MODULE 4

DATA INTENSIVE COMPUTING (MAP REDUCE PROGRAMMING)

Module 4 : Data Intensive Computing-MapReduce Programming

- Introduction to Data Intensive Computing
- Challenges Ahead
- High performance Distributed File Systems and Storage Clouds
- Developing Applications with MapReduce Programming Model

Unit 4: Objectives

After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

Data Intensive Computing: Map-Reduce Programming

- Data intensive computing focuses on a class of applications that deal with a large amount of data.
- Several application fields, ranging from computational science to social networking, produce large volumes of data that need to be efficiently stored, made accessible, indexed, and analyzed.
- These tasks become challenging as the quantity of information accumulates and increases over time at higher rates.
- Distributed computing is definitely of help in addressing these challenges by providing more scalable and efficient storage architectures and a better performance in terms of data computation and processing.

What is Data-Intensive Computing?

- Data-intensive computing is concerned with production, manipulation, and analysis of large-scale data in the range of hundreds of megabytes (MB) to petabytes (PB) and beyond.
- The term ***dataset*** is commonly used to identify a collection of information elements that is relevant to one or more applications.
- Datasets are often maintained in ***repositories***, which are infrastructures supporting the storage, retrieval, and indexing of large amount of information.
- In order to facilitate the classification and the search of relevant bits of information, ***metadata***, are attached to datasets.

Data Intensive Computations

- Data-intensive computations occur in many application domains.
  - Computational science is one of the most popular ones. Scientific simulations and experiments are often keen to produce, analyze, and process huge volumes of data. Hundreds of gigabytes of data are produced every second by telescopes mapping the sky and the collection of images of the sky easily reaches the scale of petabytes over a year.
  - Bioinformatics applications mine databases that may end up containing terabytes of data.
  - Earthquakes simulators process a massive amount of data, which is produced as a result of recording the vibrations of the Earth across the entire globe.

Data Intensive Computations

- Besides scientific computing, several IT industry sectors require support for data-intensive computations.
- A customer data of any telecom company would easily be in the range of 10-100 terabytes.
- Social networking and gaming are two other sectors where data intensive computing is now a reality.
- *Facebook* inbox search operations involve crawling about 150 terabytes of data and the whole uncompressed data stored by the distributed infrastructure reaches to 36 petabytes.
- *Zynga,* social gaming platform, moves 1 petabyte of data daily and it has been reported to add 1000 servers every week to sustain to store the data generated by games.

Unit 4: Objectives

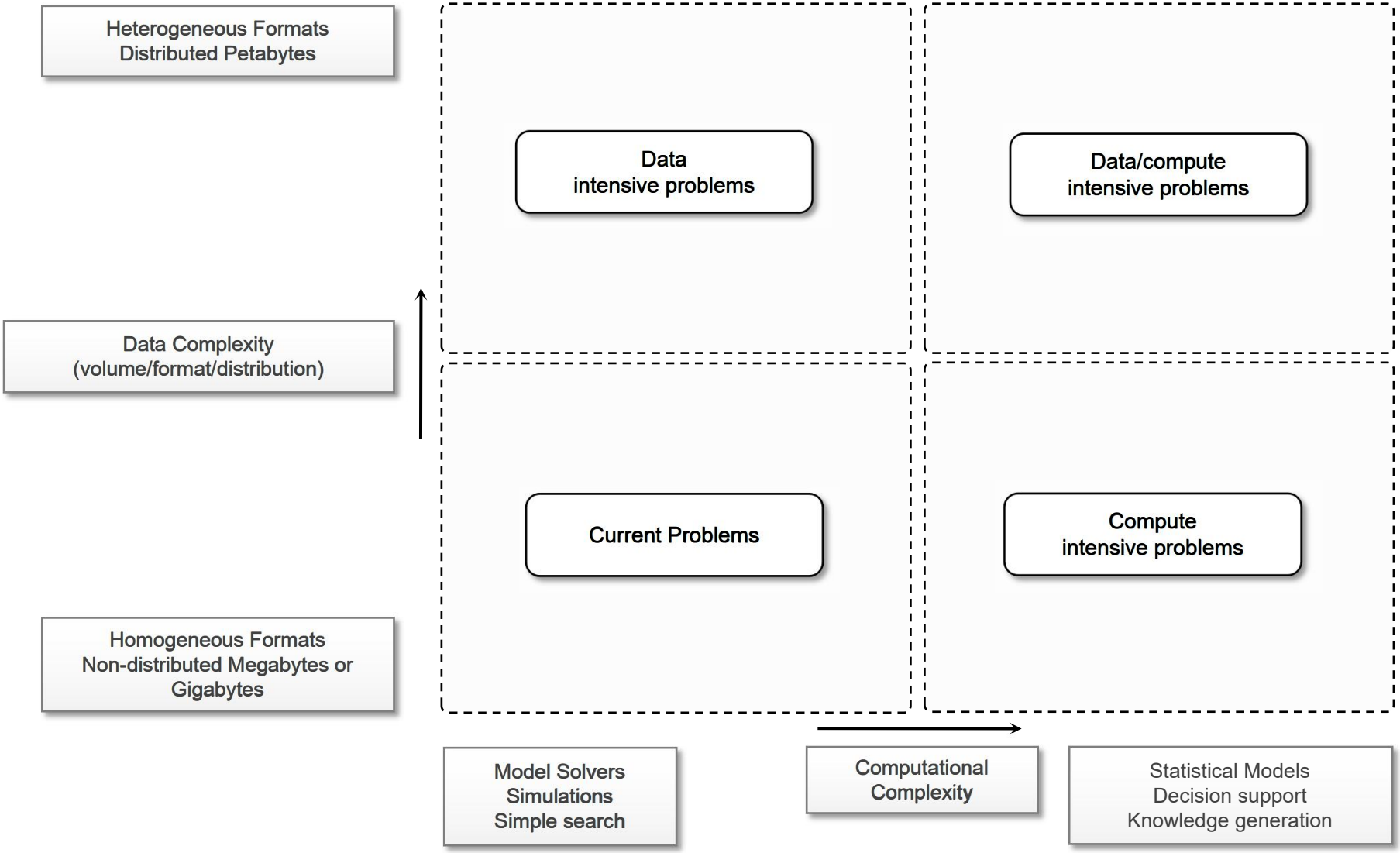
After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

Characterizing Data-Intensive Computations

- Data-intensive applications do not only deal with huge volumes of data but, very often, also exhibit compute-intensive properties.
- Data intensive applications handle datasets in the scale of multiple terabytes and petabytes.
- Datasets are commonly persisted in several formats and distributed across different locations.
- Such applications process data in multistep analytical pipelines including transformation and fusion stages.
- The processing requirements scale almost linearly with the data size and they can be easily processed in parallel.
- They also need efficient mechanisms for data management, filtering and fusion, and efficient querying and distribution.

Data Intensive Research Issues



Challenges Ahead

- Scalable algorithms that can search and process massive datasets.
- New metadata management technologies that can scale to handle complex, heterogeneous, and distributed data sources.
- Advances in high-performance computing platform aimed at providing a better support for accessing in-memory multi-terabyte data structures.
- High-performance, high-reliable, petascale distributed file systems.
- Data signature generation techniques for data reduction and rapid processing.
- New approaches to software mobility for delivering algorithms able to move the computation where the data is located.
- Specialized hybrid interconnection architectures providing a better support for filtering multi-gigabyte data streams coming from high speed networks and scientific instruments.
- Flexible and high-performance software integration techniques facilitating the combination of software modules running on different platform to quickly form analytical pipelines.

