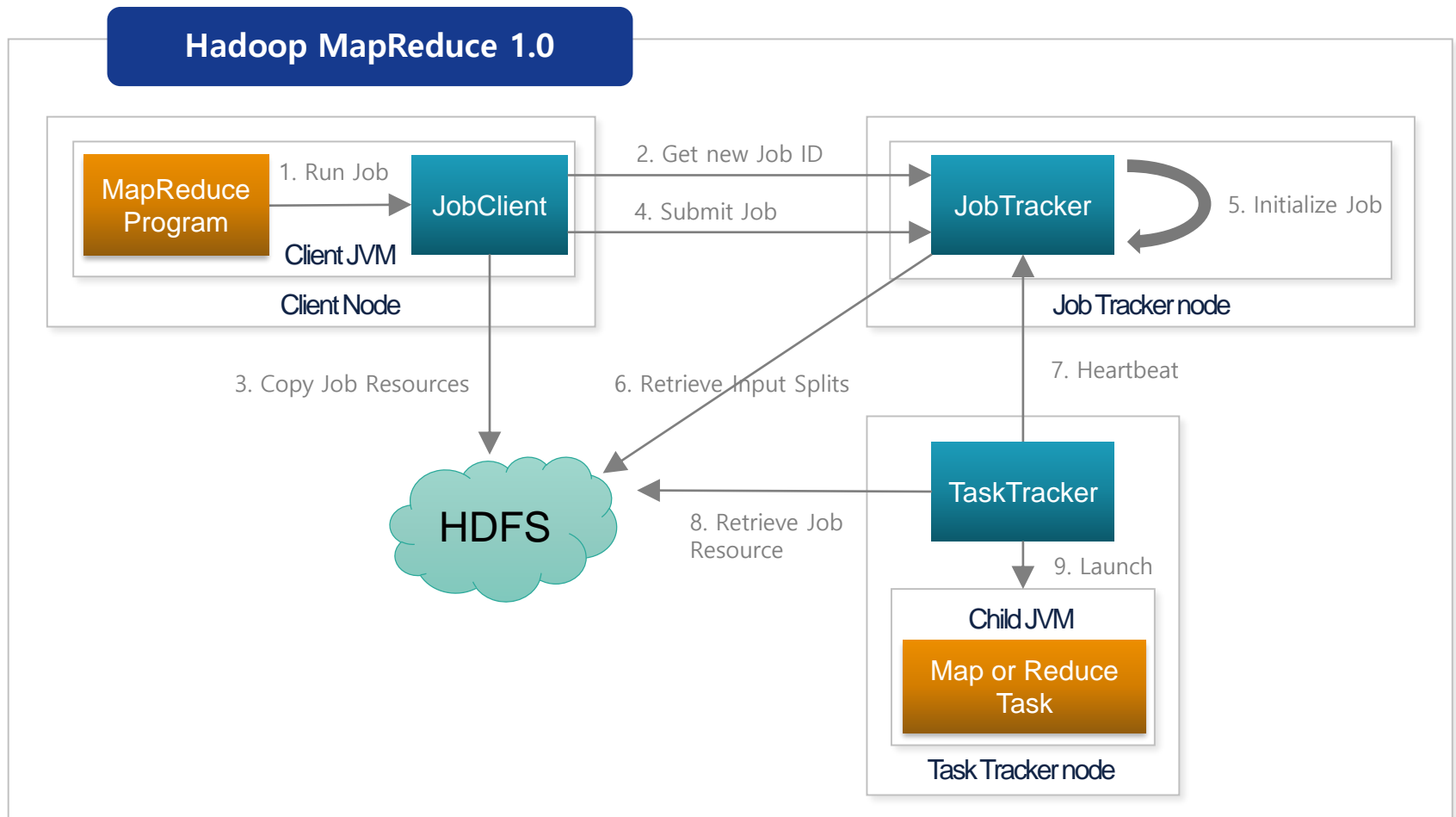
- Hadoop YARN



MapReduce

MapReduce Architecture

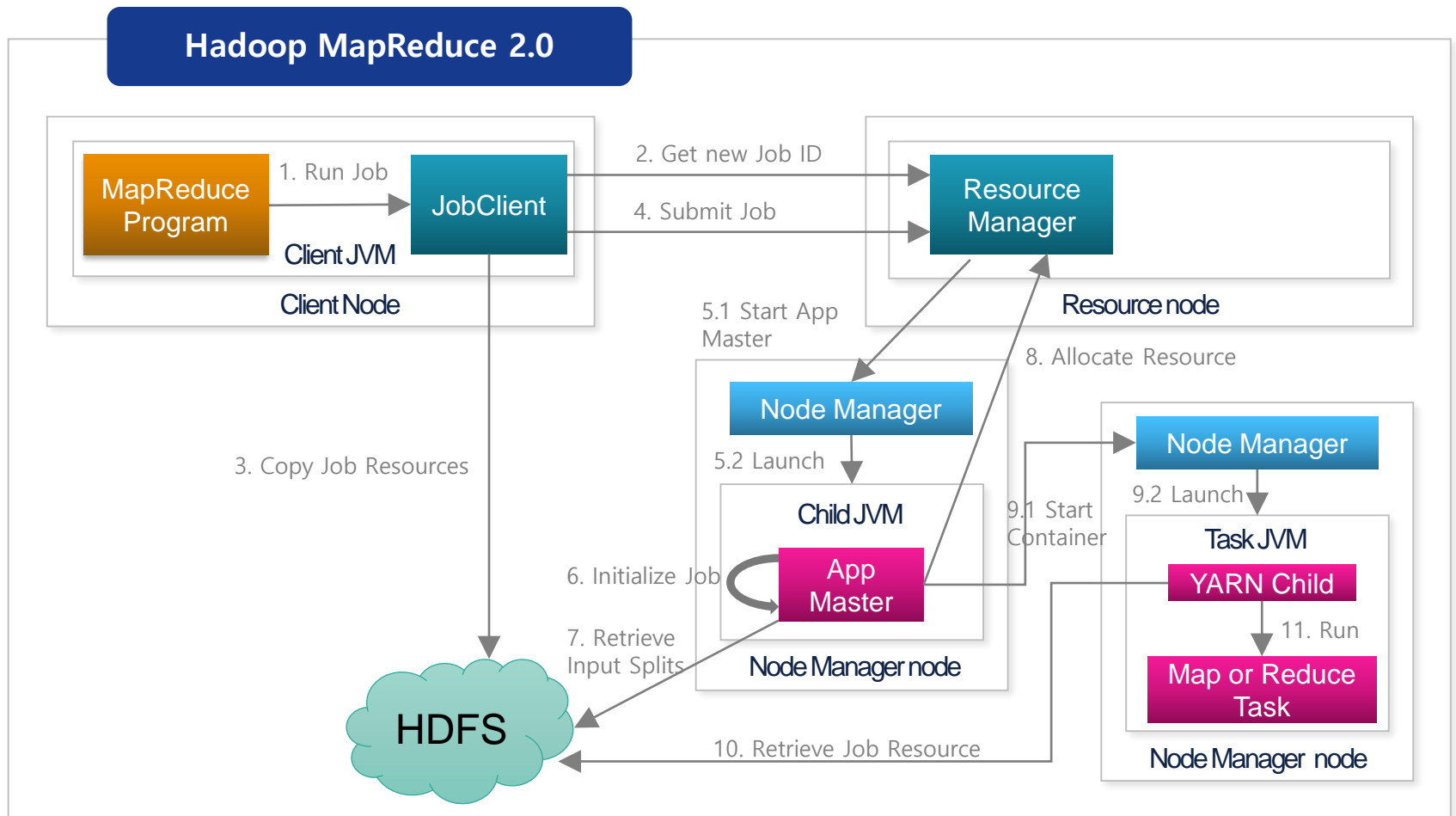
- Hadoop MapReduce 1.0



MapReduce

MapReduce Architecture

- Hadoop MapReduce 2.0



MapReduce

MapReduce Architecture

- MapReduce Key Aspects

Key Aspects

Key Aspects of MapReduce

- It's an API, or set of libraries

Job – unit of MapReduce work / instance

Map task – runs on each node

Reduce task – runs on some nodes

Source data – HDFS or other location

MapReduce Daemons and Services

- JVM or services – isolated processes

Job tracker – one (controller and scheduler)

Task tracker – one per cluster (monitor tasks)

- Job configurations

Specify input/output locations for job instances

Job clients submit jobs for execution

MapReduce

MapReduce Architecture

- Hadoop Java API

Hadoop MapReduce

1st Generation

Classic MapReduce

MapReduce Libraries

```
import org.apache.hadoop.fs.path;  
import org.apache.hadoop.conf.*;  
import org.apache.hadoop.io.*;  
import org.apache.hadoop.mapred.*;  
import org.apache.hadoop.util.*;
```

