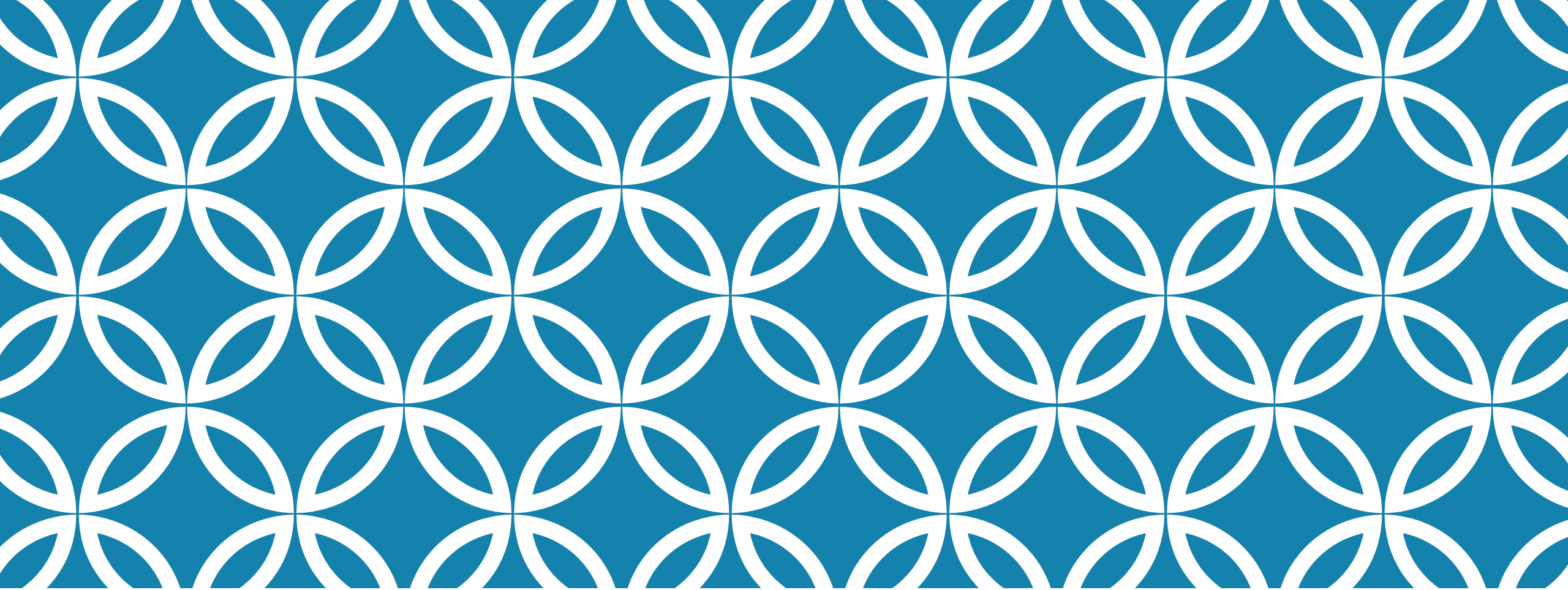
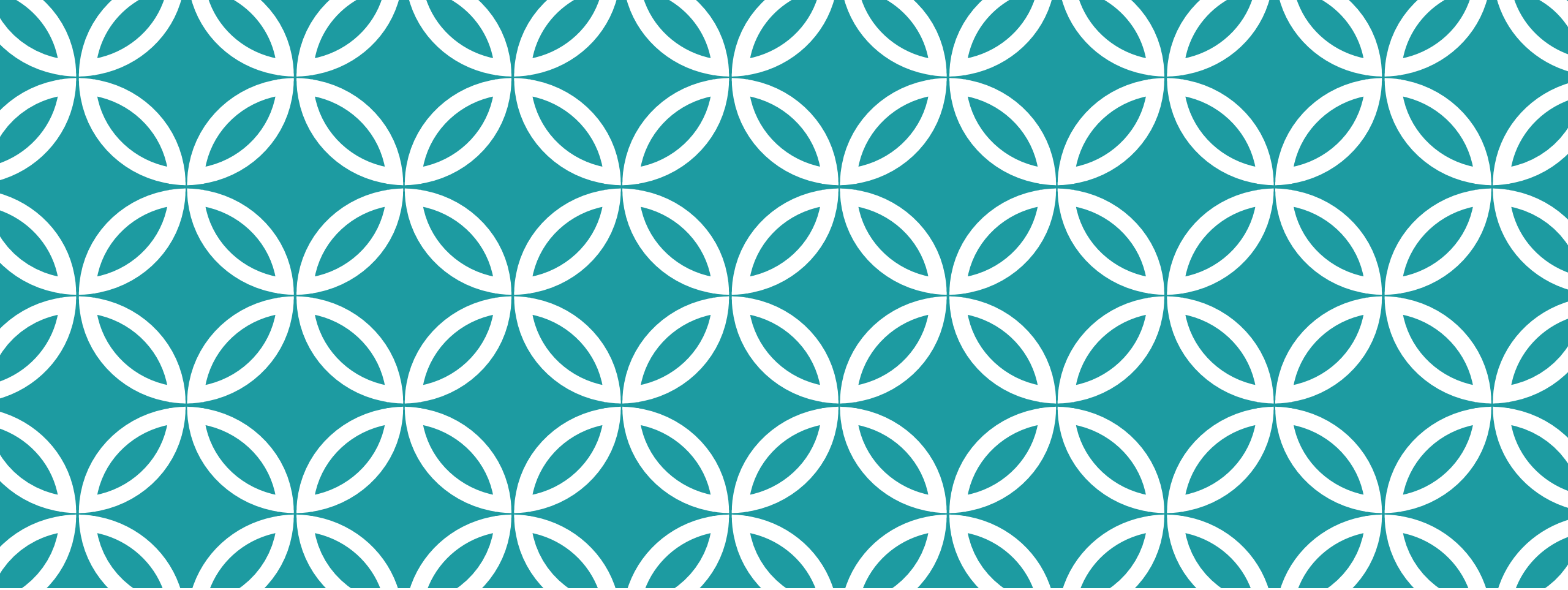


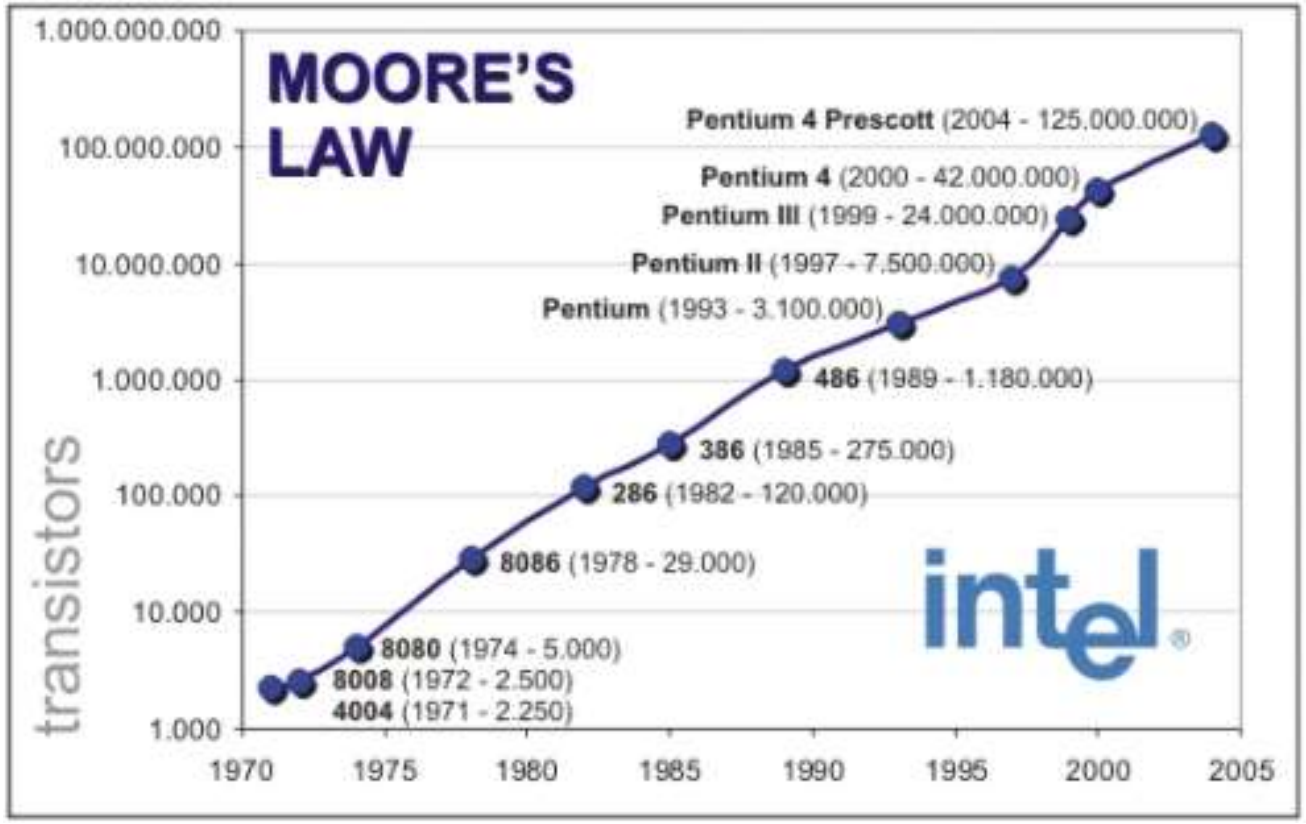
BIG DATA ANALYTICS AND HADOOP

Atul Kahate



MOTIVATION FOR BIG DATA

GORDON MOORE



MOORE'S LAW

MOORE'S LAW AND PARALLEL PROCESSING

Moore's Law, which says that the performance of computers has historically doubled approximately every 2 years, initially helped computing resources to keep pace with data growth.

However, this pace of improvement in computing resources started tapering off around 2005.

The computing industry started looking at other options, namely parallel processing to provide a more economical solution.

If one computer could not get faster, the goal was to use many computing resources to tackle the same problem in parallel.

Hadoop is an implementation of the idea of multiple computers in the network applying **MapReduce** to scale data processing.

The evolution of cloud-based computing through vendors such as Amazon, Google, and Microsoft provided a boost to this concept because we can now rent computing resources for a fraction of the cost it takes to buy them.

WHAT IS BIG DATA?

Any dataset that cannot be processed or (in some cases) stored using the resources of a single machine to meet the required service level agreements (SLAs).

Consider a simple example. If the average size of the job processed by a business unit is 200 GB, assume that we can read about 50 MB per second.

Given the assumption of 50 MB per second, we will need 2 seconds to read 100 MB of data from the disk sequentially, and it would take us approximately 1 hour to read the entire 200 GB of data.

Now imagine that this data was required to be processed in under 5 minutes.

If the 200 GB required per job could be evenly distributed across 100 nodes, and each node could process its own data (consider a simplified use-case such as simply selecting a subset of data based on a simple criterion: SALES_YEAR>2001), discounting the time taken to perform the CPU processing and assembling the results from 100 nodes, the total processing can be completed in under 1 minute.

DATA STORAGE AND ANALYSIS

Although the storage capacities of hard drives have increased massively over the years, access speeds, i.e. the rate at which data can be read from drives have not kept up.

One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s, so you could read all the data from a full drive in around five minutes.

Over 30 years later, 20-terabyte drives are the norm, but the transfer speed is around 100-200 MB/s, so it takes more than two and a half hours to read all the data off the disk.

The obvious way to reduce the time is to read from multiple disks at once.

Imagine if we had 100 drives, each holding one hundredth of the data.

Working in parallel, we could read the data in under two minutes.

PROBLEMS AND SOLUTIONS

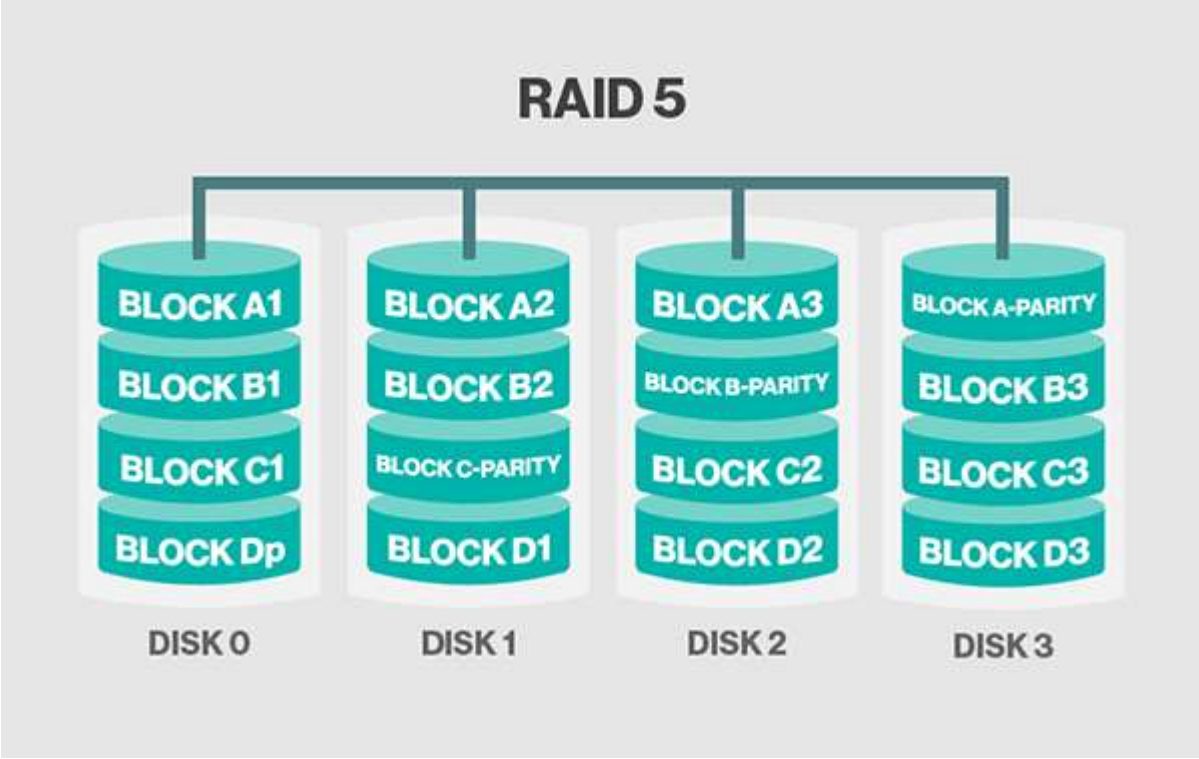
Hardware failure

- As soon as you start using many pieces of hardware, the chance that one will fail is fairly high.
- A common way of avoiding data loss is through replication: redundant copies of the data are kept by the system so that in the event of failure, there is another copy available.
- This is how RAID works, for instance, although Hadoop's filesystem, the **Hadoop Distributed Filesystem (HDFS)**, takes a slightly different approach.

Data processing

- Most analysis tasks need to be able to combine the data in some way, and data read from one disk may need to be combined with data from any of the other 99 disks.
- Various distributed systems allow data to be combined from multiple sources, but doing this correctly is very challenging.
- **MapReduce** provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values.

