Introduction to Big Data Science

04th Period

HDFS, NoSQL, Map-Reduce Programming

Contents

- What is Big Data?
- Big Data Source
- Distributed File System
- Hadoop Distributed File System (HDFS)
- SQL and NoSQL
- What is Map-Reduce Operation?

Overview of Big Data Science

- What is Big Data?
  - Recently, there have been very large and complex data sets from nature, sensors, social networks, enterprises increasingly based on high speed computers and networks together.
  - Big data is the term for a collection of the data sets that it becomes difficult to process using on-hand database management tools or traditional data processing applications.

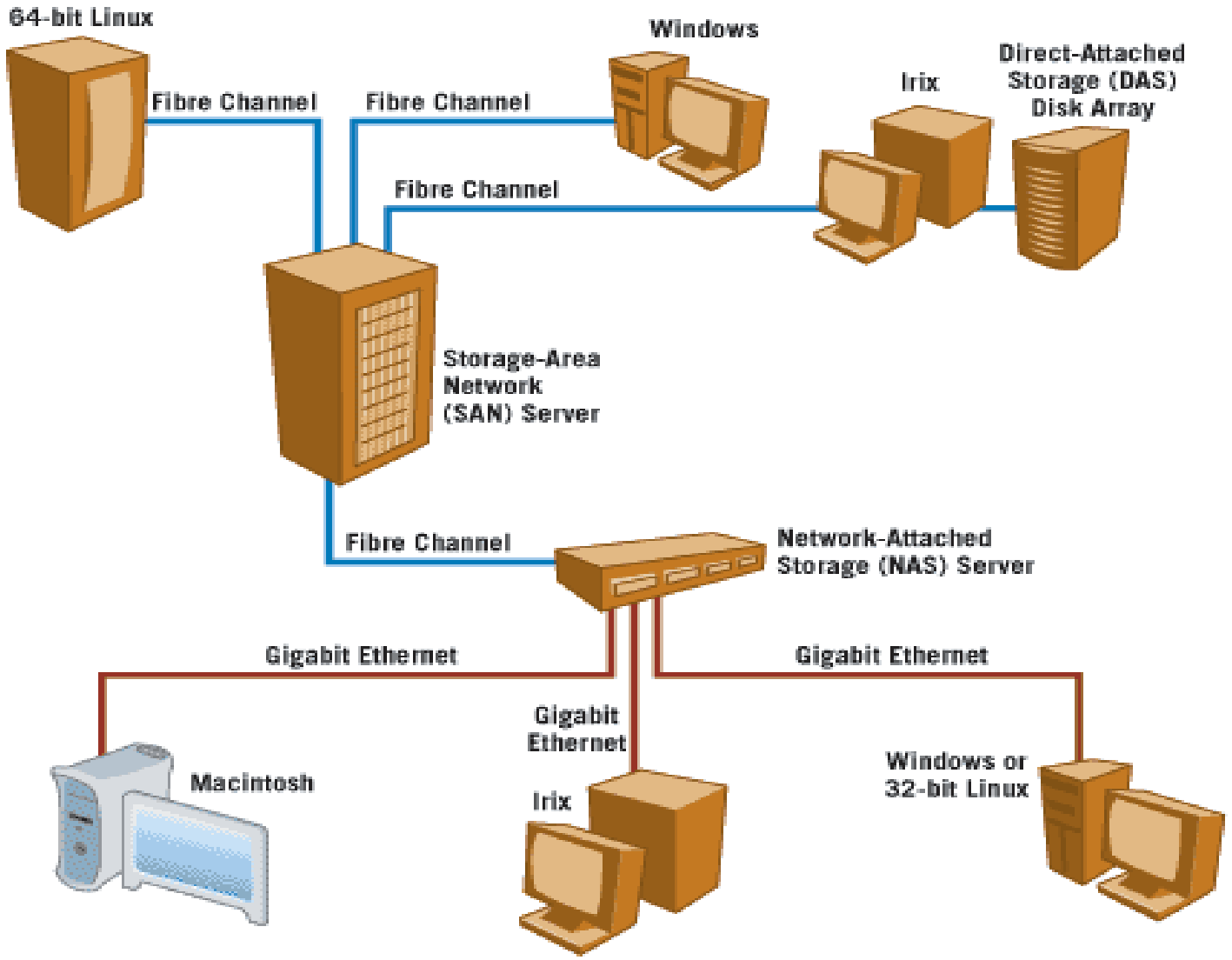
Overview of Big Data Science

- Several Areas
  - Meteorology
  - Genomics
  - Connectomics
  - Complex physics simulations
  - Biological and environmental research
  - Internet search
  - Finance and business informatics
  - Data sets ubiquitous information-sensing mobile devices
  - Aerial sensory technologies (remote sensing)
  - Software logs
  - Cameras
  - Microphones
  - Radio-frequency identification readers
  - Wireless sensor networks

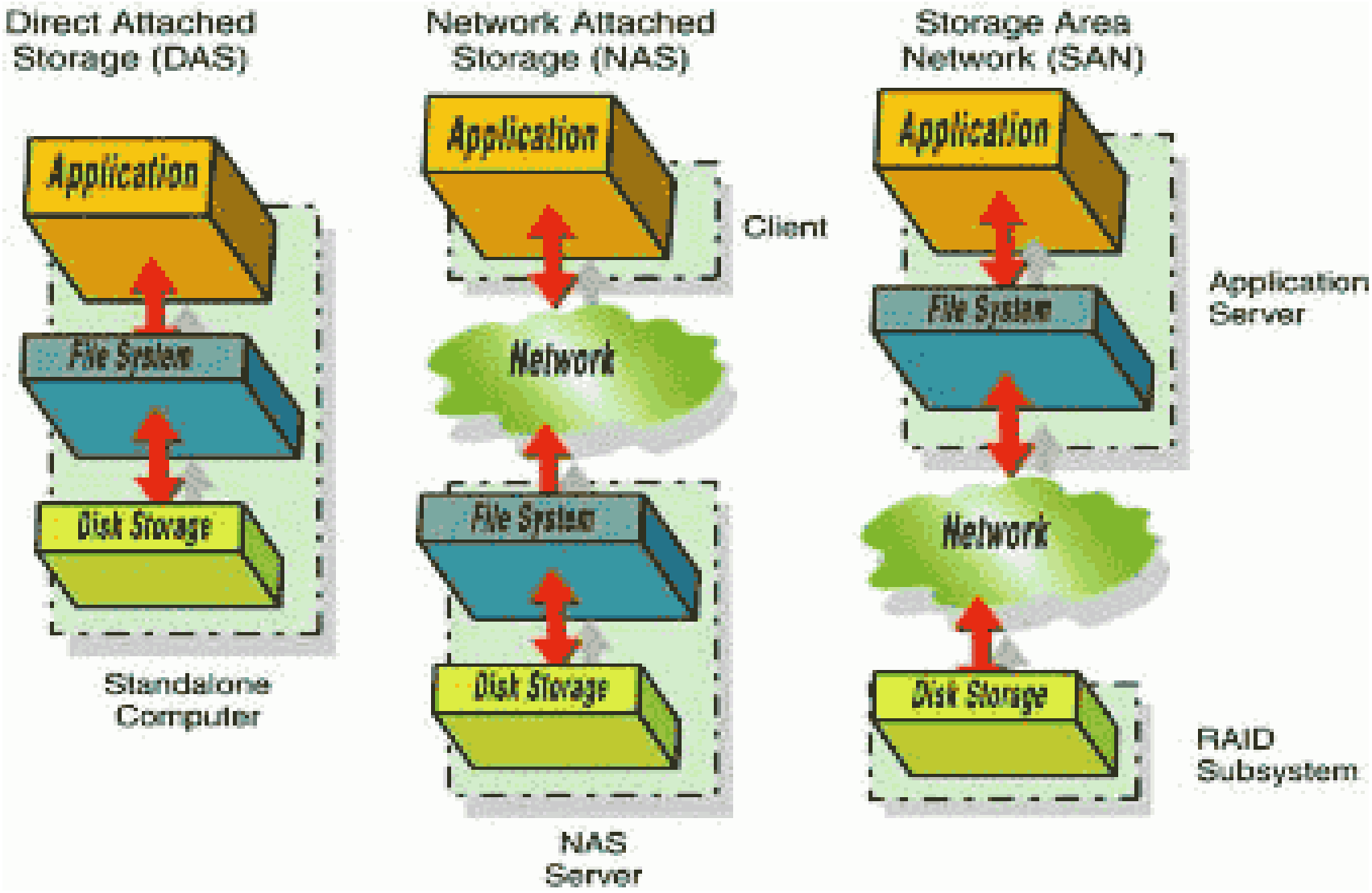
Clustered File System

- A file system which is shared by being simultaneously mounted on multiple servers.
- Clustered file systems can provide features like
  - Location-independent addressing
  - Redundancy which improve reliability or reduce the complexity of the other parts of the cluster.
- Parallel file systems are a type of clustered file system that spread data across multiple storage nodes, usually for redundancy or performance.

