CLOUD COMPUTING
Cloud Applications

Zeinab Zali
Isfahan University Of Technology

Cloud Applications

- Cloud application developers are free to design an application without being concerned where the application will run.
- When the workload can be partitioned in n segments, the application can spawn n instances of itself and run them concurrently resulting in dramatic speedups.
- Web services, database services, and transaction-based services are ideal applications for cloud computing

Not Suitable applications for cloud computing

- Applications where the workload cannot be arbitrarily partitioned
- Applications that require intensive communication among concurrent instances
- An application with a complex workflow and multiple dependencies

Cloud application development challenges

- Most of the challenges posed by the inherent imbalance between computing, I/O, and communication bandwidth of processors
- cloud computing infrastructures attempt to automatically distribute and balance the workload
- application developers have the responsibility to
  - identify optimal storage for the data
  - exploit spatial and temporal data and code locality
  - minimize communication among running treads and instances

Cloud application development challenges

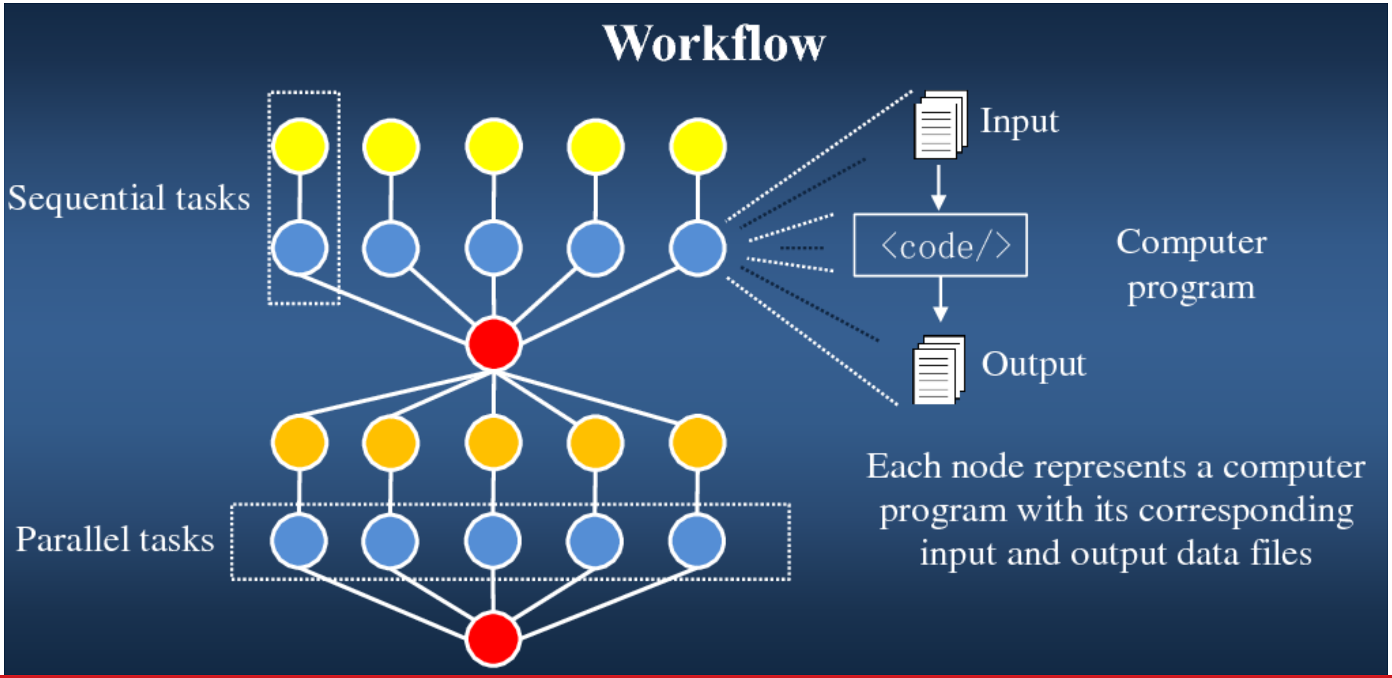
- Security isolation, performance isolation and reliability of instances
- efficiency, consistency, and communication scalability
- The organization and the location of data storage, as well as the storage bandwidth
- The ability to identify the source of unexpected results and errors is helped by frequent logging, but performance considerations limit the amount of data logging.

Cloud application architectural styles

- The vast majority of cloud applications take advantage of request-response communication between clients and stateless servers.
- A stateless server does not require a client to first establish a connection to the server, instead it views a client request as an independent transaction and responds to it.
  - Benefit: Recovering from a server failure requires a considerable overhead for a server which maintains the state of all its connections

Workflow

- The description of a complex activity involving an ensemble of tasks is known as a workflow.
  - Task is the central concept in workflow modeling
  - A task is a unit of work to be performed on the cloud

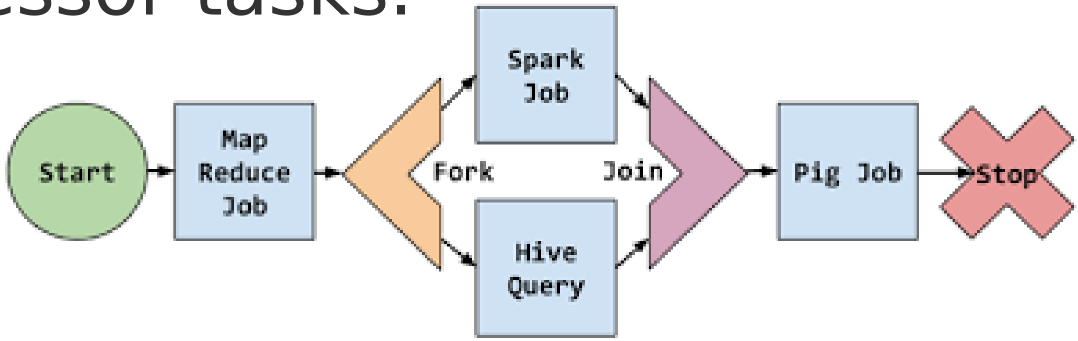


Task attributes

- Preconditions – boolean expressions that must be true before the task can take place.
- Postconditions – boolean expressions that must be true after the task do take place.
- Attributes – provide indications of the type and quantity of resources necessary for the execution of the task
- the security requirements
- Exceptions – provide information on how to handle abnormal events.

Task concepts

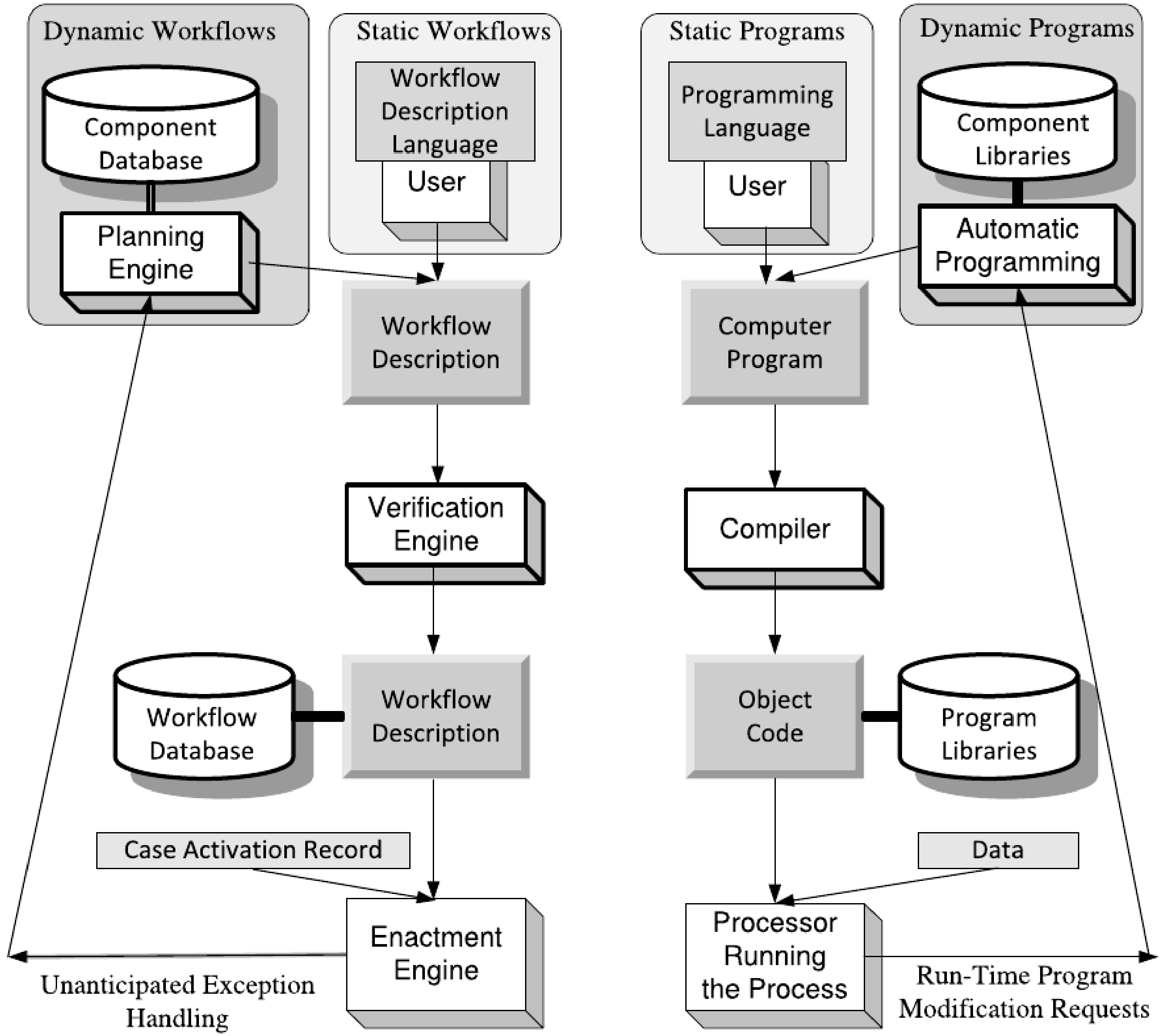
- The task that has just completed execution is called the predecessor task
- the one to be initiated next is called the successor task
- A fork routing task triggers execution of several successor tasks
- A join routing task waits for completion of its predecessor tasks.



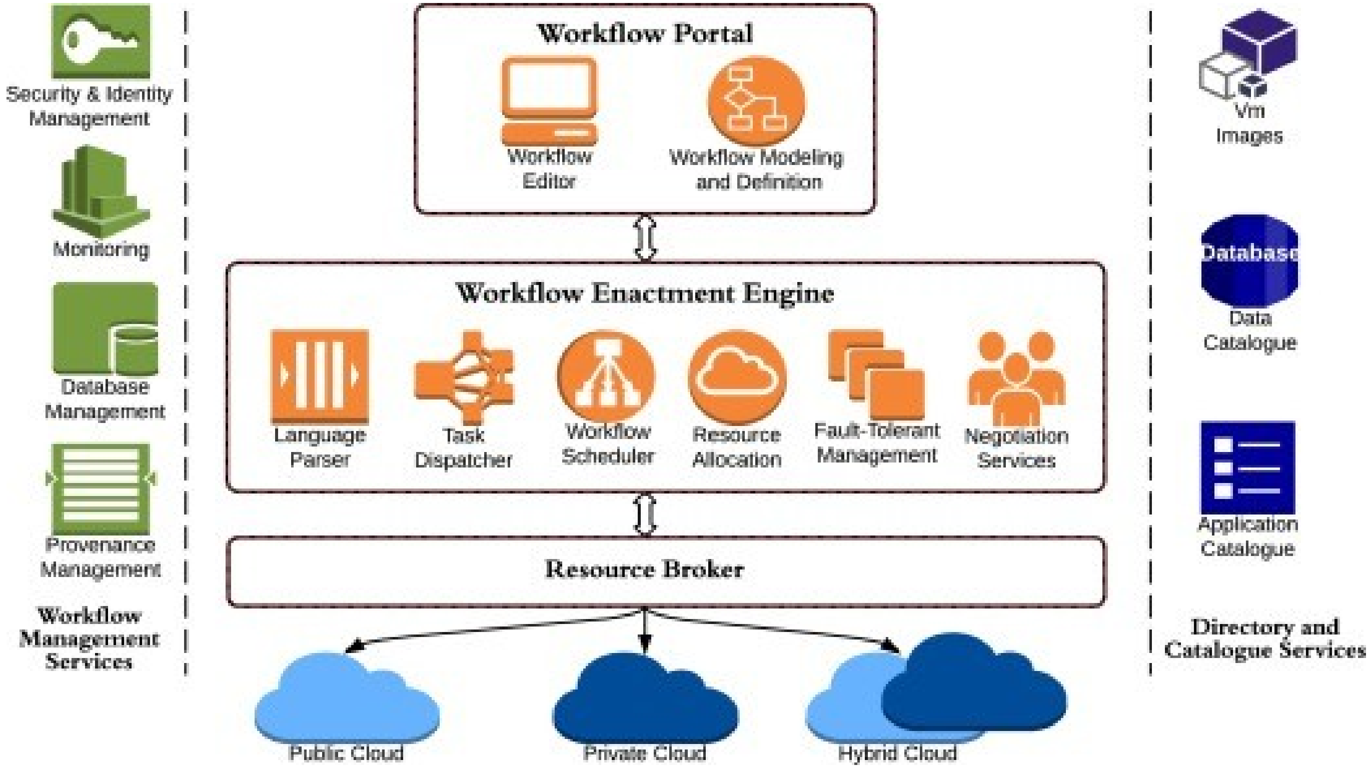
Workflow Definition Language (WFDL)

- A process description can be provided in a Workflow Definition Language (WFDL), supporting constructs for choice, concurrent execution, the classical fork, join constructs, and iterative execution.
  - The workflow specification by means of a workflow description language is analogous to writing a program
- Life-cycle of a traditional program
  - creation, compilation, and execution
- Life-cycle of a workflow:
  - creation, verification, and enactment