2nd Generation

YARN MapReduce

MapReduce API versions

- Version 1.0
org.apache.hadoop.mapred
- Version 2.0
org.apache.hadoop.mapreduce

MapReduce

MapReduce Architecture

- Hadoop MapReduce Data Types

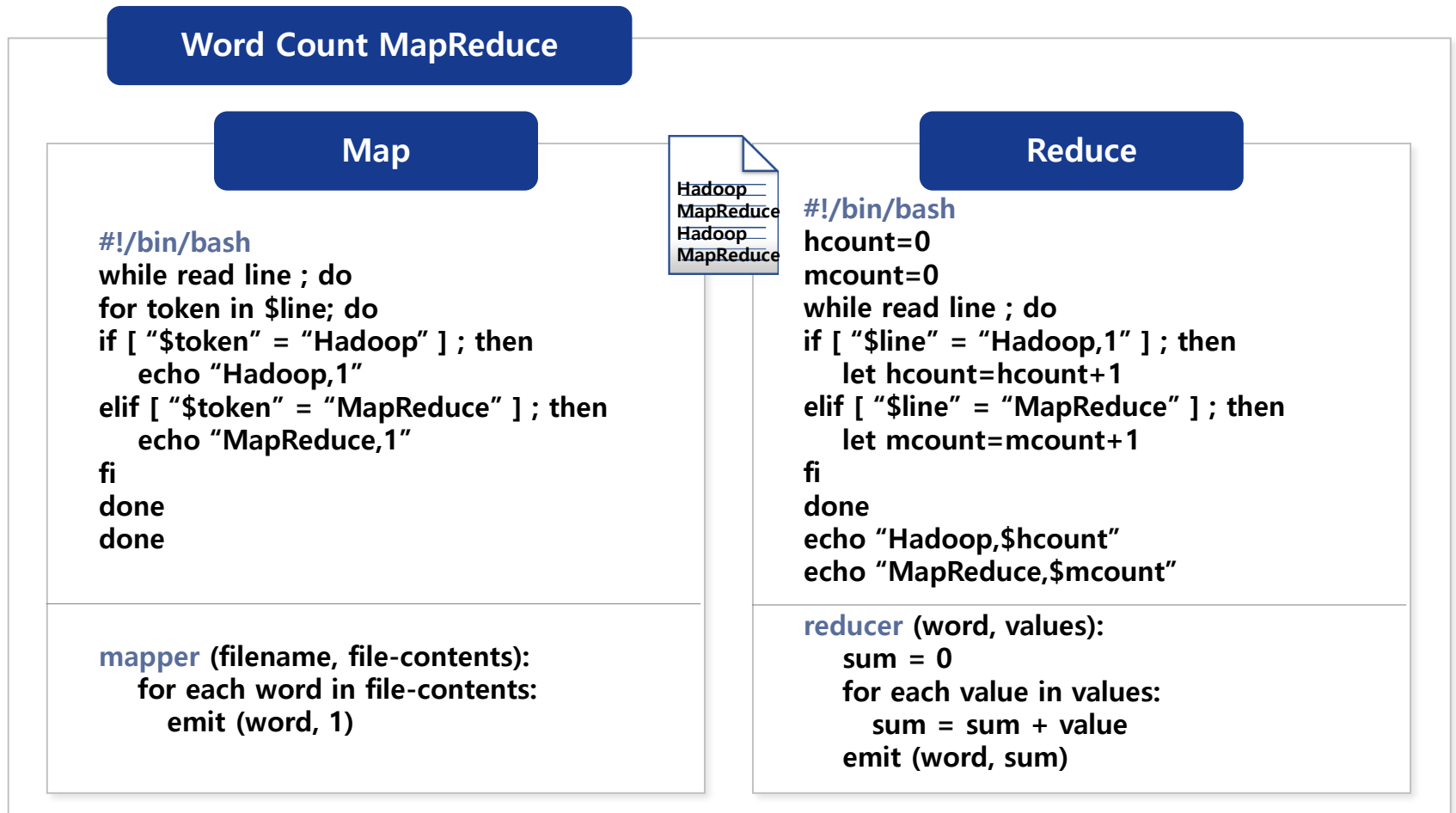
MapReduce Data Types

Java primitive	Writable Implementation	Serialized Size
boolean	BooleanWritable	1
byte	ByteWritable	1
short	ShortWritable	2
int	IntWritable	4
	VIntWritable	1 – 5
float	FloatWritable	4
Long	LongWritable	8
	VLongWritable	1 – 9
Double	DoubleWritable	8
String	Text	-

MapReduce

MapReduce Architecture

- Word Count MapReduce Pseudocode



MapReduce

MapReduce Architecture

- MapReduce Coding Steps

Coding Steps

- Create a class
- Create a static Map class
- Create a static Reduce class
- Create a main function
 - Create a job
 - Job calls the Map and Reduce classes

```
public class MapReduce {  
  
    public static void Main(String[] args)  
    {  
        //create JobRunnerInstance  
        //call MapInstance on JobInstance  
        //call ReduceInstance on JobInstance  
    }  
  
    public void Map()  
    {  
        //write Mapper  
    }  
  
    public void Reduce()  
    {  
        //write Reducer  
    }  
}
```

MapReduce

MapReduce Architecture

- Mapper & Reducer Class

Mapper & Reducer Class

```
public class Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
```

```
    protected void map(KEYIN key, VALUEIN value, Context context) throws IOException,  
                                                                    InterruptedException
```

```
    {  
        context.write((KEYOUT)key, (VALUEOUT) value);  
    }
```

```
}
```

```
KeyIn    = LongWritable  
ValueIn  = Text  
KeyOut   = LongWritable  
ValueOut = Text
```

```
public class Reducer <KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
```

```
    protected void reduce(KEYIN key, Iterable<VALUEIN> values, Context context)  
                                                                    throws IOException, InterruptedException
```

```
    {  
        for (VALUEIN value : values) {  
            context.write((KEYOUT)key, (VALUEOUT) value);  
        }
```

```
    }
```

```
}
```

```
KeyIn    = LongWritable  
ValueIn  = Text  
KeyOut   = LongWritable  
ValueOut = Text
```

MapReduce

MapReduce Architecture

- Word Count Mapper Code

Word Count Mapper Code

```
public static class Map extends MapReduceBase {
    implements Mapper<LongWritable, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value,
                        OutputCollector<Text, IntWritable> output, Reporter reporter)
            throws IOException {

            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);

            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }
}
```

MapReduce

MapReduce Architecture

- Word Count Reducer Code

Word Count Reducer Code

```
public static class Reduce extends MapReduceBase {
    implements Reducer<Text, IntWritable, Text, IntWritable> {

        public void reduce(Text key, Iterator<IntWritable> values,
                           OutputCollector<Text, IntWritable> output, Reporter reporter)
            throws IOException {

            int sum = 0;

            while (values.hasNext()) {
                sum += values.next().get();
            }

            output.collect(key, new IntWritable(sum));
        }
    }
}
```

MapReduce

MapReduce Architecture

- Word Count Driver Code

Word Count Driver Code

```
public static void main(String[] args) throws Exception {  
    JobConf conf = new JobConf(WordCount.class);  
    conf.setJobName("wordcount");  
  
    conf.setOutputKeyClass(Text.class);  
    conf.setOutputValueClass(IntWritable.class);  
  
    conf.setMapperClass(Map.class);  
    conf.setCombinerClass(Reduce.class);  
    conf.setReducerClass(Reduce.class);  
  
    conf.setInputFormat(TextInputFormat.class);  
    conf.setOutputFormat(TextOutputFormat.class);  
  
    FileInputFormat.setInputPaths(conf, new Path(args[0]));  
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
  
    JobClient.runJob(conf);  
}
```


MapReduce

MapReduce Demo

Demo



1. MapReduce Programming

MapReduce

MapReduce Demo

- MapReduce Development Environment

Development Environment

1. Install JDK
 - 1) <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
2. Download Eclipse
 - 1) <https://eclipse.org/downloads/>
3. Add Hadoop Eclipse Plugin
 - 1) <https://github.com/winghc/hadoop2x-eclipse-plugin>
4. Create a project
 - 1) Java Project
 - 2) Maven Project
5. Developemnt
 - 1) Ex : https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0
6. Build & Execution