RAID CONCEPT



HARDWARE "SCALE-UP" VERSUS "SCALE-OUT"

Traditional approach for handling scale: Scale-up

- Use a very powerful server
- Upgrade the hardware – better processor, RAM, storage capacity
- Creates single point of failure or else needs more backups and other complications

Google approach: Scale-out

- Instead of buying a single $1 million machine, buy 200 $5,000 machines
- If capacity expansion is needed, add more machines

GOOGLE'S BIG DATA STRATEGY

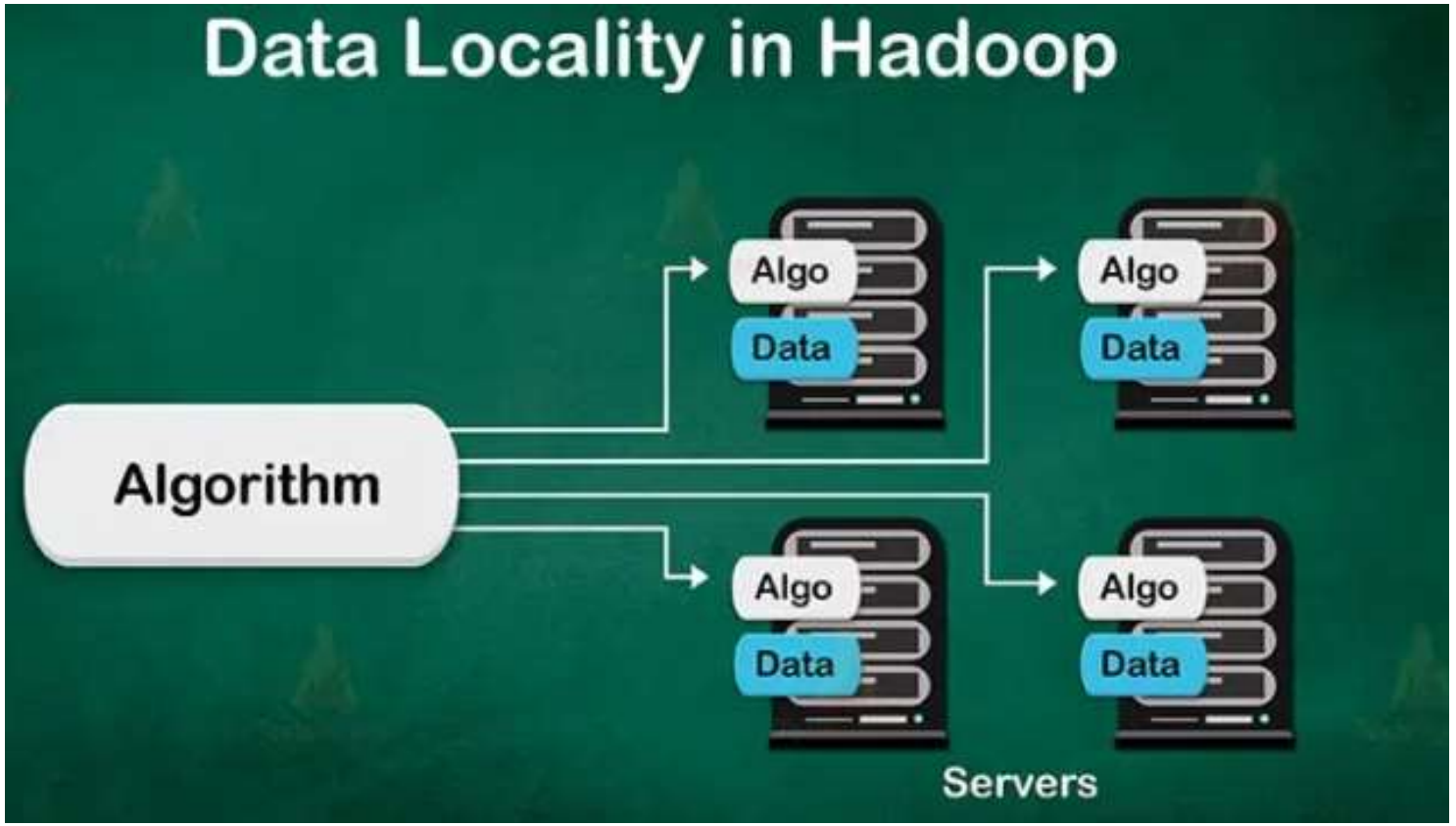
Anything can fail any time, so software should be written accordingly

Use commodity (i.e. not vendor-specific) components

Write code for scale-out, not scale-up

- May put limitations such as JOIN may have to work across multiple machines now
- If we abandon C in ACID (consistency), it becomes much easier to parallelize operations – but then programmers need to take care of consistency

HADOOP DATA LOCALITY



APPLICATIONS ARE MOVED TO WHERE THE DATA IS

In Java/.NET/PHP, the three-tier architecture was drilled into us.

In the three-tier programming model, the data is processed in the centralized application tier after being brought into it over the network. (See next slide)

We are used to the notion of data being distributed but the application being centralized.

Big Data cannot handle this network overhead. Moving terabytes of data to the application tier will saturate the networks and introduce considerable inefficiencies, possibly leading to system failure.

In the Big Data world, the data is distributed across nodes, but the application moves to the data.

It is important to note that this process is not easy.

Not only does the application need to be moved to the data but all the dependent libraries also need to be moved to the processing nodes.

If your cluster has hundreds of nodes, it is easy to see why this can be a maintenance/deployment nightmare.

Hence Big Data systems are designed to allow you to deploy the code centrally, and the underlying Big Data system moves the application to the processing nodes prior to job execution.

3-TIER ARCHITECTU

Presentation tier

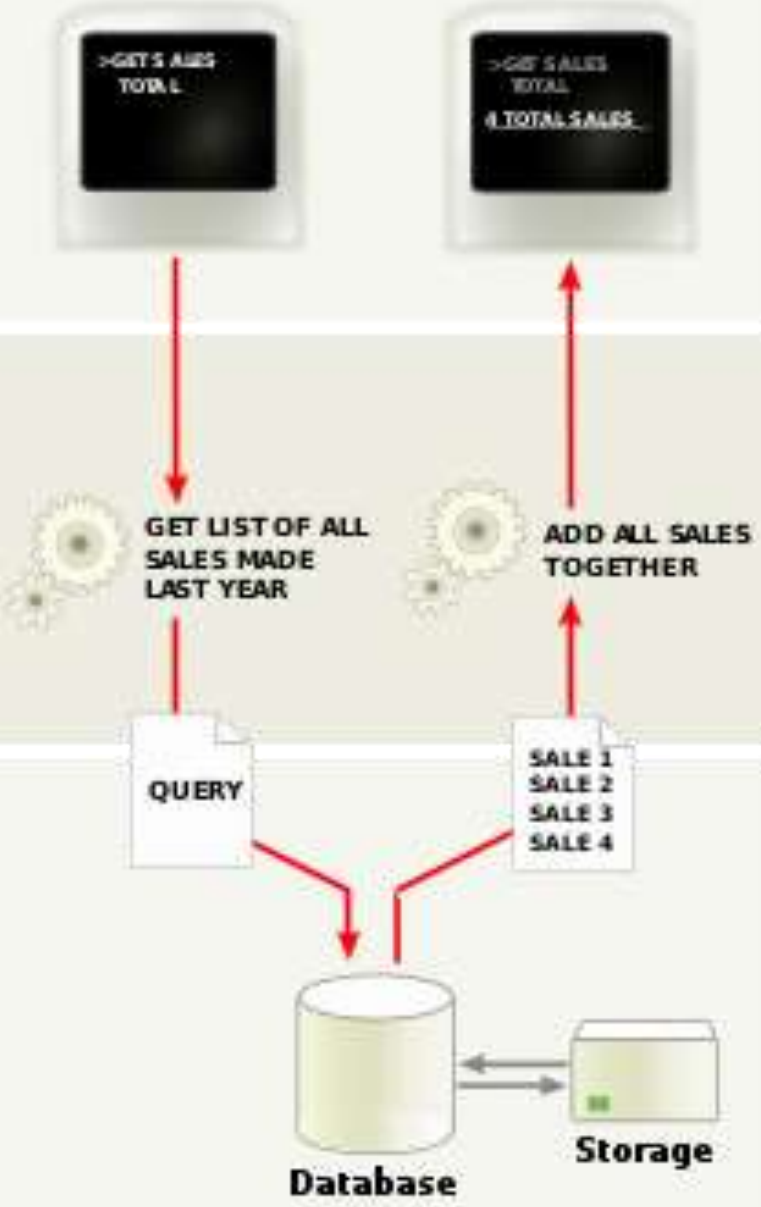
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

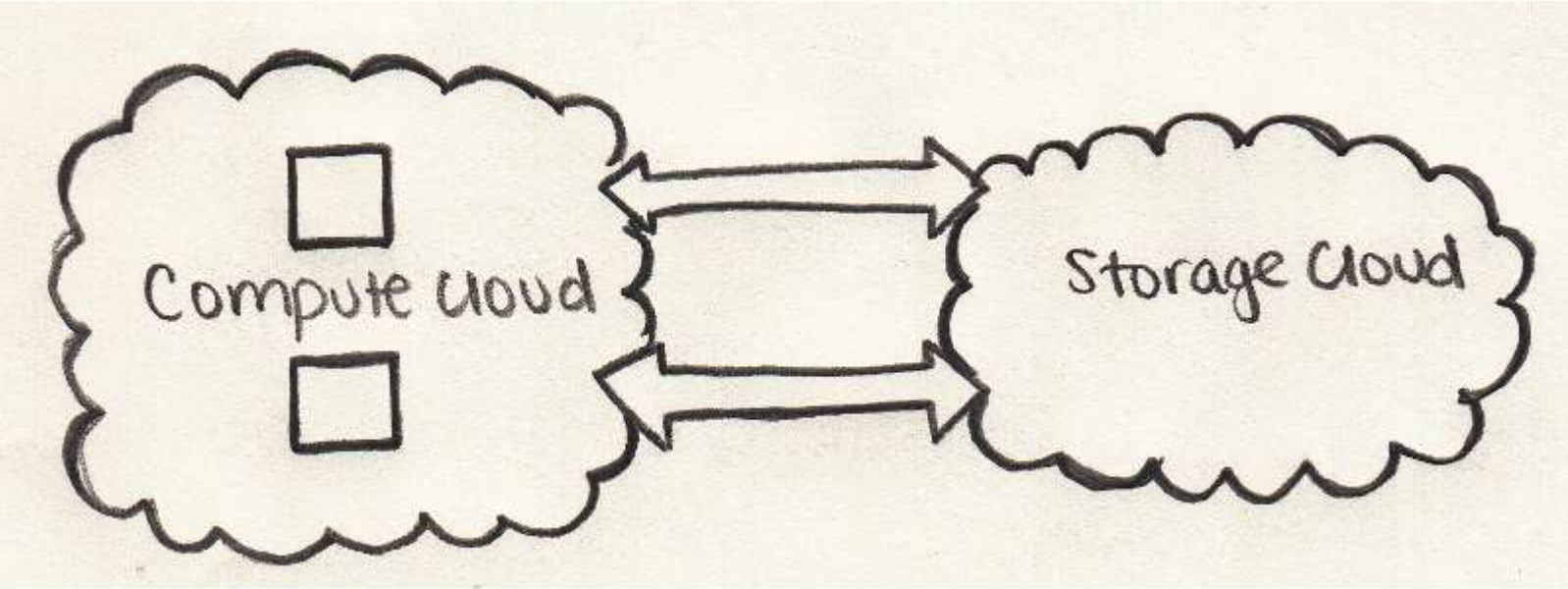
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

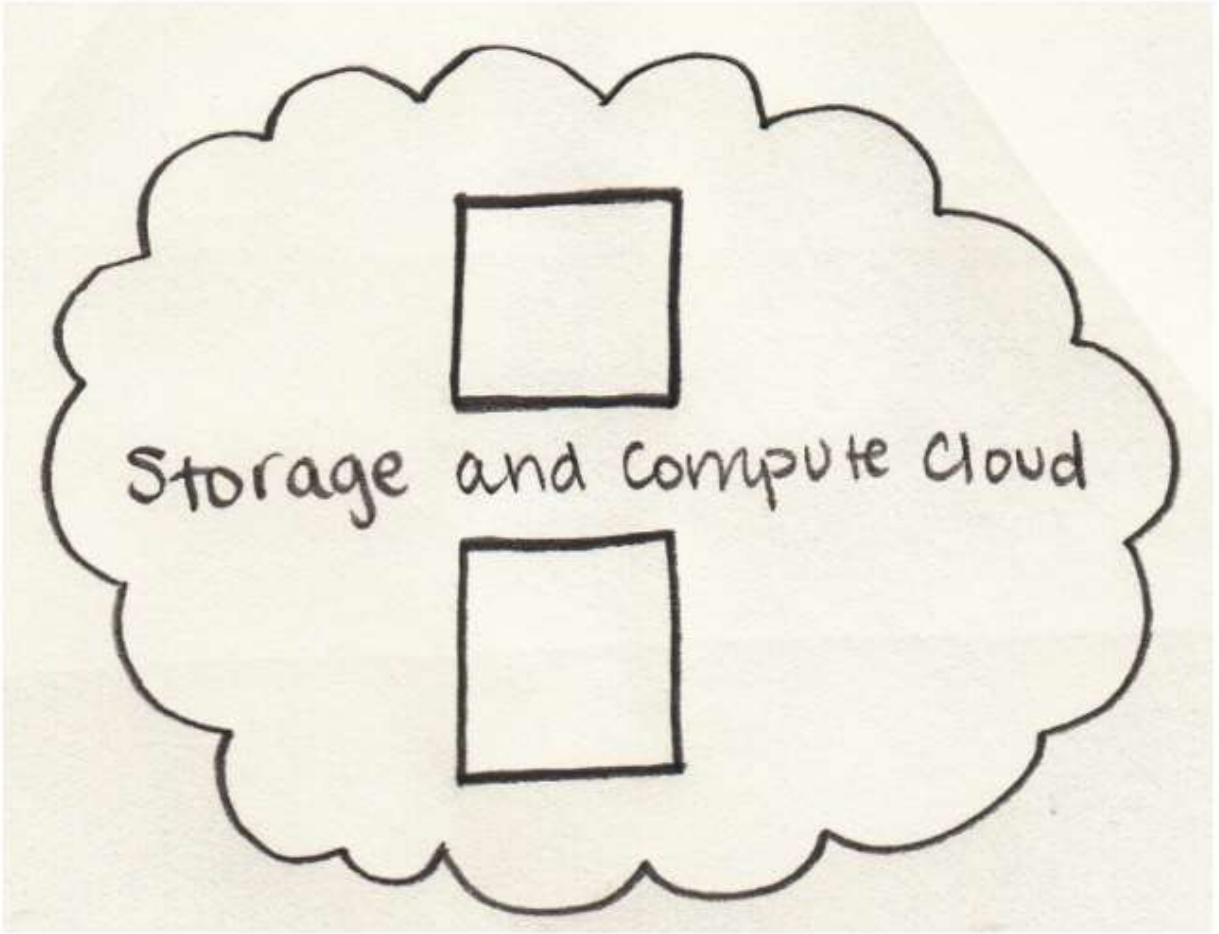
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



CLOUD WITHOUT BIG DATA APPLICATIONS



CLOUD WITH BIG DATA



WHY SO?

This makes data processing local.

All Big Data programming models are distributed- and parallel-processing based.

Network I/O is orders of magnitude slower than disk I/O.

Because data has been distributed to various nodes, and application libraries have been moved to the nodes, the goal is to process the data in place.

Although processing data local to the node is preferred by a typical Big Data system, it is not always possible.

Big Data systems will schedule tasks on nodes as close to the data as possible.

KEY CONCEPTS

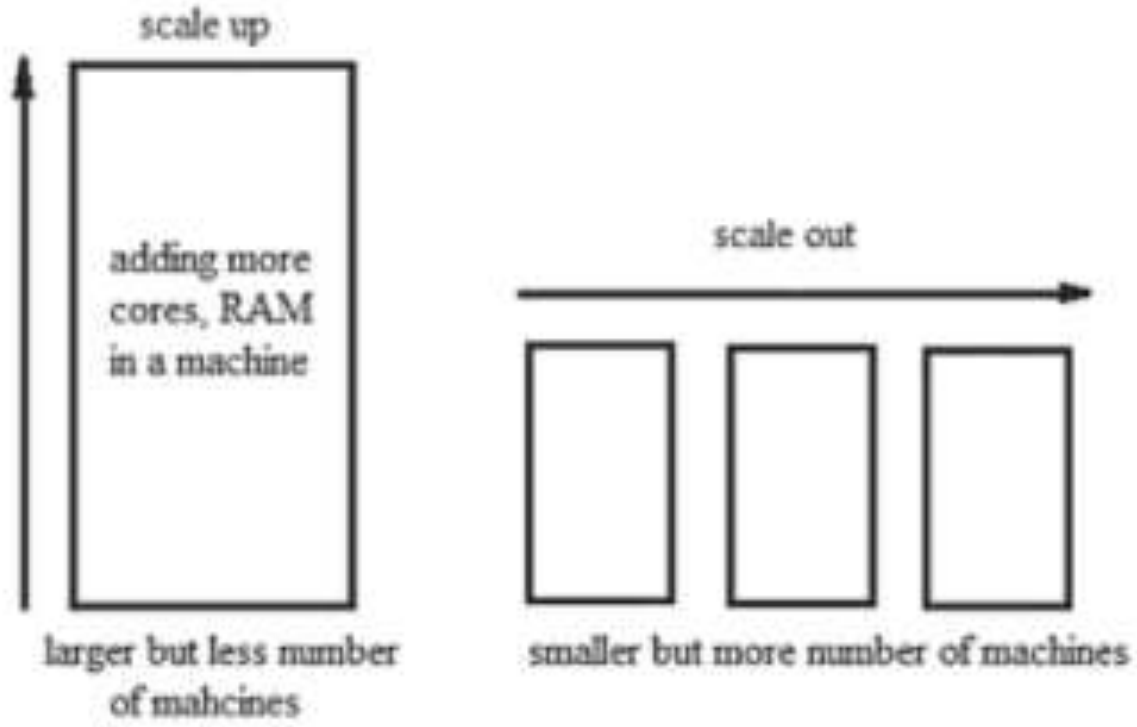
Data is distributed across several nodes (Network I/O speed << Local Disk I/O Speed).