Workflow vs traditional programming

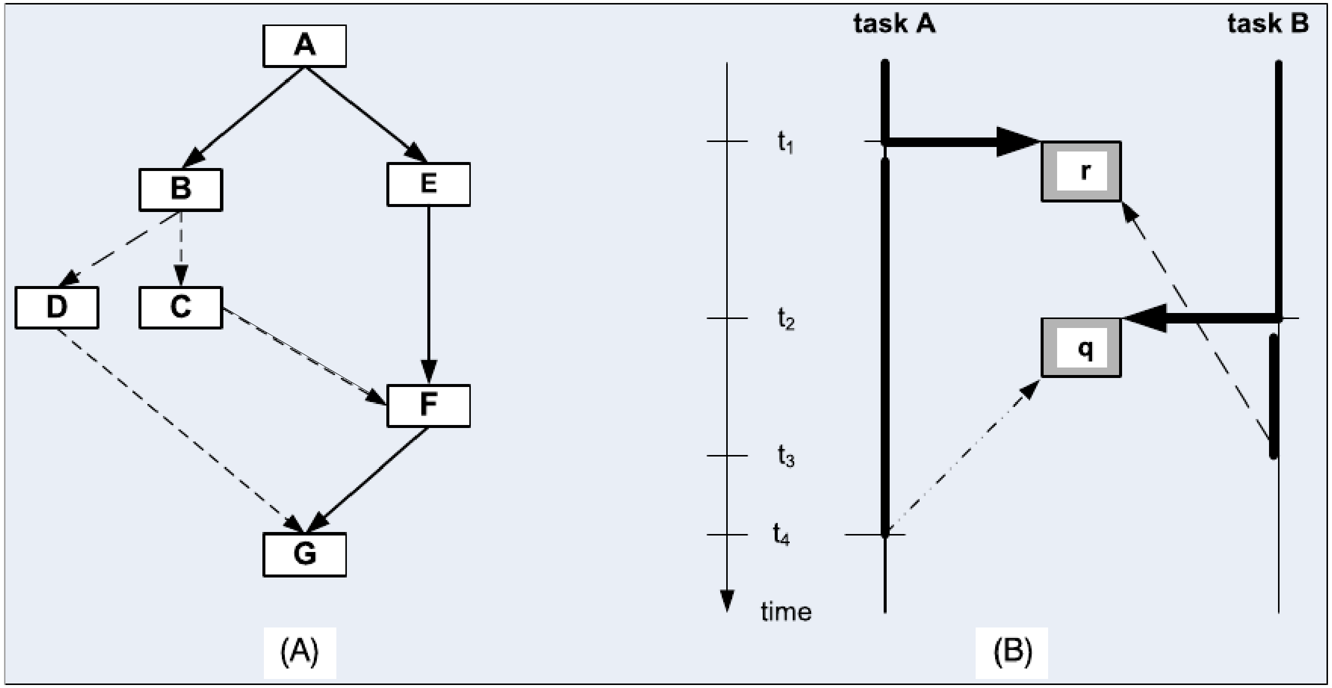


Workflow Enactment Engine



Safety and liveness

- safety means that nothing “bad” ever happens
- liveness means that something “good” will eventually take place

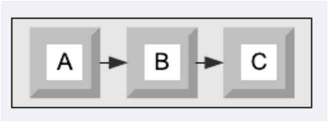


No liveness
(Dotted lines correspond to choices; either D or C are executed)

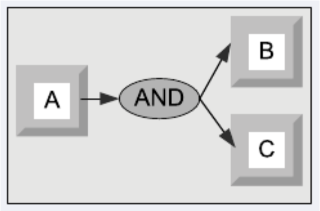
No safety

Workflow patterns

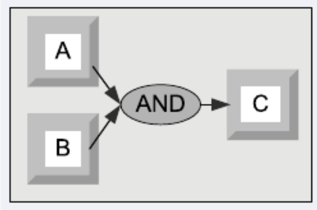
- The term workflow pattern refers to the temporal relationships among the tasks of a process
  - Sequence



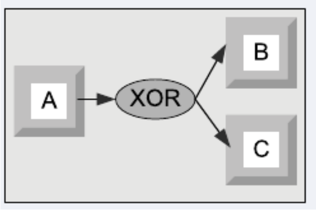
  - AND split



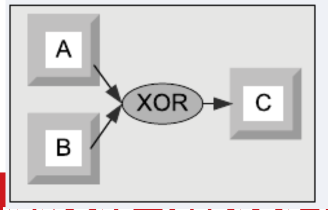
  - Synchronization



  - XOR split

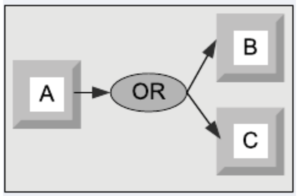


  - XOR Join

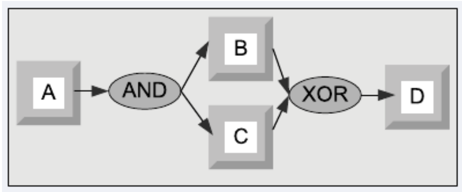


Workflow patterns

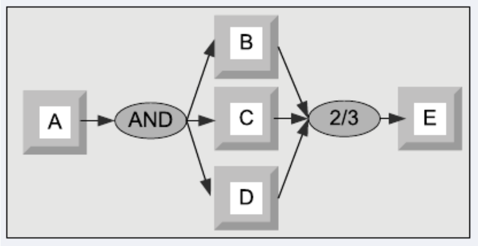
– OR split



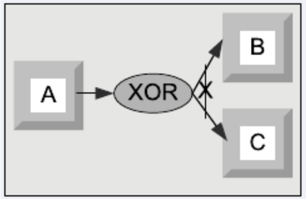
– Multiple merge



– N out of M join



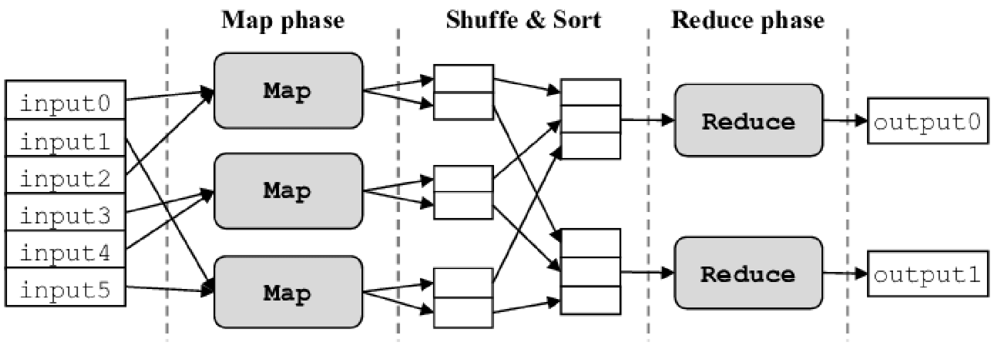
– deferred choice

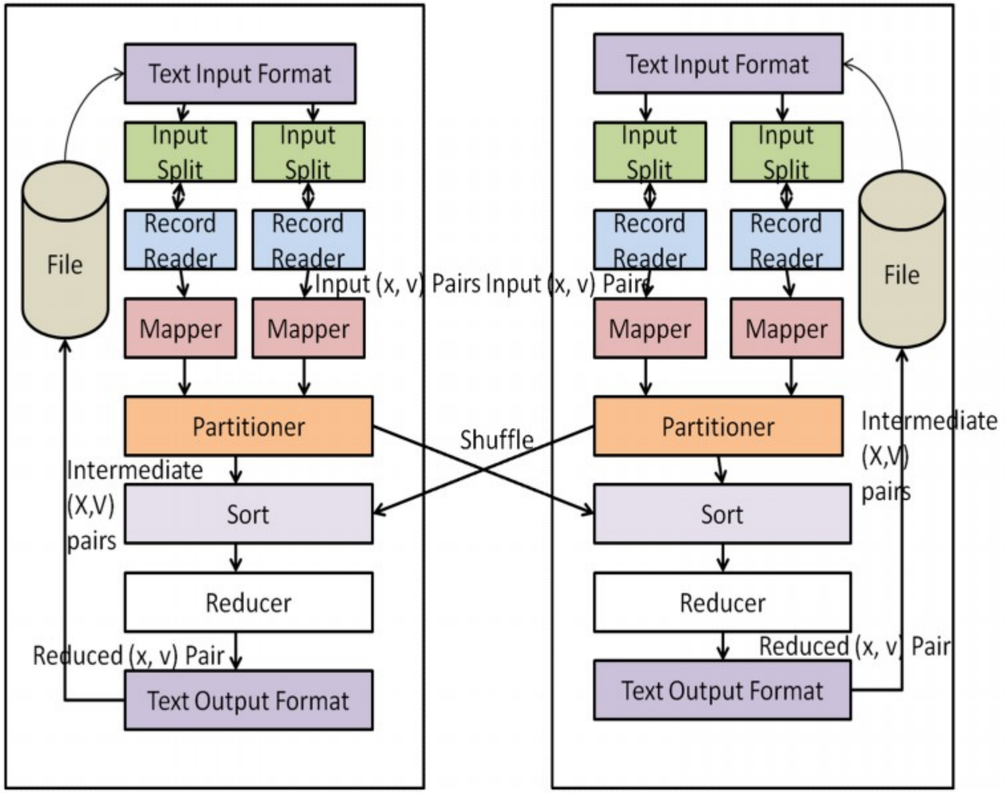


Characterizing a Multiprocess workflow

- A system Σ, an initial state of the system, $\sigma_{initial}$, and a goal state, σ_{goal}.
- A process group, $P = \{p_1 , p_2 , . . . , p_n \};$
  - each process p_i in the process group is characterized by a set of preconditions, $pre(p_i)$, postconditions, $post(p_i)$, and attributes, $atr(p_i)$.
- A workflow described by a directed activity graph A given the tuple $< P, \sigma_{initial} , \sigma_{goal} >$.
  - The nodes of A are processes in P and the edges define precedence relations among processes. $P_i \rightarrow P_j$ implies that $pre(p_j) \subset post (p_i)$.

MapReduce Workflow Example





Workflow types

- Static: the activity graph does not change during the enactment of a case
- Dynamic: the activity graph may be modified during the enactment of a case
- workflow enactment methods
  - Strong coordination models where the process group P executes under the supervision of a coordinator process.
    - Suitable for dynamic workflow
  - Weak coordination models where there is no supervisory process.
    - are based on peer-to-peer communication between processes in the process group

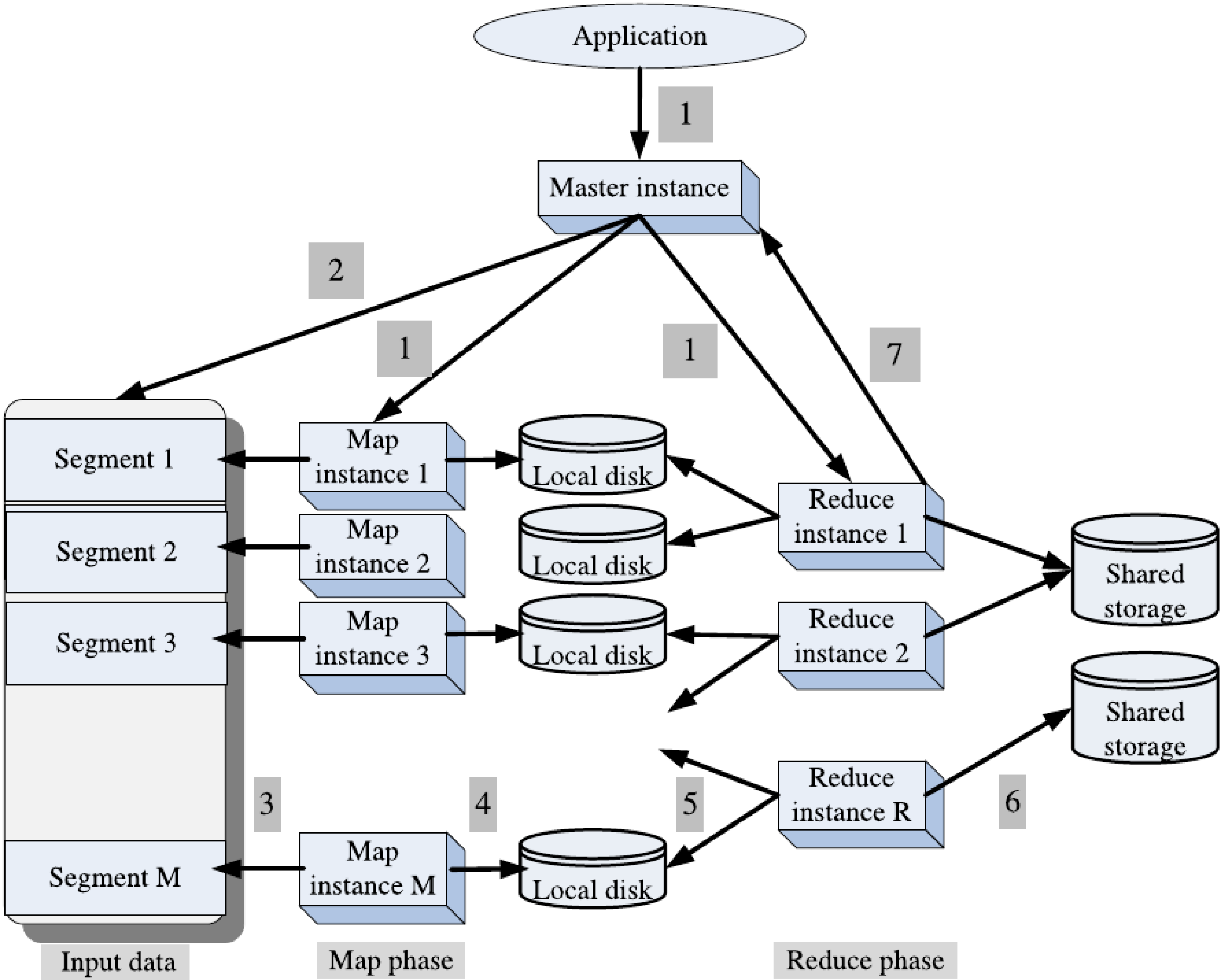
Hadoop-MapReduce

MapReduce Programming model

- MapReduce is based on a very simple idea for parallel processing of **data-intensive** applications supporting arbitrarily divisible load sharing
  - split the data into blocks
  - assign each block to an instance/process
  - run the instances in parallel
  - Once all the instances have finished the computations assigned to them, start the second phase and merge the partial results produced by individual instances
- a Master instance partitions the data and gathers the partial results