Unit 4: Objectives

After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

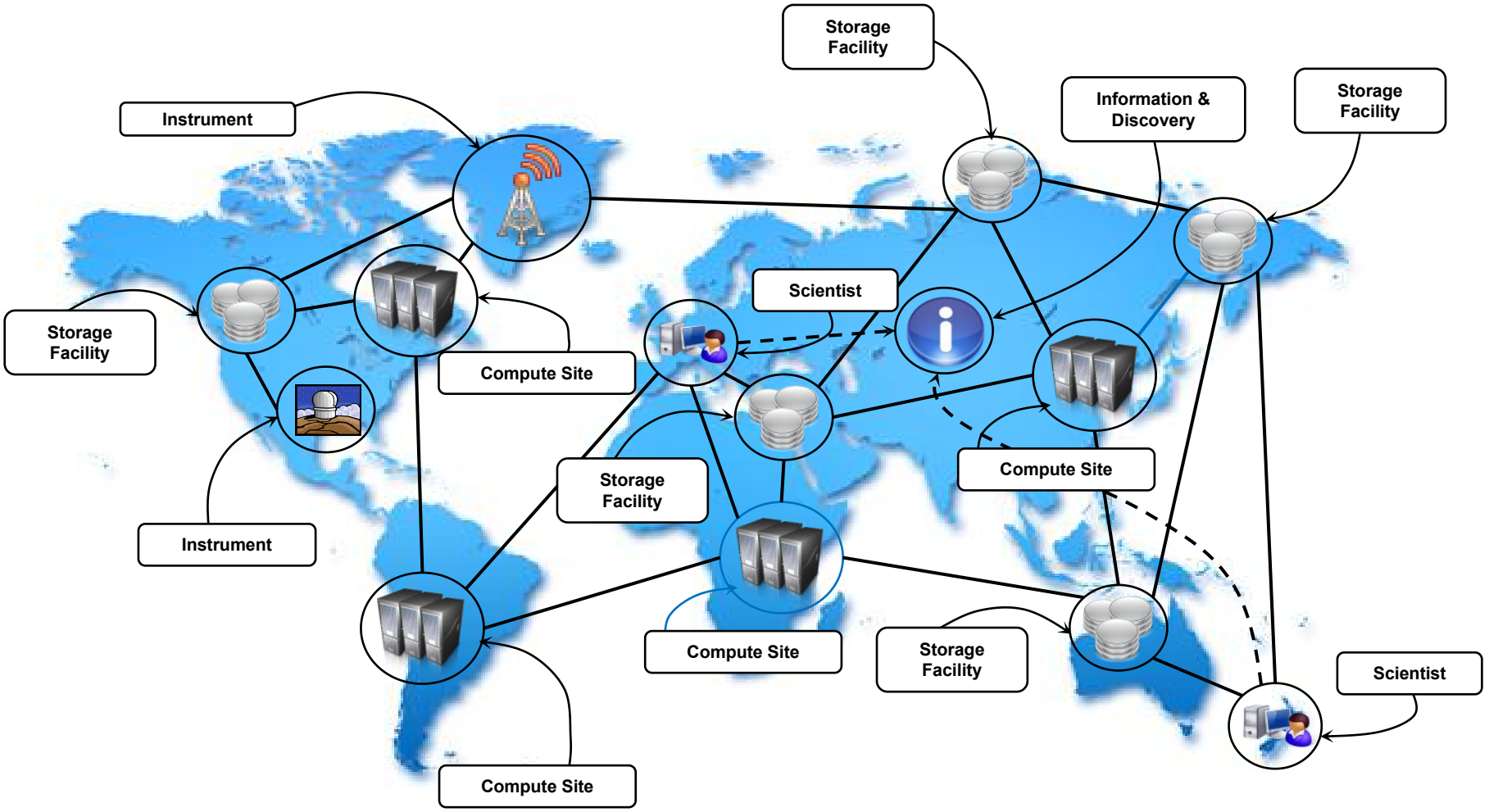
Historical Perspective

- Data-intensive computing involves the production, management, and analysis of large volumes of data.
- Support for data-intensive computations is provided by harnessing storage, networking technologies, algorithms, and infrastructure software all together.
  - The early Age: Wide Area Networking
  - Data Grids
  - Data Clouds and Big Data
  - Data bases and Data-Intensive Computing

Data Grids

- With the advent of Grid Computing, huge computational power and storage facilities could be obtained by harnessing heterogeneous resources across different administrative domains. Within this context, *Data Grids* emerge as infrastructures supporting data-intensive computing.
- A Data Grid provides services helping users to discover, transfer, and manipulate large datasets stored in distributed repositories and, also, create and manage copies of them.
- Data Grids offer two main functionalities: high-performance and reliable file transfer for moving large amounts of data; and scalable replica discovery and management mechanisms for an easy access to distributed datasets.
- Data Grids mostly provide storage and dataset management facilities as support of scientific experiments that produce huge volumes of data.

Data Grids


Storage Facility
Information & Discovery
Storage Facility
Instrument
Scientist
Storage Facility
Compute Site
Compute Site
Storage Facility
Instrument
Compute Site
Storage Facility
Compute Site
Scientist
Compute Site

Characteristics of Data Grids

- ***Massive Datasets.*** The size of datasets can easily be in the scale of gigabytes, terabytes, and beyond. It is therefore necessary to minimize latencies during bulk transfers, replicate content with appropriate strategies, and manage storage resources.
- ***Shared Data Collections.*** Resource sharing includes distributed collections of data. For example, repositories can be used to both storing and reading data.
- ***Unified Namespace.*** Data Grids impose a unified logical namespace where to locate data collections and resources. Every data element has a single logical name, which is eventually mapped to different physical file names for the purpose of replication and accessibility.
- ***Access Restrictions.*** Even though one of the purposes of Data Grids is facilitating the sharing of results and data for experiments, some users might wish to ensure confidentiality for their data and restrict access to them only to their collaborators. Authentication and authorization in Data Grids involve both coarse-grained and fine-grained access control over shared data collections.

Examples (scientific research fields) for Data Grids

- **The LHC Grid:** A project funded by the European Union to develop a world-wide Grid computing environment for use by high-energy physics researchers around the world collaborating on the LHC (Large Hadron Collider) experiment. It supports storage and analysis of large-scale datasets, from hundreds of terabytes to petabytes, generated by the LHC experiment (http://lhc.web.cern.ch/lhc/).
- **BioInformatics Research Network (BIRN).** BIRN is a national initiative to advance biomedical research through data sharing and online collaboration. Funded by the National Center for Research Resources (NCRR), a component of the US National Institutes of Health (NIH), BIRN provides data-sharing infrastructure, software tools, and strategies and advisory services (http://www.birncommunity.org).
- **International Virtual Observatory Alliance (IVOA).** IVOA is an organization aimed at providing improved access to the ever-expanding astronomical data resources available on-line. It does so by promoting standards for Virtual Observatories, which are a collection of interoperating data archives and software tools utilizing the Internet to form a scientific research environment where astronomical research programs can be conducted. This allows scientists to discover, access, analyze, and combine lab data from heterogeneous data collections (http://www.ivoa.net/).

Data Clouds and Big Data

- Large datasets have mostly been the domain of scientific computing.
- This scenario has recently started to change as massive amount of data are being produced, mined, and crunched by companies providing Internet services such as searching, on-line advertisement, and social media.
- It is critical for such companies to efficiently analyze these huge datasets as they constitute a precious source of information about their customers.
- Logs analysis is an example of data-intensive operation that is commonly performed in this context: companies such as Google have a massive amount of data in the form of logs that are daily processed by using their distributed infrastructure.

Data Clouds and Big Data

- Together with the diffusion of Cloud computing technologies supporting data-intensive computations, the term *Big Data* has become popular.
- This term characterizes the nature of data-intensive computations nowadays and currently identifies datasets that grow so large that they become complex to work with using on-hand database management tools.
- Relational databases and desktop statistics/visualization packages become ineffective for that amount of information required.

Data Clouds and Big Data

- *Big Data* problems are found in non-scientific application domains such as web logs, RFID, sensor networks, social networks, Internet text and documents, Internet search indexing, call detail records, military surveillance, medical records, photography archives, video archives, and large scale e-Commerce.
- In general, *Big Data* applies to datasets whose size is beyond the ability of commonly used software tools to capture, manage, and process the data within a tolerable elapsed time.
- Therefore, *Big Data* sizes are a constantly moving target currently ranging from a few dozen terabytes to many petabytes of data in a single dataset.

Data Clouds and Big Data

- Cloud technologies support data-intensive computing in several ways:
  - By providing a large amount of compute instances on demand that can be used to process and analyze large datasets in parallel.
  - By providing a storage system optimized for keeping large blocks of data and other distributed data store architectures.
  - By providing frameworks and programming APIs optimized for the processing and management of large amount of data. These APIs are mostly coupled with a specific storage infrastructure in order to optimize the overall performance of the system.

Data Clouds and Big Data

- Data Cloud is a combination of these components.
- An example is the ***MapReduce* framework,** which provides the best performance when leveraging the *Google File System* on top of Google's large computing infrastructure.
- Another example is the **Hadoop system,** the most mature, large, and open source Data Cloud. It consists of the *Hadoop Distributed File System (HDFS)* and Hadoop's implementation of *MapReduce*.
- A similar approach is proposed by ***Sector*,** which consists of the *Sector Distributed File System (SDFS)* and a compute service called *Sphere* that allows users to execute arbitrary *User Defined Functions (UDFs)* over the data managed by SDFS.
- ***Greenplum*** uses a shared-nothing MPP (massively parallel processing) architecture based upon commodity hardware. The architecture also integrates MapReduce-like functionality into its platform.
- A similar architecture has been deployed by ***Aster*,** which uses *MPP-based data warehousing* appliance supporting *MapReduce* and targeting 1 PB of data.

Unit 4: Objectives

After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

Technologies for Data Intensive Computing

- Research on databases and the data management industry are indeed at a turning point and new opportunities arise. Some factors contributing to this change are:
  - Growing of popularity of Big Data.
  - Growing importance of data analytics in the business chain.
  - Presence of data in several forms, not only structured.
  - New approaches and technologies for computing.