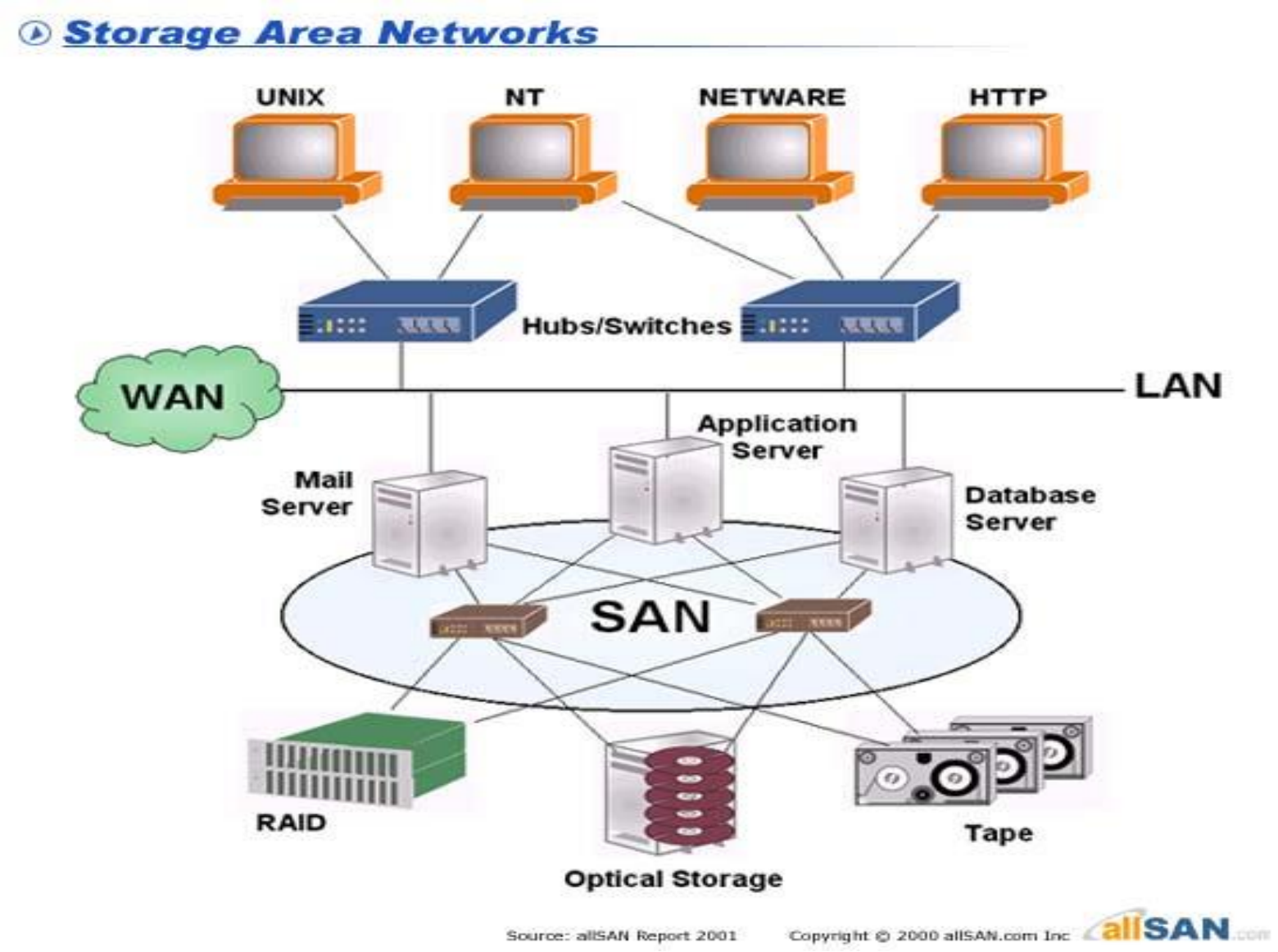
Clustered File System (DAS/NAS/SAN)



Fundamental Differences in DAS/NAS/SAN



Storage Area Network



Shared-Disk / Storage Area Network

- A shared-disk file-system uses a storage-area network (SAN) to provide direct disk access from multiple computers at the block level.
- Access control and translation from file-level operations (application) to block-level operations (by SAN) must take place on the client node.
- Adds a mechanism for concurrency control which gives a consistent and serializable view of the file system, avoiding corruption and unintended data loss
- Usually employ some sort of a fencing mechanism to prevent data corruption in case of node failures.