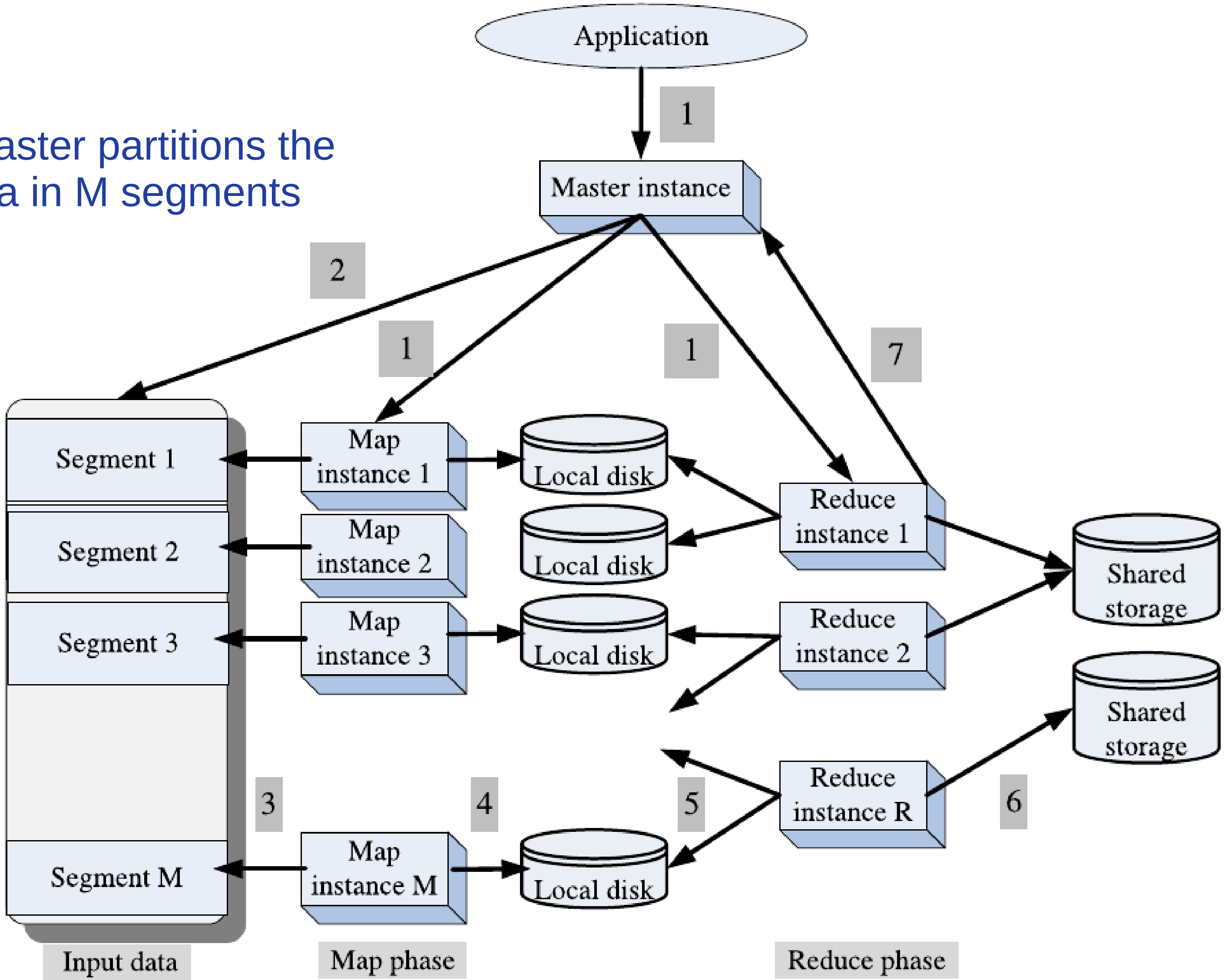
The MapReduce philosophy

(1) An application starts a Master instance and M worker instances for the Map phase and later R worker instances for the Reduce phase.



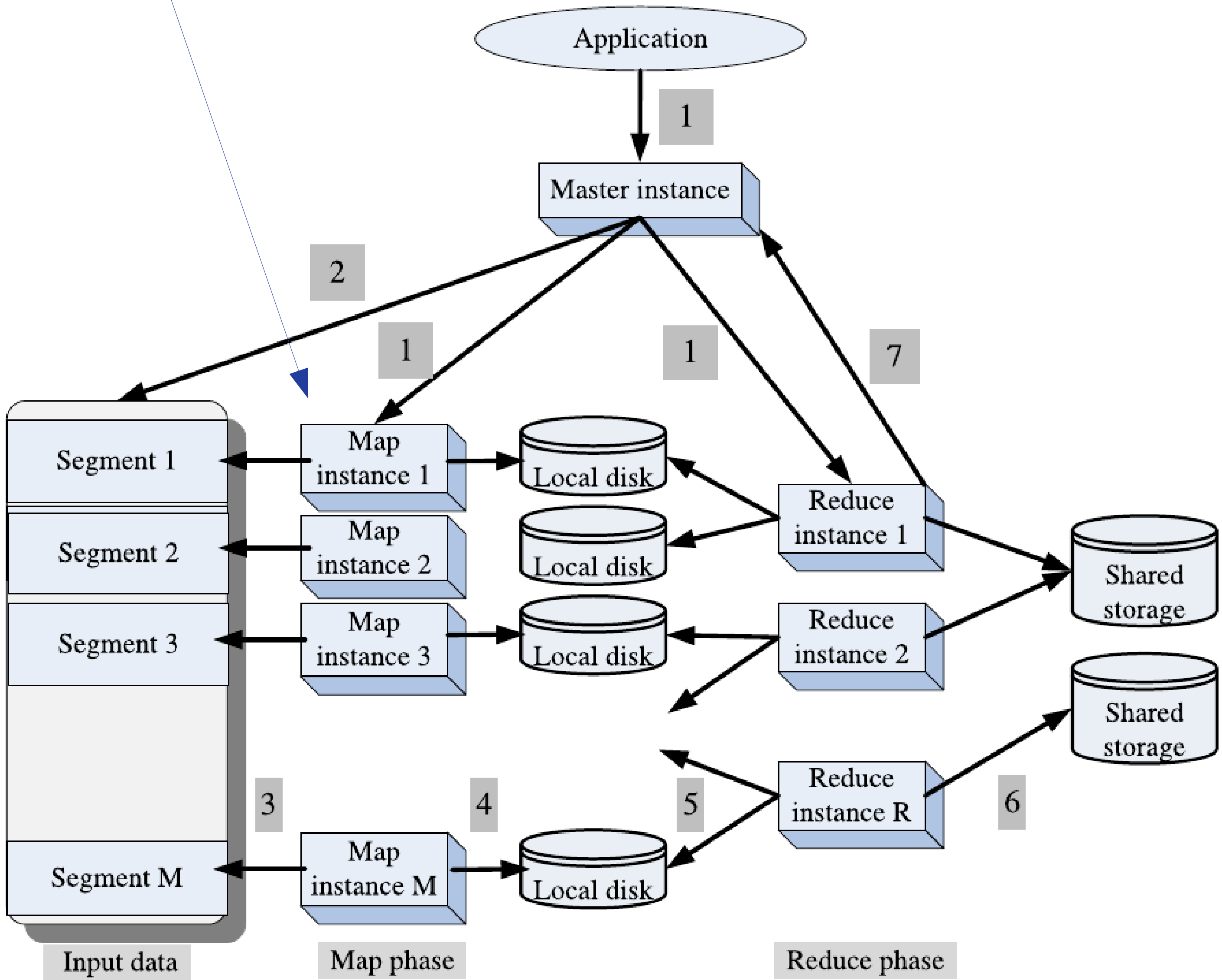
The MapReduce philosophy

(2)The Master partitions the input data in M segments



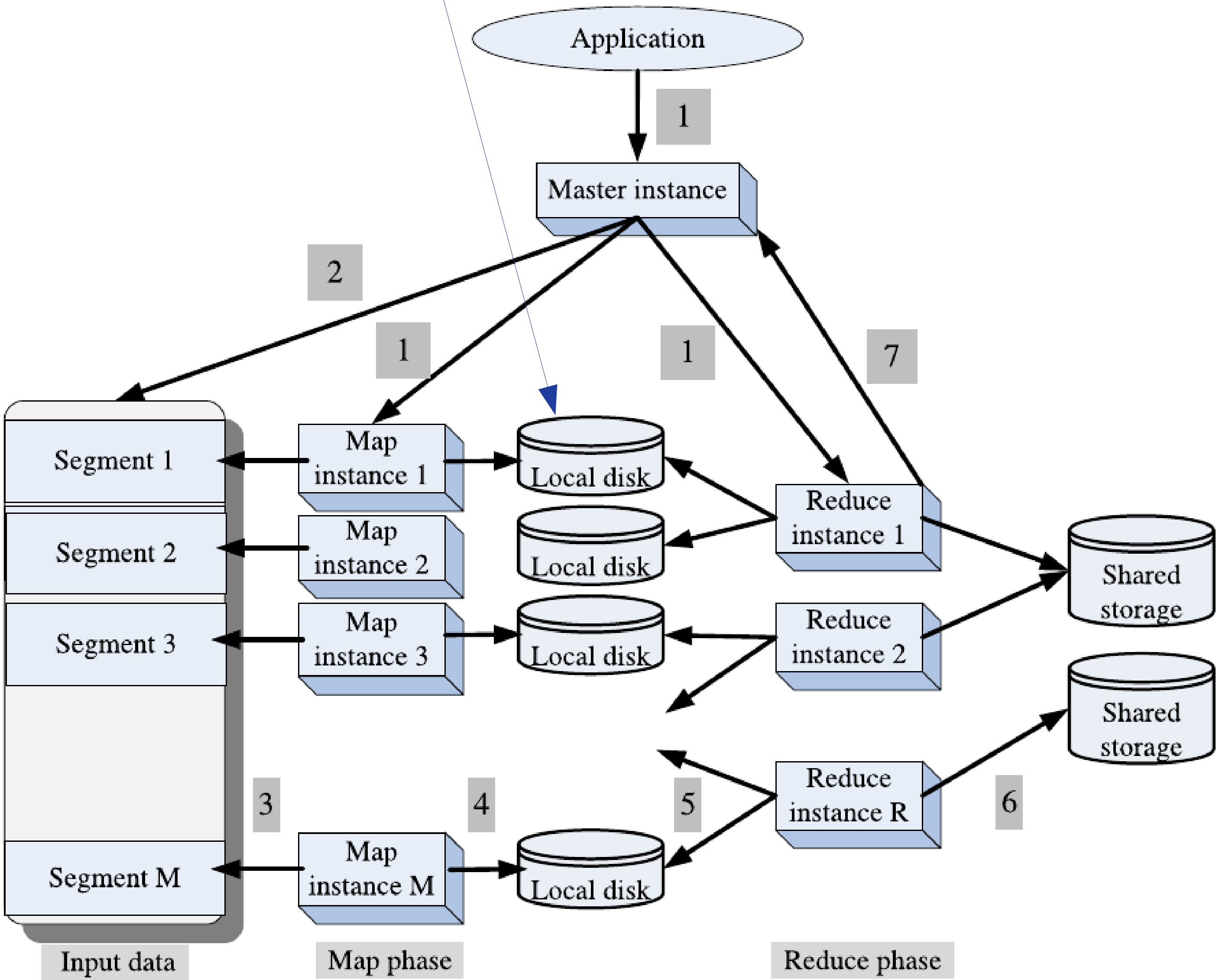
The MapReduce philosophy

(3) Each Map instance reads its input data segment and processes the data



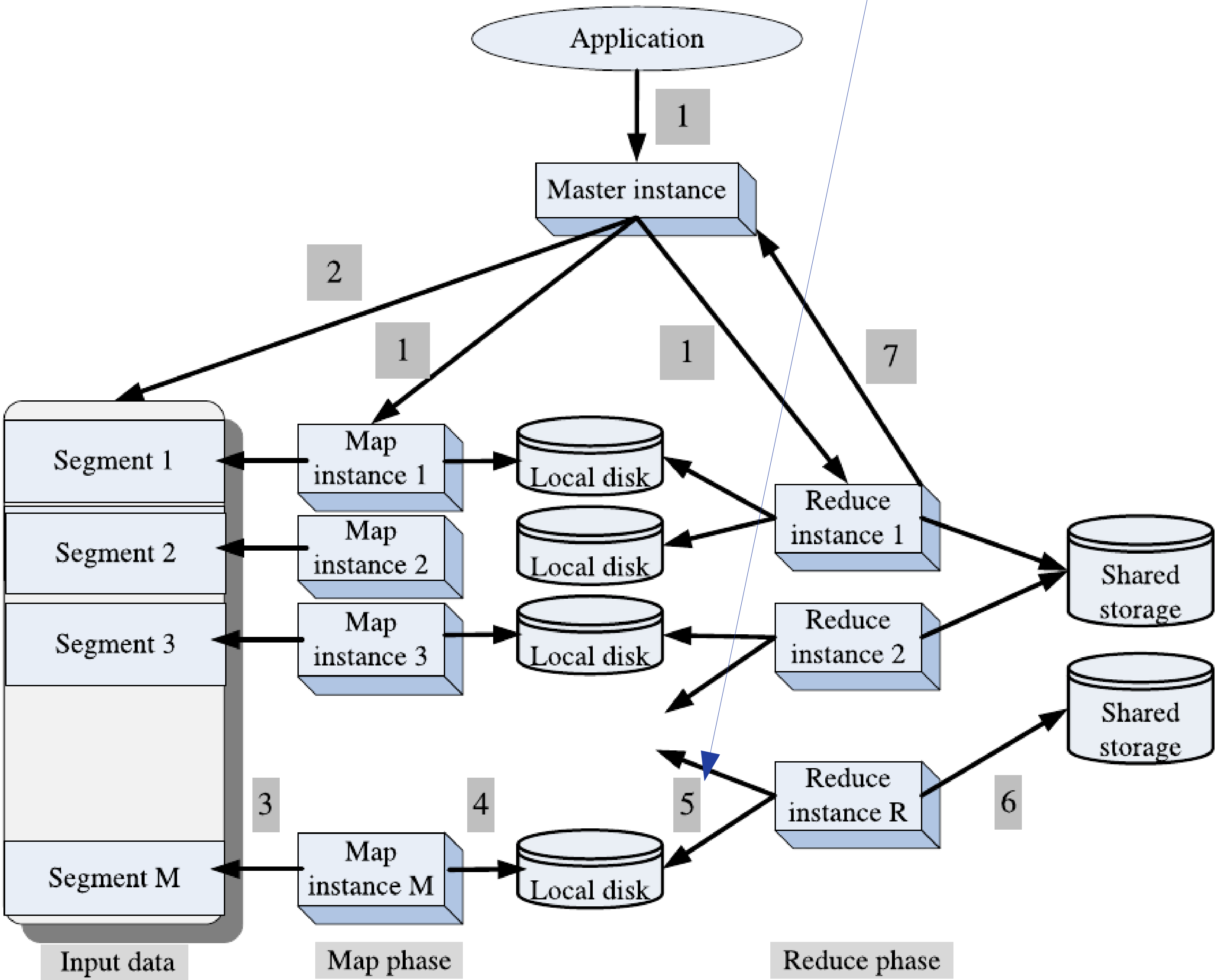
The MapReduce philosophy

(4) The results of the processing are stored on the local disks of the servers where the Map instances run