MapReduce

MapReduce Demo

- MapReduce Word Count Demo

Word Count Demo

1. Hadoop Cluster Login
 - 1) `ssh ryan@hdp.pvm.centos`
2. Assuming environment variables are set as follows
 - 1) `export JAVA_HOME=/usr/java/default`
 - 2) `export PATH=${JAVA_HOME}/bin:${PATH}`
 - 3) `export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar`
3. Compile WordCount.java and create a jar
 - 1) `$ bin/hadoop com.sun.tools.javac.Main WordCount.java`
 - 2) `$ jar cf wc.jar WordCount*.class`
4. Run the application
 - 1) `$ bin/hadoop jar wc.jar WordCount /user/ryan/wordcount/input /user/ryan/wordcount/output`
5. Result
 - 1) `$ bin/hadoop fs -cat /user/ryan/wordcount/output/part-r-00000`

MapReduce

MapReduce Demo

사용자 View Count

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha

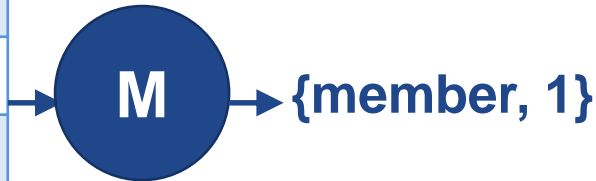


Member	View
Janani	50
Swetha	15
Shreya	22
Jitu	23
Shreya	32
Jitu	10

MapReduce

MapReduce Demo : Mapper Class Code

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha



해당 로직은 Mapper 클래스를 상속한 클래스내에 구현한다.

해당 Mapper는 데이터셋으로부터 1줄 씩 읽어 들인다.

MapReduce

MapReduce Demo : Mapper Class Code

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha

해당 Mapper는 데이터셋으로부터
1줄 씩 읽어 들인다.

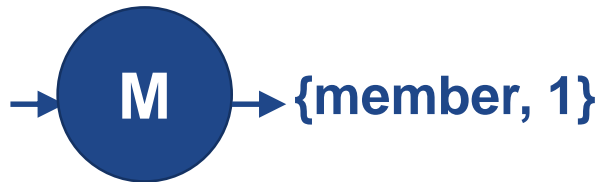
Input

{LineNum, Line}
{Long, String}

Datatypes

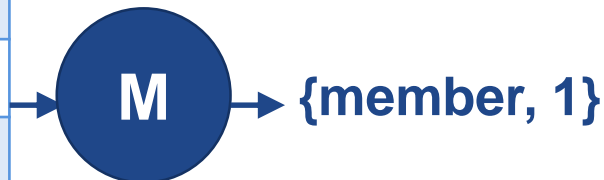
MapReduce

MapReduce Demo : Mapper Class Code



Datatypes
{String, Integer}

View ID	From Member	To Member
1	Janani	Jitu
2	Swetha	Janani
3	Shreya	Pradeep
4	Jitu	Vitthal
5	Shreya	Janani
6	Jitu	Swetha



`{ Jitu, 1 }`
`{ Janani, (1,1) }`
`{ Pradeep, 1 }`
`{ Vitthal, 1 }`
`{ Swetha, 1 }`

MapReduce

MapReduce Demo : Mapper Class Code

View Count Mapper Code

```
public class Map extends
Mapper<LongWritable, Text, Text, IntWritable> {
    @Override

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

        String[] row = value.toString().split("Wt");
        context.write(new Text(row[2]), new IntWritable(1));

    }
}
```


MapReduce

MapReduce Demo : Map Code

```
public class Map extends  
Mapper<LongWritable, Text, Text,  
IntWritable> {  
    @Override
```

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

```
String[] row =  
value.toString().split("Wt");
```

```
context.write(new Text(row[2]),  
new IntWritable(1));  
}  
}
```

Map 로직은 map 메소드를
Override하여 구현한다.

Map 메소드 인자는
key, value, context이다.

Context는 output key, value쌍을
저장하는 객체이다.

Context는 Reducer 클래스로
Data를 전송하기 전에
내부적으로 shuffle, sort로직이
발생할 것이다.

→ 라인을 가져와서 단어로 분할한다.

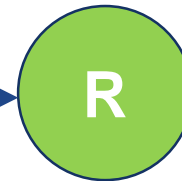
→ 세번째 단어는 멤버이름이다.
Map의 output value는 항상 1이다.

MapReduce

MapReduce Demo : Reducer Class Code

각 멤버들을 위해
Reducer는 collection을 가져온다. Sum

{member, 1} →



Collection은 같은 키에 대응하는
모든 값을 가지고 있다.