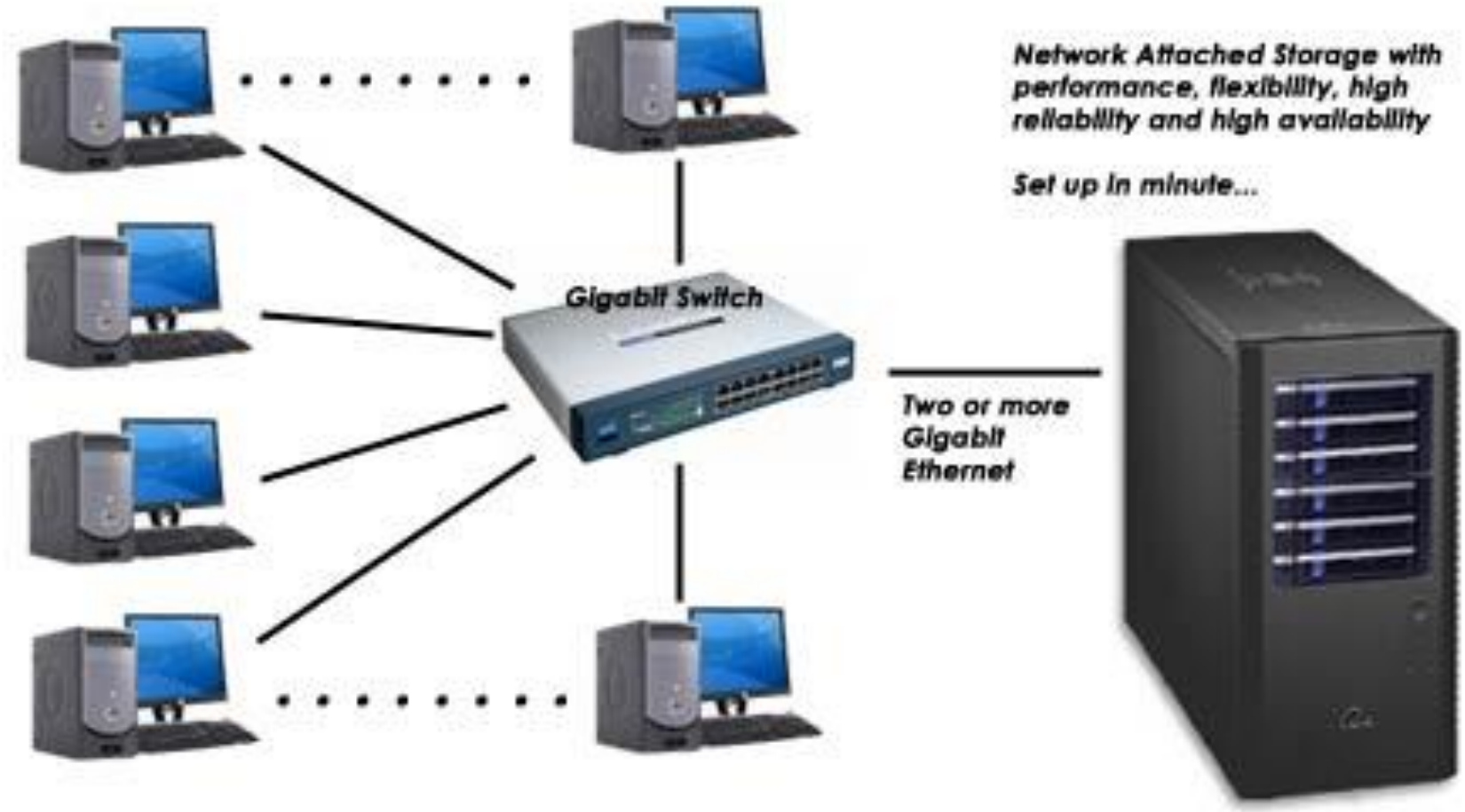
Shared-Disk / Storage Area Network

- The SAN may use any of a number of block-level protocols, including SCSI, iSCSI, HyperSCSI, ATA over Ethernet, Fibre Channel, etc.
- There are different architectural approaches to a shared-disk file-system.
  - Distribute file information across all the servers in a cluster (fully distributed).
  - Utilize a centralized metadata server.
  - Both achieve the same result of enabling all servers to access all the data on a shared storage device.

Shared-Disk / Storage Area Network

- Examples
  - Silicon Graphics (SGI) clustered file system (CXFS)
  - Veritas Cluster File System
  - DataPlow Nasan File System
  - DataPlow SAN File System (SFS)
  - IBM General Parallel File System (GPFS)
  - Microsoft Cluster Shared Volumes (CSV)
  - Oracle Cluster File System (OCFS)

Network Attached Storage (NAS)



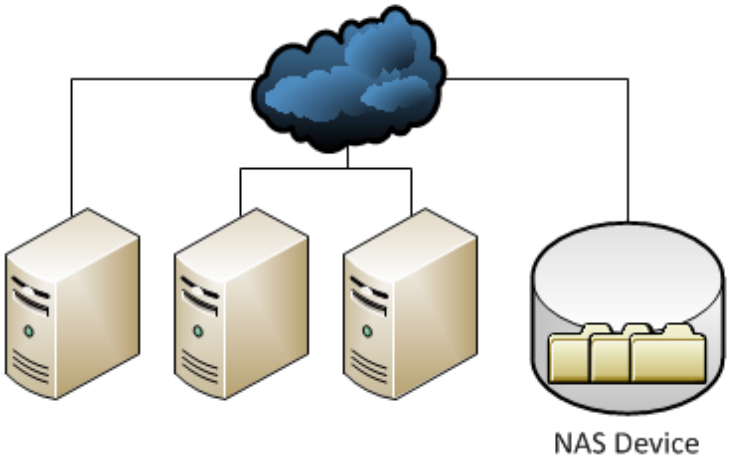
Network Attached Storage (NAS)

- Provides both storage and a file system, like a shared disk file system on top of a SAN.
- Typically uses file-based protocols (as opposed to block-based protocols a SAN would use) such as NFS, SMB/CIFS, AFP, or NCP.
- Design Considerations
  - Avoiding single point of failure: Fault tolerance and high availability by data replication of one sort or another
  - Performance: fast disk-access time and small amount of CPU-processing time over distributed structure
  - Concurrency: for consistent and efficient multiple accesses to the same file or block by concurrency control or locking which may either be built into the file system or provided by an add-on protocol

NAS vs. SAN

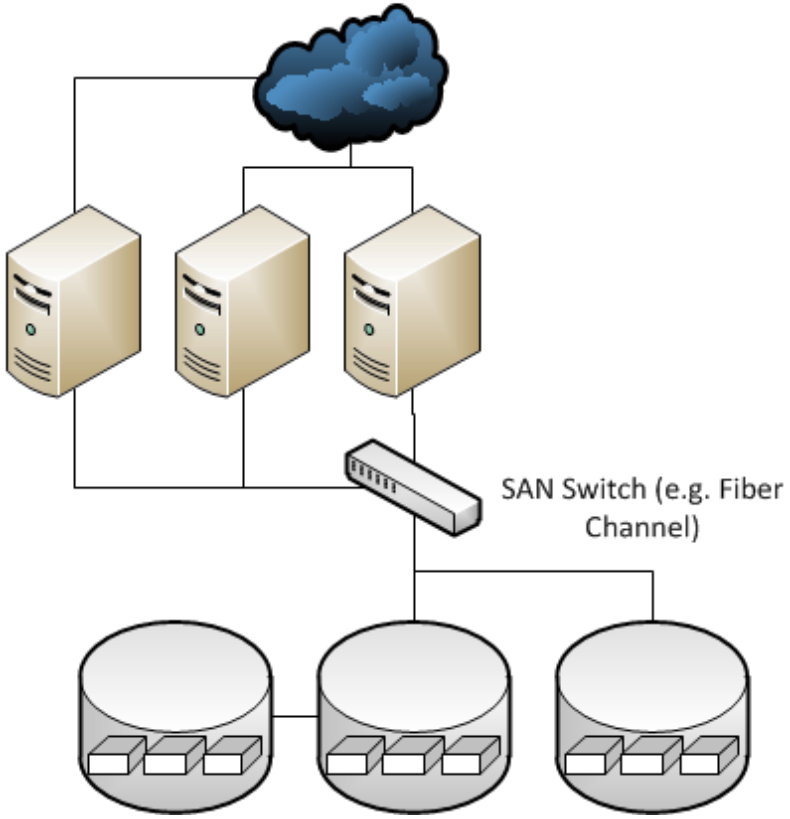
Network Attached Storage

- Shared storage over shared network
- File system
- Easier management



Storage Area Network

- Shared storage over dedicated network
- Raw storage
- Fast, but costly



Distributed File System

- Distributed file systems do not share block level access to the same storage but use a network protocol.
- These are commonly known as network file systems, even though they are not the only file systems that use the network to send data.
- Distributed file systems can restrict access to the file system depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed.

Distributed File System

- The difference between a distributed file system and a distributed data store
  - Distributed file system (DFS) allows files to be accessed using the same interfaces and semantics as local files - e.g. mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model.
  - Distributed data stores, by contrast, require using a different API or library and have different semantics (most often those of a database).

Distributed File System

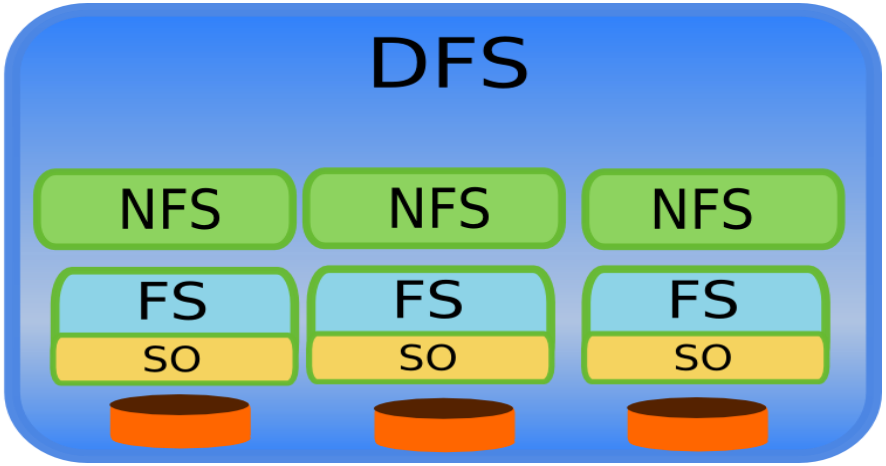
- Design Goal of DFS
  - **Access transparency**: Clients are unaware that files are distributed and can access them in the same way as local files are accessed.
  - **Location transparency**: A consistent name space exists encompassing local as well as remote files. The name of a file does not give its location.
  - **Concurrency transparency**: All clients have the same view of the state of the file system. This means that if one process is modifying a file, any other processes on the same system or remote systems that are accessing the files will see the modifications in a coherent manner.