Storage systems

- Distributed file systems constitute the primary support for the management of data. They provide an interface where to store information in the form of files and later access them for read and write operations.
- 1. Lustre
- 2. IBM General Parallel File System (GPFS)
- 3. Google File System (GFS)
- 4. Sector
- 5. Amazon Simple Storage Service (S3)

1. Lusture

- The Lustre file system is a massively parallel distributed file system that covers the needs of a small workgroup of clusters to a large scale computing cluster.
- The file system is used by several of the Top 500 supercomputing systems including the one rated as the most powerful supercomputer in the June 2012 list.
- Lustre is designed to provide access to petabytes (PBs) of storage, to serve thousands of clients with an IO throughput of hundreds of gigabytes per second (GB/s).
- The system is composed by a metadata server containing the metadata information about the file system and a collection of object storage servers that are in charge of providing storage.
- Users access the file system via a POSIX compliant client, which can be either mounted as a module in the kernel or through a library.
- The file system implements a robust failover strategy and recovery mechanism, making server failures and recoveries transparent to clients.

2. IBM General Parallel File System (GPFS)

- GPFS is the high performance distributed file system developed by IBM providing support for RS/6000 supercomputer and Linux computing clusters.
- GPFS is a multi-platform distributed file system built over several years of academic research and provides advanced recovery mechanisms.
- GPFS is built on the concept of shared disks, where a collection of disks is attached to the file systems nodes by means of some switching fabric.
- The file system makes this infrastructure transparent to users and stripes large files over the disk array also by replicating portion of the file in order to ensure high availability.
- By means of this infrastructure, the system is able to support petabytes of storage, which is accessed at a high throughput and without losing consistency of data.
- Compared to other implementations, GPFS distributes also the metadata of the entire file system and provides transparent access to it, thus eliminating a single point of failure.

3. Google File System (GFS)

- GFS is the storage infrastructure supporting the execution of distributed applications in the Google's computing Cloud.
- The system has been designed to be a fault tolerant, high available, distributed file system built on commodity hardware and standard Linux operating systems.
- It has been designed with the following assumptions:
  - The system is built on top of commodity hardware that often fails.
  - The system stores a modest number of large files, multi-GB files are common and should be treated efficiently, small files must be supported but there is no need to optimize for that.
  - The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
  - The workloads also have many large, sequential writes that append data to files.
  - High-sustained bandwidth is more important than low latency.

4. Sector

- Sector is the storage cloud that supports the execution of data-intensive applications defined according to the Sphere framework.
- It is a user space file system that can be deployed on commodity hardware across a wide-area network.
- Compared to other file systems, Sector does not partition a file into blocks but replicates the entire files on multiple nodes, allowing users to customize the replication strategy for better performance.
- The system's architecture is composed of four nodes: a security server, one or more master nodes, slave nodes, and client machines.
- The security server maintains all the information about access control policies for user and files, whereas master servers coordinate and serve the I/O requests of clients, which ultimately interact with slave nodes to access files.
- The protocol used to exchange data with slave nodes is UDT, which is a lightweight connection-oriented protocol optimized for wide-area networks.

5. Amazon Simple Storage Service (S3)

- Amazon S3 is the on-line storage service provided by Amazon, claimed to support high availability, reliability, scalability, infinite storage, and low latency at commodity cost.
- The system offers a flat storage space organized into buckets, which are attached to an AWS (Amazon Web Services) account.
- Each bucket can store multiple objects, each of them identified by a unique key.
- Objects are identified by unique URLs and exposed through the HTTP protocol, thus allowing a very simple *get-put* semantics.
- Because of the use of the HTTP protocol, there is no need of any specific library for accessing the storage system, whose objects can also be retrieved through the Bit Torrent protocol.
- Bit Torrent is a P2P file sharing protocol used to distribute large amounts of data. The key characteristic of the protocol is the ability to allow users to download a file in parallel from multiple hosts.

Not Only SQL (NoSQL) Systems

- NoSQL cannot be considered a relational database since it is not a monolithic piece of software organizing the information according to the relational model, but, rather, is a collection of scripts that allow users to manage most of the simplest and more common database tasks by using text files as information store.
- Nowadays, the term NoSQL is a big umbrella encompassing all the storage and database management systems that differ in some way from the relational model.
- The general philosophy is to overcome the restrictions imposed by the relational model and to provide more efficient systems.
- This often implies the use of tables to accommodate a larger range of data types or avoiding joins to increase the performance and scale horizontally.

Broad Classification of NoSQL

- *Document stores* (Apache Jackrabbit, Apache CouchDB, SimpleDB, Terrastore).
- *Graphs* (AllegroGraph, Neo4j, FlockDB, Cerebrum).
- *Key-value stores*. This is a macro classification that is further categorized into key-value store on disk, key-value caches in RAM, hierarchically key-value stores, eventually consistent key-value stores, and ordered key-value store.
- *Multi-value databases* (OpenQM, Rocket U2, OpenInsight).
- *Object databases* (ObjectStore, JADE, ZODB).
- *Tabular stores* (Google BigTable, Hadoop HBase, Hypertable).
- *Tuple stores* (Apache River).

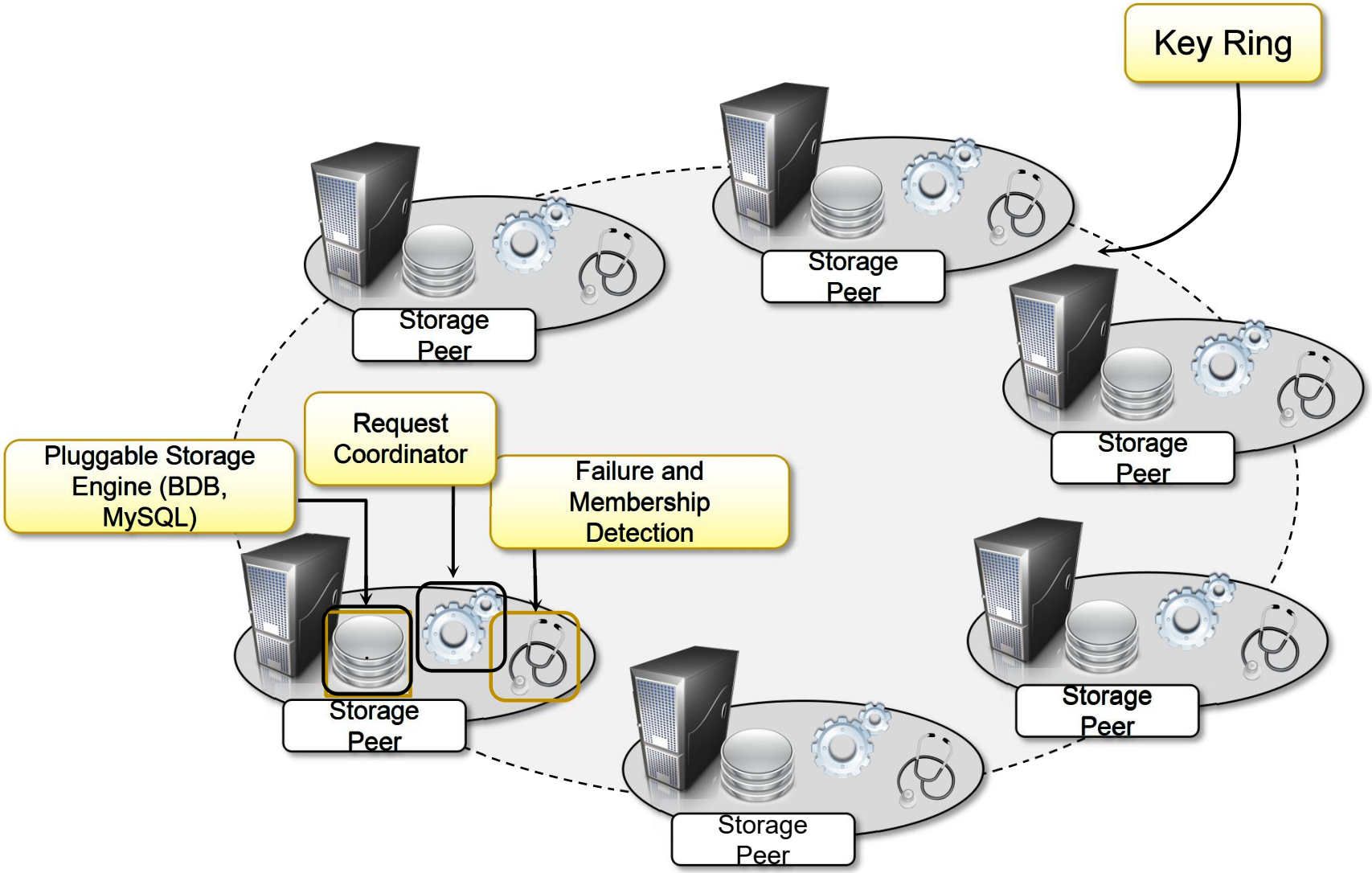
Apache CouchDB and MongoDB

- Document stores, both of them provide a schema-less store where the primary objects are documents organized into a collection of key-value fields. The value of each field can be of type string, integer, float, date, or an array of values.
- The databases expose a RESTful interface and represents data in JSON format. Both of them allow querying and indexing data by using the MapReduce programming model, expose Javascript as a base language for data querying and manipulation rather than SQL, and support large files as documents.
- From an infrastructure point of view, the two systems support data replication and high-availability.
- CouchDB ensures ACID properties on data.
- MongoDB supports *sharding*, which is the ability to distribute the content of a collection among different nodes.