Applications are distributed to data (nodes in the cluster) instead of the other way around.

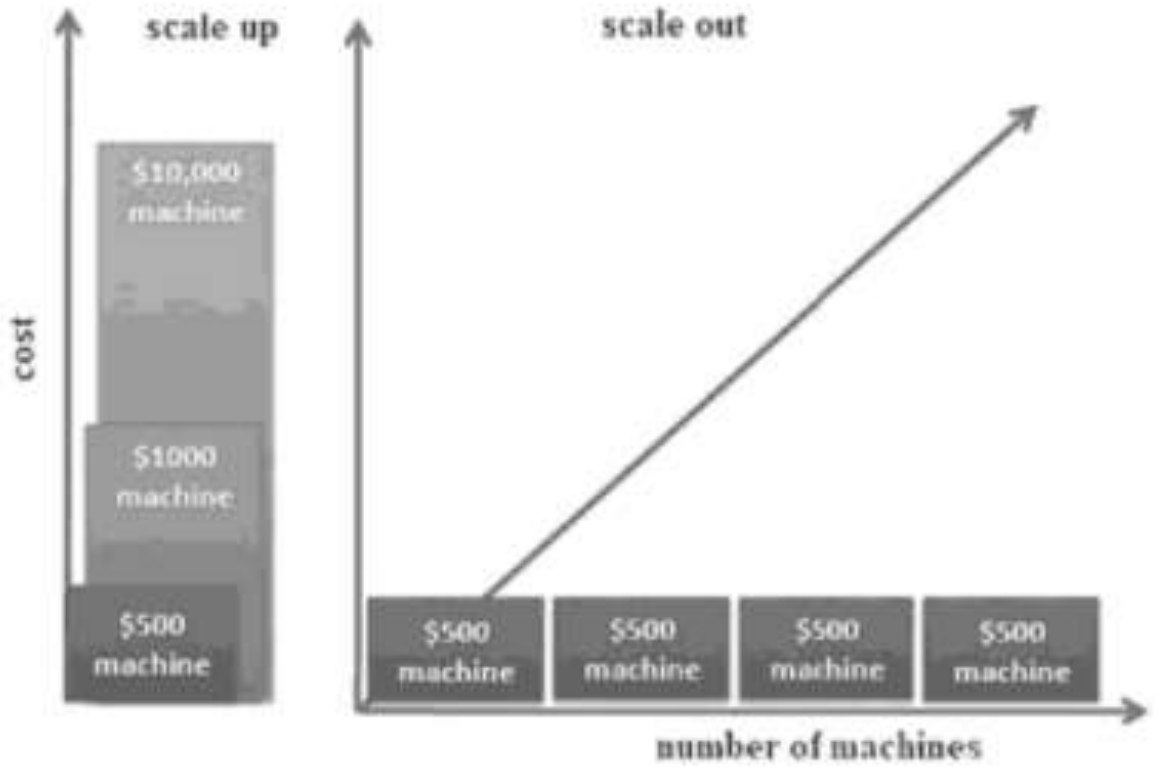
As much as possible, data is processed local to the node (Network I/O speed << Local Disk I/O Speed).

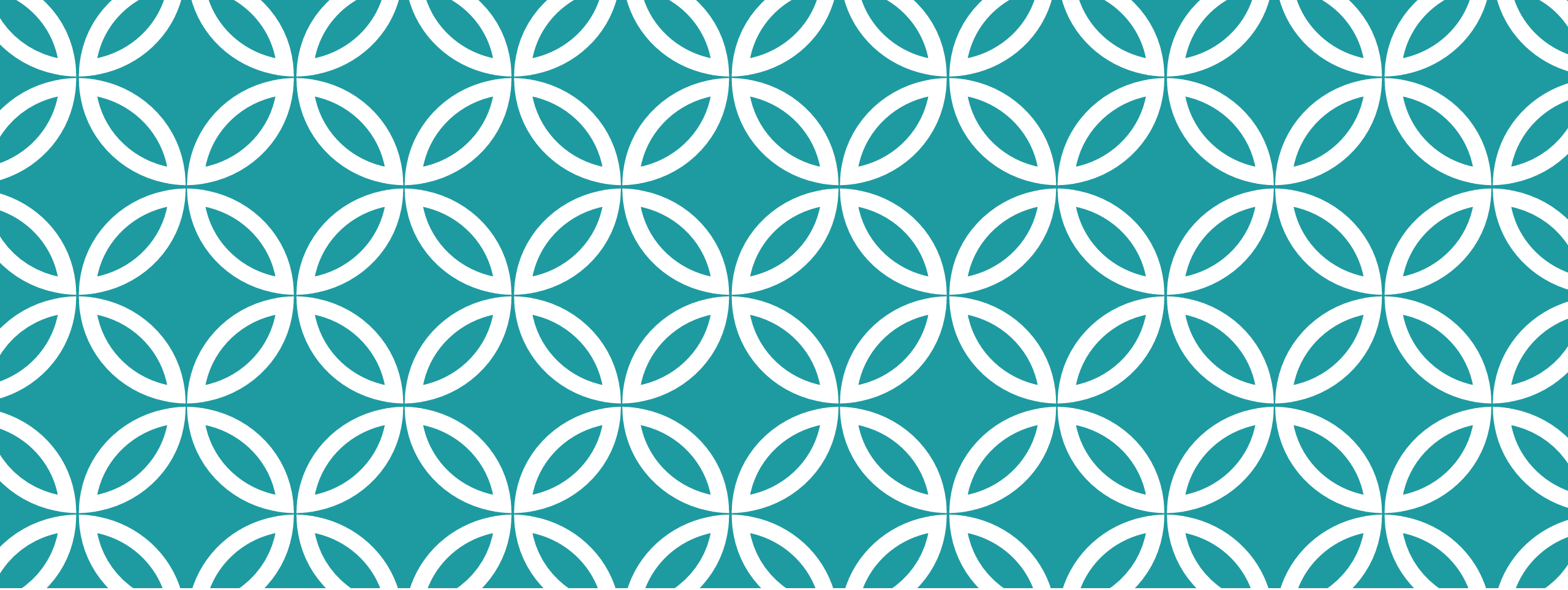
Random disk I/O is replaced by sequential disk I/O (Transfer Rate << Disk Seek Time).

SCALE OUT VERSUS SCALE UP



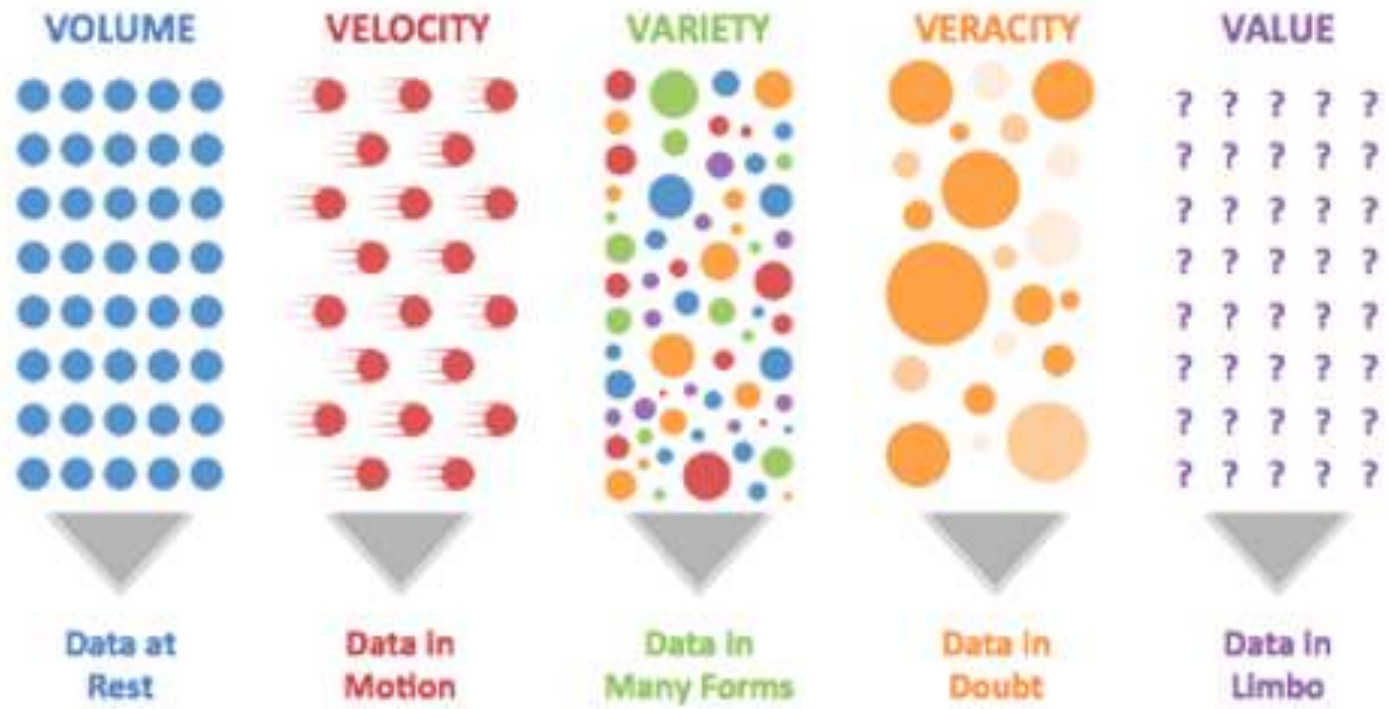
COST OF SCALE UP VERSUS SCALE OUT





BIG DATA CHARACTERISTICS

THE "V"S


VOLUME
VELOCITY
VARIETY
VERACITY
VALUE
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
? ? ? ? ?
Data at Rest
Data in Motion
Data in Many Forms
Data in Doubt
Data in Limbo

VOLUME

Big data is a form of data whose volume is so large that it would not fit on a single machine, therefore specialized tools and frameworks are required to store process and analyze such data.

For example, social media applications process billions of messages everyday, industrial and energy systems can generate terabytes of sensor data everyday, cab aggregation applications can process millions of transactions in a day, etc.

The volumes of data generated by modern IT, industrial, healthcare, Internet of Things, and other systems is growing exponentially driven by the lowering costs of data storage and processing architectures and the need to extract valuable insights from the data to improve business processes, efficiency and service to consumers.

Though there is no fixed threshold for the volume of data to be considered as big data, however, typically, the term big data is used for massive scale data that is difficult to store, manage and process using traditional databases and data processing architectures.

VELOCITY

Velocity of data refers to how fast the data is generated.

Data generated by certain sources can arrive at very high velocities, for example, social media data or sensor data.

Velocity is another important characteristic of big data and the primary reason for the exponential growth of data.

High velocity of data results in the volume of data accumulated to become very large, in short span of time.

Some applications can have strict deadlines for data analysis (such as trading or online fraud detection) and the data needs to be analyzed in real-time.

Specialized tools are required to ingest such high velocity data into the big data infrastructure and analyze the data in real-time.

VARIETY

Variety refers to the forms of the data. Big data comes in different forms such as structured, unstructured or semi-structured, including text data, image, audio, video and sensor data.

Big data systems need to be flexible enough to handle such variety of data.

VERACITY

Veracity refers to how accurate is the data.

To extract value from the data, the data needs to be cleaned to remove noise.

Data-driven applications can reap the benefits of big data only when the data is meaningful and accurate.

Therefore, cleansing of data is important so that incorrect and faulty data can be filtered out.

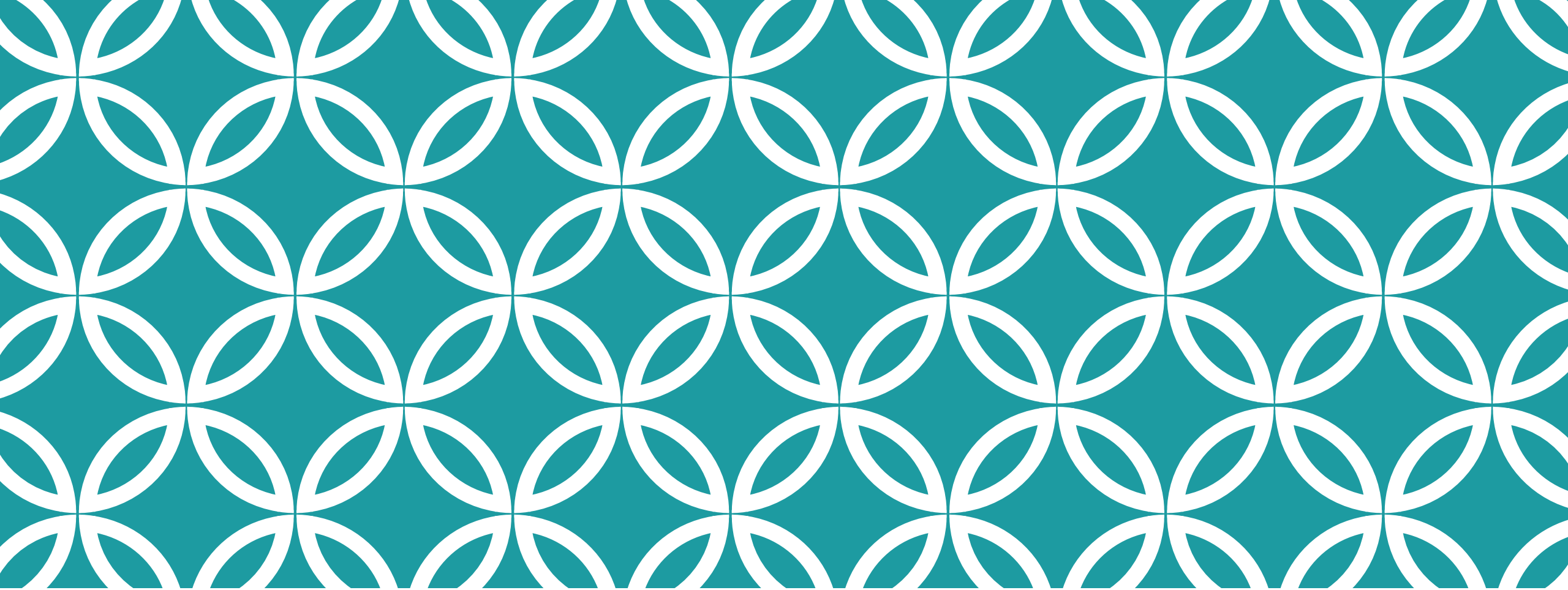
VALUE

Value of data refers to the usefulness of data for the intended purpose.

The end goal of any big data analytics system is to extract value from the data.

The value of the data is also related to the veracity or accuracy of the data.