Distributed File System

- Design Goal of DFS
  - **Failure transparency**: The client and client programs should operate correctly after a server failure.
  - **Heterogeneity**: File service should be provided across different hardware and operating system platforms.
  - **Scalability**: The file system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to huge ones (hundreds through tens of thousands of systems).
  - **Replication transparency**: To support scalability, we may wish to replicate files across multiple servers. Clients should be unaware of this.
  - **Migration transparency**: Files should be able to move around without the client's knowledge.

Distributed File System

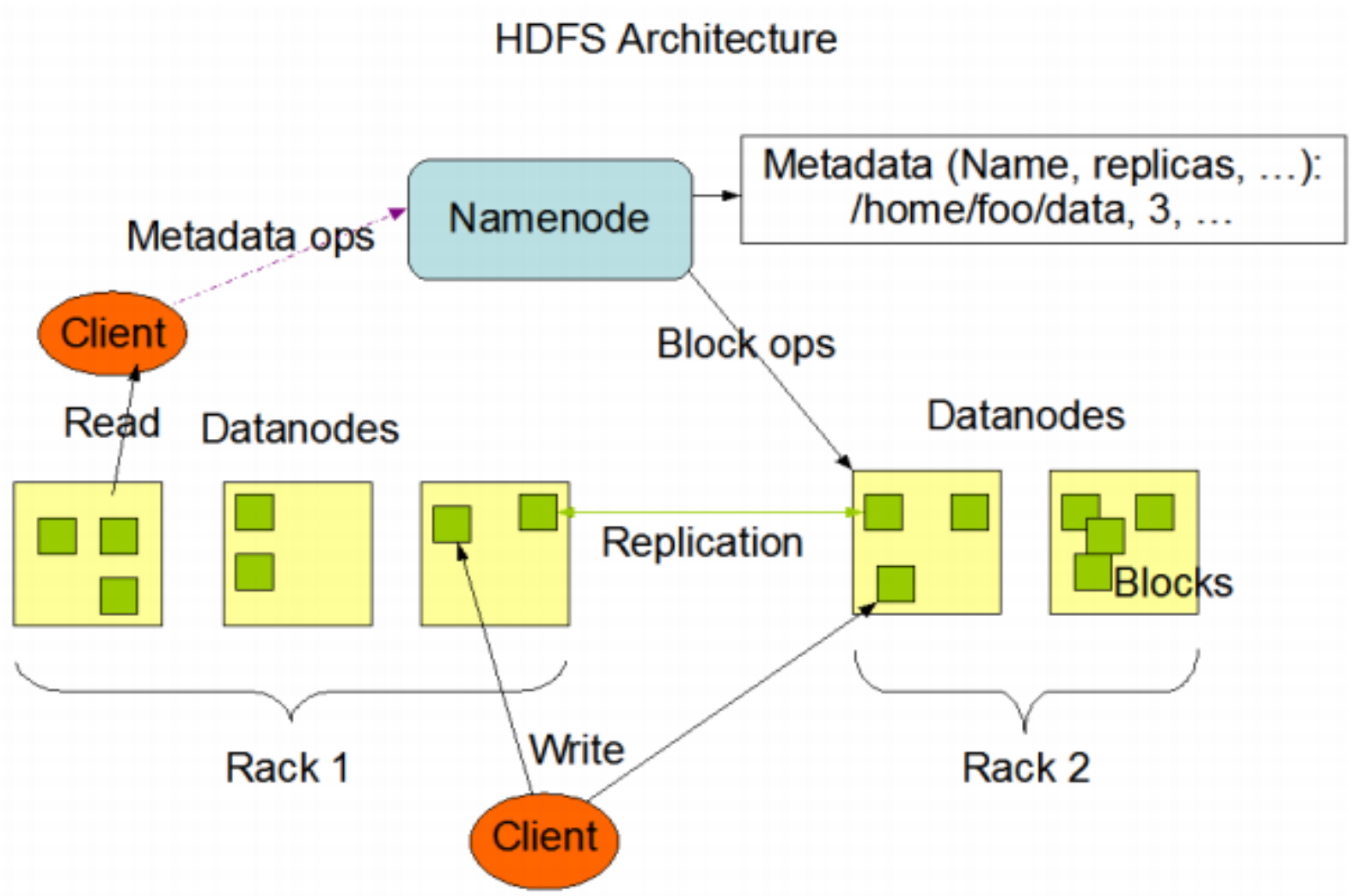
- Examples
  - GFS (Google Inc.)
  - Ceph (Inktank)
  - Windows Distributed File System (DFS) (Microsoft)
  - FhGFS (Fraunhofer)
  - GlusterFS (Red Hat)
  - Lustre
  - Ibrix



Hadoop Distributed File System (HDFS)

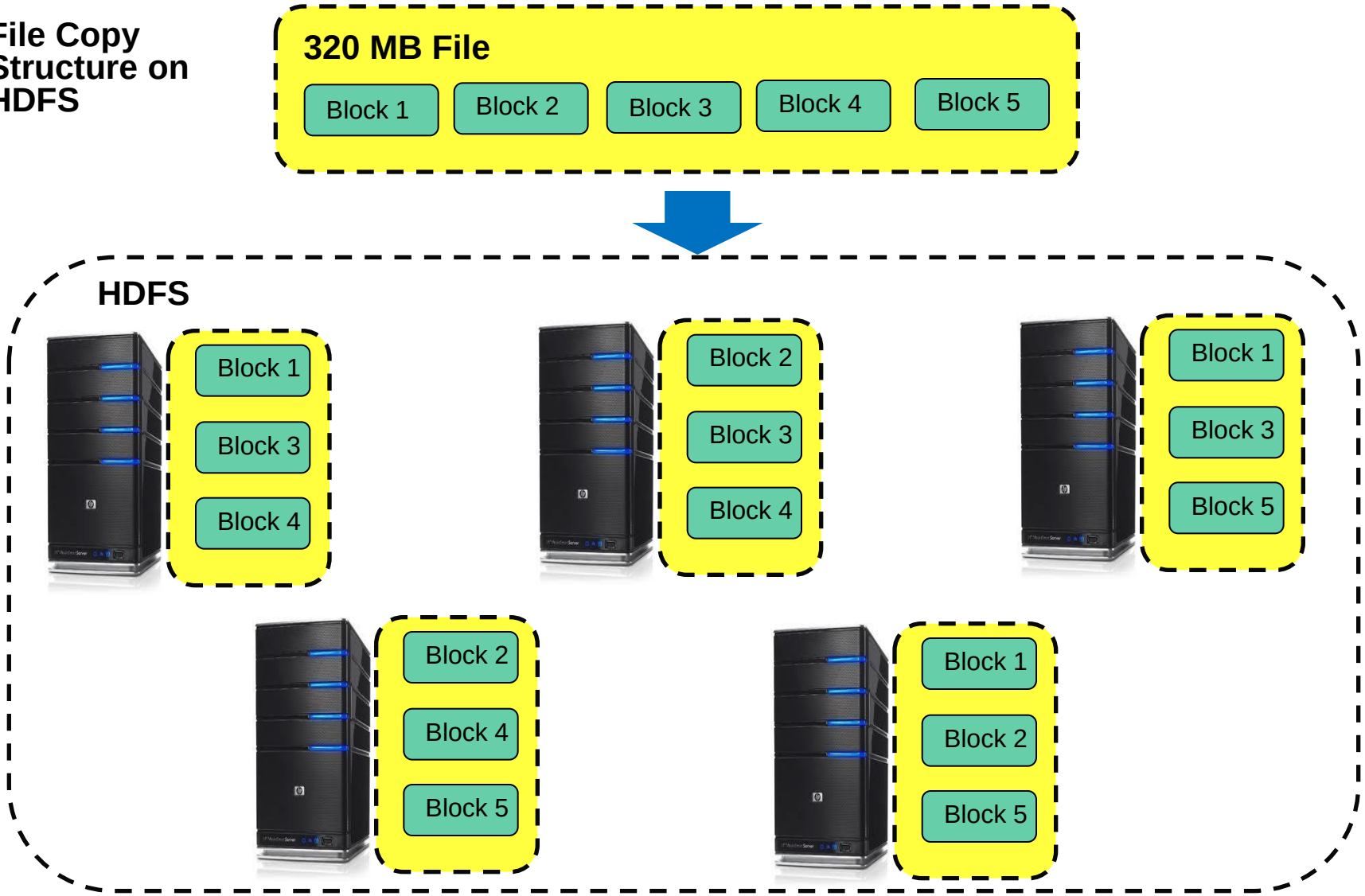
- The existing file system such as large file storage , NAS or SAN is expensive. It needs very high performance server.
- HDFS can combine Web server level hosts into a disk storage.
- Some high performance large storage system will be needed. SAN is suitable for DBMS, NAS for safe file store.
- Four Objectives of HDFS
  - Error recovery
  - Access data using streaming
  - Large data storage
  - Data integrity