1. Amazon Dynamo

- It is distributed key-value store supporting the management of information of several of the business services offered by Amazon.
- The main goal of Dynamo is to provide an incrementally scalable and highly available storage system.
- Dynamo provides a simplified interface based on a *get/put* semantics, where objects are stored and retrieved with a unique identifier (key).
- This creates an *eventually consistent* model, which means that in the long term all the users will see the same data.

Amazon Dynamo Architecture



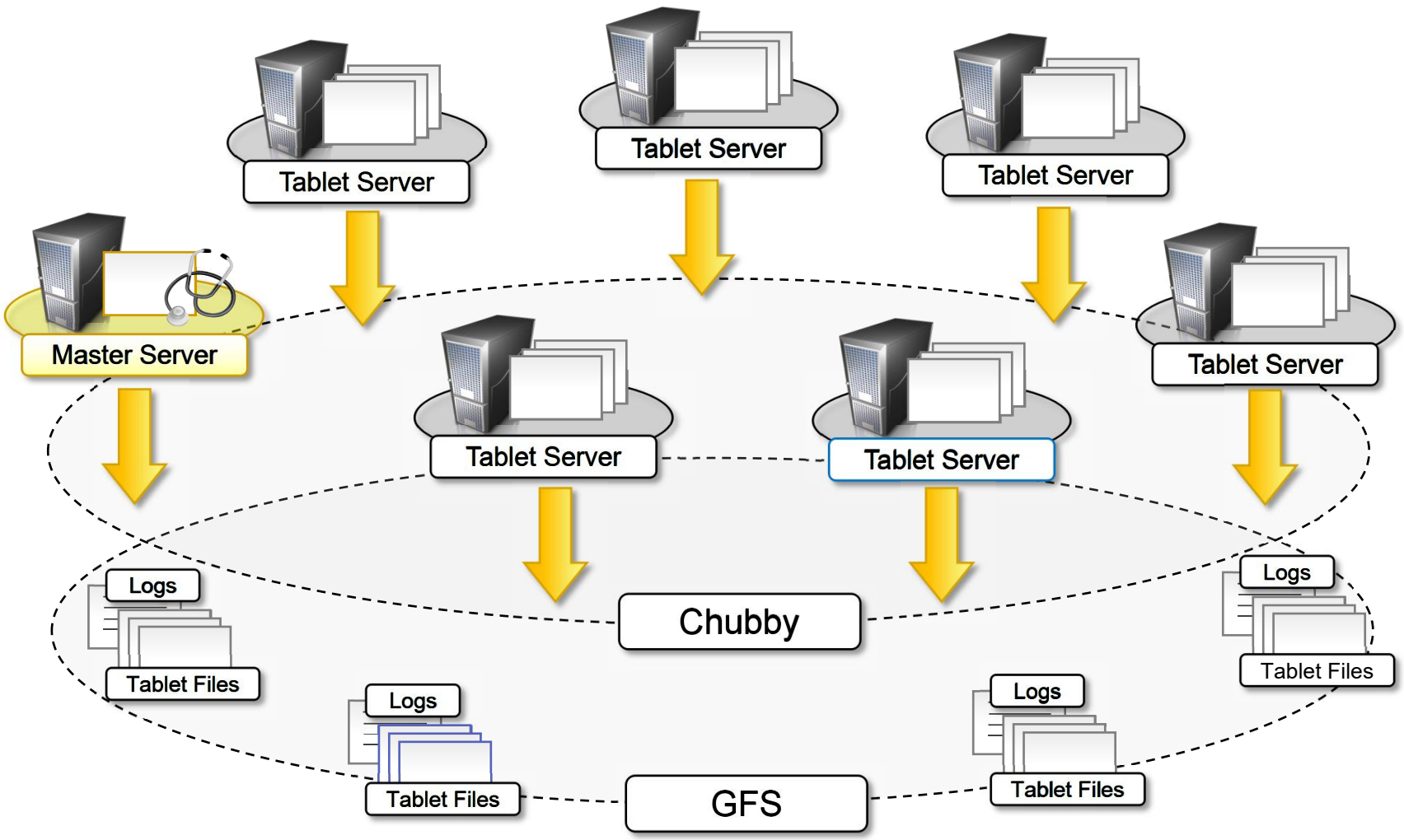
2. Google Bigtable

- Distributed storage system designed to scale up to petabytes of data across thousands of servers.
- Bigtable provides storage support for several Google applications exposing different types of workload: from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users.
- The key design goals of Bigtable are wide applicability, scalability, high performance, and high availability.
- To achieve these goals, Bigtable organizes the data storage in tables whose rows are distributed over the distributed file system supporting the middleware, which is the Google File System.
- A table is organized in rows and columns, columns can be grouped in column family, which allow for specific optimization for better access control, the storage and the indexing of data.

Google Bigtable

- Besides the basic data access, Bigtable APIs also allow more complex operations such as single row transactions and advanced data manipulation by means of the Sazwall scripting language or the MapReduce APIs.
- Sazwall is an interpreted procedural programming language developed at Google for the manipulation of large quantities of tabular data.
- It includes specific capabilities for supporting statistical aggregation of values read or computed from the input and other features that simplify the parallel processing of petabytes of data.

Bigtable Architecture



Apache Cassandra

- distributed object store for managing large amounts of structured data spread across many commodity servers.
- The system is designed to avoid a single point of failure and offer a highly reliable service. Cassandra was initially developed by Facebook and now it is part of the Apache incubator initiative.
- Currently, it provides storage support for several very large web applications such as *Facebook* itself, *Digg*, and *Twitter*.
- Cassandra is defined as a second generation distributed database that builds on the concept of *Amazon Dynamo* which follows a fully distributed design and *Google Bigtable,* from which inherits the “Column Family” concept.

Cassandra

- The data model exposed by Cassandra is based on the concept of table that is implemented as a distributed multi-dimensional map indexed by a key.
- The value corresponding to a key is a highly structured object and constitutes the row of a table. Cassandra organizes the row of a table into columns and sets of columns can be grouped into column families.
- The APIs provided by the system to access and manipulate the data are very simple: insertion, retrieval, and deletion. The insertion is performed at row level, while retrieval and deletion can operate at column level.

Hadoop HBase

- HBase is the distributed database supporting the storage needs of the Hadoop distributed programming platform.
- HBase is designed by taking inspiration from Google Bigtable and its main goal is to offer real time read/write operations for tables with billions of rows and millions of columns by leveraging clusters of commodity hardware.
- The internal architecture and logic model of HBase is very similar to Google Bigtable and the entire system is backed by the Hadoop Distributed File System (HDFS), which mimics the structure and the services of GFS.

Unit 4: Objectives

After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

Programming Platforms

- Platforms for programming data intensive applications provide abstractions helping to express the computation over a large quantity of information and runtime systems able to manage efficiently huge volumes of data.
- Traditionally, database management systems based on the relational model have been used to express the structure and the connections between the entities of a data model.
- This approach has proven to be not successful in the case of "Big Data" where information is mostly found unstructured or semi-structured and where data is most likely to be organized in files of large size or a huge number of medium size files, rather than rows in a database.