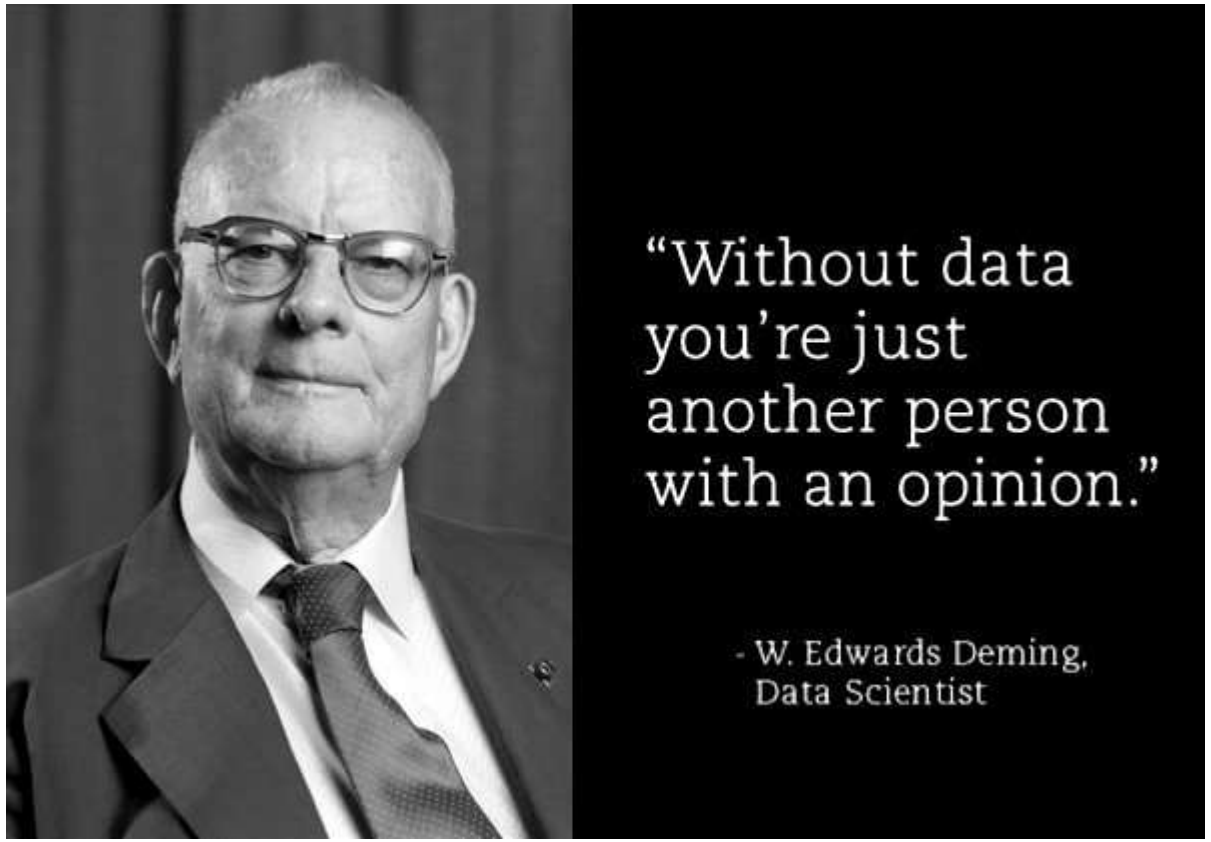
For some applications value also depends on how fast we are able to process the data.



WHAT IS DATA ANALYTICS?

DATA AND DECISIONS

WHY DATA/BUSINESS ANALYTICS



WHY DATA/BUSINESS ANALYTICS

Decisions used to be taken based on opinions

Opinion-based decisions

- Risky -> Lead to incorrect decisions

Business Analytics involves data analysis to improve decision making quality

DATA AND METRICS

Every organization uses performance metrics such as market share, profitability, sales growth, return on investment (ROI) etc

Used for quantifying, monitoring, benchmarking, and improving performance

ANALYTICS

A concept that uses statistical, mathematical and operation research techniques

Uses Artificial Intelligence (AI) techniques such as Machine Learning (ML) and Deep Learning

Based on data collection and storage

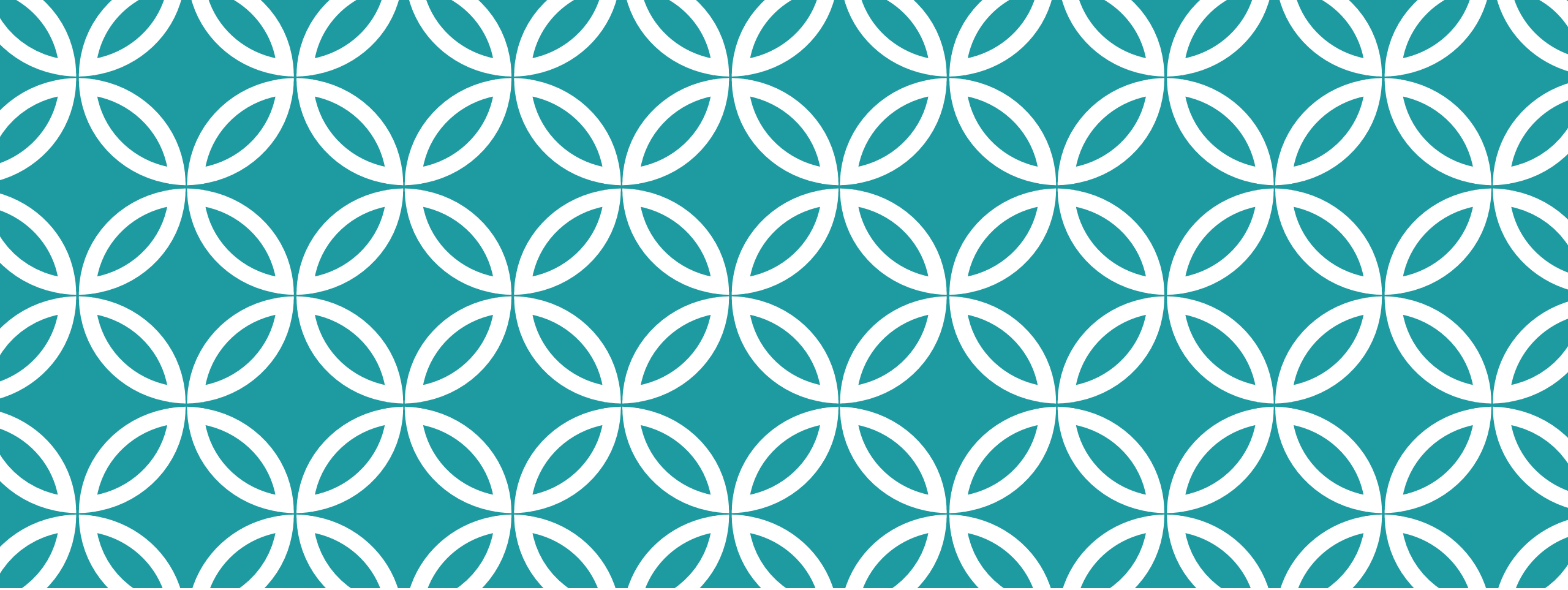
Uses data management processes such as data Extraction, Transformation, and Loading (ETL)

Uses computing and big data technologies such as Hadoop, Spark, Pig, Hive

OBJECTIVES OF ANALYTICS

Problem solving

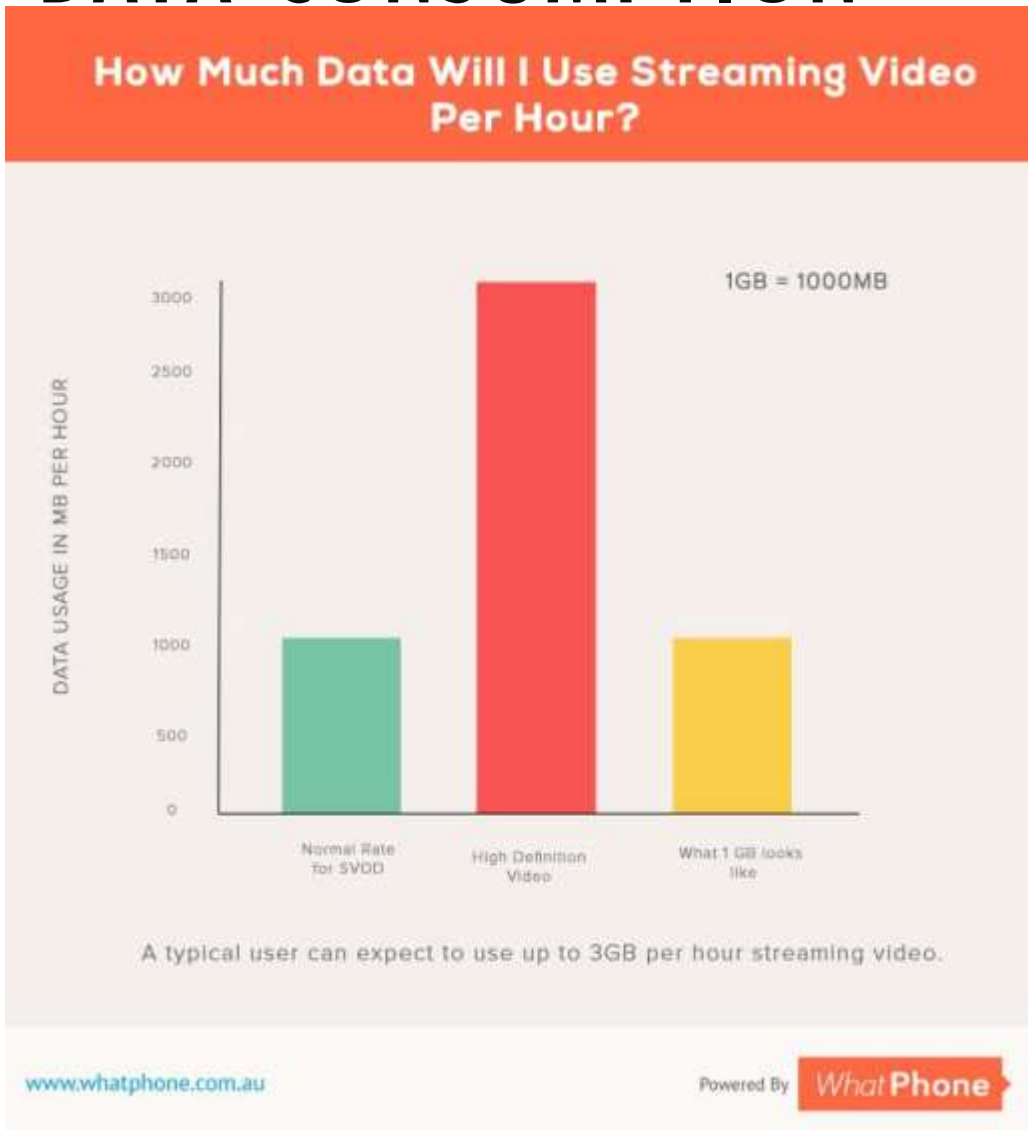
Decision making

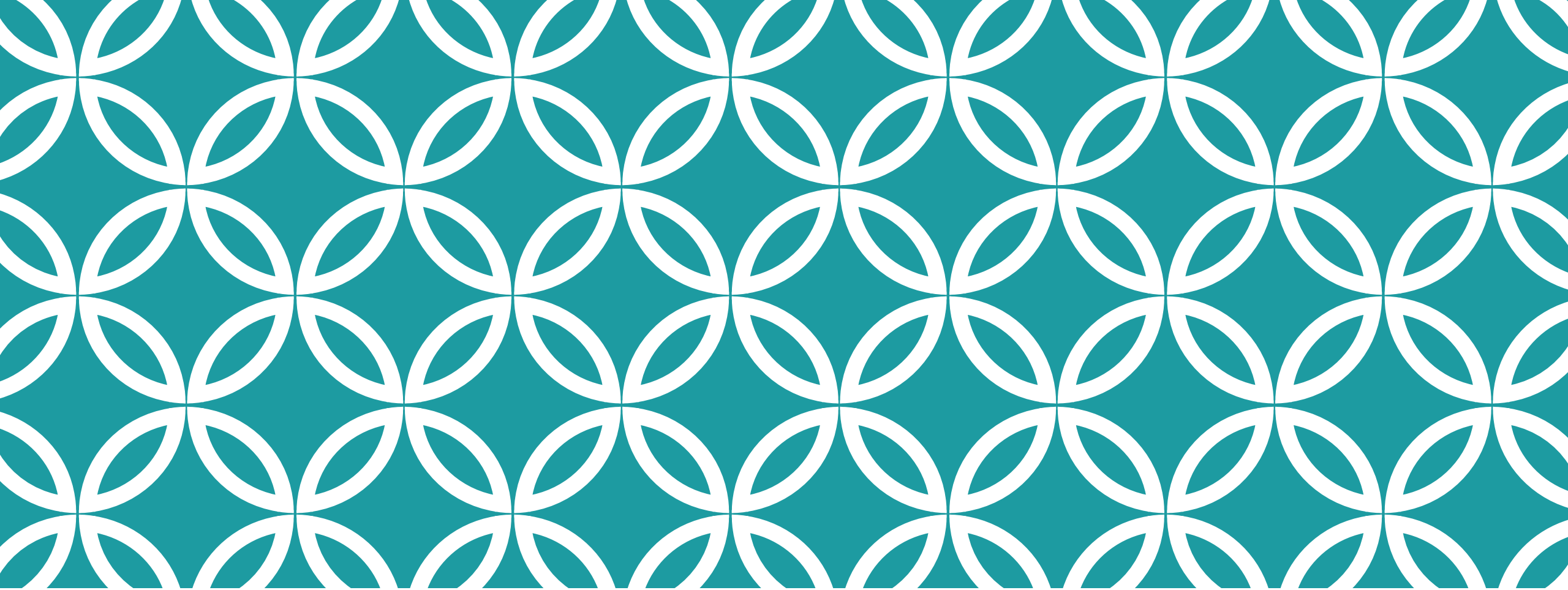


SOME STATISTICS

BIG DATA ANALYTICS (2017)

STREAMING DATA CONSUMPTION





HADOOP

HISTORY

The set of storage and processing methodologies commonly known as “Big Data” emerged from the search engine providers in the early 2000s, principally Google and Yahoo!.

The search engine providers were the first group of users faced with Internet scale problems,

mainly how to process and store indexes of all of the documents in the Internet universe.

Yahoo! and Google independently set about developing a set of capabilities to meet this

challenge.

In 2003, Google released a whitepaper called “The Google File System.” Subsequently,

in 2004, Google released another whitepaper called “MapReduce: Simplified Data Processing on Large Clusters.” At the same time, at Yahoo!, Doug Cutting (who is generally acknowledged as the initial creator of Hadoop) was working on a web indexing project called Nutch.

HISTORY … CONTINUED

The Google whitepapers inspired Doug Cutting to take the work he had done to date on the Nutch project and incorporate the storage and processing principles outlined in these whitepapers. The resultant product is what is known today as Hadoop.

Hadoop was born in 2006 as an open source project under the Apache Software Foundation licensing scheme.

The name Hadoop was after Cutting's son's toy elephant

WHAT IS HADOOP?

Hadoop is a distributed data storage and processing platform, based upon a central concept: data locality.

Data locality refers to the processing of data where it resides by bringing the computation to the data, rather than the typical pattern of requesting data from its location (for example, a database management system) and sending this data to a remote processing system or host.

Hadoop is schemaless with respect to its write operations (it is what's known as a schema-on-read system). This means that it can store and process a wide range of data, from unstructured text documents, to semi-structured JSON (JavaScript Object Notation) or XML documents, to well structured extracts from relational database systems.

DISTRIBUTED SYSTEMS AND HADOOP

In order to perform computation at scale, Hadoop distributes an analytical computation that involves a massive dataset to many machines that each simultaneously operate on their own individual chunk of data.

Any distributed system needs to address these points:

Fault tolerance: If a component fails, it should not result in the failure of the entire system. The system should gracefully degrade into a lower performing state. If a failed component recovers, it should be able to rejoin the system.

Recoverability: In the event of failure, no data should be lost.

Consistency: The failure of one job or task should not affect the final result.

Scalability: Adding load (more data, more computation) leads to a decline in performance, not failure; increasing resources should result in a proportional increase in capacity.

HOW HADOOP ACHIEVES THIS?

Data is distributed immediately when added to the cluster and stored on multiple nodes. Nodes prefer to process data that is stored locally in order to minimize traffic across the network.

Data is stored in blocks of a fixed size (usually 128 MB) and each block is duplicated multiple times across the system to provide redundancy and data safety.

A computation is usually referred to as a job; jobs are broken into tasks where each individual node performs the task on a single block of data.