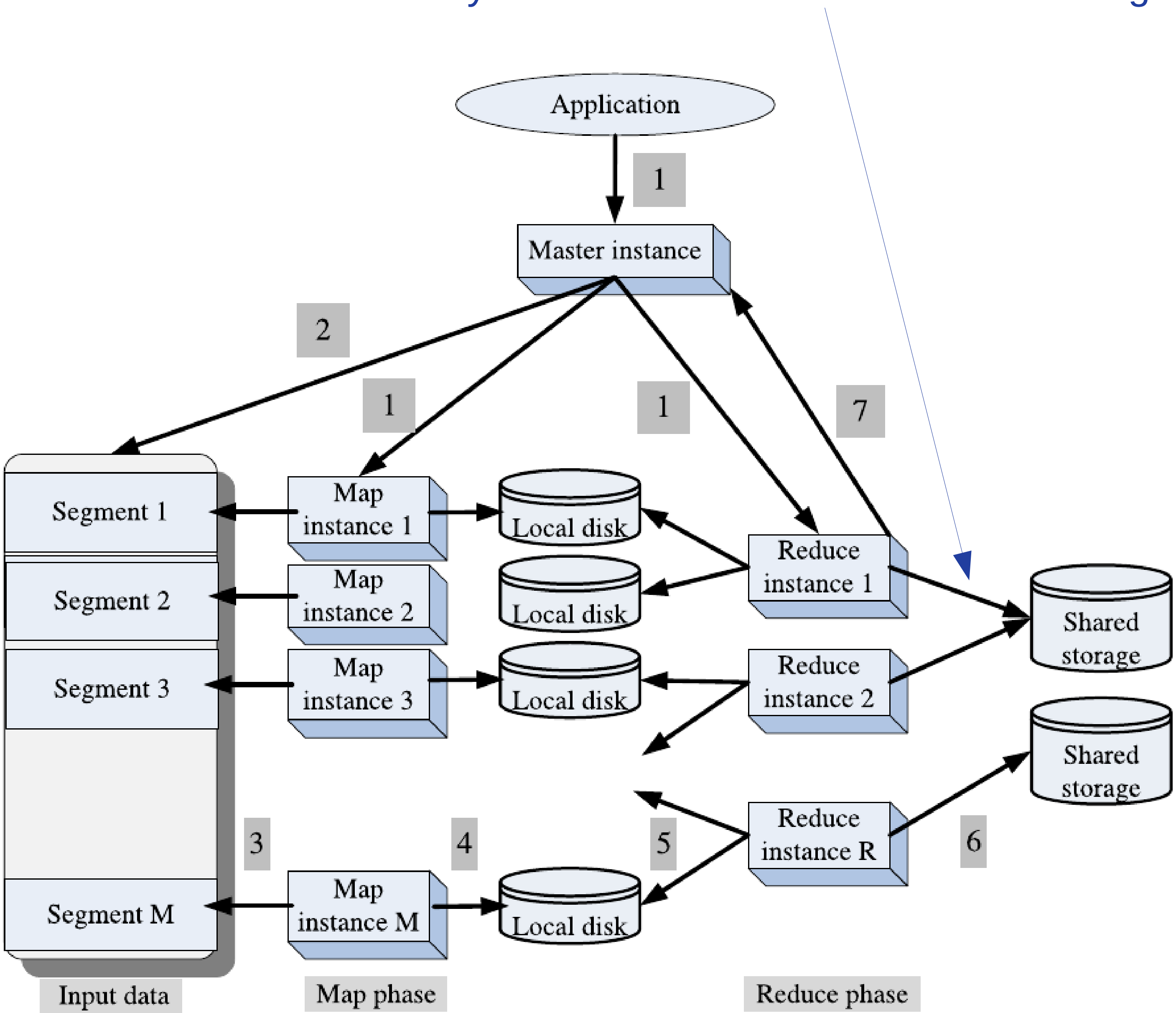
The MapReduce philosophy

(5)When all Map instances have finished processing their data the R Reduce instances read the results of the first phase and merges the partial results



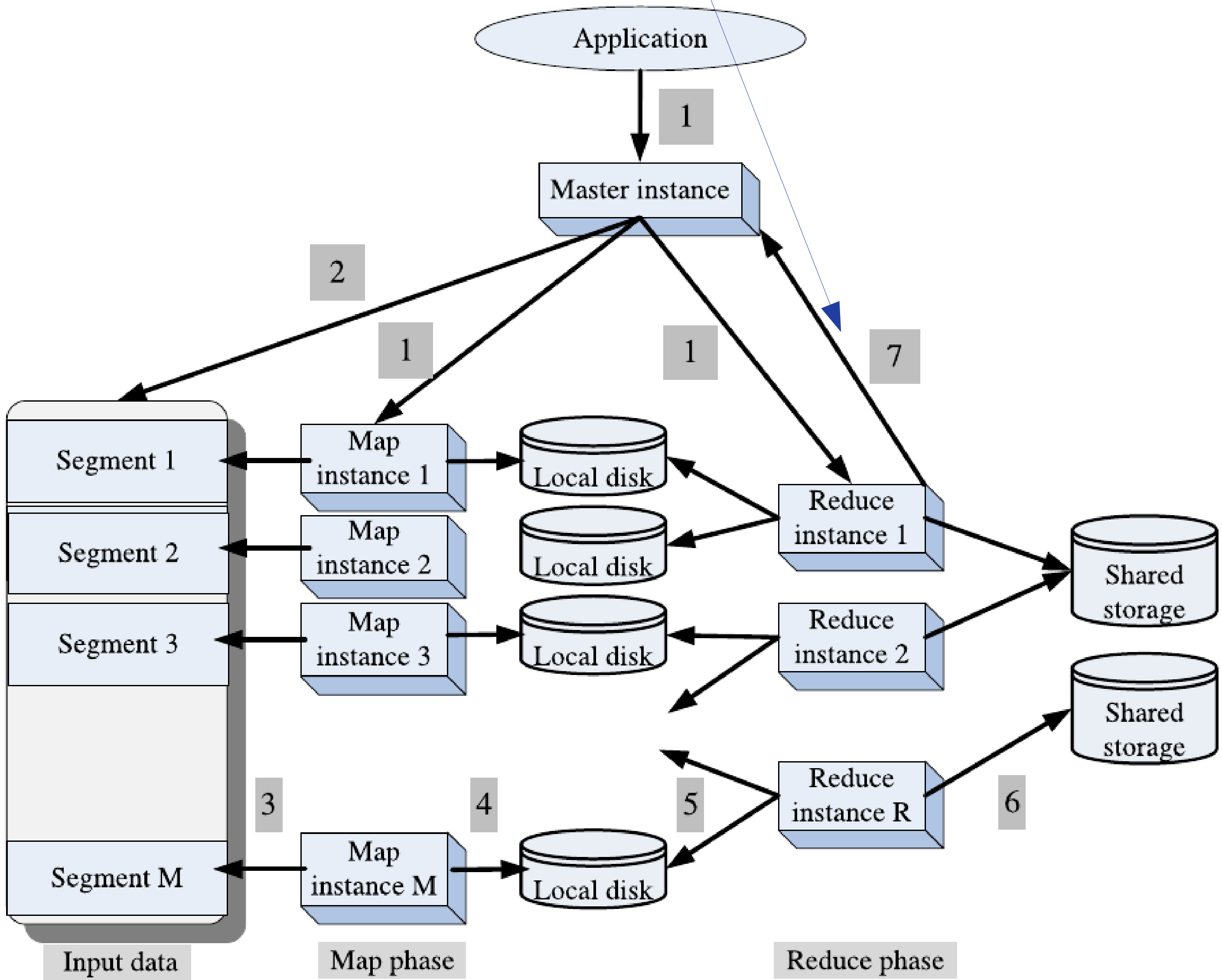
The MapReduce philosophy

(6) The final results are written by Reduce instances to a shared storage server



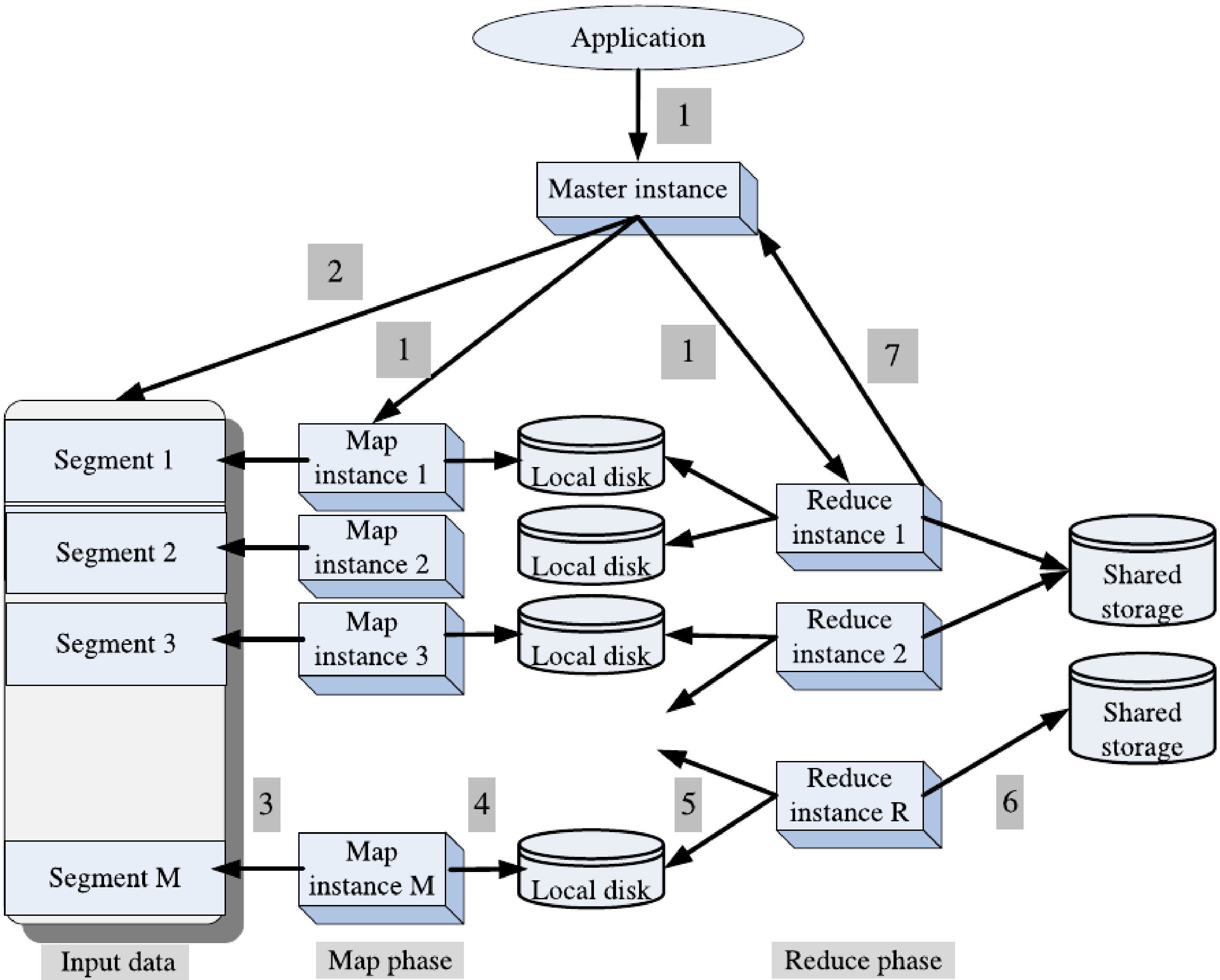
The MapReduce philosophy

(7) The Master instance monitors the Reduce instances and when all of them report task completion the application is terminated.



The MapReduce philosophy

Finally a set of input < key, value > pairs is transformed into a set of output < key, value > pairs

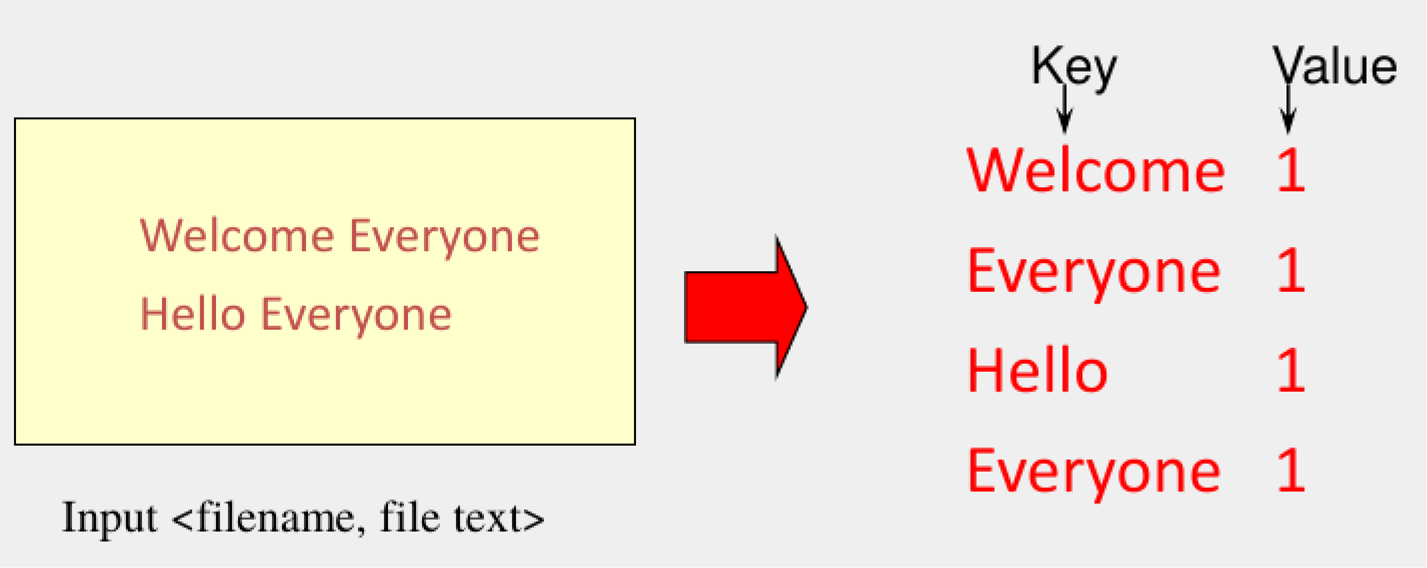


Map-Reduce Examples

- Processing logs of web page requests and count the URL access frequency
  - the Map functions produce the pairs <URL, 1 >
  - The Reduce functions produce <URL, totalcount >
- Another trivial example is distributed sort
  - the Map function extracts the key from each record and produces a < key, record > pair
  - the Reduce function outputs these pairs unchanged