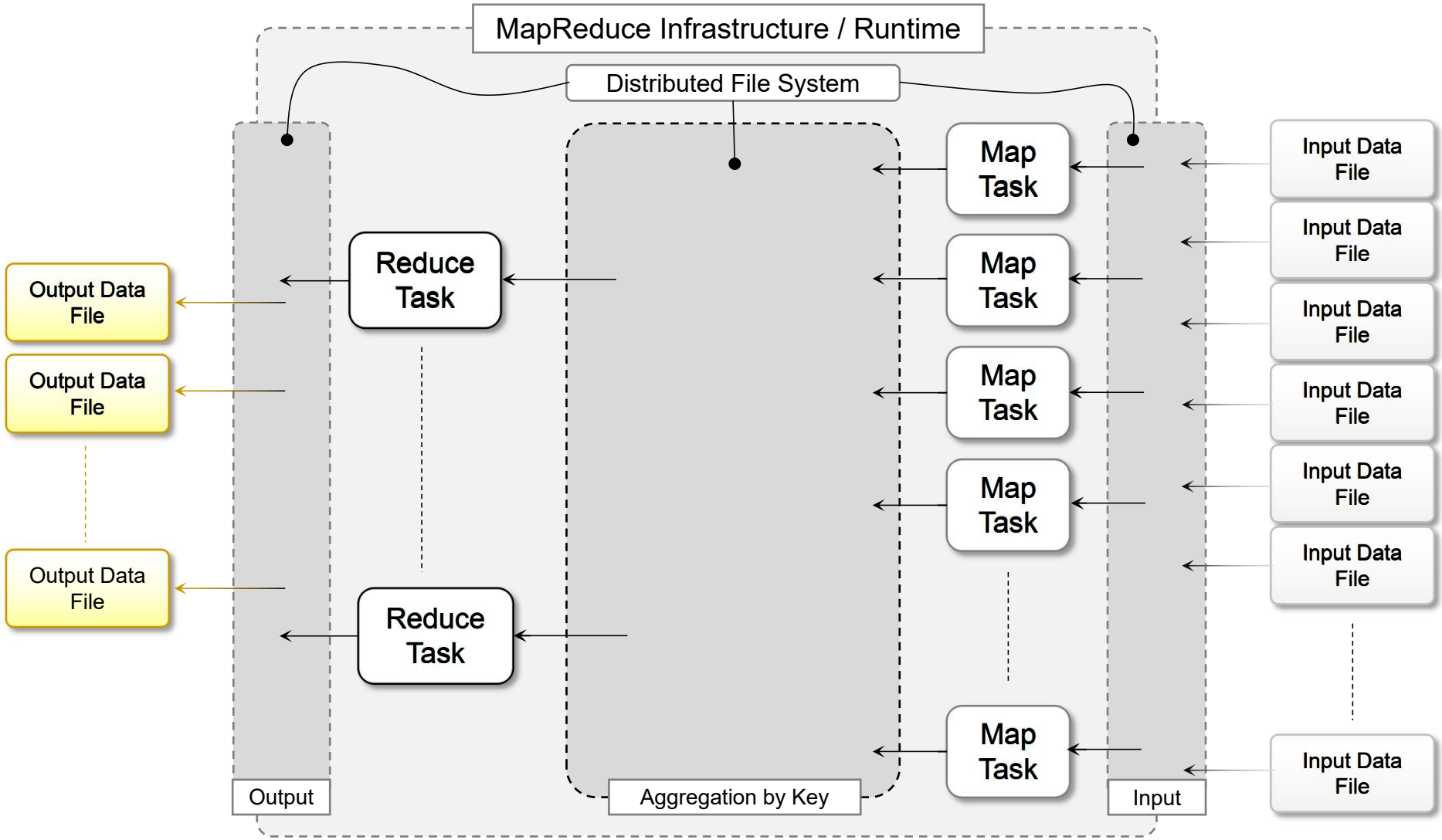
The MapReduce Programming model

- MapReduce is a programming platform **introduced by Google for processing large quantities of data.**
- It expresses the computation logic of an application into two simple functions: ***map* and *reduce*.**
- Data transfer and management is completely handled by the distributed storage infrastructure (i.e. the Google File System), which is in charge of providing access to data, replicate files, and eventually move them where needed.
- Therefore, developers do not have to handle anymore these issues and are provided with an interface that presents data at a higher level: as a collection of key-value pairs.
- The computation of MapReduce applications is then organized in a workflow of **map and reduce operations** that is entirely controlled by the runtime system and developers have only to specify how the map and reduce functions operate on the key value pairs.

MapReduce programming model

- More precisely, the model is expressed in the form of two functions, which are defined as follows:
  - ***map (k1,v1) → list(k2,v2)***
  - ***reduce(k2,list(v2)) → list(v2)***
- The *map* function reads a key-value pair and produces a list of key-value pairs of different types.
- The *reduce* function reads a pair composed by a key and a list of values and produces a list of values of the same type.
- The types (*k1,v1,k2,kv2*) used in the expression of the two functions provide hints on how these two functions are connected and are executed to carry out the computation of a MapReduce job: the output of map tasks is aggregated together by grouping the values according to their corresponding keys and constitute the input of reduce tasks that, for each of the keys found, reduces the list of attached values to a single value.
- Therefore, the input of a MapReduce computation is expressed as a collection of key-value pairs <*k1,v1*> and the final output is represented by a list values: *list(v2).*

MapReduce computation Workflow



MapReduce computation workflow

- The computation model expressed by MapReduce is very straightforward and allows a greater productivity for those who have to code the algorithms for processing huge quantities of data.
- This model has proven to be successful in the case of Google, where the majority of the information that needs to be processed is stored in textual form and represented by web pages or log files.
- Some of the examples that show the flexibility of MapReduce are the following….

1. Distributed Grep

- It performs the recognition of patterns within text streams, and performed across a wide set of files.
- MapReduce provides a parallel and faster execution of this operation.
- The input file is a plain text file and the map function emits a line into the output each time it recognizes the given pattern.
- The reduce task aggregates all the lines emitted by the map tasks into a single file.

2. Count of URL-Access Frequency.

- MapReduce is used to distribute the execution of web-server log parsing.
- In this case the map function takes as input the log of a web server and emits into the output file a key-value pair *<URL,1>* for each page access recorded in the log.
- The reduce function aggregates all these lines by the corresponding URL thus summing the single accesses and outputs a <*URL, total-count*> pair.

3. Reverse Web-Link Graph

- The Reverse web-link graph keeps track of all the possible web pages that might lead to a given link.
- In this case input files are simple HTML pages that are scanned by map tasks emitting <*target, source*> pairs for each of the links found given in the web page *source*.
- The reduce task will collate all the pairs that have the same target into a <*target, list(source)*> pair.
- The final result is given one or more files containing these mappings.

3. Term-Vector per Host.

- A term vector recaps the most important words occurring in a set of documents in the form of *list(<word, frequency>)*, where the number of occurrences of a word is taken as a measure of its importance.
- MapReduce is used to provide a mapping between the origin of a set of document, obtained as the host component of the URL of a document, and the corresponding term vector.
- In this case, the map task creates a pair *<host, term-vector>* for each text document retrieved and the reduce task aggregates the term vectors corresponding to documents retrieved from the same host.

4. Inverted Index

- The inverted index contains information about the presence of words in documents.
- This information is useful to allow fast full text searches if compared to direct document scans.
- In this case, the map task takes as input a document and for each document it emits a collection of <*word, document-id*>.
- The reduce function aggregates the occurrences of the same word, producing a pair <*word, list(document-id)*>.

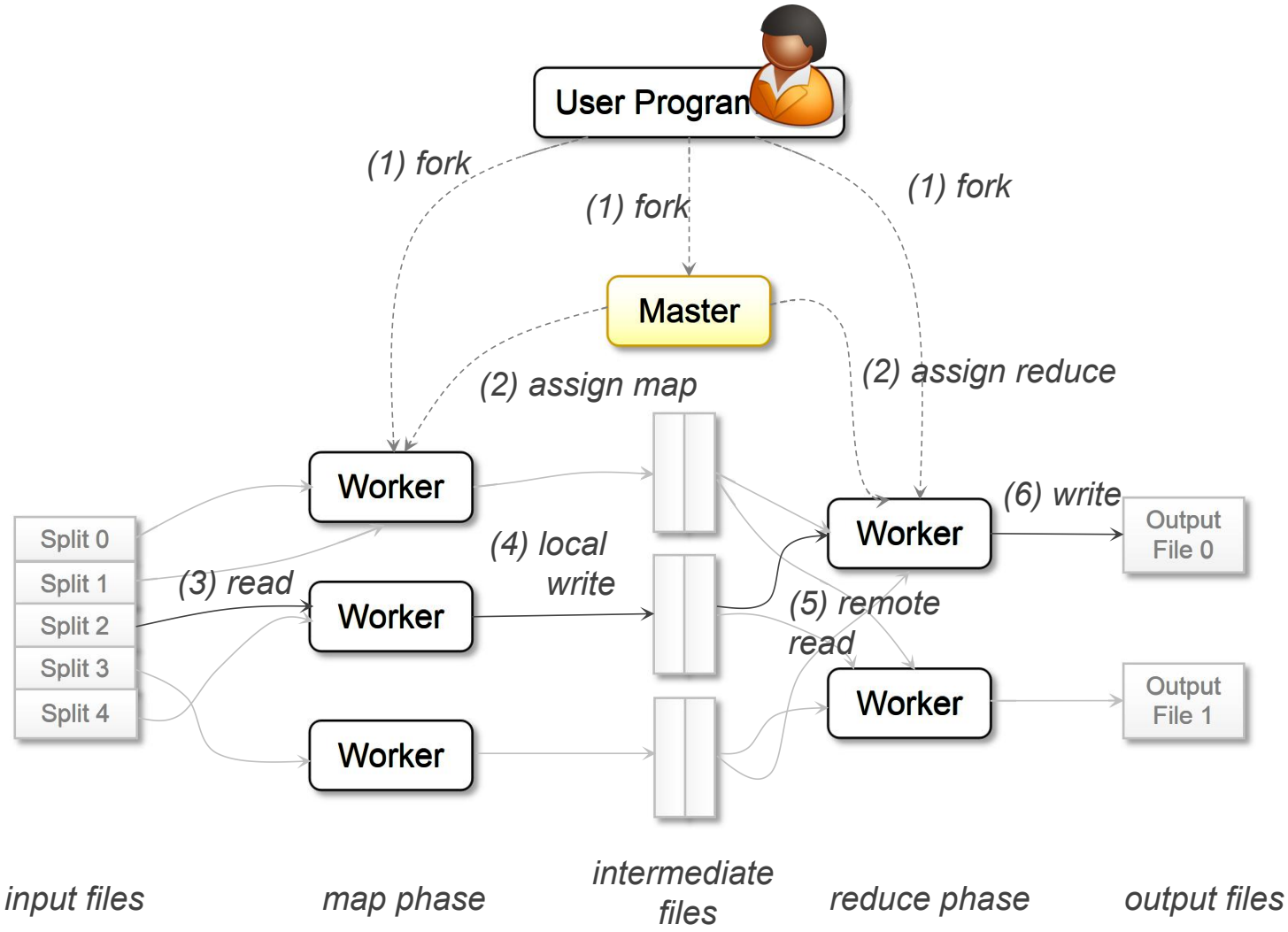
6. Distributed Sort.