Hadoop Distributed File System

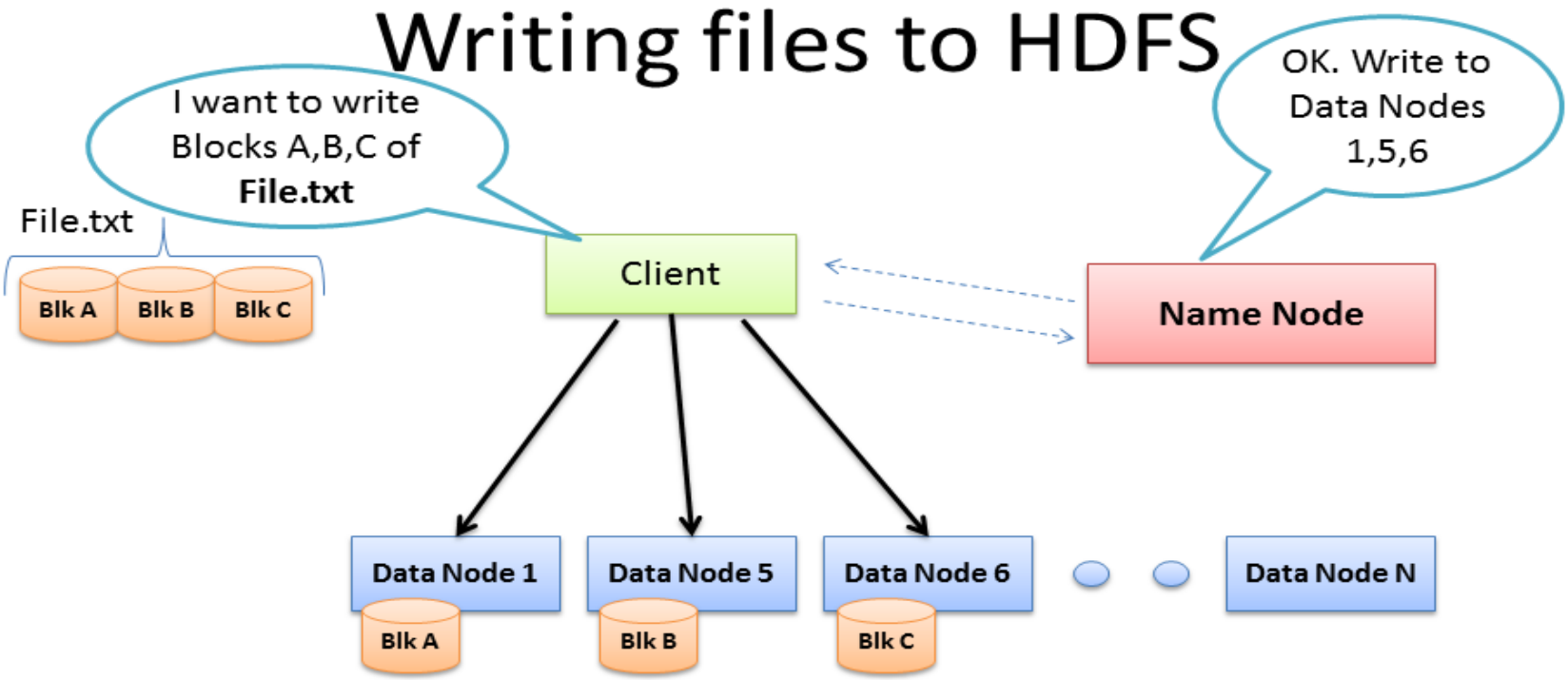


Block Structured File System

File Copy Structure on HDFS



Block Structured File System

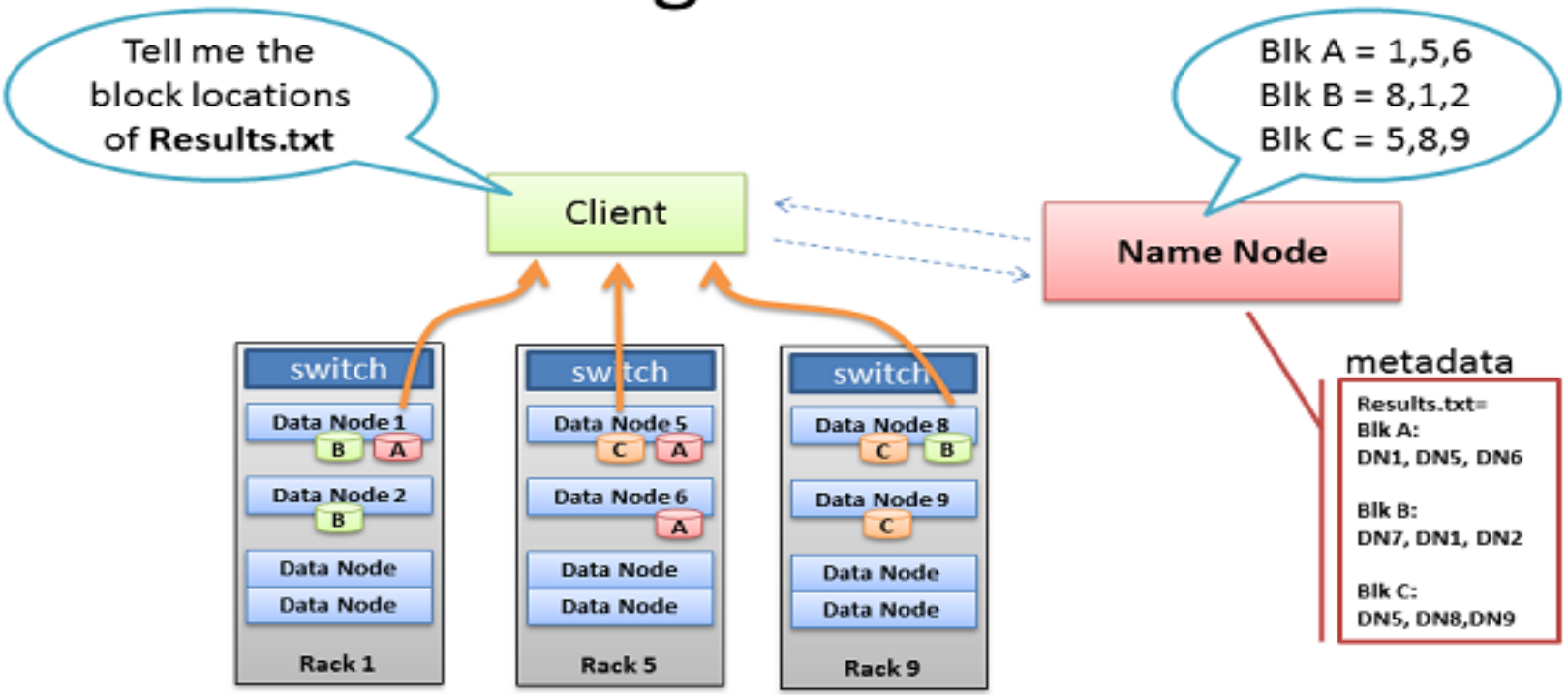


- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

BRAD HEDLUND .com

Block Structured File System

Client reading files from HDFS



- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

BRAD HEDLUND .com

HDFS,NoSQL, Map-Reduce

- NoSQL
  - No SQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational databases.
  - Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability.
  - NoSQL databases are often highly optimized key–value stores intended for simple retrieval and appending operations, with the goal being significant performance benefits in terms of latency and throughput.