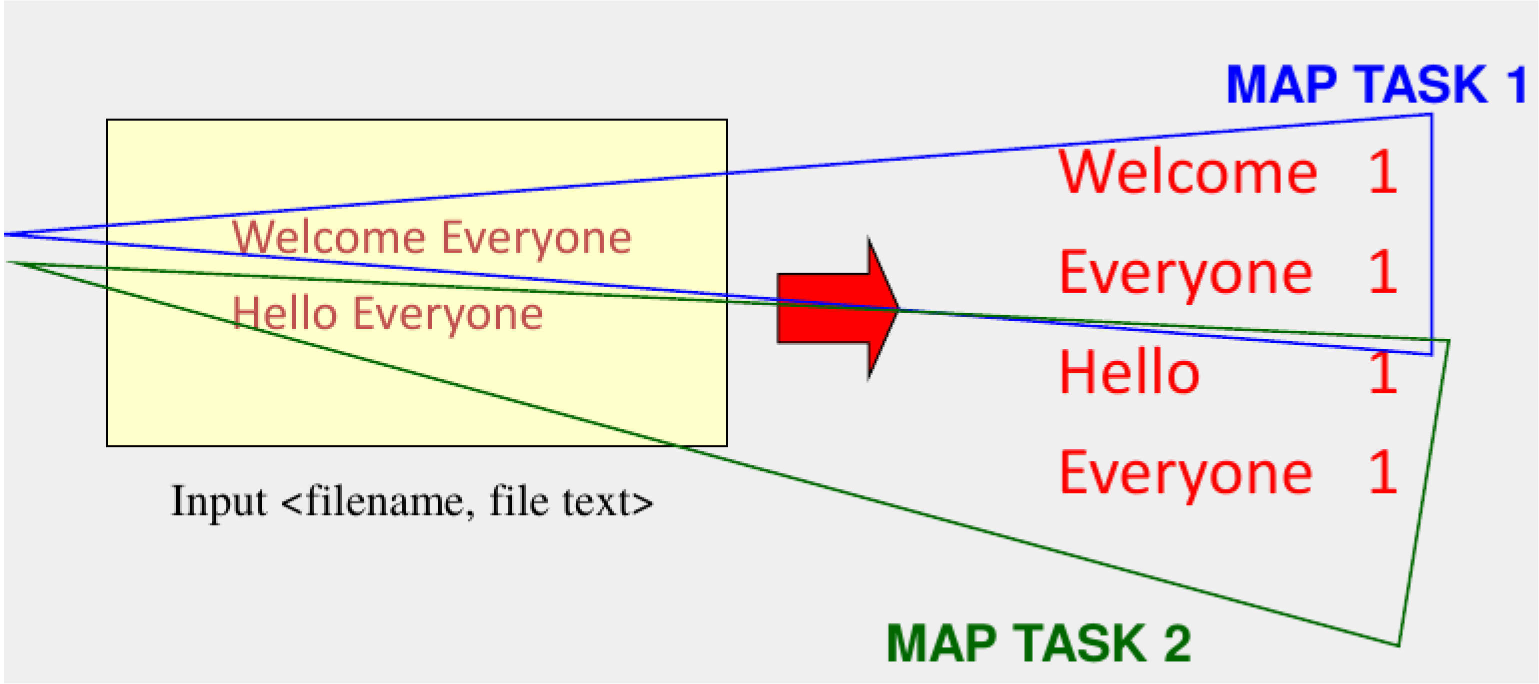
Word (URL) Count Example MAP

- Process individual records to generate intermediate key-value pairs



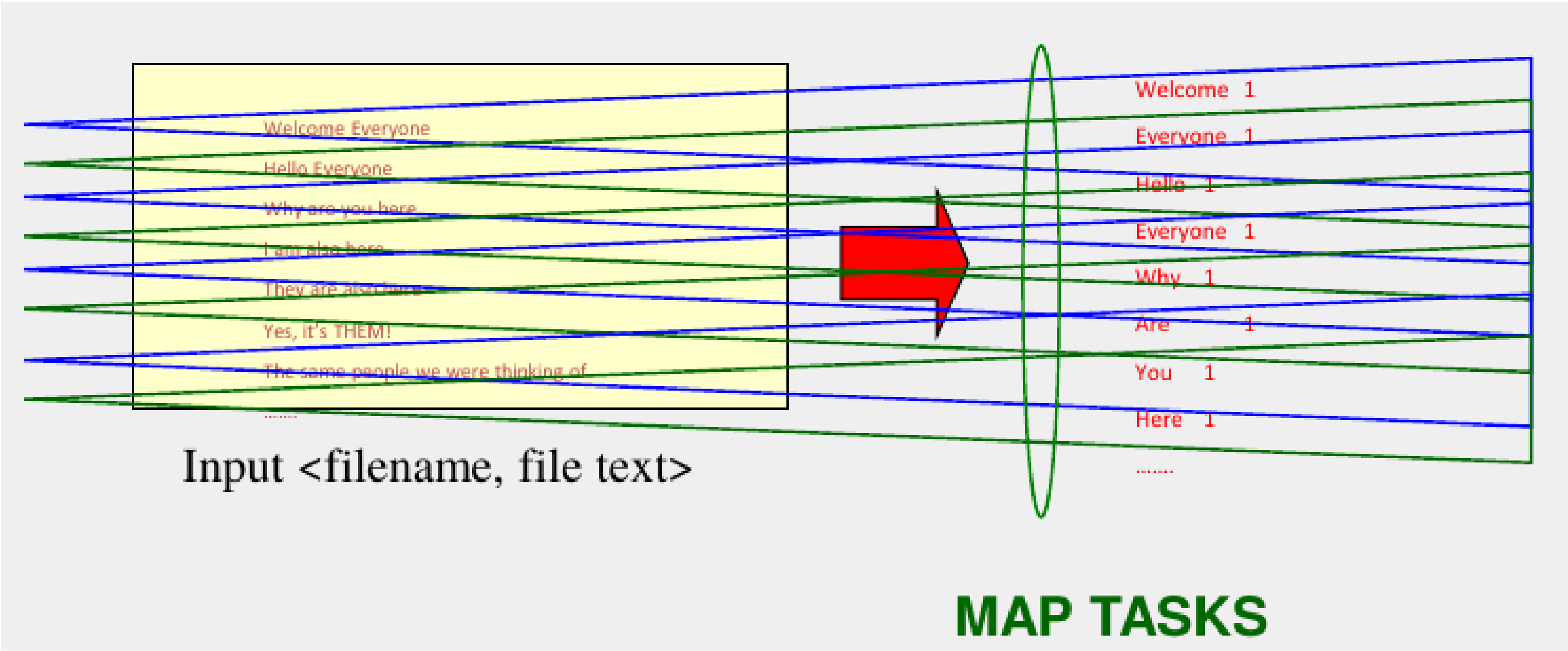
Word (URL) Count Example MAP

- Process individual records to generate intermediate key-value pairs



Word (URL) Count Example MAP

- Parallelly process a large number of individual records to generate intermediate key/value pairs

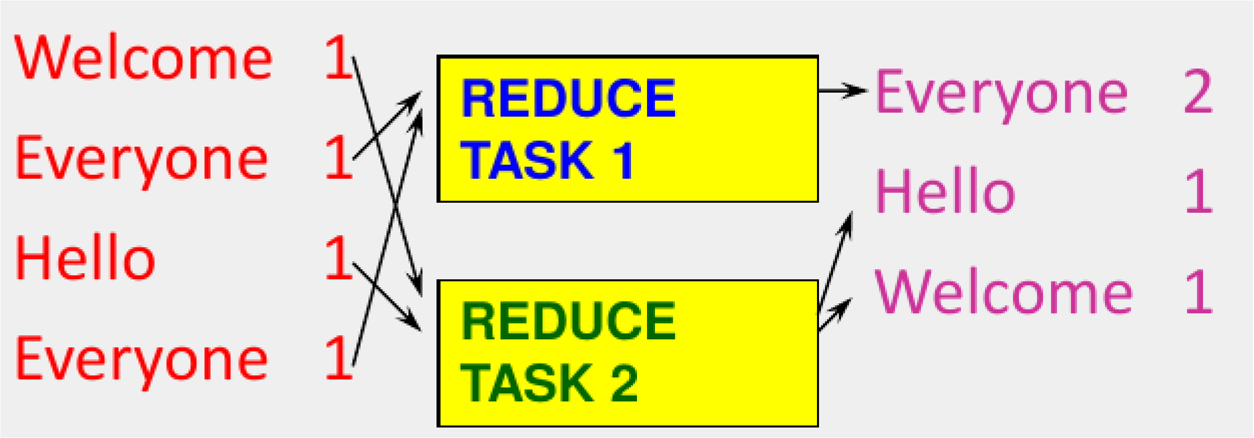


Word (URL) Count Example Reduce

- Reduce processes and merges all intermediate values associated per key

Word (URL) Count Example Reduce

- Each Key assigned to one Reduce
- Parallelly processes and merges all intermediate values by partitioning keys
- Popular: hash partitioning, i.e. key is assigned to reduce#= hash(key)%number of reduce servers



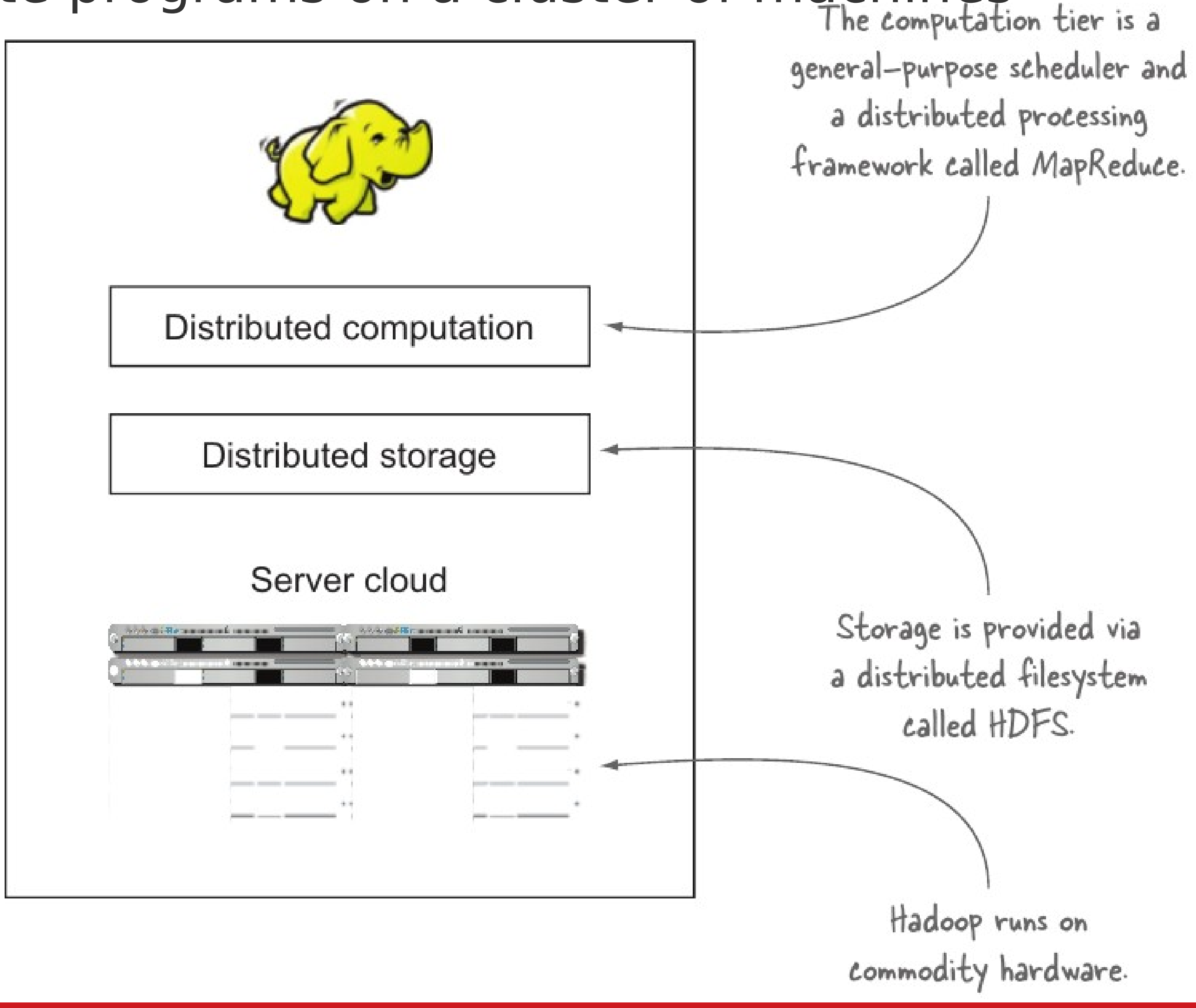
URL Count sudo code

```
map(String key, String value):
  // key: document name;  value: document contents
  for each word w in value:
  EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word;  values: a list of counts
  int result = 0;
  for each v in values:
  result += ParseInt(v);
  Emit(AsString(result));
```