Member	View
Janani	50
Swetha	15
Shreya	22
Jitu	23
Shreya	32
Jitu	10

MapReduce

MapReduce Demo : Reducer Class Code

Member	View
Janani	50
Swetha	15
Shreya	22
Jitu	23
Shreya	32
Jitu	10

Input

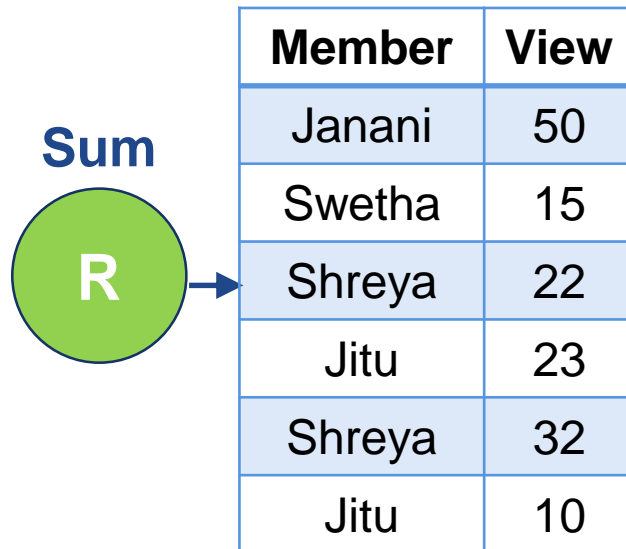
**{Member, Counts of
Collection}**

{String, Iterable<Integer>}

Datatypes

MapReduce

MapReduce Demo : Reducer Class Code



Datatypes
{String, Integer}

MapReduce

MapReduce Demo : Reducer Class Code

View Count Reducer Code

```
public class Reduce extends
Reducer<Text, IntWritable, Text, IntWritable> {
    @Override

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

MapReduce

MapReduce Demo: Reduce Code

```
public class Reduce extends
Reducer<Text, IntWritable, Text,
IntWritable> {
    @Override

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

    int count = 0;
    for (IntWritable value:values)
    {
        count += value.get();
    }

    context.write(key, new
IntWritable(count));

}
}
```

Reduce 로직은 reduce 메소드를
Override하여 구현한다.

Reduce 메소드 인자는
key, iterable of values, context이다.

Context는 output key, value쌍을
저장하는 객체이다..

→ Values를 통해 반복하고 sum을 한다.

→ 적합한 데이터 타입으로
Context에 key value쌍을 저장한다.

MapReduce

MapReduce Demo : Main Class

View Count Main Code

```
public class ViewCount extends Configured implements Tool {  
    @Override  
    public int run(String[] args) throws Exception {...}  
    public static void main(String[] args) throws Exception {  
        int exitCode = ToolRunner.run(new ViewCount(), args)  
        System.exit(exitCode);  
    }  
}
```

MapReduce

MapReduce Demo: Main class

Hive

Hive Architecture

Hive Demo

HIVE Overview

Hive Architecture

하이프 시작 동기

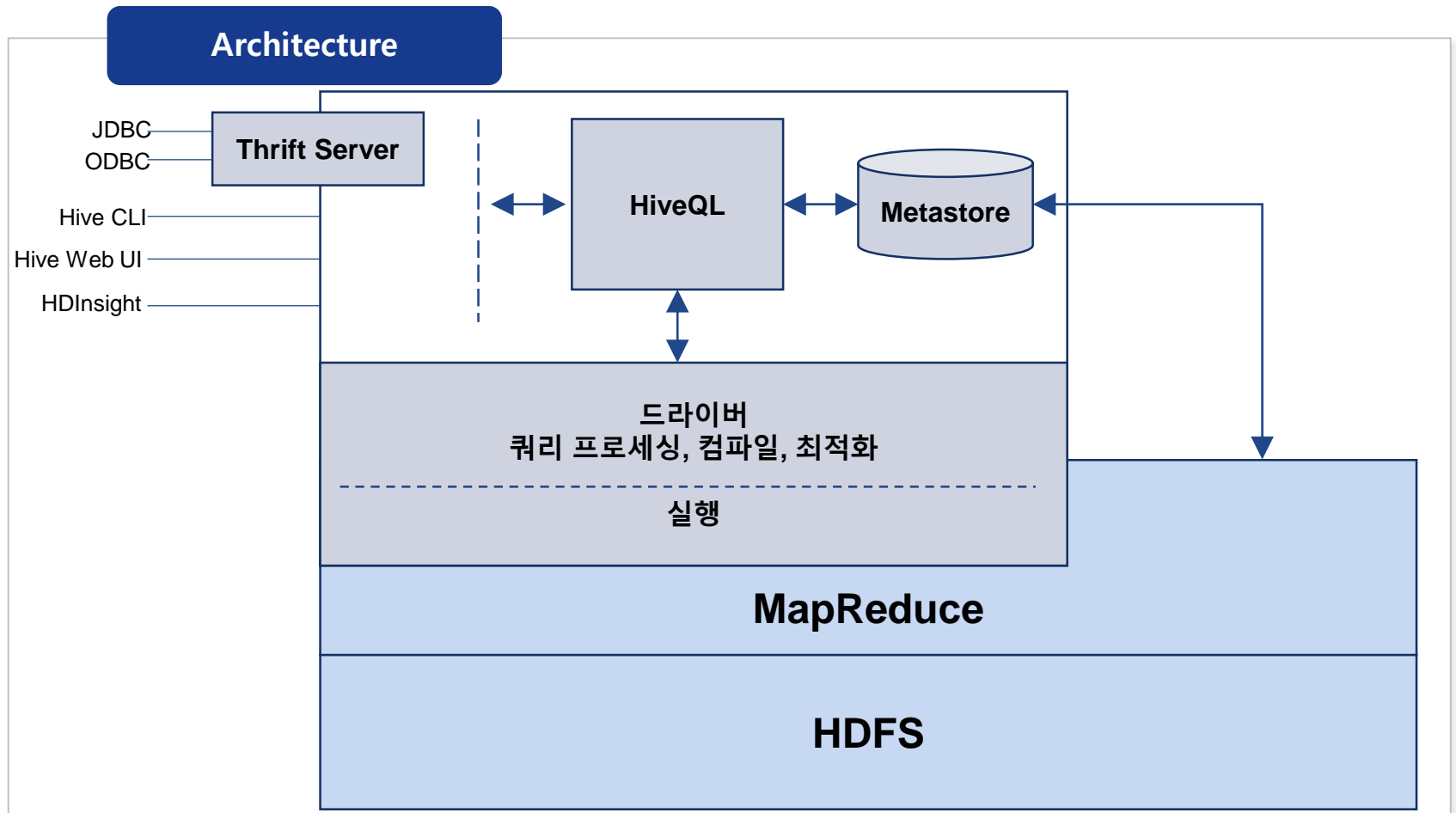


- 큰 덩어리의 빅 데이터를 열기 위해서...
- SQL-like한 쿼리 언어와 인터페이스를 제공하기 위해서...
- 실행을 위해 하둡 코어기반의 맵 리듀스를 사용하여 구축하기 위해서...
- 페이스북에서 먼저 시작
 - 맵 리듀스 개발에 시간이 많이 소비됨
 - 프레임워크에 대한 전문지식이 요구됨
 - 필요한 전문 지식을 갖춘 인적자원이 제한적임
 - HDFS내의 데이터를 이해하는데 도움이 되는 스키마가 없음

HIVE

Hive Architecture

- Hive Architecture



HIVE

Hive Architecture

- Hive Architecture

Metastore