Map Operation

- What is a Map operation
  - Doing something to every element in an array is a common operation. The operation can be considered as Map function.

```
int a = {1,2,3};
for (I = 0; I < a.length; i++)
    a[i] = a[i] * 2;
```

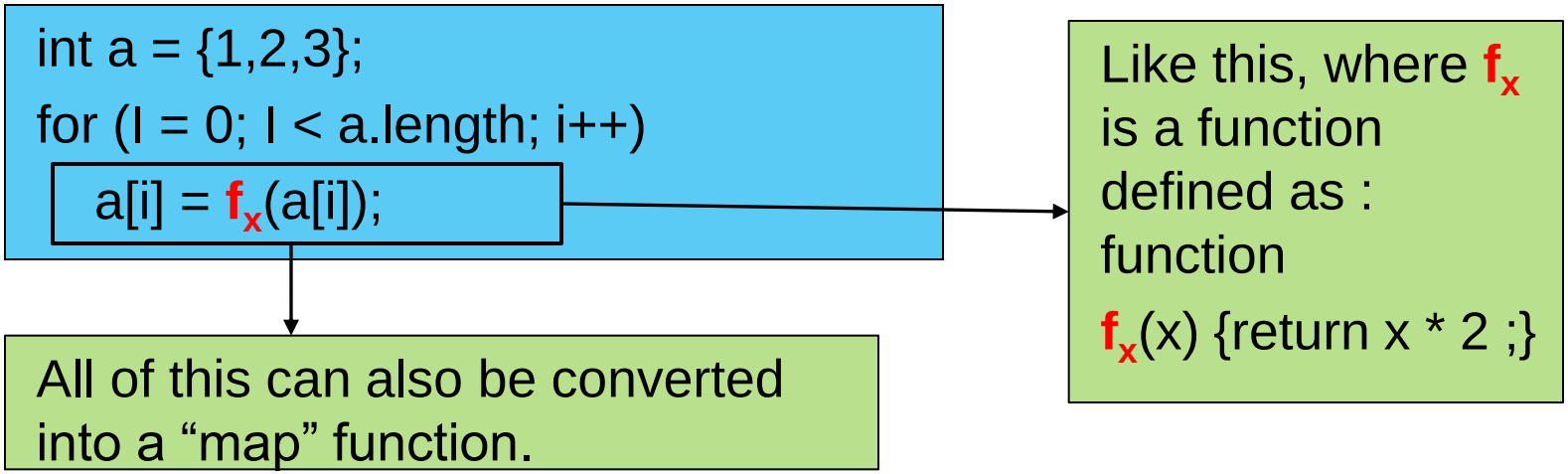
This can be written as a function.

  - New value for the variable **a** would be

```
int a = {2,4,6};
```

Map Operation

- What is a Map operation
  - Doing something to every element in an array is a common operation. The operation can be considered as Map function.



  - New value for the variable **a** would be

```
int a = {2,4,6};
```

Map Operation

- What is a Map operation
  - …like this, where f_x is a function passed as an argument:

```
function map (fx , a) {
  for (I = 0; I < a.length; i++)
      a[i] = fx(a[i]);
}
```

  - You can invoke this map function as follows

```
map(function(x) {return x*2; }, a);
```

This is function f_x whose defintion is included in the call.

Reduce Operation

- What is a Reduce operation
  - Another common operation on arrays is to combine all their value:

```
function sum(a) {
  int s = 0;
  for (i = 0; I < a.length; i++)
    s += a[i] ;
  return s;
}
```

This can be written as a function.

Reduce Operation

- What is a Reduce operation
  - Another common operation on arrays is to combine all their values:

```
function sum(a) {
  int s = 0;
  for (i = 0; I < a.length; i++)
    s = fx (s, a[i]) ;
  return s;
}
```

Like this, where a function f_x is defined so that it adds its arguments: function

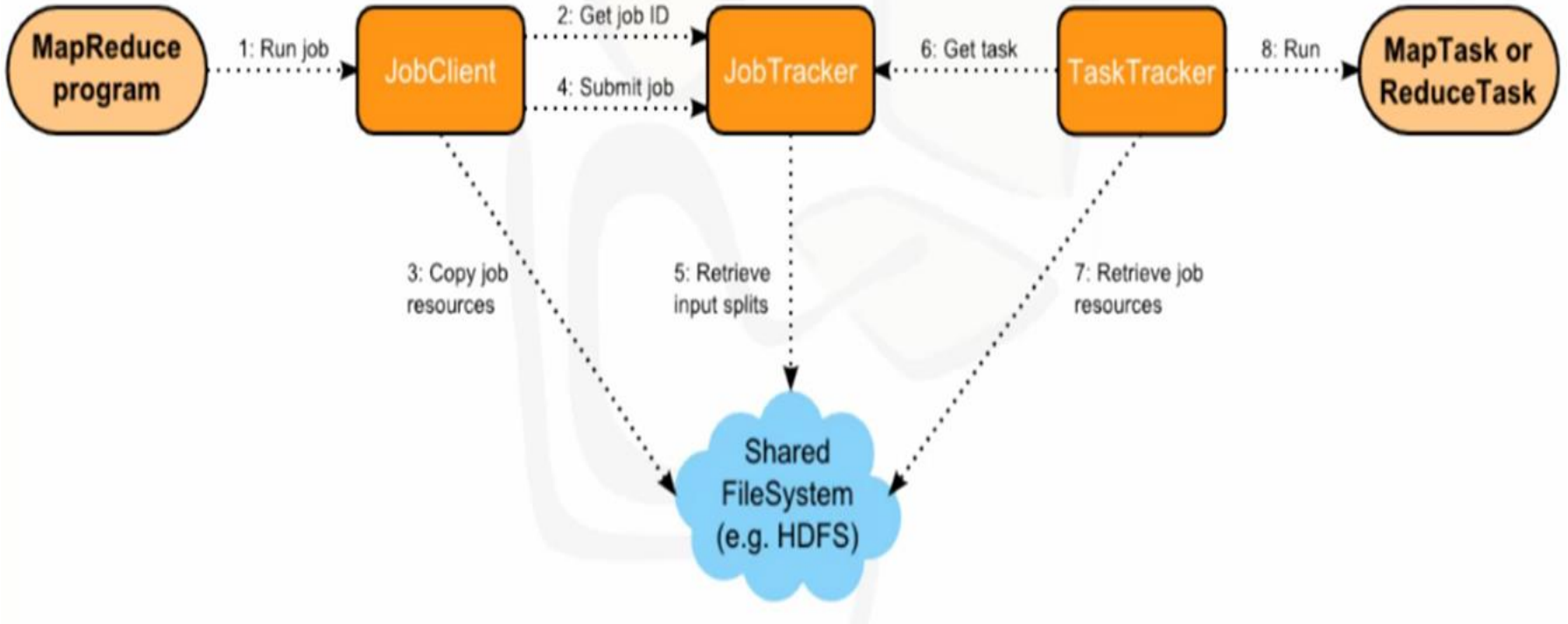
f_x(a,b) {return a+b ;}

The whole function ***sum*** can also be rewritten so that f_x is passed as an argument.