Hadoop

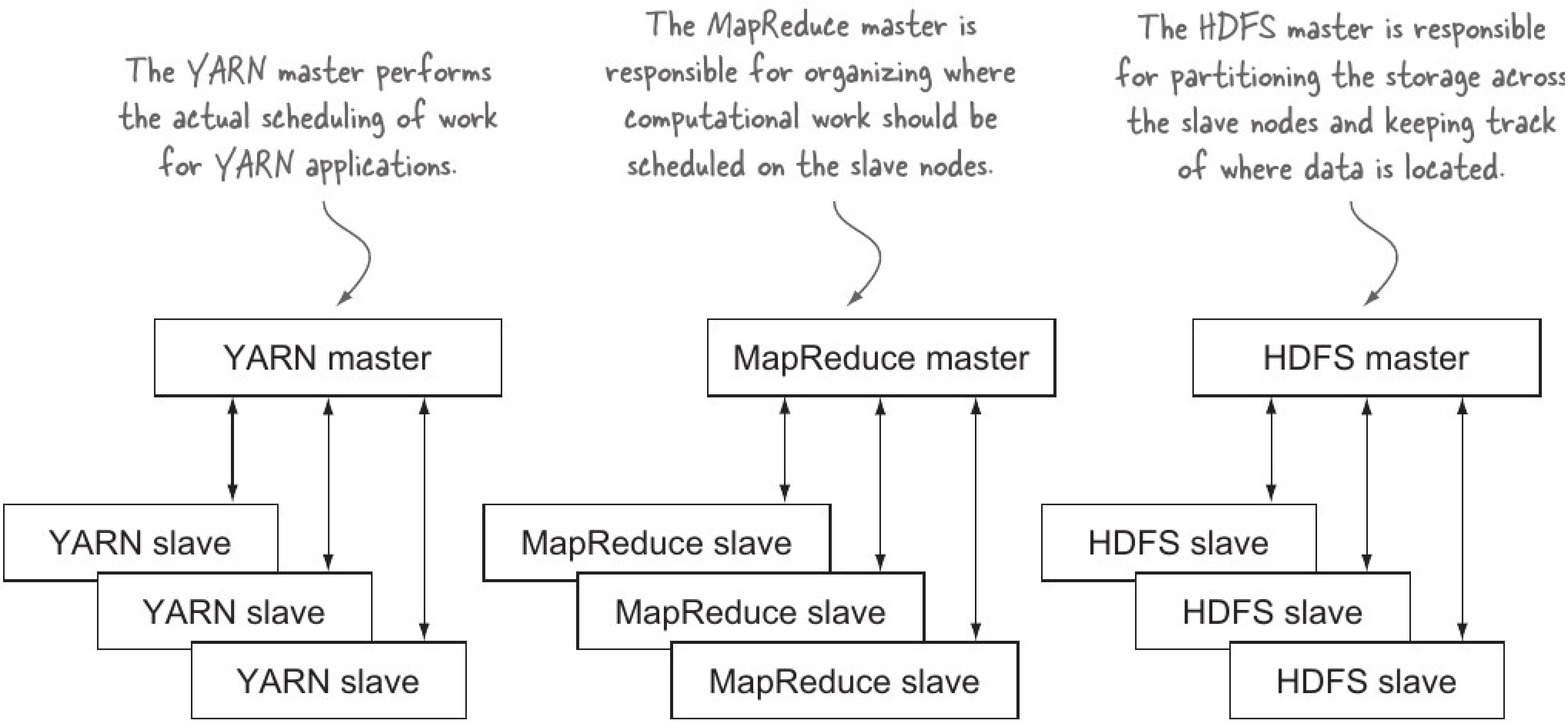
Hadoop is a platform that provides both distributed storage and computational capabilities. it offers a way to parallelize and execute programs on a cluster of machines



Hadoop components

- Hadoop is a distributed master-slave architecture that consists of the following primary components
  - Hadoop Distributed File System (HDFS) for data storage.
    - The FS could be also Amazon S3, CloudStore, or an implementation of GFS
  - Yet Another Resource Negotiator (YARN), a general purpose scheduler and resource manager.
    - Any YARN application can run on a Hadoop cluster
  - MapReduce, a batch-based computational engine.

Hadoop Master-slave arch

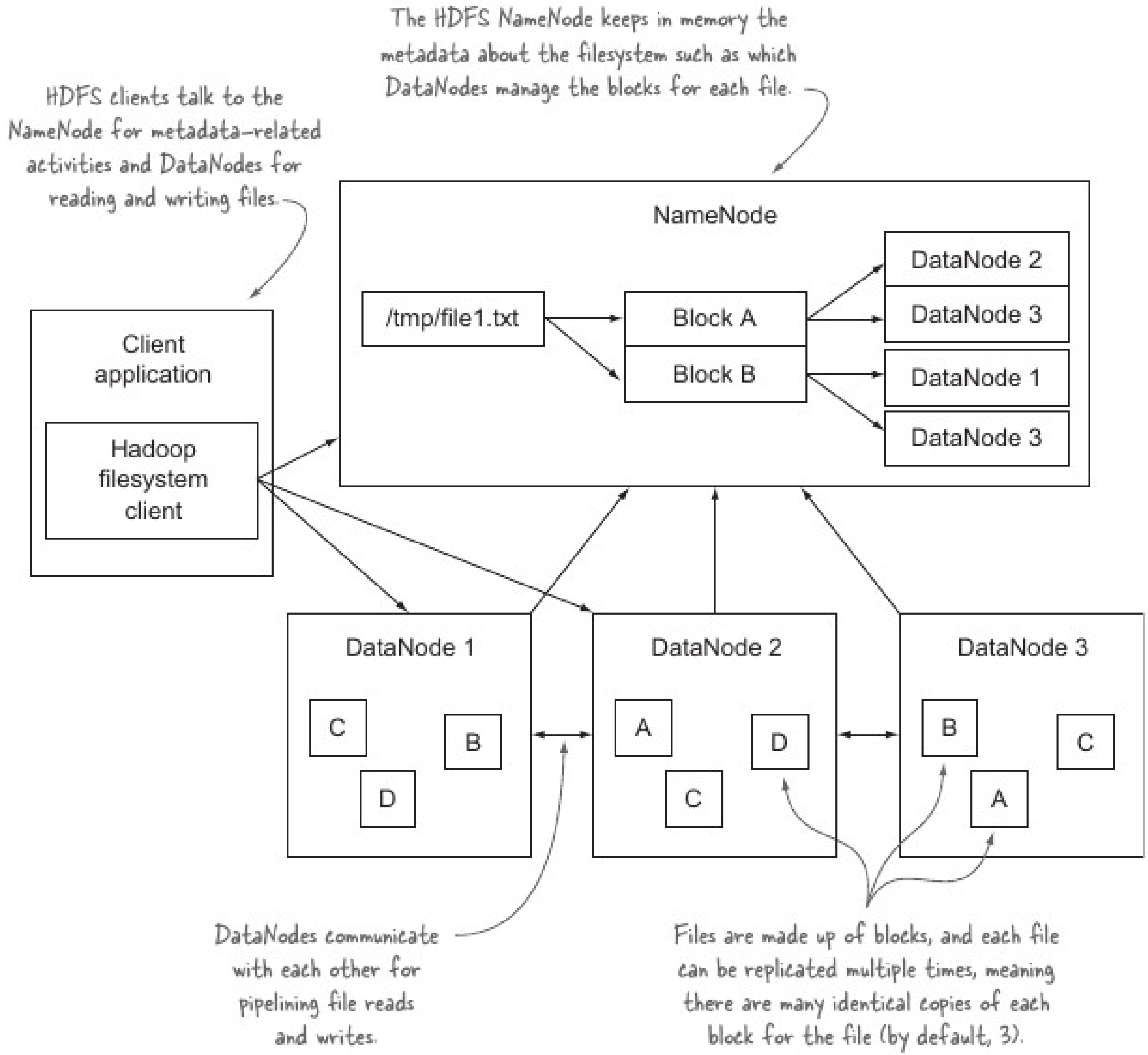


HDFS

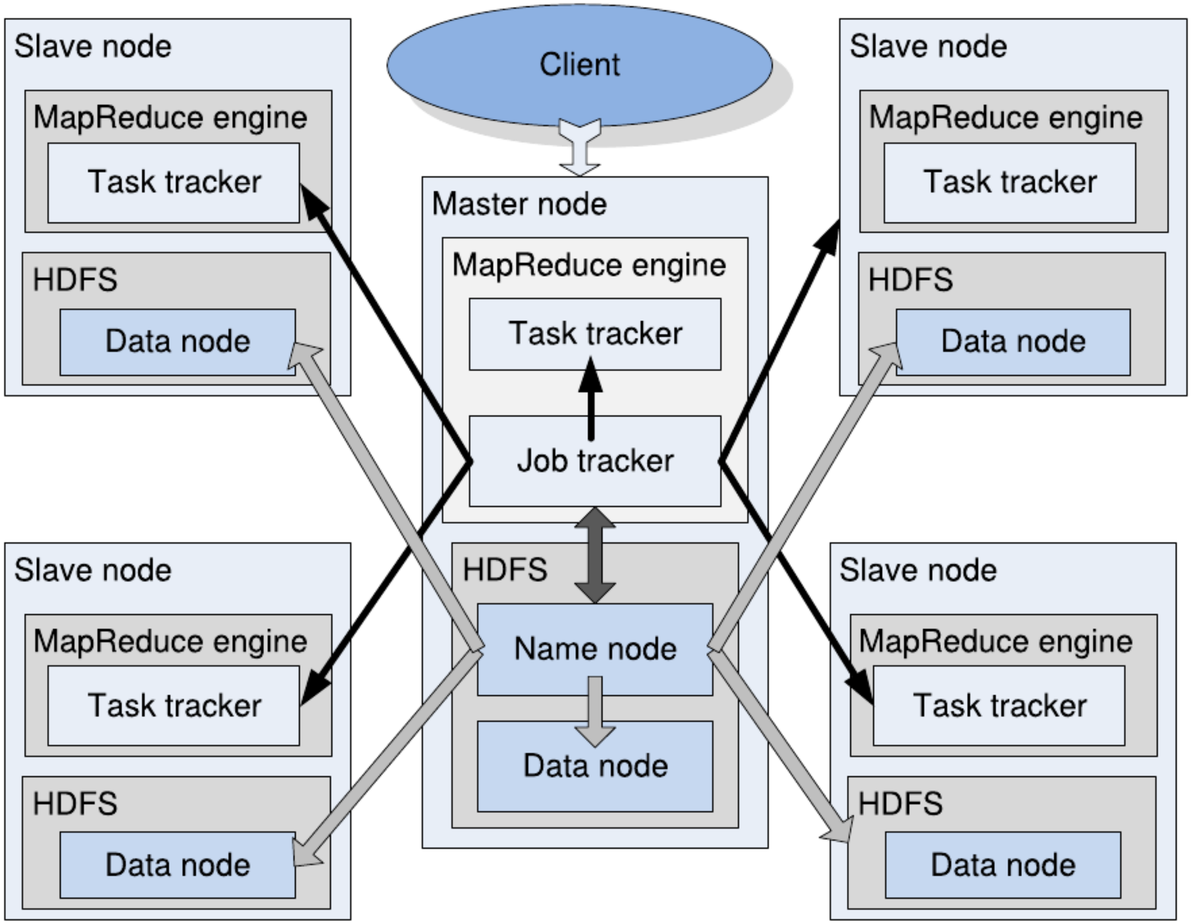
- It's a distributed filesystem that's modeled after the Google File System (GFS) paper
- HDFS replicates files for a configured number of times
  - So it is tolerant of both software and hardware failure, and automatically replicates data blocks on nodes that have failed