```
CREATE TABLE IF NOT EXISTS stocks (  
  exch string,  
  symbol string,  
  ymd string,  
  price_open float,  
  price_high float,  
  price_low float,  
  price_close float )  
ROW FORMAT DELEMITED FIELDS TERMIATED BY ';;'
```

Metadata



MySQL

Dataset



```
ABCDs, BAC, 2015-10-20, 30.10, 30.18, 30.03, 30.15  
ABCDs, BAC, 2015-10-21, 30.20, 30.23, 30.17, 30.25  
ABCDs, BAC, 2015-10-22, 30.30, 30.34, 30.24, 30.48  
ABCDs, BAC, 2015-10-23, 30.50, 30.52, 30.46, 30.30  
ABCDs, BAC, 2015-10-24, 30.30, 30.35, 30.35, 30.20  
ABCDs, BAC, 2015-10-25, 30.20, 30.40, 30.18, 30.10  
ABCDs, BAC, 2015-10-26, 30.10, 30.23, 30.09, 30.20  
ABCDs, BAC, 2015-10-27, 30.20, 30.32, 30.19, 30.38  
ABCDs, BAC, 2015-10-28, 30.40, 30.52, 30.32, 30.49  
ABCDs, BAC, 2015-10-29, 30.50, 30.68, 30.46, 30.39
```



HDFS

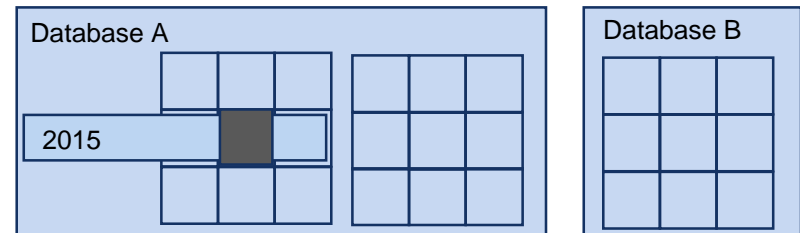
HIVE

Hive Architecture

- Hive Architecture

Hive Warehouse

- Hive의 모든 객체에 관한 메타 데이터는 메타 저장소에 보관
- 구성
 - Database
 - Tables
 - Partitions
 - Buckets/Clusters
- 로컬 Hive warehouse
 - Hive에 의해 관리
 - 기본적으로 /hive/warehouse 아래 위치
 - 테이블을 삭제하면 데이터 뿐만 아니라 메타데이터도 삭제
- 외부 테이블
 - Hive는 단지 메타 데이터만 관리
 - HDFS의 어디든지 위치
 - Hive에서 테이블 삭제 시 단지 테이블 정의만 삭제됨
데이터는 건드리지 않음



- Hive Architecture

Schema on READ

- Hive는 특정 형식을 강요하지 않습니다.
- 직렬화 / 역 직렬화를 사용해서 데이터를 읽고 씁니다.
- 데이터를 읽을 때(Read) Schema 형태에 맞게 됐는지 확인 (최대한 유연하게 데이터를 읽어 들임).
 - Ex) 데이터가 선언된 테이블의 컬럼 수 보다 작으면?
 - ✓ null로 처리하거나 데이터 형을 임의로 바꿔서 처리

HIVE

Hive Architecture

- Hive CLI

CLI

- Hive
 - `hive -e 'select a, b, from t1 where c=15'`
 - `hive -S -e 'select a, b, from t1' > result.txt`
 - `hive -f /my/local/file/get-data.sql`
- Variable Substitution (4 namespaces)
 - **hivevar**
 - `-d, --define, --hivevar`
 - `set hivevar:name=value`
 - **hiveconf**
 - `--hiveconf`
 - `set hiveconf:property=value`
 - **system**
 - `set system:property=value`
 - **env**
 - `set env:property=value`

```
$ hive -d srctable=movies
hive> set hivevar:cond=123;
hive> select a,b,c from demodb.${hivevar.srctable}
      where a = ${hivevar.cond}
```

```
$ hive -v -d src=movies -d db=demodb -e 'select *
      from ${hivevar:db}.${hivevar.src} LIMIT 100;
```

HIVE

Hive Architecture

- Basic commands using HiveQL

Create Database & Table

Create Database

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name  
[COMMENT some_comment]  
[LOCATION hdfs_path]  
[WITH DBPROPERTIES (property_name=property_value, ...)];
```

```
USE database_name;  
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name;
```