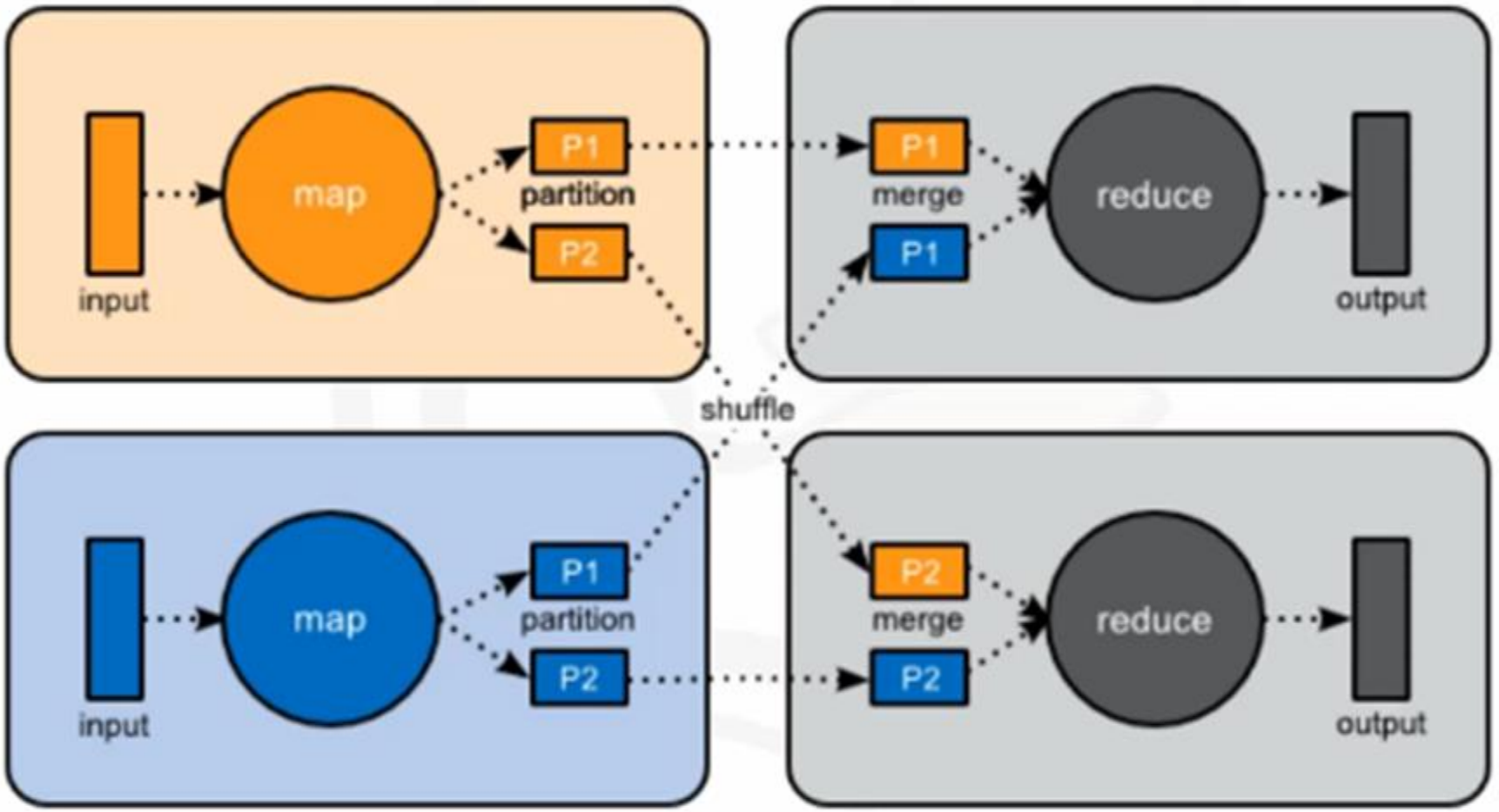
as a reduce operation:

```
reduce(function(a,b) {return a+b; }, a, 0);
```