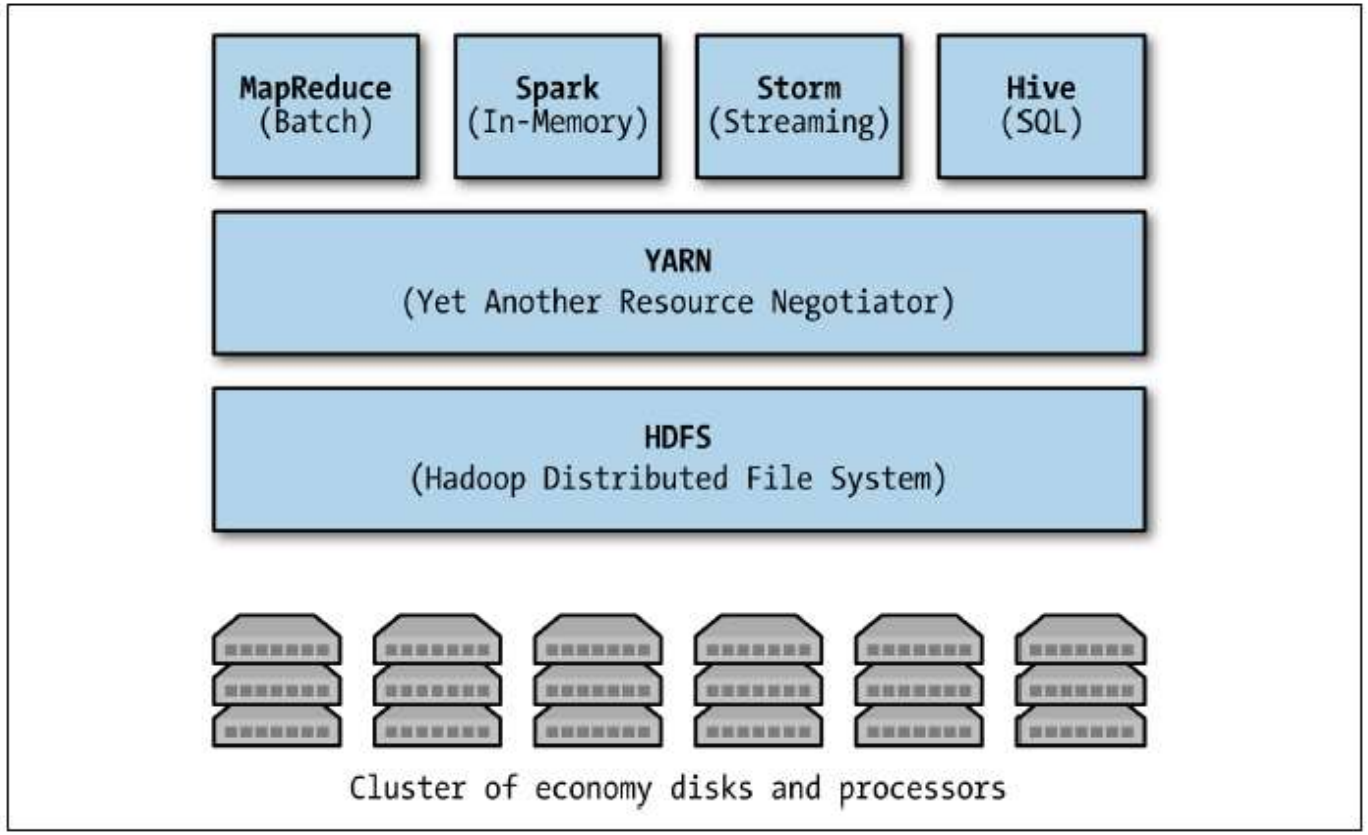
Jobs are written at a high level without concern for network programming, time, or low-level infrastructure, allowing developers to focus on the data and computation rather than distributed programming details.

The amount of network traffic between nodes should be minimized transparently by the system. Each task should be independent and nodes should not have to communicate with each other during processing to ensure that there are no interprocess dependencies that could lead to deadlock.

Jobs are fault tolerant usually through task redundancy, such that if a single node or task fails, the final computation is not incorrect or incomplete.

Master programs allocate work to worker nodes such that many worker nodes can operate in parallel, each on their own portion of the larger dataset.

HADOOP ECOSYSTEM



HOW IS HADOOP DIFFERENT?

Because the schema is not interpreted during write operations to Hadoop, there are no indexes, statistics, or other constructs typically employed by database systems to optimize query operations and filter or reduce the amount of data returned to a client.

This further necessitates the requirement for data locality.

WHAT IS HADOOP?

Apache Hadoop is a set of open-source software utilities that facilitate usage of a network of many computers to solve problems involving massive amounts of data.

It provides a software framework for distributed storage and distributed computing.

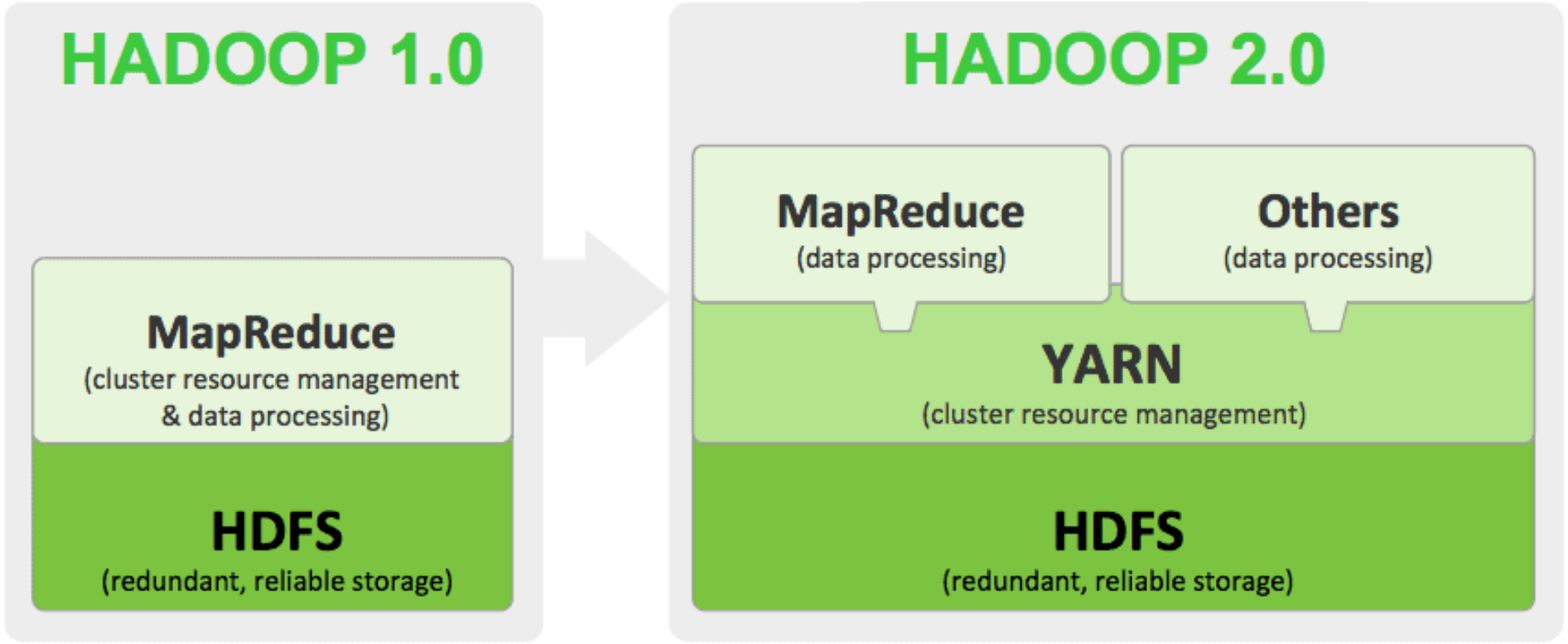
It divides a file into the number of blocks and stores it across a cluster of machines.

Hadoop also achieves fault tolerance by replicating the blocks on the cluster.

It enables distributed processing by dividing a job into a number of independent tasks.

These tasks run in parallel over the computer cluster.

KEY HADOOP COMPONENTS



HADOOP COMPONENTS

HDFS – Hadoop Distributed File System provides for the storage of Hadoop. As the name suggests it stores the data in a distributed manner. The file gets divided into a number of blocks which spreads across the cluster of commodity hardware.

MapReduce – This is the processing engine of Hadoop. MapReduce works on the principle of distributed processing. It divides the task submitted by the user into a number of independent subtasks. These sub-task executes in parallel, thereby increasing the throughput.

Yarn – Yet Another Resource Manager provides resource management for Hadoop. There are two daemons running for Yarn. One is NodeManager on the slave machines and other is the Resource Manager on the master node. Yarn looks after the allocation of the resources among various slave competing for it.

MAPREDUCE

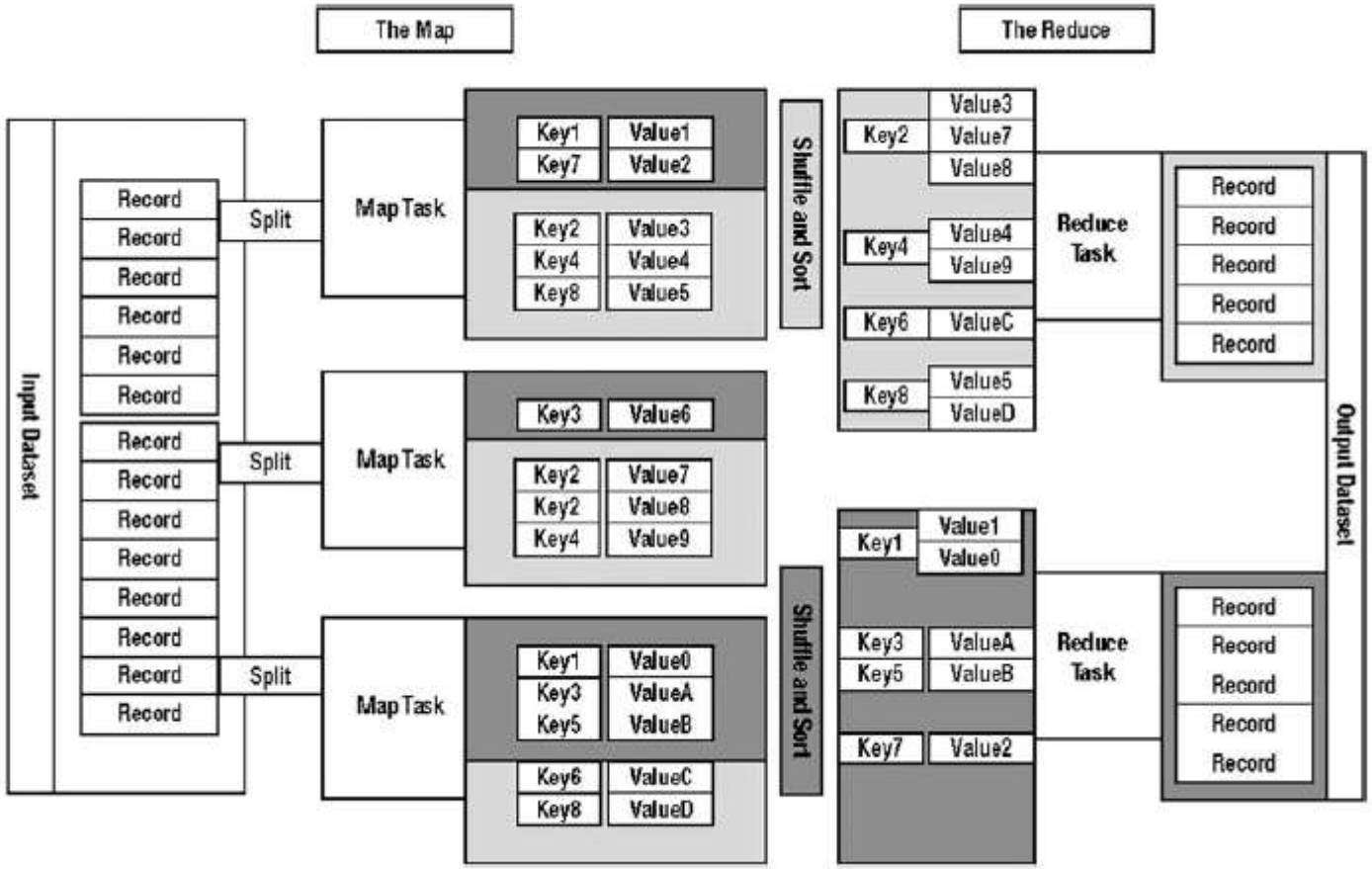
The idea of the MapReduce algorithm is to process the data in parallel on your distributed cluster.

It is subsequently combined into the desired result or output.

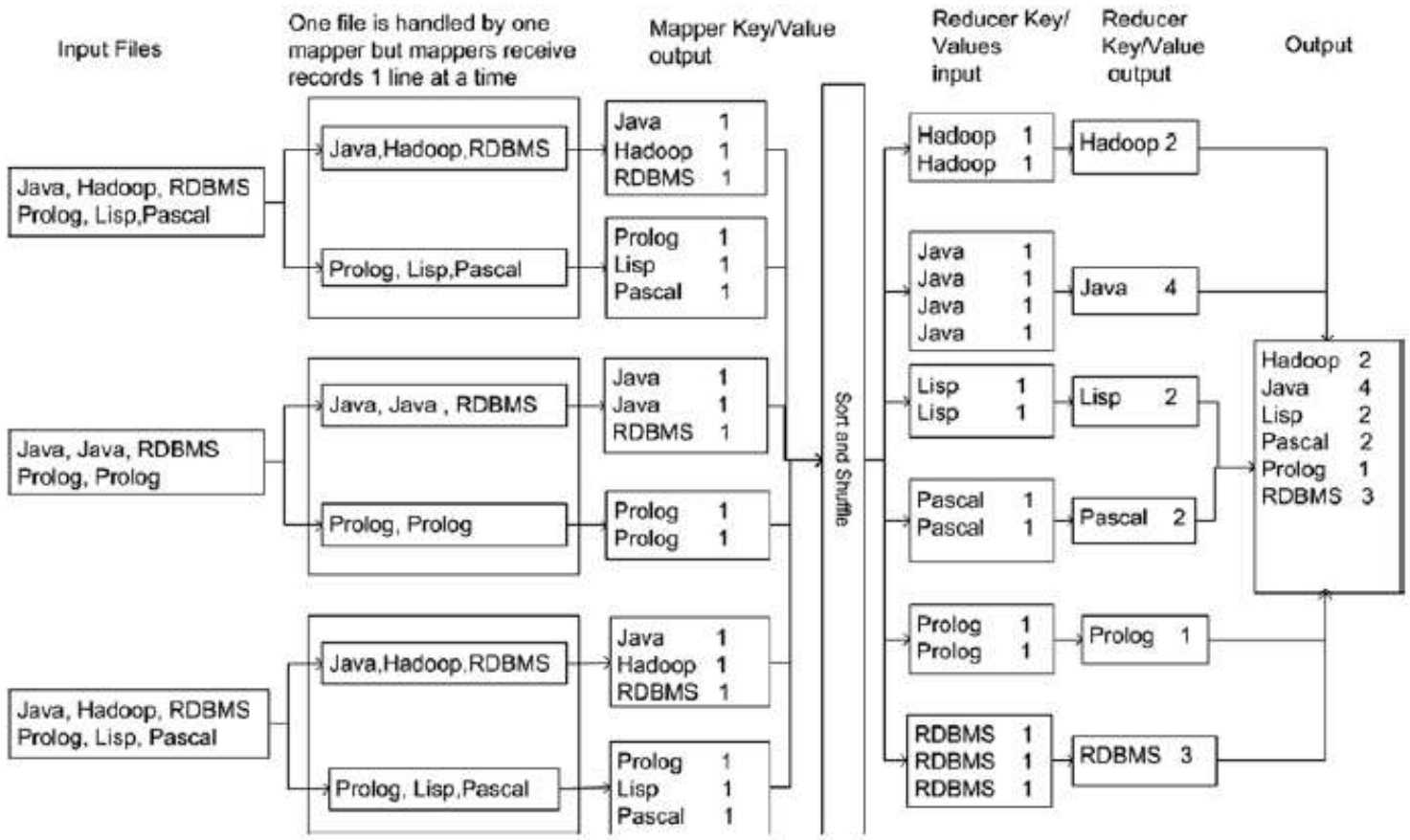
MAPREDUCE COMPARED TO RDBMS

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Transactions	ACID	None
Structure	Schema-on-write	Schema-on-read
Integrity	High	Low
Scaling	Nonlinear	Linear