HDFS main Components

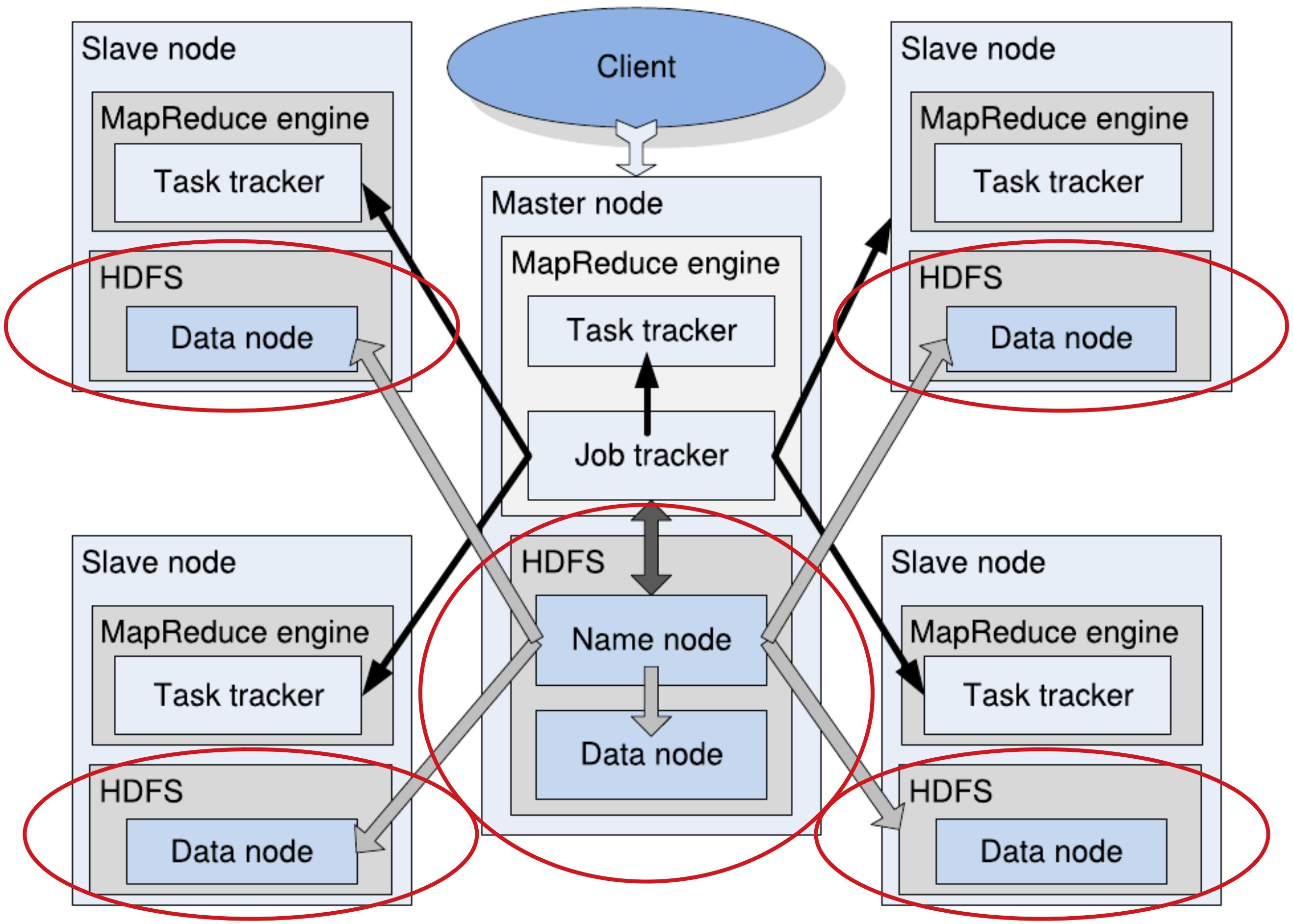
1. NameNode
2. DataNode



HDFS in Hadoop Cluster



HDFS in Hadoop1 Cluster

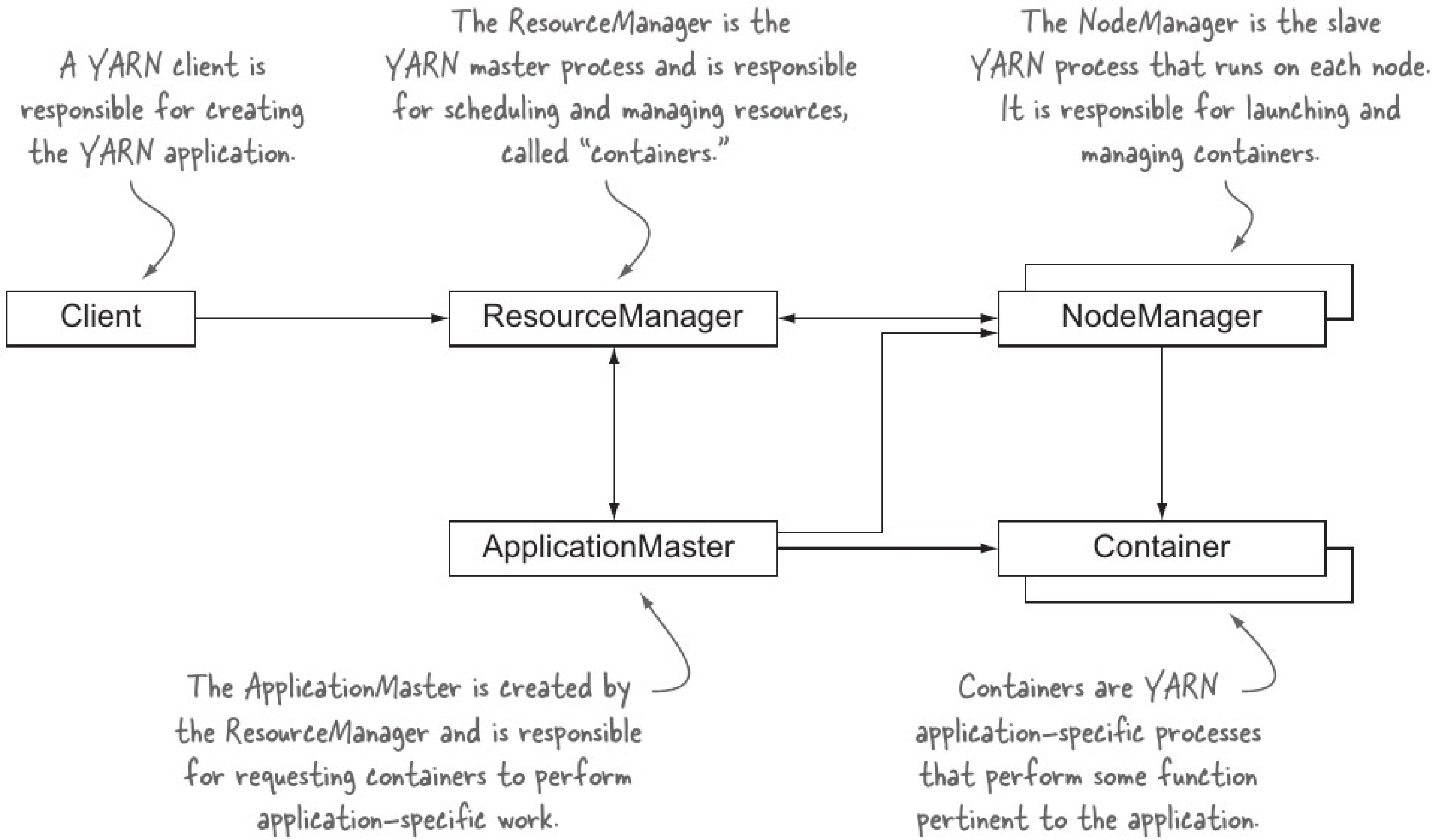


Yarn

- YARN is Hadoop’s distributed resource scheduler
  - a resource management system supplying CPU cycles, memory, and other resources needed by a single job or to a DAG of MapReduce applications

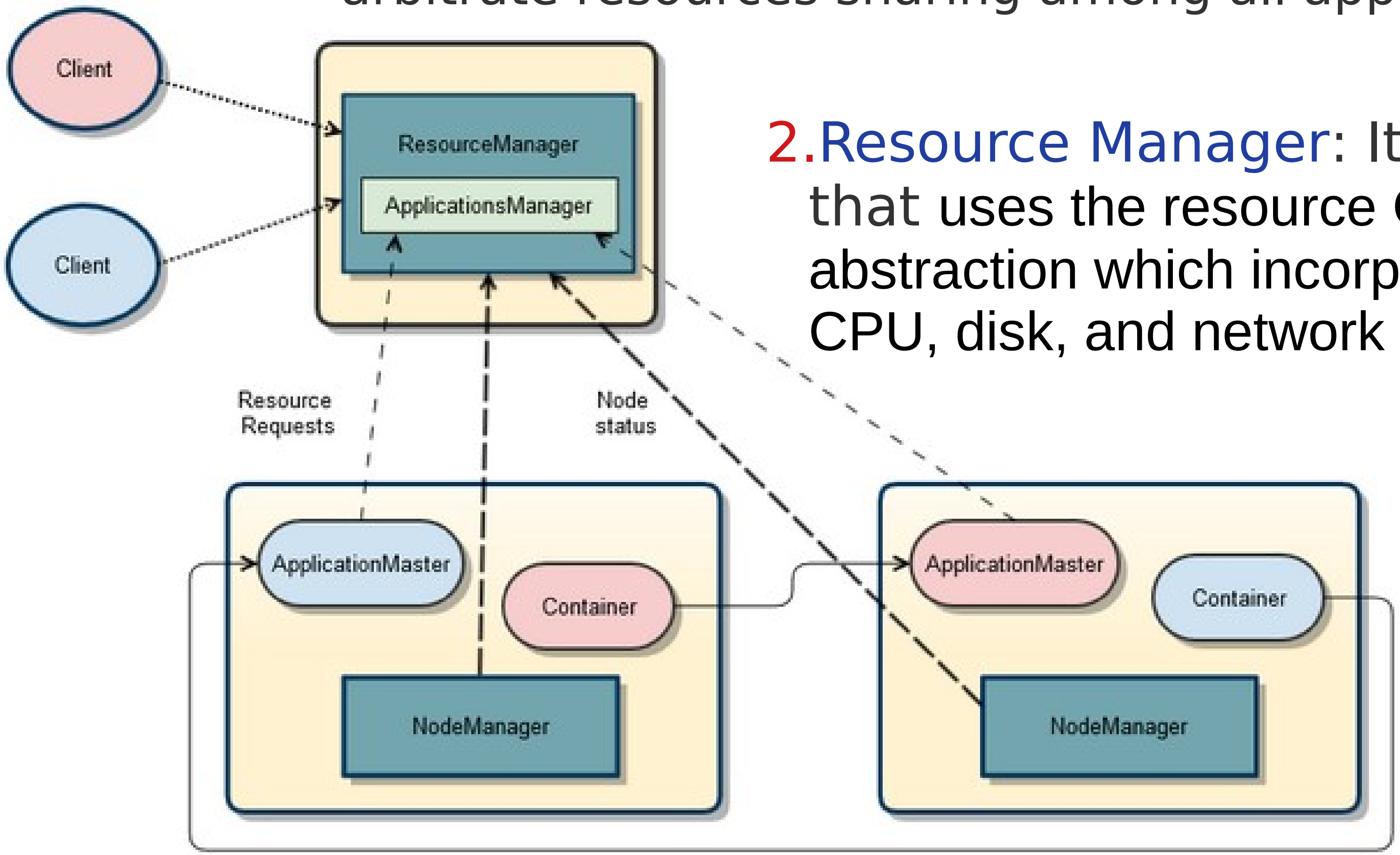
Yarn main components(I)

1. ResourceManager
2. NodeManager



Yarn main components(II)

1. Node Manager: responsible for containers, monitors their resource usage (CPU, memory, disk, network)
   - It reports the resource usage to the resource manager to arbitrate resources sharing among all application



2. Resource Manager: It has a scheduler that uses the resource Container abstraction which incorporates memory, CPU, disk, and network for an application

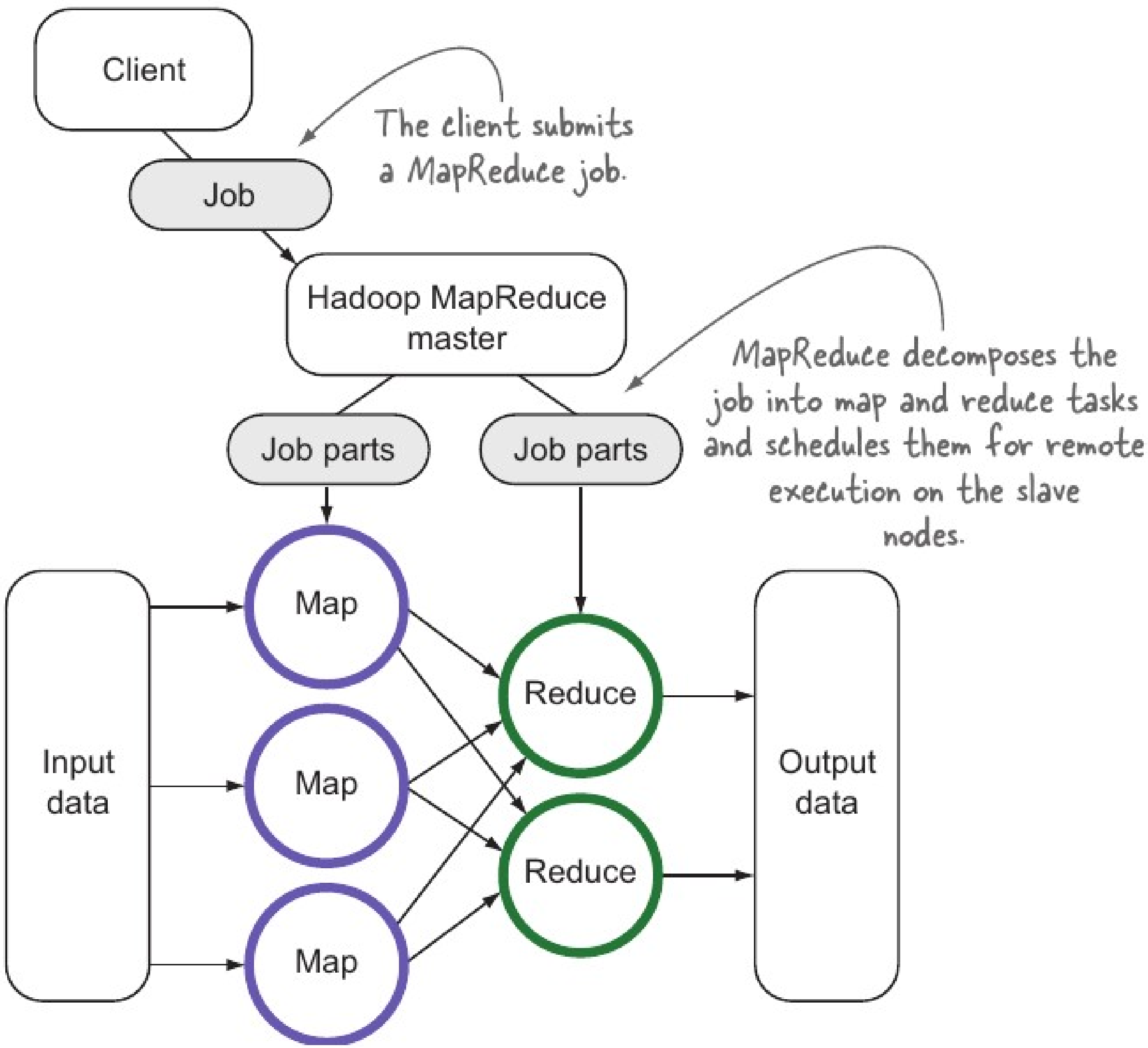
process of starting an application(I)

1. A client submits an application to the YARN Resource Manager, including the information required for the CLC (Container Launch Context)
2. The Applications Manager negotiates a container and bootstraps the Application Master instance for the application.
3. The Application Master registers with the Resource Manager and requests containers.
4. The Application Master communicates with Node Managers to launch the containers it has been granted, specifying the CLC for each container.