HDFS

/hive/warehouse

aaa.db

bbb.db

/somewhere/on/hdfs

ccc.db

Create Table

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name  
[(col_name datatype [COMMENT col_comment], ...)]  
[PARTITIONED BY (col_name datatype [COMMENT col_comment], ...)]  
[ROWFORMAT rowformat] [STORED AS file_format]  
[LOCATION hdfs_path]  
[TBLPROPERTIES (property_name=property_value, ...)];
```

HDFS

/hive/warehouse

aaa.db

sales

/somewhere/on/hdfs

ccc.db

employees

- Basic commands using HiveQL

LOAD DATA

LOAD DATA INPATH [path]

- Moves data if source is HDFS
- **LOAD DATA INPATH '/somewhere/on/hdfs/data'**
[OVERWRITE] INTO TABLE table_name;

LOAD DATA LOCAL INPATH [path]

- Copies data if source is LOCAL
- **LOAD DATA LOCAL INPATH '/somewhere/on/local/data'**
[OVERWRITE] INTO TABLE table_name;

- Basic commands using HiveQL

Table Partitions

Managed Partitioned Tables

```
CREATE TABLE page_views (userid STRING, page STRING)
PARTITIONED BY (dt STRING, ostype STRING)
STORED AS TEXTFILE;
```

```
LOAD DATA INPATH '/data/linux/20151218/pageviews/'
[OVERWRITE] INTO TABLE page_views
PARTITION (dt='2015-12-18', ostype='linux');
```

```
ALTER TABLE page_views ADD PARTITION (dt='2015-12-18', ostype='linux');
LOCATION '/data/linux/20151218/pageviews/';
```

HDFS

/hive/warehouse/page_views

dt=2015-12-18

ostype=linux

Virtual Partition Columns

dt STRING
ostype STRING

- **SELECT** dt as eventdate, page, count(*) as pviewcount FROM page_views
WHERE ostype='linux'

HIVE

Hive Architecture

- Basic commands using HiveQL

Buketing

Bucketing

```
CREATE TABLE t1 (a INT, b STRING, c STRING)  
CLUSTERD BY (b) INTO 256 BUCKETS
```

```
CREATE TABLE t1 (a INT, b STRING, c STRING)  
PARTITIONED BY (dt STRING)  
CLUSTERD BY (b) SORTED BY (c) INTO 64 BUCKETS
```

Enforce Bucketing

- set **mapred.reduce.tasks=64;**
 - INSERT OVERWRITE TABLE t1
SELECT a, b, c FROM t2 **CLUSTER BY b;**
- set **hive.enforce.bucketing=true;**
 - INSERT OVERWRITE TABLE t1
SELECT a, b, c FROM t2;

- Basic commands using HiveQL

Bucket, Block Sampling

Bucket Sampling

```
SELECT * FROM source TABLESAMPLE (BUCKET x OUT OF y [ON colname])
```

```
CREATE TABLE page_views (userid INT, page STRING, views INT)  
PARTITIONED BY (dt STRING)  
CLUSTERED BY (userid) SORTED BY (dt) INTO 64 BUCKETS
```

```
SELECT * FROM page_views TABLESAMPLE (BUCKET 3 OUT OF 64 ON userid);
```

Block Sampling

```
SELECT * FROM source TABLESAMPLE (n PERCENT);  
SELECT * FROM source TABLESAMPLE (x M);  
SELECT * FROM source TABLESAMPLE (10 ROWS);
```

```
SELECT * FROM source TABLESAMPLE (0.1 PERCENT);  
SELECT * FROM page_views TABLESAMPLE (90M);
```

- Basic commands using HiveQL

SELECT Statement

```
SELECT
    exp1, exp2, exp3
FROM
    some_table
WHERE
    where_condition
LIMIT
    number_of_records;
```

```
FROM
    some_table
SELECT
    exp1, exp2, exp3
WHERE
    where_condition
```

DISTINCT Clause

```
SELECT DISTINCT col1,col2,col3 FROM some_table;
```

Aliasing

```
SELECT col1 + col2 AS col3 FROM some_table;
```

REGEX Column Specification

```
SELECT '(ID|Name)?+.' FROM some_table;
```

- Interchangeable constructs
- Hive is not case sensitive
- Semicolon to terminate statements

- Basic commands using HiveQL

Sub queries & Union

```
SELECT mycol
FROM(
    SELECT col_a + col_b AS mycol
    FROM some_table;
)subq;
```

```
SELECT col_a + col_b AS mycol
FROM some_table
UNION ALL
SELECT col_y AS mycol
FROM another_table;
```

```
SELECT t3.mycol
FROM(
    SELECT col_a+col_b AS mycol
    FROM some_table
    UNION ALL
    SELECT col_y AS mycol
    FROM another_table
)t3
JOIN t4 ON (t4.col_x=t3.mycol);
```

- Basic commands using HiveQL

a	b	c
1	H	10
2	A	10
1	H	20
1	B	10
1	S	10

HAVING & GROUP BY

```
SELECT
  a, b, SUM(c)
FROM
  t1
GROUP BY
  a, b
```

a	b	_c0
1	B	10
1	H	30
1	S	10
2	A	10

```
SELECT
  a, b, SUM(c)
FROM
  t1
GROUP BY
  a, b
HAVING
  SUM(c) > 2
```

a	b	_c0
1	B	10
1	H	30
1	S	10
2	A	10