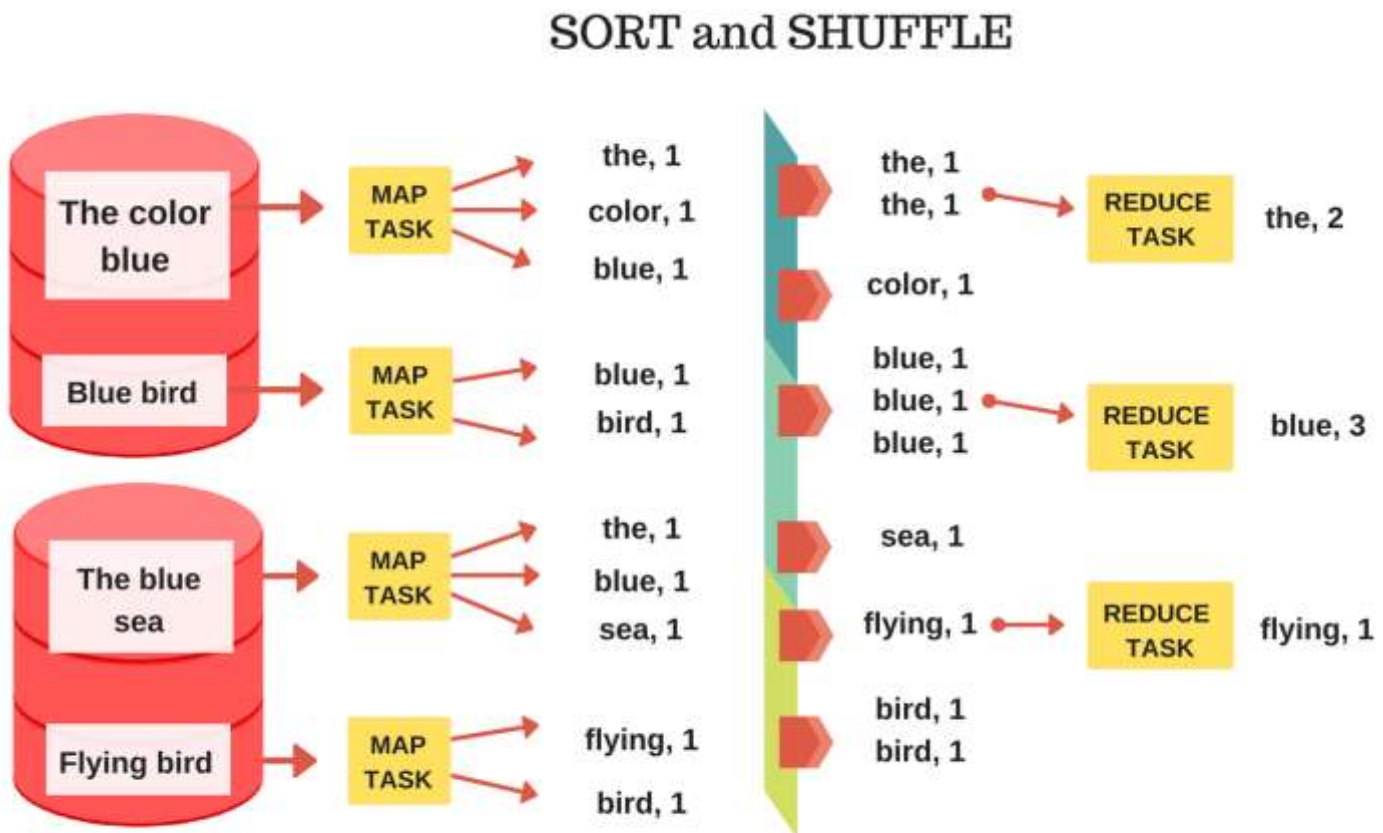
MAPREDUCE CONCEPT



WORD COUNT MAPREDUCE EXAMPLE



MAPREDUCE EXAMPLE



MAPREDUCE

MapReduce (**MR**) is a framework to write analytical applications in batch mode on terabytes or petabytes of data stored on HDFS. An MR job usually processes each block (excluding replicas) of input file(s) in HDFS with the mapper tasks in a parallel manner.

The MR framework sorts and shuffles the outputs of the mappers to the reduce tasks in order to produce the output.

The framework takes care of computing the number of tasks needed, scheduling tasks, monitoring them, and re-executing them if they fail.

The developer needs to focus only on writing the business logic, and all the heavy lifting is done by the HDFS and MR frameworks.

MAPREDUCE

For example, if an MR job is submitted for **File1**, one map task will be created and run on any Node 1, 2, or 3 to achieve data locality.

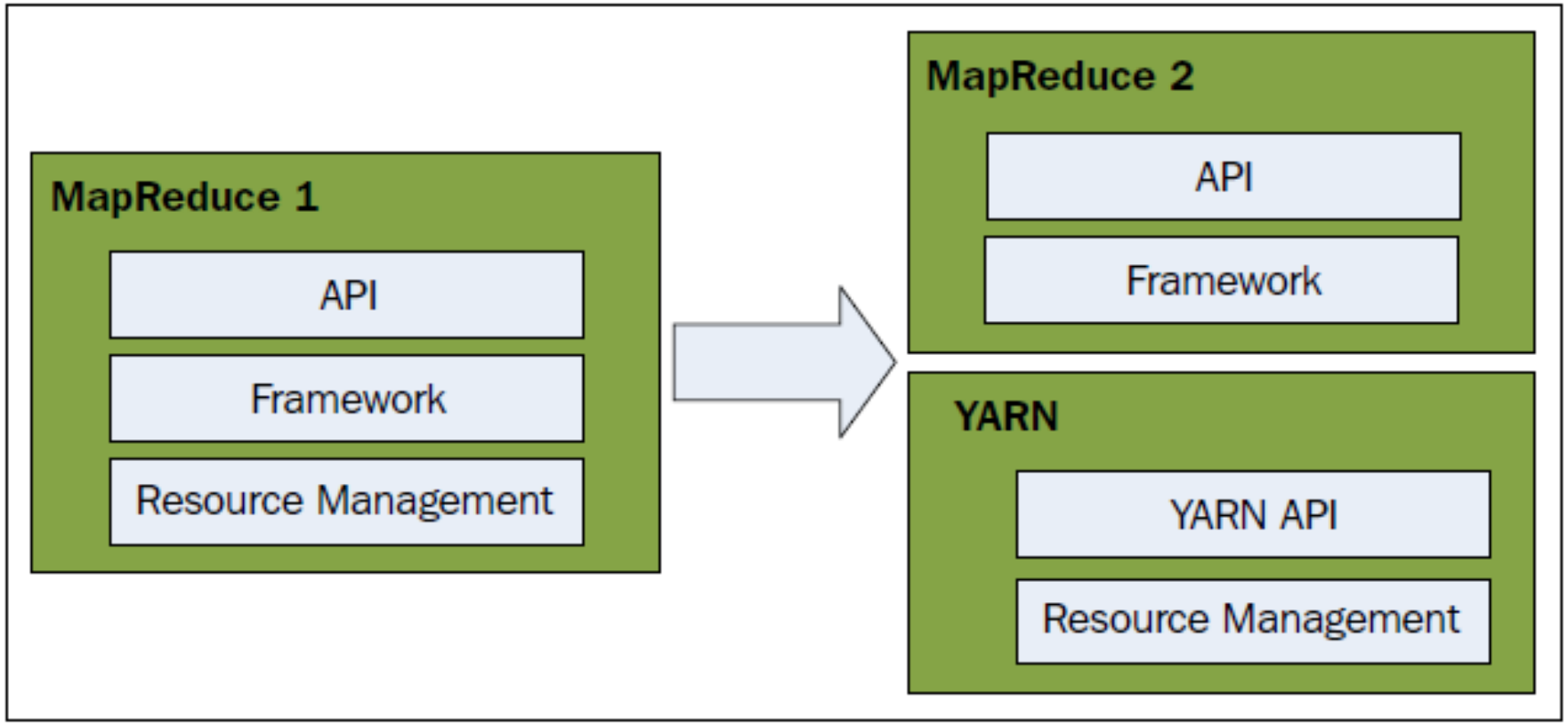
In the case of **File2**, two map tasks will be created with map task 1 running on Node 1, 3, or 4, and map task 2 running on Node 1, 2, or 3, depending on resource availability.

The output of the mappers will be sorted and shuffled to reducer tasks.

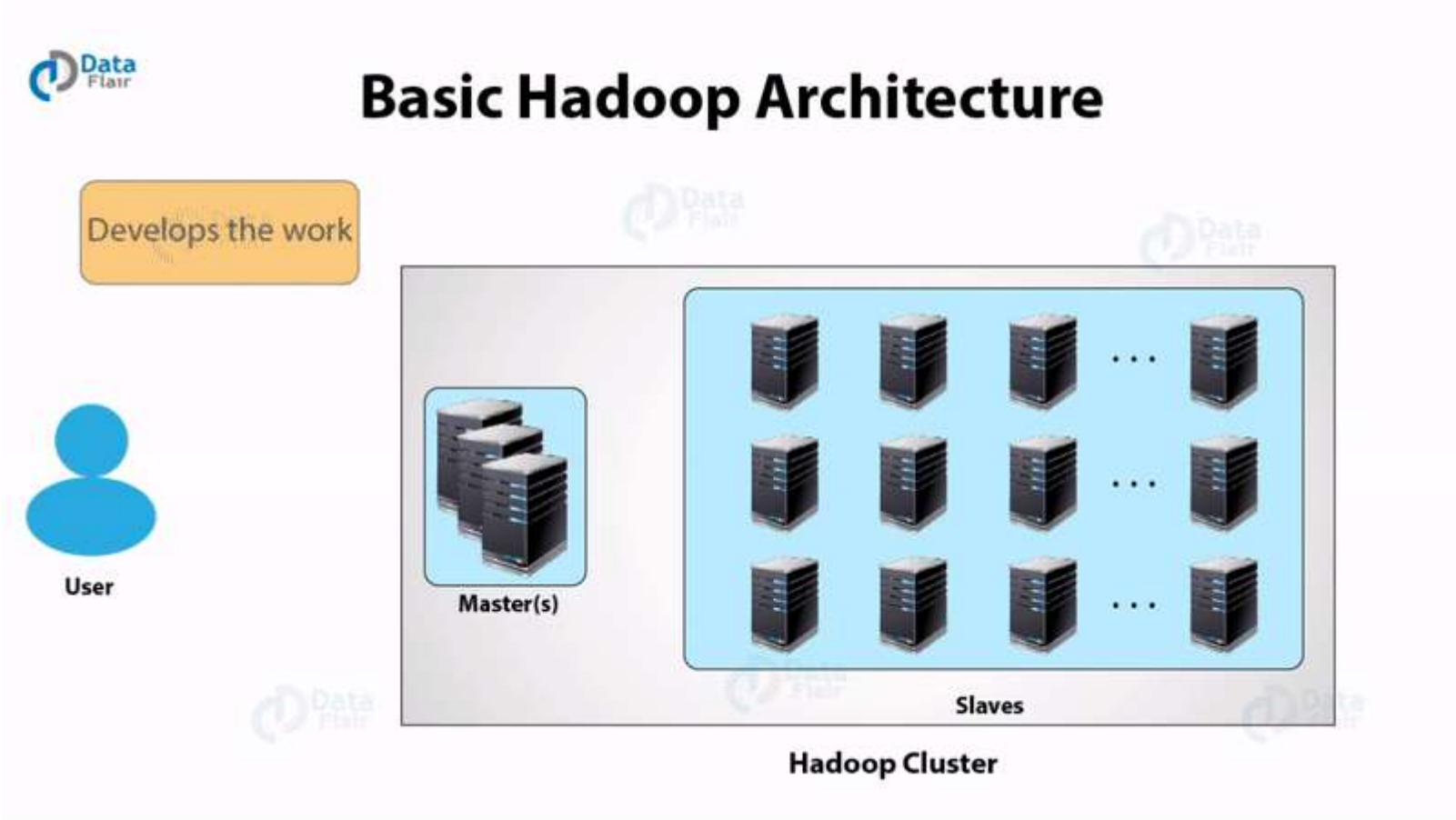
By default, the number of reducers is one.

However, the number of reducer tasks can be increased to provide parallelism at the reducer level.

MAPREDUCE 1 AND 2



BASIC HADOOP ARCHITECTURE



HADOOP DAEMONS

Daemons are the processes that run in the background. The Hadoop Daemons are:-

a) Namenode – It runs on master node for HDFS.

b) Datanode – It runs on slave nodes for HDFS.

c) Resource Manager – It runs on YARN master node for MapReduce.

d) Node Manager – It runs on YARN slave node for MapReduce.

These 4 daemons run for Hadoop to be functional.

HADOOP CLUSTER

A Hadoop cluster is a group of computers connected together via a network.

We use it for storing and processing large data sets.

Hadoop clusters communicate with a high-end machine, which acts as a master.

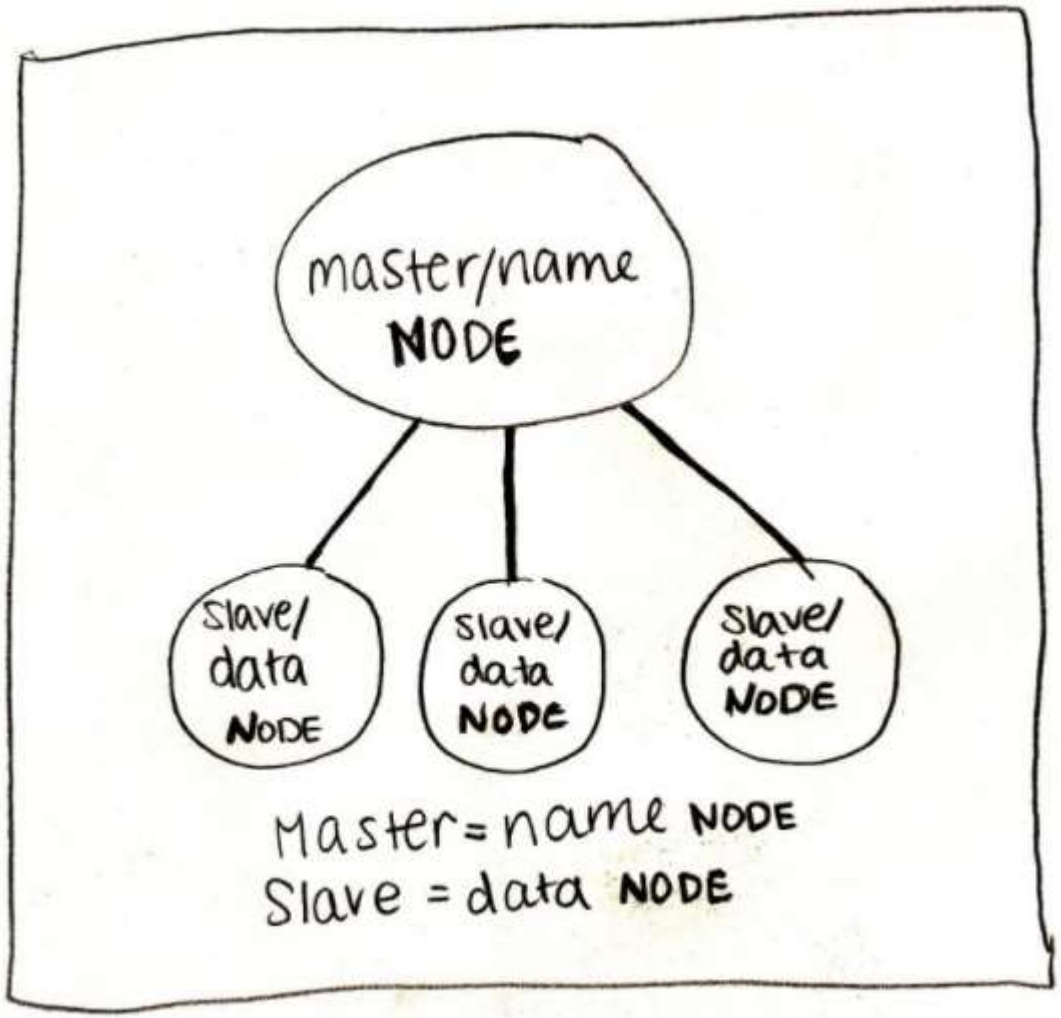
These master and slaves implement distributed computing over distributed data storage.

It runs software for providing distributed functionality.