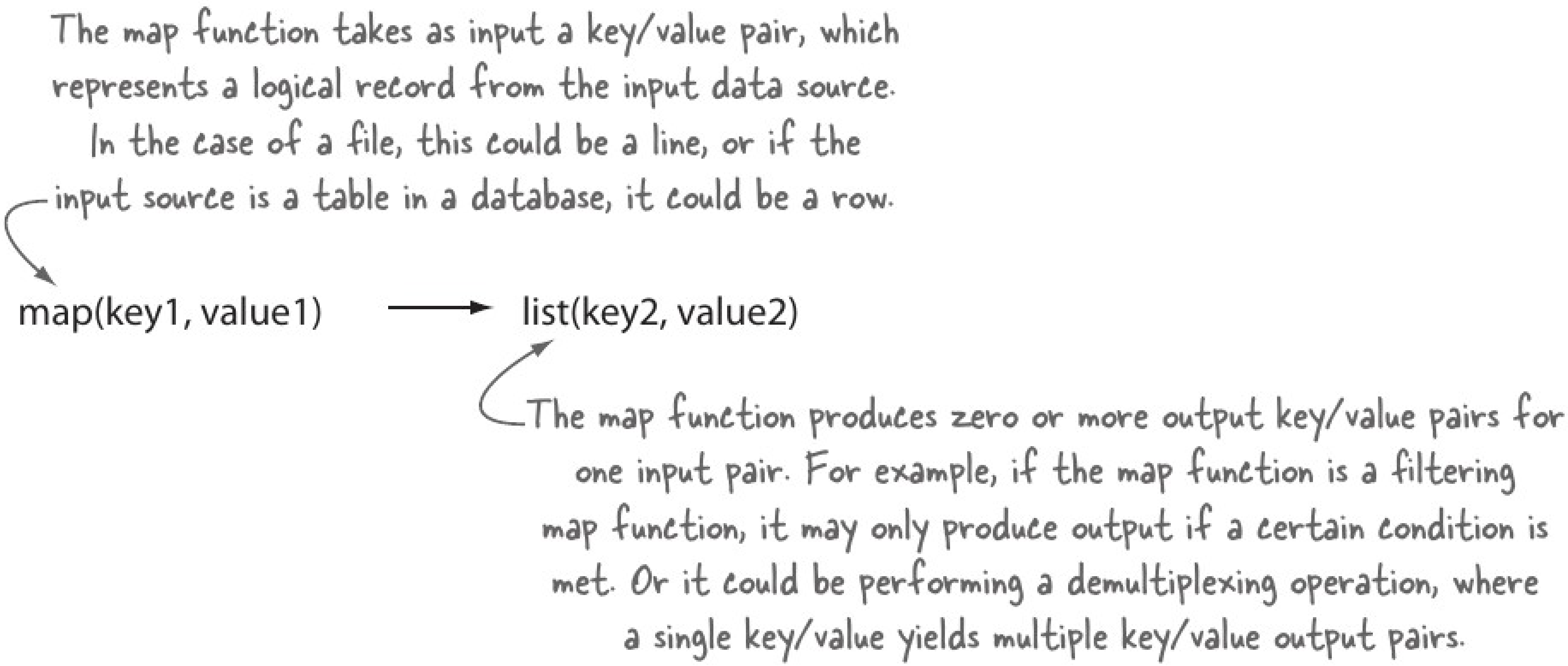
process of starting an application(II)

5. The Application Master manages application execution.
   - During execution, the application provides progress and status information to the Application Master.
   - The client can monitor the application’s status by querying the Resource Manager or by communicating directly with the Application Master.
6. The Application Master reports completion of the application to the Resource Manager.
7. The Application Master un-registers with the Resource Manager, which then cleans up the Application Master container.

MapReduce framework

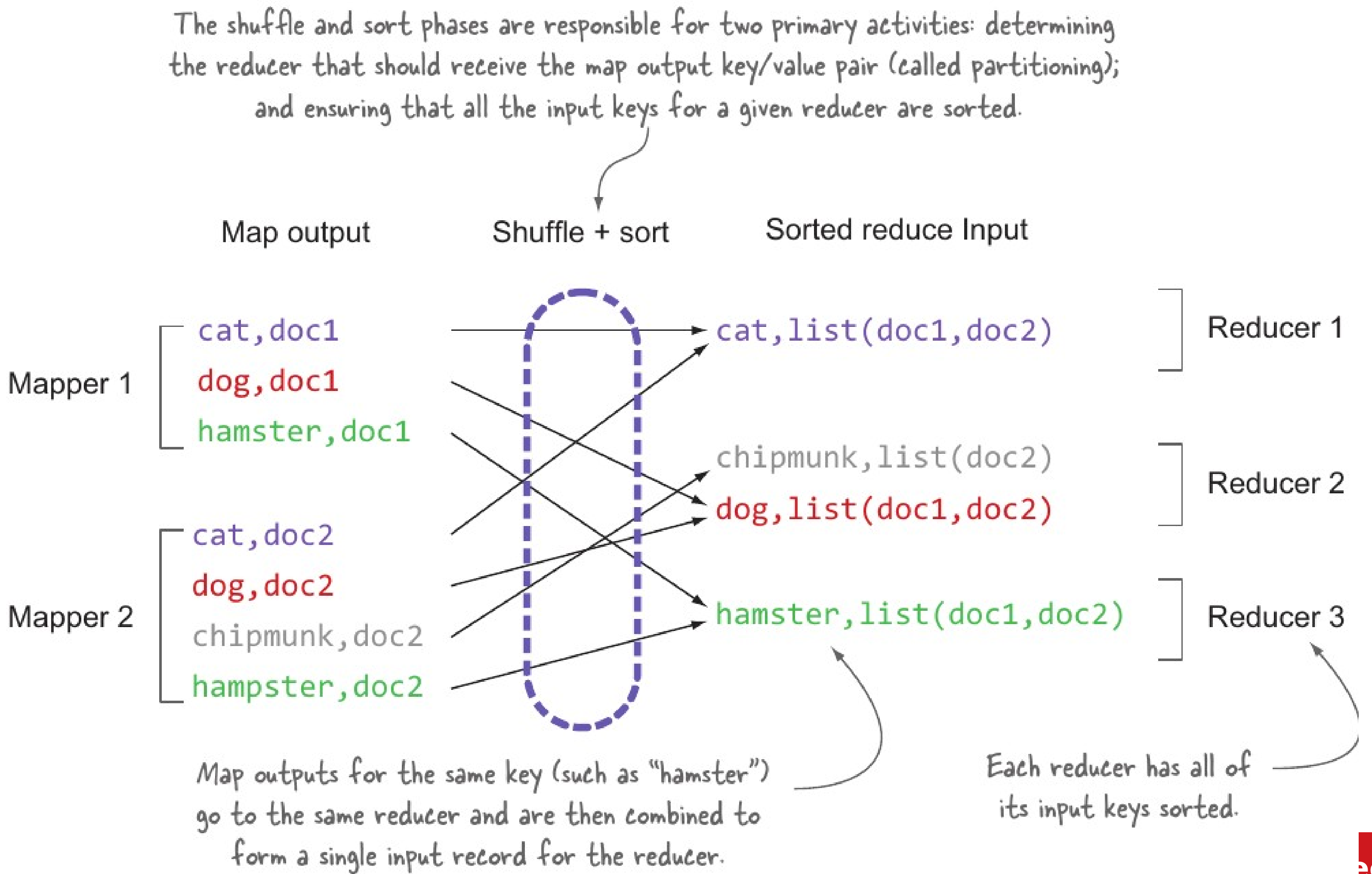


Map input/output

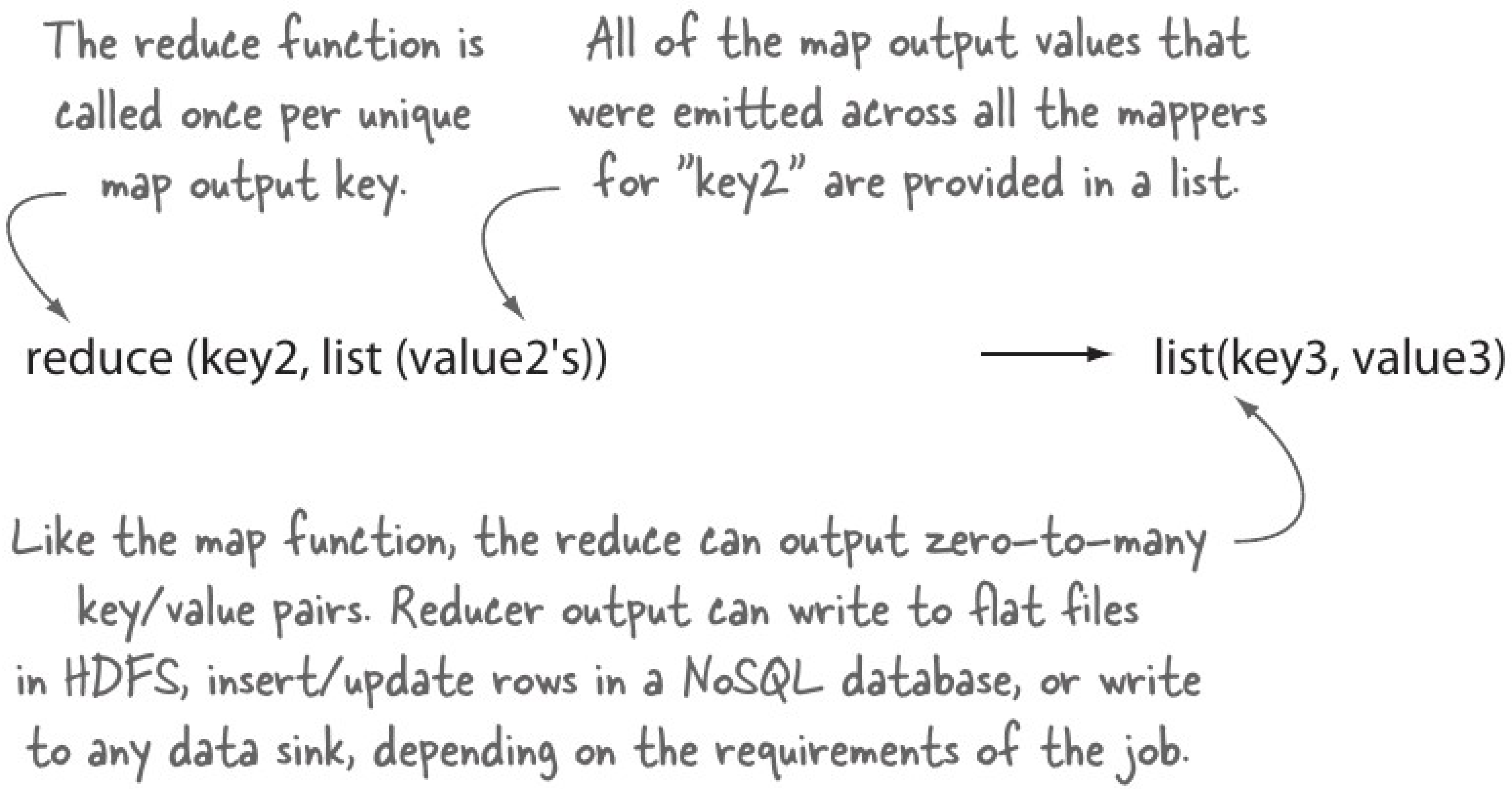


MapReduce Shuffle&Sort

The power of MapReduce occurs between the map output and the reduce input in the shuffle and sort phases

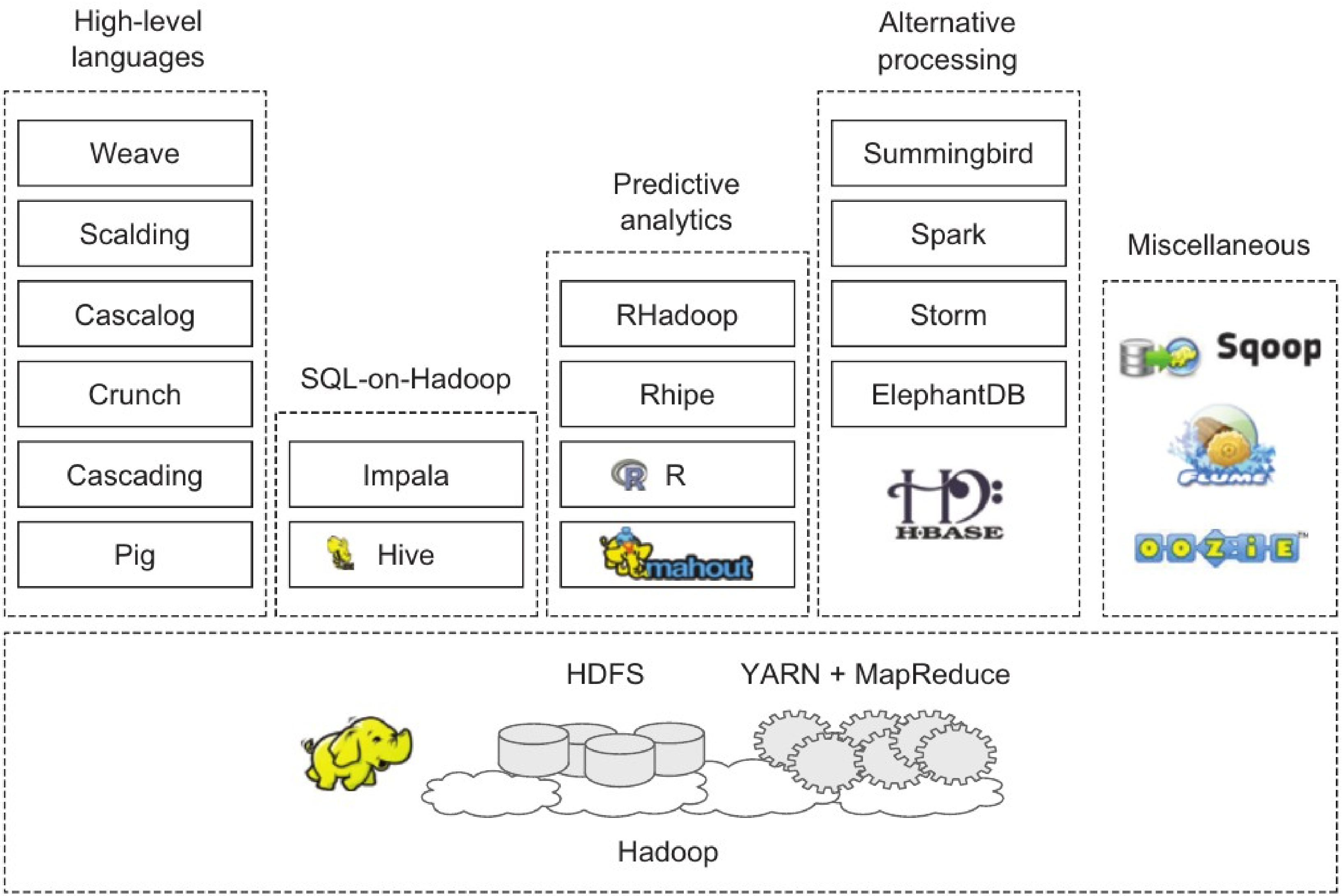


Reduce input/output



Hadoop echosystem

Hadoop is an apache Open Source project
https://hadoopecosystemtable.github.io/



WordCount Java Code

Map Class

```
public static class MapClass 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 itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        output.collect(word, one);
      }
    }
}
```

WordCount Java Code

Reduce Class

```
/**
 * A reducer class that just emits the sum of the input values.
 */
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));
  }
}
```

WordCount Java Code

Driver

```
public void run(String inputPath, String outputPath) throws Exception {
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("wordcount");

  // the keys are words (strings)
  conf.setOutputKeyClass(Text.class);
  // the values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(MapClass.class);
  conf.setReducerClass(Reduce.class);

  FileInputFormat.addInputPath(conf, new Path(inputPath));
  FileOutputFormat.setOutputPath(conf, new Path(outputPath));

  JobClient.runJob(conf);
}
```