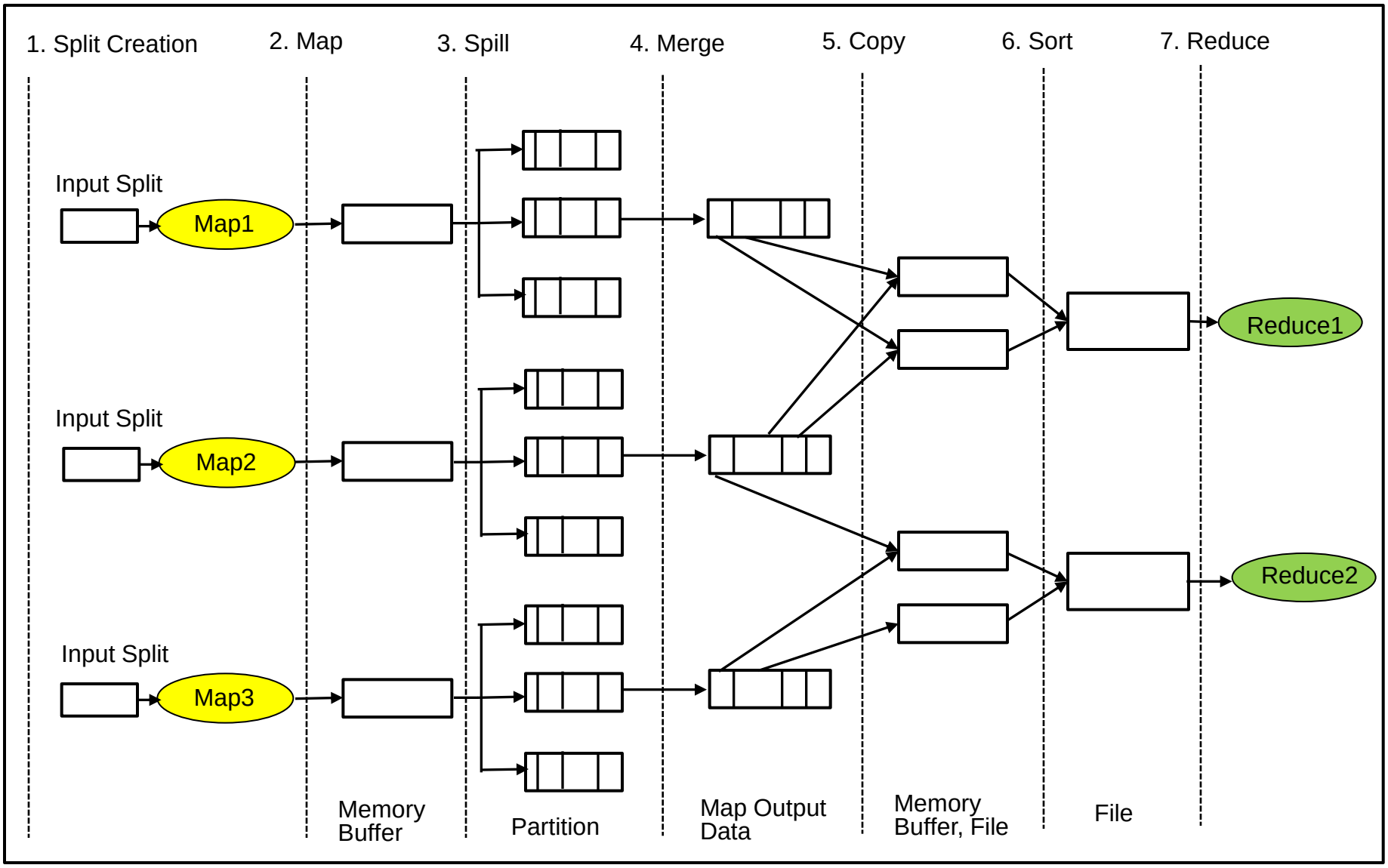
Submitting a MapReduce Job



MapReduce – The Shuffle



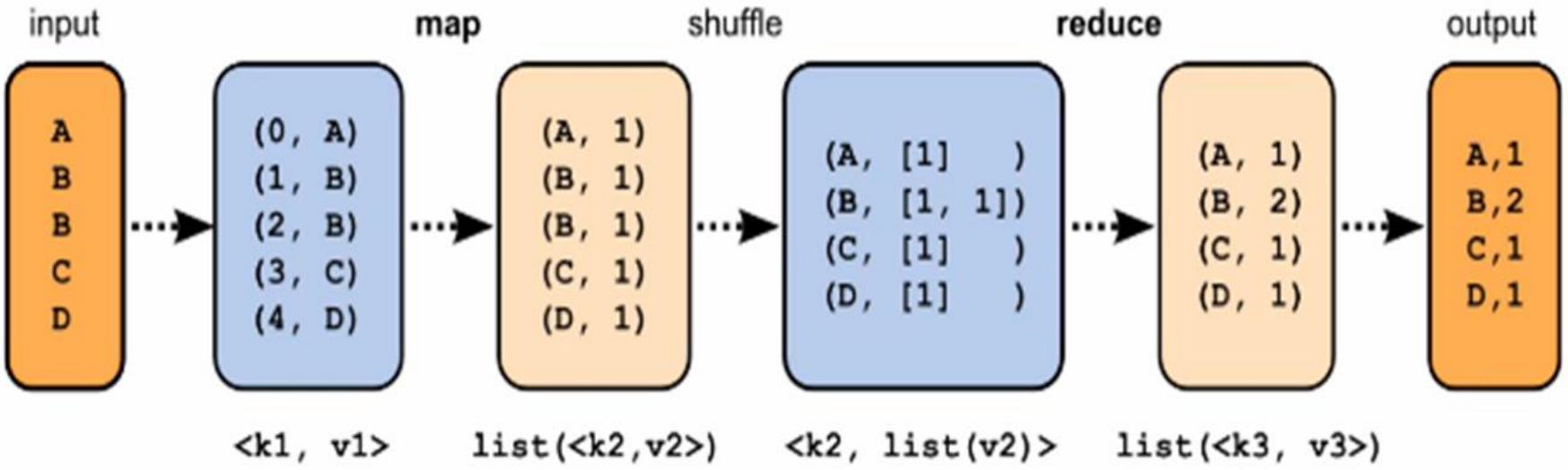
Shuffle Process



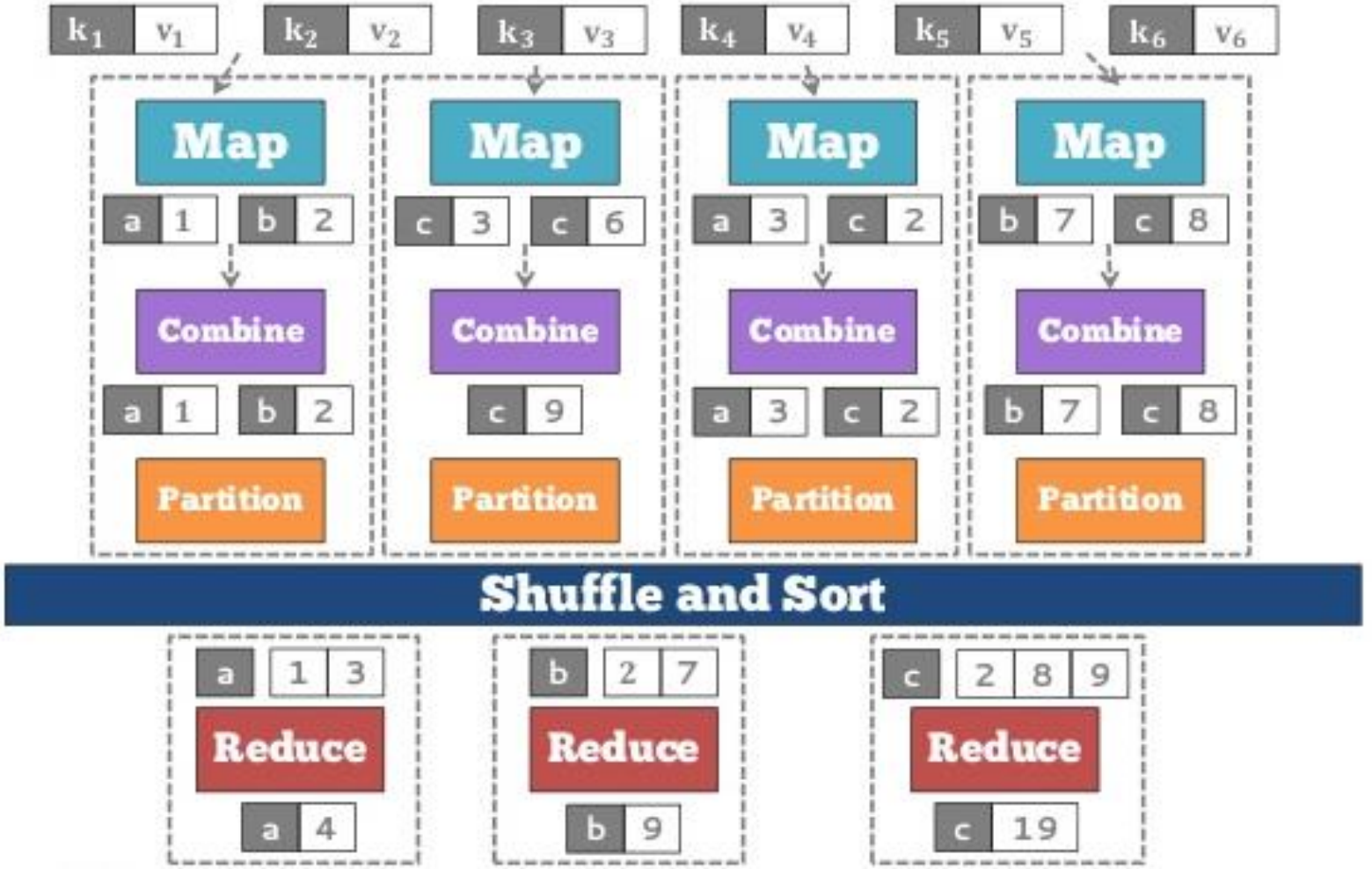
MapReduce – Type and Data Flow

- Lists

	Input	Output
map	<k1, v1>	list(<k2, v2>)
reduce	<k2, list(v2)>	list(<k3, v3>)



Flow of Map and Reduce



HADOOP MapReduce Program : WordCount.java

```
package net.kzk9;

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

import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {
  // Mapperの実装
  public static class Map extends
    Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    @Override
      protected void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        context.write(word, one);
      }
    }
  }
```

```
  // Reducerの実装
  public static class Reduce extends
    Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable value = new IntWritable(0);

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

  public static void main(String[] args) throws Exception {
    // 設定情報の読み込み
    Configuration conf = new Configuration();
    // 引数のパース
    GenericOptionsParser parser = new GenericOptionsParser(conf,
args);
    args = parser.getRemainingArgs();

    // ジョブの作成
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
    // Mapper/Reducerに使用するクラスを指定
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    // ジョブ中の各種型を設定
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
```

HADOOP MapReduce Program：WordCount.java

```
// 入力 /出力パスを設定
    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // ジョブを JobTrackerにサブミット
    boolean success = job.waitForCompletion(true);
    System.out.println(success);
  }
}
```

```
TestWordCount.java
package net.kzk9;
import net.kzk9.WordCount;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import junit.framework.TestCase;
import org.junit.Before;
import org.junit.Test;

public class WordCountTest extends TestCase {
  // テストする対象の Mapper
  private WordCount.Map mapper;

  // Mapper実行用のドライバクラス
  private MapDriver driver;

  @Before
  public void setUp() {
    // テスト対象の Mapperの作成
    mapper = new WordCount.Map();
    // テスト実行用 MapDriverの作成
    driver = new MapDriver(mapper);
  }
```

```
  @Test
  public void testWordCountMapper() {
    // 入力を設定
    driver.withInput(new LongWritable(0), new Text("this is a pen"))
      // 期待される出力を設定
      .withOutput(new Text("this"), new IntWritable(1))
      .withOutput(new Text("is"), new IntWritable(1))
      .withOutput(new Text("a"), new IntWritable(1))
      .withOutput(new Text("pen"), new IntWritable(1))
      // テストを実行
      .runTest();
  }
}
```

```
#!/bin/bash
export HADOOP_HOME=/usr/lib/hadoop-0.20/
export DIR=wordcount_classes

# CLASSPATHの設定
CLASSPATH=$HADOOP_HOME/hadoop-core.jar
for f in $HADOOP_HOME/lib/*.jar; do
CLASSPATH=${CLASSPATH}:$f;
done

# ディレクトリの初期化
rm -fR $DIR
mkdir $DIR

# コンパイル
javac -classpath $CLASSPATH -d $DIR WordCount.java

# jarファイルの作成
jar -cvf wordcount.jar -C $DIR .
```