Master has two parts: NameNode and Resource Manager

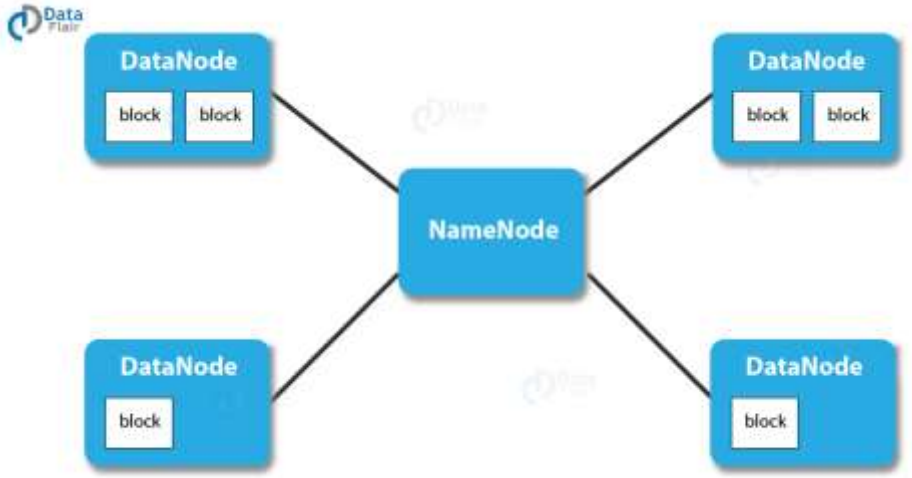
Slave also has two parts: DataNode and Node Manager

NAME NODE AND DATA NODE



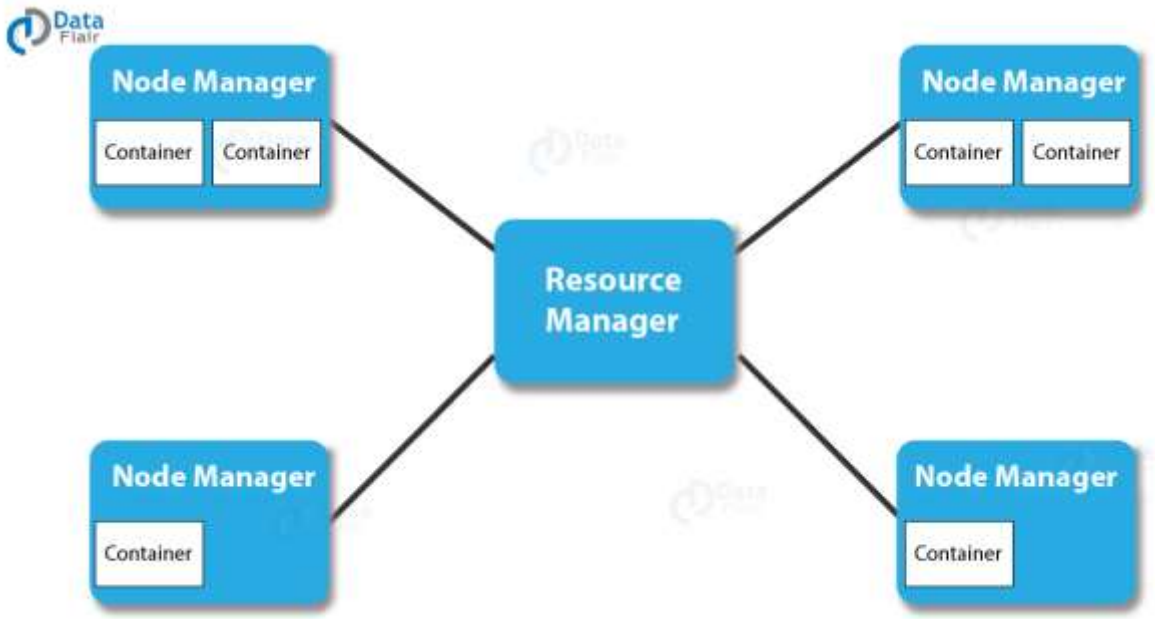
NAMENODE

The master NameNode keeps track of what blocks make up a file and where those blocks are located. The NameNode communicates with the DataNodes, the processes that actually hold the blocks in the cluster. Metadata associated with each file is stored in the memory of the NameNode master for quick lookups, and if the NameNode stops or fails, the entire cluster will become inaccessible



RESOURCE MANAGER

Arbitrates resources among competing nodes, Keeps track of live and dead nodes



HADOOP CLUSTER: SLAVES

There are two daemons running on Slave machines: DataNode and Node Manager

DataNode

Stores the business data

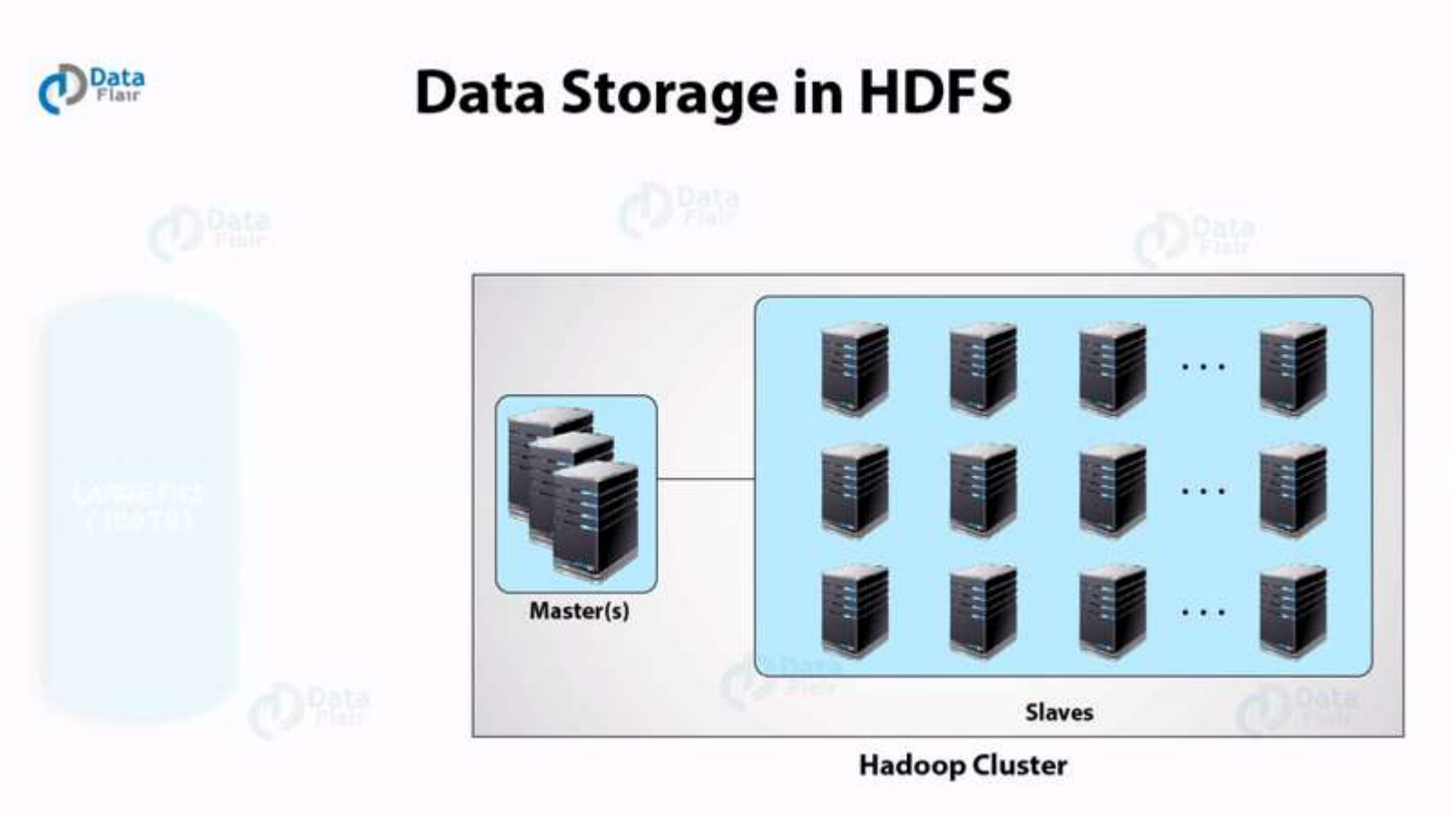
Performs read, write and data processing operations

Upon instruction from a master, it does creation, deletion, and replication of data blocks.

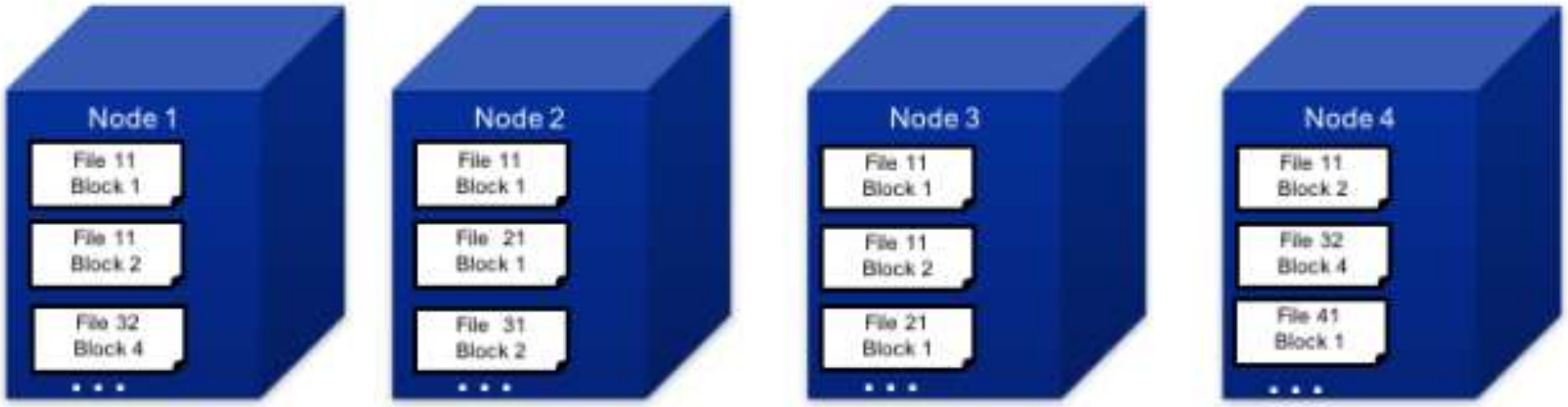
Functions of NodeManager

It runs services on the node to check its health and reports the same to the ResourceManager.

HDFS


Data Flair
Data Storage in HDFS
Master(s)
Slaves
Hadoop Cluster

HDFS


Node 1
File 11
Block 1
File 11
Block 2
File 32
Block 4
Node 2
File 11
Block 1
File 21
Block 1
File 31
Block 2
Node 3
File 11
Block 1
File 11
Block 2
File 21
Block 1
Node 4
File 11
Block 2
File 32
Block 4
File 41
Block 1

HDFS

Makes multiple copies of each block (by default, three copies) and stores these copies on different nodes.

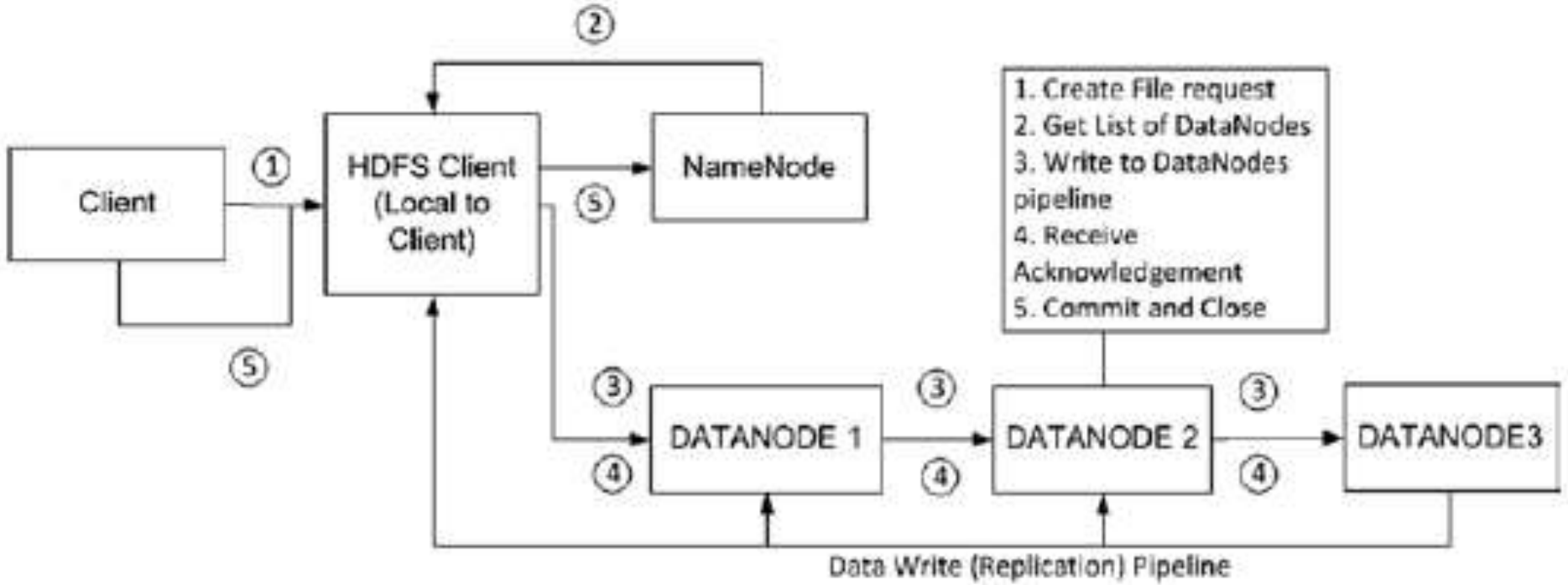
This way, if one node dies, two other copies are still available and the block will be copied to a third node once the failure is detected without affecting availability.

The multiple copies also facilitate load balancing, because we can choose to send the work to the least busy node that contains the data.

In our example, file 11 is stored across different nodes. It has two blocks.

- Block 1 is stored on nodes 1, 2, and 3, while block 2 is stored on nodes 1, 3, and 4. When the file is being processed, work for block 1 can be performed on any of nodes where it is stored—whichever is less busy.
- Similarly, work on block 2 can be performed on any of the three nodes where it is stored.
- If one of the nodes—say, node 3—were to get corrupted or otherwise become unavailable, HDFS would still have access to the two other copies of each block stored on node 3 and would make sure to create replacement copies for the blocks originally stored on node 3 on other nodes that are still up and running.

HDFS WRITE OPERATION



HDFS FEATURES

High availability: Enabling high availability is done by creating a standby NameNode.

Data integrity: When blocks are stored on HDFS, computed checksums are stored on the DataNodes as well. Data is verified against the checksum.

HDFS ACLs: HDFS implements POSIX-style permissions that enable an owner and group for every file with read, write, and execute permissions. In addition to POSIX permissions, HDFS supports POSIX **Access Control Lists** (**ACLs**) to provide access for specific named users or groups.

Snapshots: HDFS Snapshots are read-only point-in-time copies of the HDFS filesystem, which are useful to protect datasets from user or application errors.

HDFS rebalancing: The HDFS rebalancing feature will rebalance the data uniformly across all DataNodes in the cluster.

HDFS FEATURES

APIs: HDFS provides a native Java API, Pipes API for C++, and Streaming API for scripting languages such as Python, Perl, and others. FileSystem Shell and web browsers can be used to access data as well. Also, WebHDFS and HttpFs can be used to access data over HTTP.

Data encryption: HDFS will encrypt the data at rest once enabled. Data encryption and decryption happens automatically without any changes to application code.

Kerberos authentication: When Kerberos is enabled, every service in the Hadoop cluster being accessed will have to be authenticated using the Kerberos principle. This provides tight security to Hadoop clusters.

NFS access: Using this feature, HDFS can be mounted as part of the local filesystem, and users can browse, download, upload, and append data to it.

HDFS DAEMONS

NameNode is the daemon running of the master machine. It is the centerpiece of an HDFS file system. NameNode stores the directory tree of all files in the file system. It tracks where across the cluster the file data resides. It does not store the data contained in these files.

When the client applications want to add/copy/move/delete a file, they interact with NameNode. The NameNode responds to the request from client by returning a list of relevant DataNode servers where the data lives.

Datanode daemon runs on the slave nodes. It stores data in the HadoopFileSystem. In functional file system data replicates across many DataNodes.

On startup, a DataNode connects to the NameNode. It keeps on looking for the request from NameNode to access data. Once the NameNode provides the location of the data, client applications can talk directly to a DataNode, while replicating the data, DataNode instances can talk to each other.