```
SELECT
  a, SUM(c)
FROM
  t1
GROUP BY
  a
```

a	_c0
1	50
2	10

```
SELECT
  CONCAT(a, b) as r
  , SUM(c)
FROM
  t1
GROUP BY
  CONCAT(a, b)
HAVING
  SUM(c) > 2
```

r	_c0
1B	10
1H	30
1S	10
2A	10

HIVE

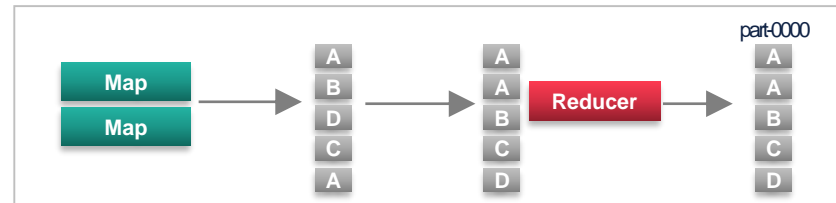
Hive Architecture

- Basic commands using HiveQL

ORDER, SORT, DISTRIBUTED, CLUSTER BY

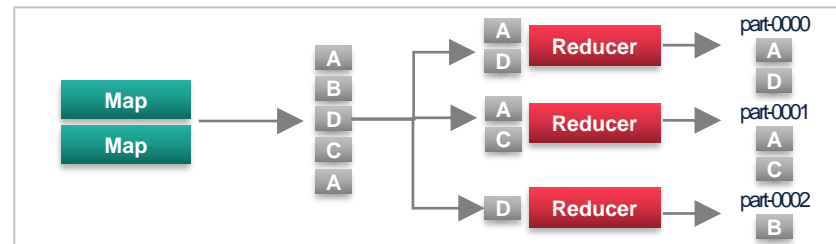
ORDER BY

```
SELECT x, y, z FROM t1 ORDER BY x ASC
```



SORT BY

```
SELECT x, y, z FROM t1 SORT BY x
```

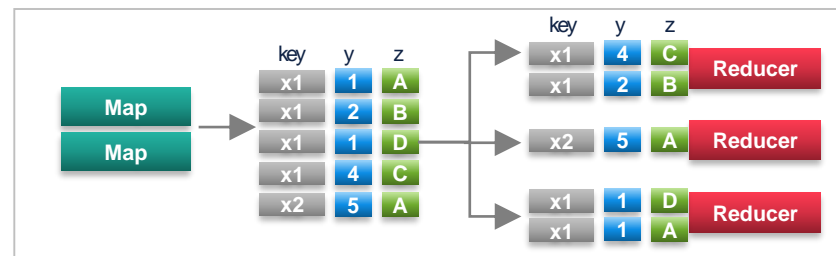


DISTRIBUTED BY

```
SELECT x, y, z FROM t1 DISTRIBUTED BY y
```

CLUSTER BY

```
SELECT x, y, z FROM t1 CLUSTER BY y
```



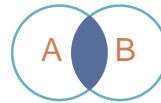
HIVE

Hive Architecture

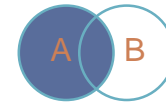
- Basic commands using HiveQL

JOIN

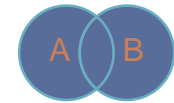
Inner Join



Left Join



Full Join



JOIN (Inner Join)

```
SELECT a.val, b.val FROM a JOIN b ON (a.key = b.key);
```

LEFT, RIGHT, FULL [OUTER] JOIN

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key = b.key) JOIN c ON (c.key = a.key);
```

LEFT SEMI JOIN

```
SELECT a.val FROM a WHERE a.key IN (SELECT b.key FROM b) – Not Supported
```

```
SELECT a.val FROM a WHERE EXISTS (SELECT 1 FROM b WHERE b.key=a.key) – Not Supported
```

```
SELECT a.val FROM a LEFT SEMI JOIN b ON (a.key=b.key);
```

HIVE

Hive Architecture

- Basic commands using HiveQL

Multiple Inserts

Interchangeability of blocks

```
FROM table_name  
SELECT *;
```

Syntax

```
FROM from_statement  
INSERT OVERWRITE TABLE table1 [PARTITION (partcol1=val1, partcol2=val2)] select_statement1  
INSERT INTO TABLE table2 [PARTITION (partcol1=val1, partcol2=val2) [IF NOT EXISTS]]  
select_statement2  
INSERT OVERWRITE DIRECTORY 'path' select_statement3
```

Example

```
FROM movies  
INSERT OVERWRITE TABLE horror_movies SELECT * WHERE horror = 1 AND  
release_date='12/18/2015'  
INSERT INTO action_movies SELECT * WHERE action=1 AND release_date='12/18/2015'
```

- Basic commands using HiveQL

Dynamic Partition Inserts

```
CREATE TABLE views_stg (eventTime STRING, userid STRING  
PARTITIONED BY(dt STRING, ostype STRING, page STRING);
```

```
FROM page_views src  
INSERT OVERWRITE TABLE views_stg PARTITION(dt='2015-12-18',ostype='linux',page='Home')  
  SELECT src.eventTime, src.userid, WHERE dt='2015-12-18' AND ostype='linux',page='Home'  
INSERT OVERWRITE TABLE views_stg PARTITION(dt='2015-12-19',ostype='linux',page='Cart')  
  SELECT src.eventTime, src.userid, WHERE dt='2015-12-19' AND ostype='linux',page='Cart'  
INSERT OVERWRITE TABLE views_stg PARTITION(dt='2015-12-20',ostype='linux',page='Checkout')  
  SELECT src.eventTime, src.userid, WHERE dt='2015-12-20' AND ostype='linux',page='Checkout'
```

```
FROM page_views src  
INSERT OVERWRITE TABLE views_stg PARTITION(ostype='linux',dt,page)  
SELECT src.eventTime, src.userid, src.dt, src.page WHERE ostype='linux'
```

- Basic commands using HiveQL

Dynamic Partition Inserts

Default maximum dynamic partitions = 1000

- `hive.exec.max.dynamic.partitions`
- `hive.exec.max.dynamic.partitions.pernode`

Enable/Disable dynamic partition inserts

- `hive.exec.max.dynamic.partition=true`

Use strict mode when in doubt

- `hive.exec.max.dynamic.partition.mode=strict`

Increase max number of files a data node can service in (hdfs-site.xml)

- `dfs.datanode.max.xcievers=4096`

Demo



1. Hive Query

Thank you.

More security, More freedom