- In this case, MapReduce is used to parallelize the execution of a sort operation over a large number of records.
- This application mostly rely on the properties of the MapReduce runtime, which sorts and creates partitions of the intermediate files, rather than in the operations performed in the map and reduce tasks.
- Indeed, these are very simple: the map task extracts the key from a record and emits a <key, record> pair for each record; the reduce task will simply copy through all the pairs.
- The actual sorting process is performed by the MapReduce runtime which will emit and partition the key value pair by ordering them according to the key.

MapReduce computation-stages

- In general, any computation that can be expressed in the form of **two major stages** can be represented in terms of MapReduce computation. These stages are:
  - **Analysis.** This phase operates directly to the data input file and corresponds to the operation performed by the map task. Moreover, the computation at this stage is expected to be embarrassingly parallel since map tasks are executed without any sequencing or ordering.
  - **Aggregation.** This phase operates on the intermediate results and it is characterized by operations which are aimed at aggregating, summing, and or elaborating the data obtained at the previous stage to present it into its final form. This is the task performed by the reduce function.

Google MapReduce Infrastructure Overview



Unit 4: Objectives

After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

Variations and Extensions of MapReduce

- For extending MapReduce application space and providing an easier interface to developers for designing distributed algorithms.
- Hadoop
- Pig
- Hive
- Map-Reduce Merge
- Twister

1. Hadoop

- Apache Hadoop is a collection of software projects for reliable and scalable distributed computing.
- It is an open source implementation of the MapReduce framework supported by a GFS-like distributed file system.
- The initiative consists of mostly two projects: Hadoop Distributed File System (HDFS) and Hadoop MapReduce.
- The former is an implementation of the Google File System, while the latter provides the same features and abstractions of Google MapReduce.

2. Pig

- allows the analysis of large data sets. Developed as an Apache project.
- Pig infrastructure's layer consists of a compiler for a high-level language that produces a sequence of MapReduce jobs.
- Developers can express their data analysis programs in a textual language called *Pig Latin*, which exposes a SQL-like interface and it is characterized by major expressiveness, reduced programming effort, and a familiar interface with respect to MapReduce.

3.Hive

- **It** is another Apache initiative that provides a data warehouse infrastructure on top of Hadoop MapReduce.
- It provides tools for easy data summarization, ad-hoc queries, and analysis of large datasets stored in Hadoop MapReduce files.
- The major advantage of Hive resides in the ability to scale out and in the ability of providing a data warehouse infrastructure in environments where there is already a Hadoop system running.

4. Map-Reduce-Merge

- It is an extension to the MapReduce model introducing a third phase to the standard MapReduce pipeline—the Merge phase—that allows efficiently merging data already partitioned and sorted (or hashed) by map and reduce modules.
- The Map-Reduce-Merge framework simplifies the management of heterogeneous related datasets and provides an abstraction able to express the common relational algebra operators as well as several join algorithms.

5. Twister

- **Twister:** an extension to the MapReduce model that allows the creation of iterative executions of MapReduce jobs. With respect to the normal MapReduce pipeline the model proposed by twister proposes the following extensions:
  1. Configure Map
  2. Configure Reduce
  3. While Condition Holds True Do
     a) Run MapReduce
     b) Apply Combine Operation to Result
     c) Update Condition
  4. Close

Alternatives to MapReduce-sphere

- **1. Sphere:** distributed processing engine that leverages the Sector Distributed File System (SDFS).
  - Sphere implements the stream processing model (***Single Program Multiple Data***) and allows developer to express the computation in terms of ***User Defined Functions (UDF)*** which are run against the distributed infrastructure.
  - Sphere strongly leverages the **Sector distributed file systems** and it is built on top of the Sector's API for data access. User defined functions are expressed in terms of programs that reads and write streams.
  - A stream is a data structure that provides access to a collection of data segments mapping one or more files in the SDFS.
  - The collective execution of UDFs is achieved through the distributed execution of ***Sphere Process Engines (SPEs)*** which are assigned with a given stream segment.
  - The execution model is master-slave model that is client controlled: a Sphere client sends a request for processing to the master node that returns the list of available slaves and the client will choose the slaves where to execute Sphere processes.

Alternatives to MapReduce: All- Pairs

- **2. All-Pairs :** an abstraction and a run-time environment for the optimized execution of **data-intensive workloads-** that is common in many scientific computing domains:

  All-pairs(A:set, B:set, F:function) → M:matrix

- Examples of problems that can be represented into this model can be found in the field of biometrics where similarity matrices are composed as a result of the comparison of several images containing subject pictures.
- Another example is constituted by several applications and algorithms in data mining. The model expressed by the All-Pairs function can be easily solved by the following algorithm:

```
1.For each $i in A
2.  For each $j in B
3.    Submit job F $i $j
```

Alternatives to MapReduce- DryadLINQ

- **3. DryadLINQ:** Microsoft Research project investigating programming models for writing parallel and distributed programs to scale from a small cluster to a large data-center.
- The aim of Dryad is to provide an infrastructure for automatically parallelizing the execution of application without requiring the developer to know about distributed and parallel programming.
- In Dryad, developers can express distributed applications as a set of sequential programs that are connected together by means of channels.
- More precisely, a Dryad computation can be expressed in terms of a directed acyclic graph where nodes are the sequential programs and vertices represent the channels connecting such programs.

Unit 4: Objectives

After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

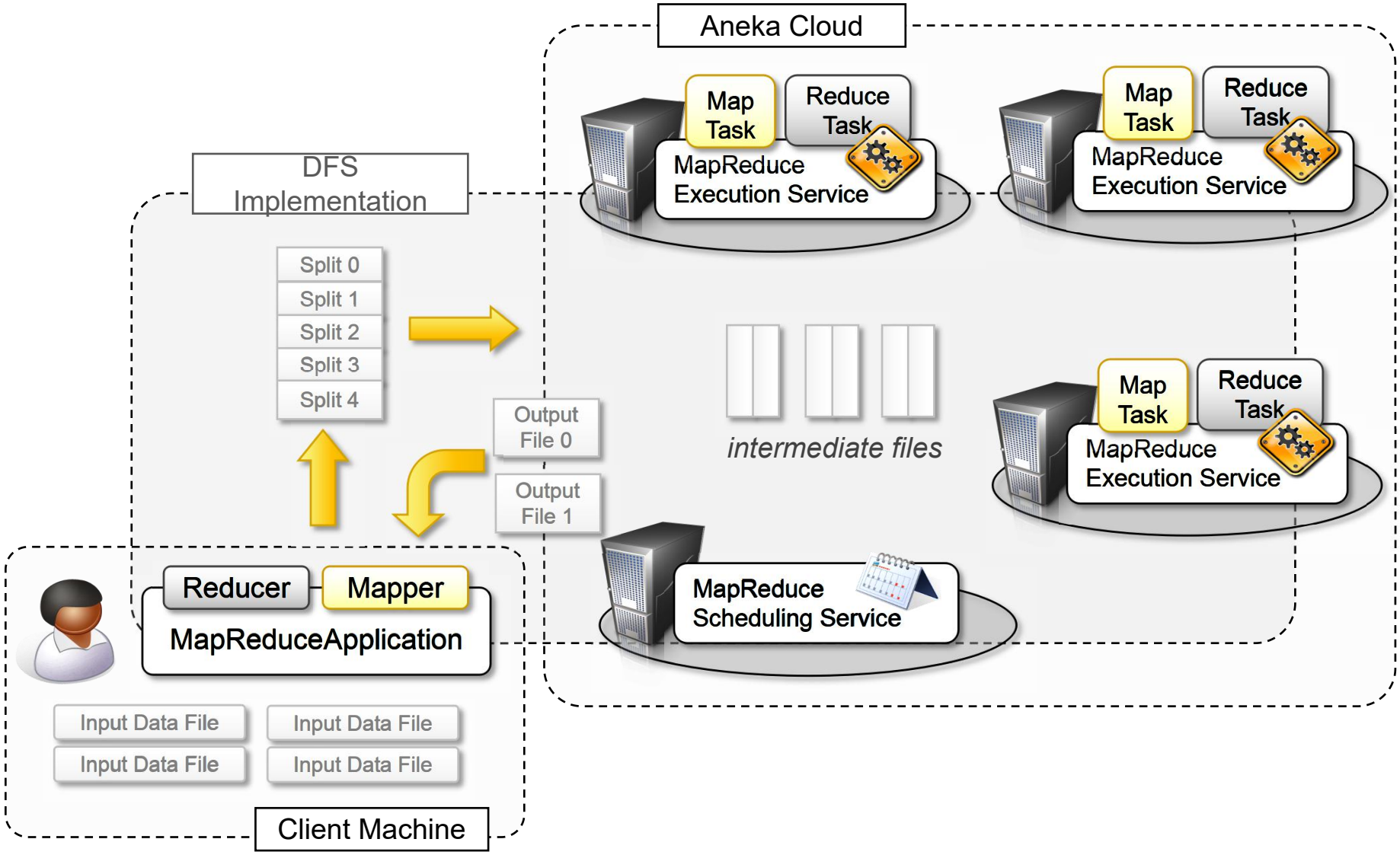
Aneka MapReduce Programming

- Aneka provides an implementation of the MapReduce abstractions by following the reference model introduced by Google and implemented by Hadoop.
- MapReduce is supported as one of the available programming models that can be used to develop distributed applications.

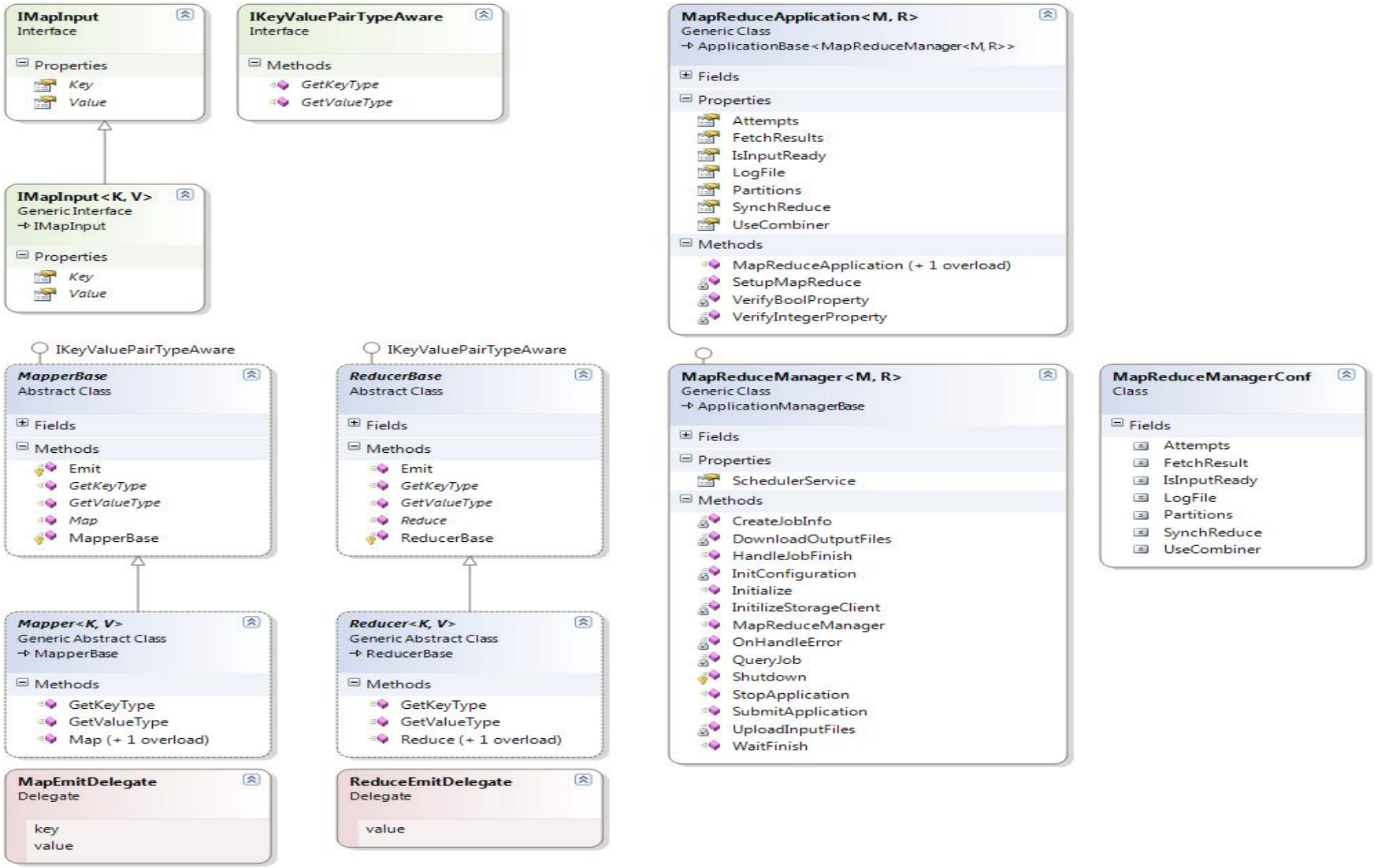
MapReduce Programming model-Aneka

- A MapReduce job in Google MapReduce or Hadoop corresponds to the execution of a MapReduce application in Aneka.
- The application instance is specialized with components that identify the map and reduce functions to use. These functions are expressed in the terms of a **Mapper and Reducer** class that are extended from the AnekaMapReduce APIs. The **runtime support** is constituted by **three main elements:**
  - ***MapReduce Scheduling Service***, which plays the role of the master process in the Google and Hadoop implementation.
  - *MapReduce* ***Execution Service,*** which plays the role of the worker process in the Google and Hadoop implementation.
  - A specialized **distributed file system** that is used to move data files.

Aneka MapReduce Infrastructure



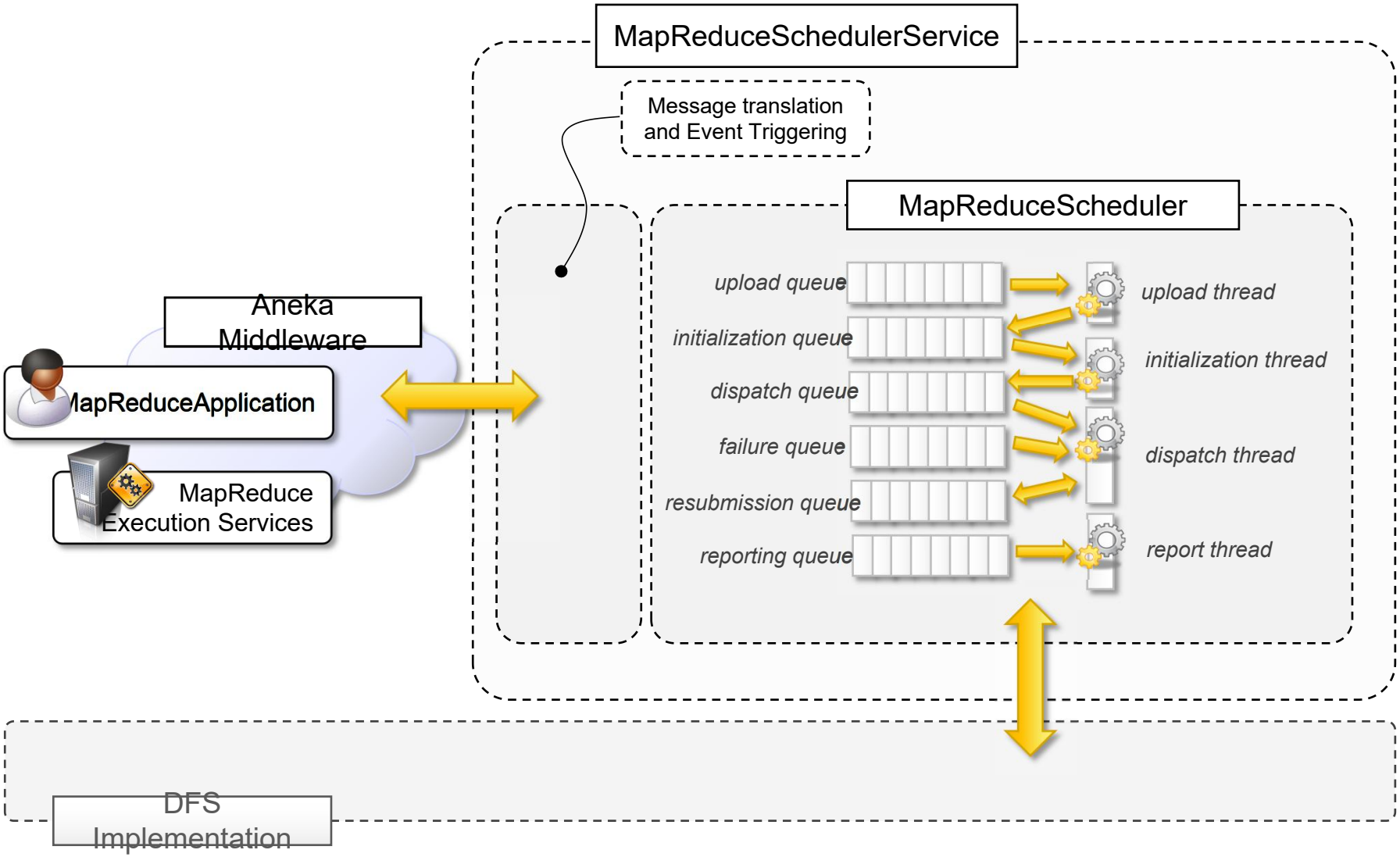
Programming abstractions



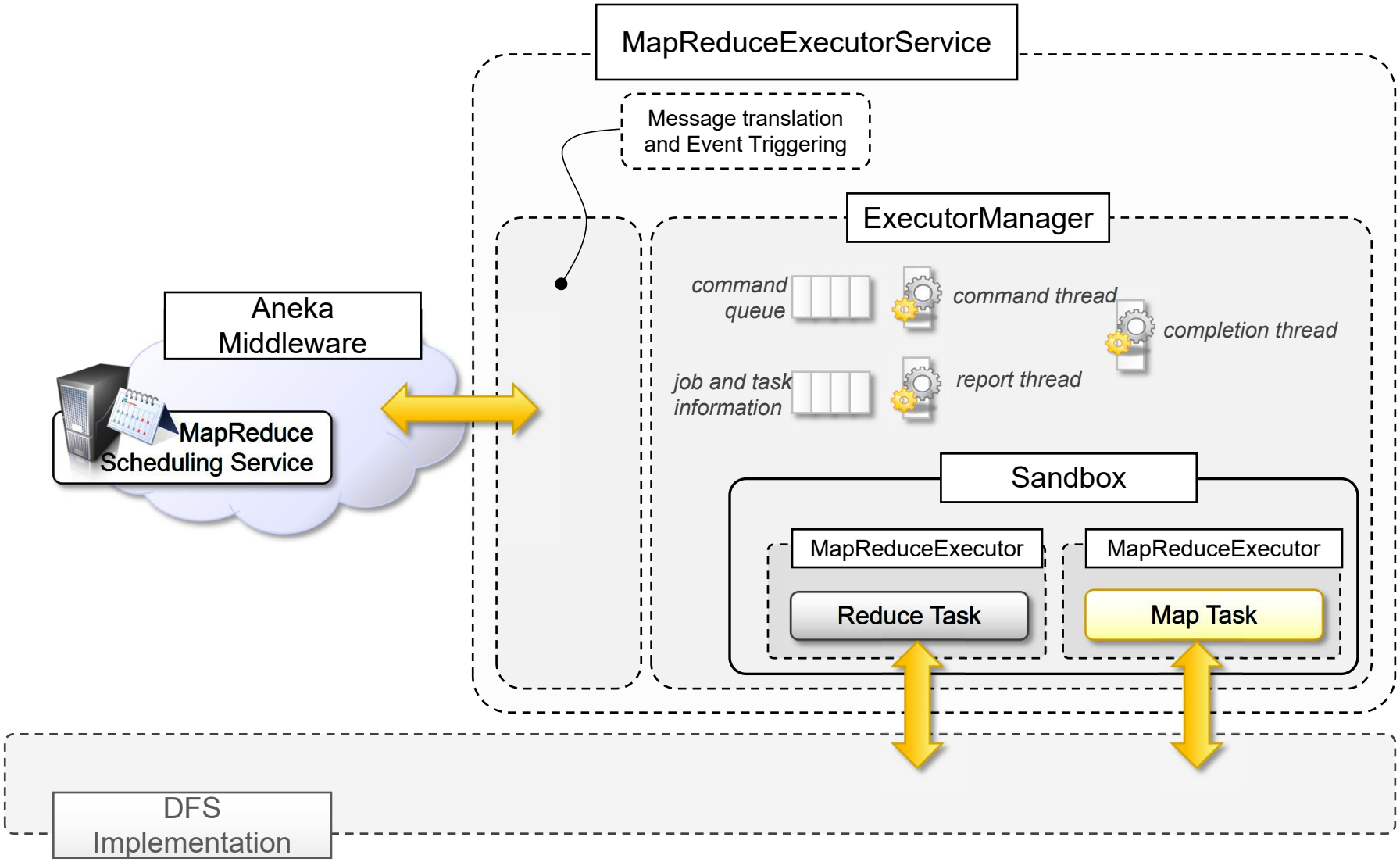
Runtime support

- Contains collection of services that deal with scheduling and executing MapReduce Tasks.
- **MapReduce Scheduling service and**
- **MapReduce Execution Service**
- These services integrate with the existing services of the framework in order to provide persistence, application accounting etc.

Scheduling Service Architecture



Execution Service Architecture



Unit 4: Objectives

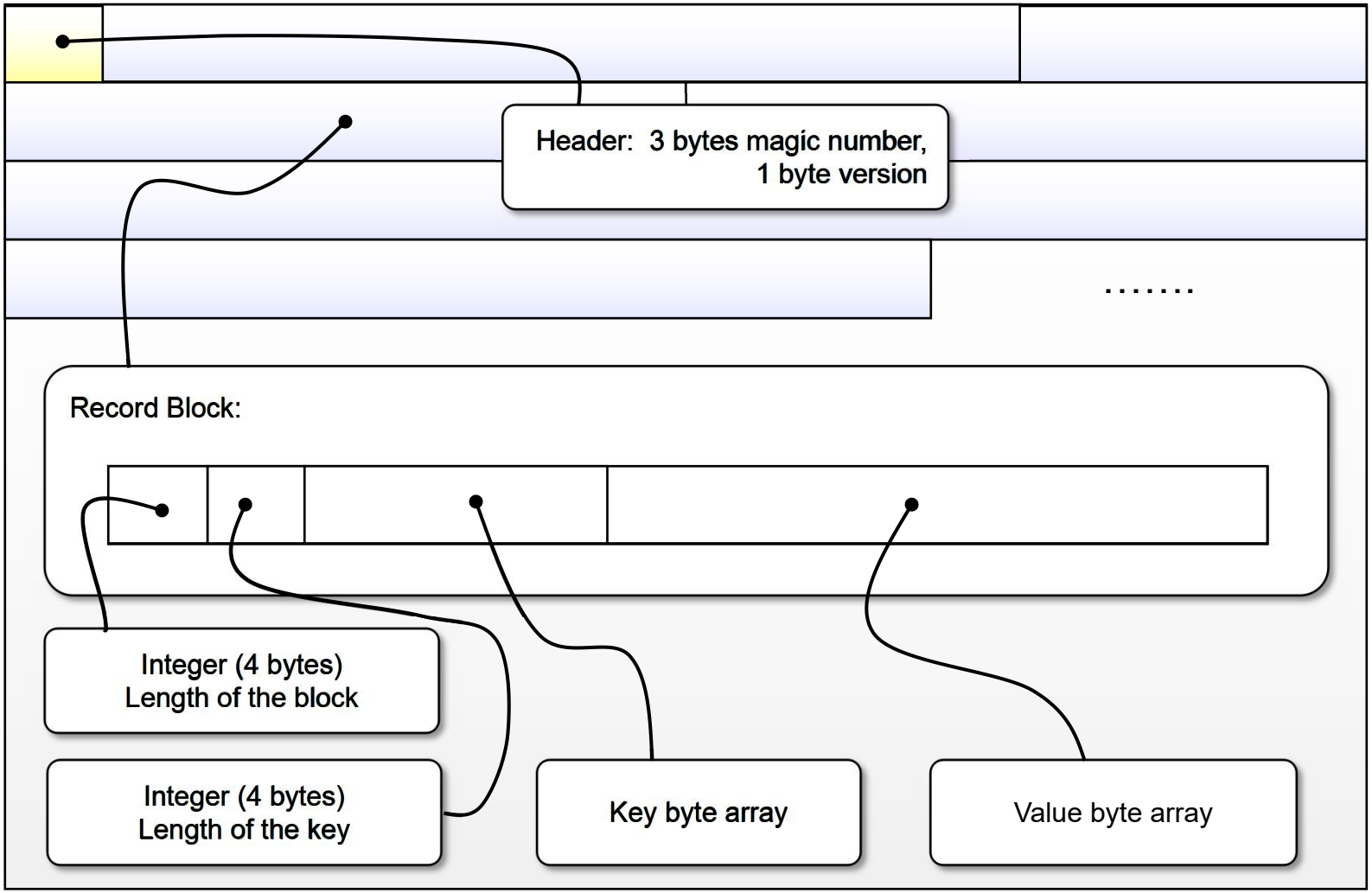
After completing this unit you should be able to

- *Data Intensive Computing: MapReduce Programming*
- *Characterizing Data Intensive Computations*
- *Historical Perspective*
- *Technologies for Data-Intensive Computing*
- *Programming Platforms*
- *Aneka MapReduce Programming Model*
- *Variations and Extensions of MapReduce*
- *Aneka MapReduce Programming*
- *Distributed File System Support*
- *Example Application*
- *Summary*

Distributed File System Support

- MapReduce has been designed to process large quantities of data stored in files of large dimensions.
- Therefore, the support provided by a distributed file system, which can leverage multiple nodes for storing data, is more appropriate.
- Distributed file system implementations guarantee high availability and a better efficiency by means of replication and distribution.
- Moreover, the original MapReduce implementation assumes the existence of a distributed and reliable storage; hence, the use of a distributed file system for implementing the storage layer is natural.

Aneka MapReduce Data File Format



END OF MODULE 4