HADOOP FILE SYSTEM (FS) COMMANDS

Create a directory

- hadoop fs -mkdir /sicsr

List the contents of a directory

- hadoop fs -ls /sicsr

Upload a file in HDFS

- hadoop fs -put testfile.txt /sicsr OR hadoop fs -copyFromLocal testfile.txt /sicsr

Download a file from HDFS

- hadoop fs -get /sicsr testfile.txt

View the contents of a file

- hadoop fs -cat /sicsr/testfile.txt

HADOOP FILE SYSTEM (FS) COMMANDS …

Copy a file

- hadoop fs -cp /user/root/dir1/testfile.txt /user/root/dir2

Move a file

- hadoop fs -mv /user/root/dir1/testfile.txt /user/root/dir2

Remove a file

- hadoop fs -rm /user/root/dir2/testfile.txt

Remove a directory

- hadoop fs -rm -r /user/root/dir2

Display the length of a file

- hadoop fs -du /user/root/dir1/testfile.txt

HADOOP FILE SYSTEM (FS) COMMANDS …

Display details of space in the system

- Entire system: hadoop fs -df
- Specific path: hadoop fs -df /user/root

- Also see: https://mindmajix.com/hadoop-hdfs-commands-with-examples

YARN

YARN is the resource management framework that enables an enterprise to process data in multiple ways simultaneously for batch processing, interactive analytics, or real-time analytics on shared datasets.

While HDFS provides scalable, fault-tolerant, and cost-efficient storage for Big Data, YARN provides resource management to clusters.

YARN is like an operating system for Hadoop, which manages the cluster resources (CPU and Memory) efficiently.

Applications such as MapReduce, Spark, and others request YARN to allocate resources for their tasks.

YARN allocates containers on nodes with the requested amount of RAM and virtual CPU from the total available on that node

YARN

YARN divides the task on resource management and job scheduling/monitoring into separate daemons.

- **ResourceManager** keeps track of the resource availability of the entire cluster and provides resources to applications when requested by ApplicationMaster.
- **ApplicationMaster** negotiates the resources needed by the application to run their tasks. ApplicationMaster also tracks and monitors the progress of the application.

The ResourceManger has two components – Scheduler and ApplicationManager.

- The **scheduler** is a pure scheduler i.e. it does not track the status of running application. It only allocates resources to various competing applications. Also, it does not restart the job after failure due to hardware or application failure. The scheduler allocates the resources based on an abstract notion of a container. A container is nothing but a fraction of resources like CPU, memory, disk, network etc.
- **ApplicationManager** accepts submission of jobs by client. Negotiates first container for specific ApplicationMaster, restarts the container after application failure.

MAPREDUCE PROGRAMMING

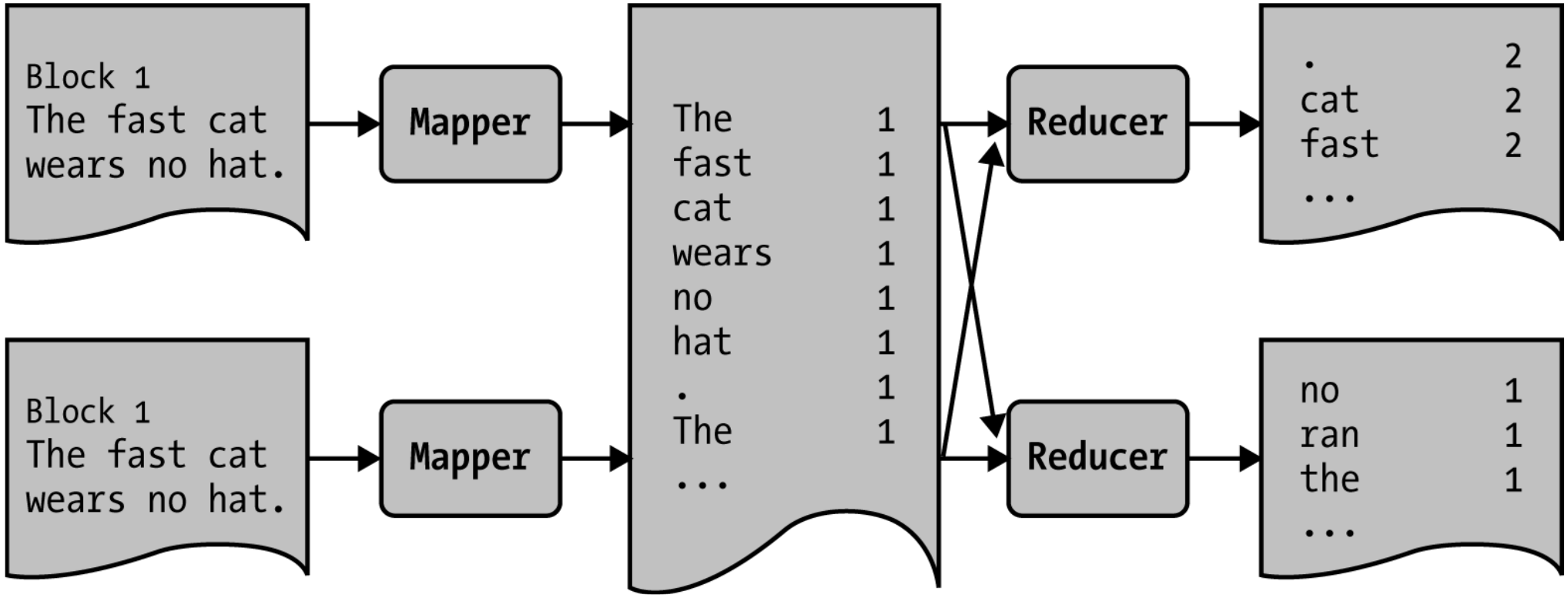
The *map* function has been a part of many functional programming languages for years, first gaining popularity with an artificial intelligence language called LISP.

Good software developers understand the value of reuse, so map has been reinvigorated as a core technology for processing lists of data elements (keys and values).

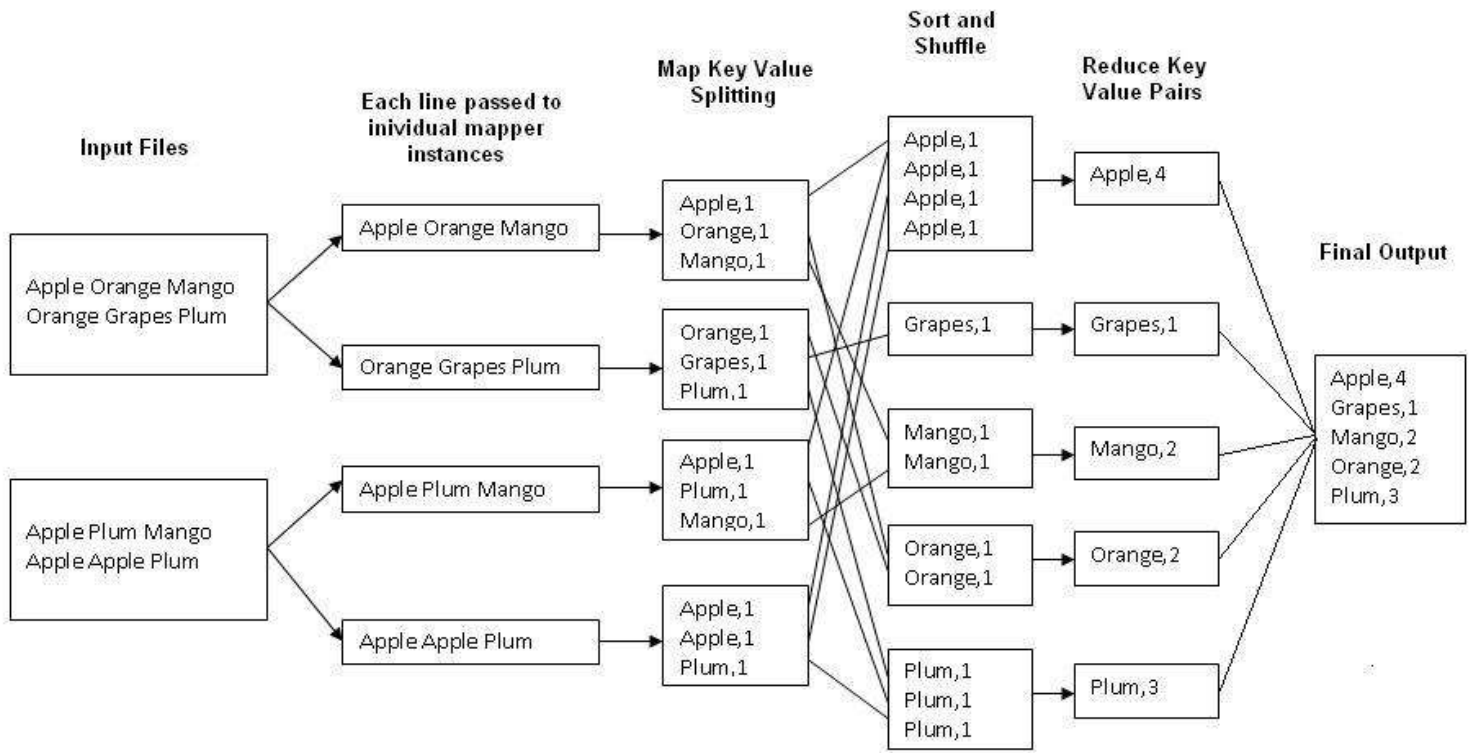
It applies a function to each element (defined as a key-value pair) of a list and produces a new list.

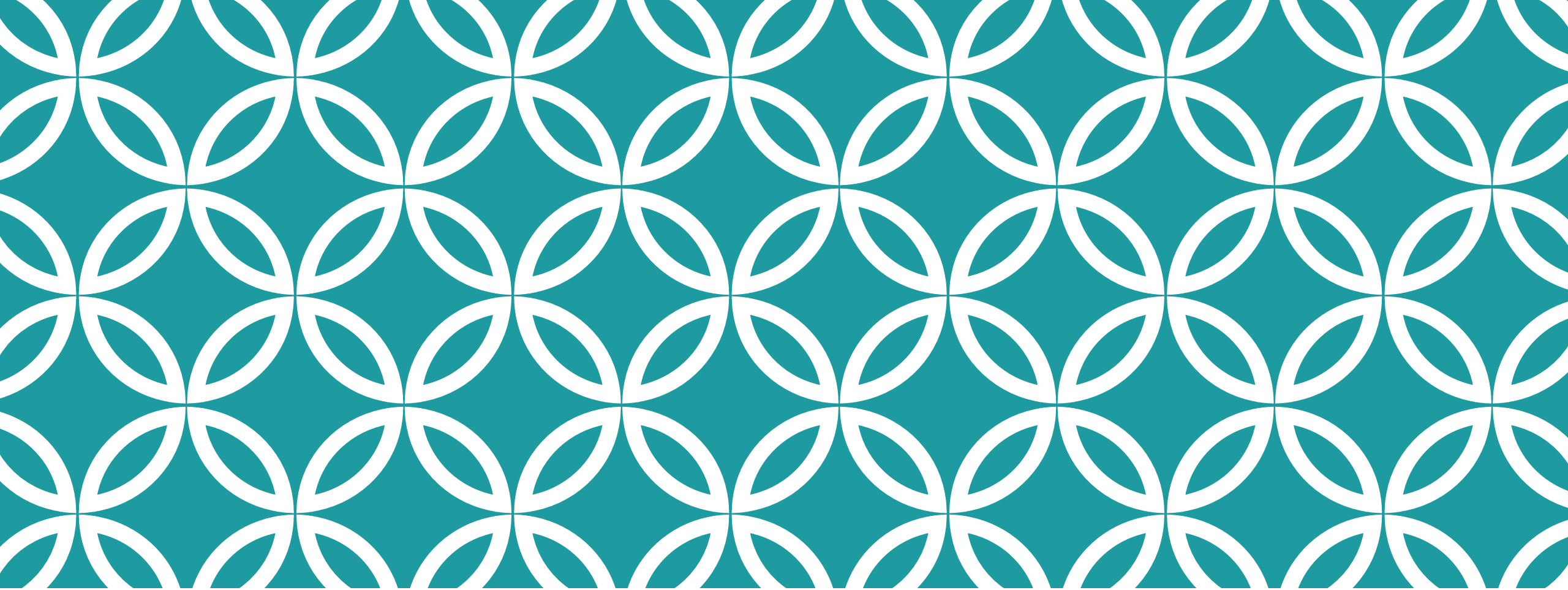
Reduce summarizes the results

MAPREDUCE EXAMPLE

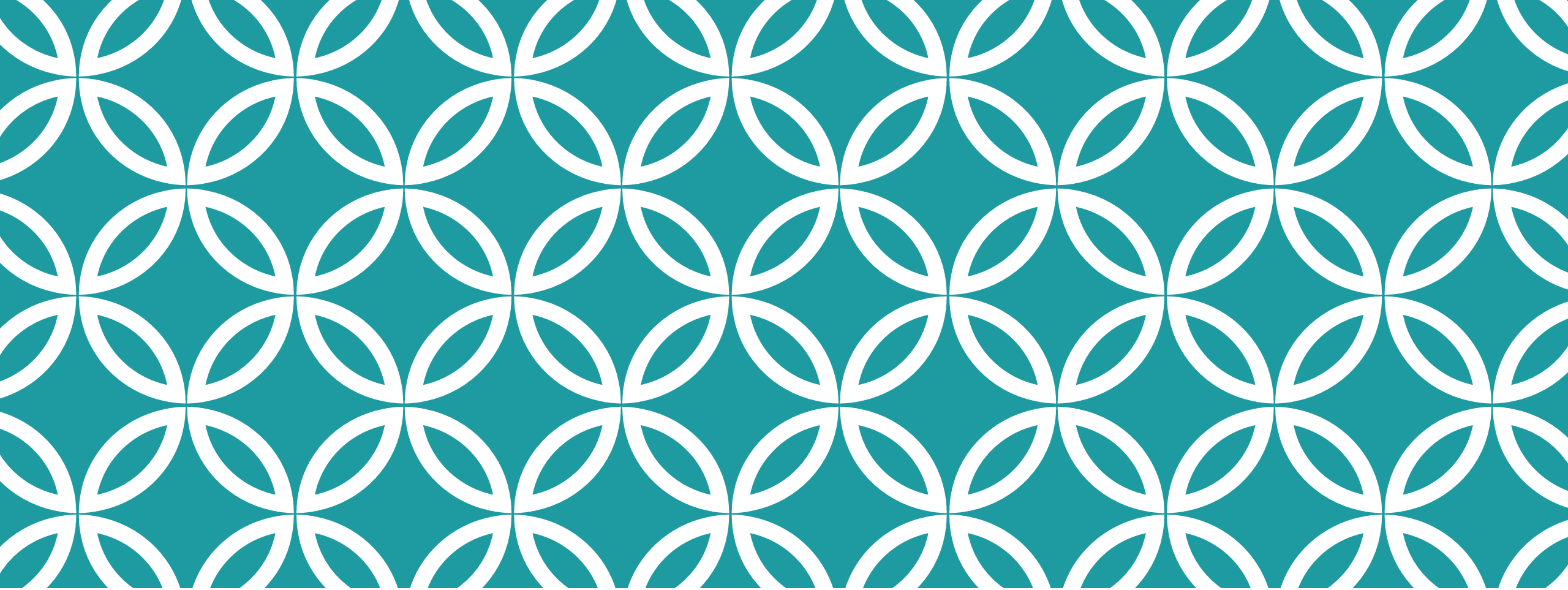


ANOTHER MAPREDUCE EXAMPLE





HANDS-ON



JAVA AND HADOOP MAPREDUCE

MAPREDUCE USE CASES

https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/

(Also stored in d:\lectures\sicsr\big data analytics folder)

WHEN *NOT* TO USE HADOOP

Hadoop is probably not the ideal solution if you need really fast access to data.

Your Queries Are Complex and Require Extensive Optimization.

You Require Random, Interactive Access to Data.

You Want to Store Sensitive Data

You Want to Replace Your Data Warehouse

CODE EXAMPLES (D:\CODE\HADOOP)

Python

- pythonMR
- pythonAge
- pythonBal
- Python country example
- Python
- pythonStudentMaxMarks
- pythonCreditCard
- pythonGenome

Java

- Jcount
- Sales
- ProcessUnits.java

HADOOP-JAVA EXPLANATION

https://dzone.com/articles/introduction-to-hadoop-mapnbspreduce

https://www.edureka.co/blog/mapreduce-tutorial/

https://www.guru99.com/create-your-first-hadoop-program.html

WORDCOUNT EXAMPLE

Refer to d:\code\hadoop\JavaBookEx\chapter_1

Code WordCount.java

Create input.txt

Set HADOOP_CLASSPATH to D:\Java\Jdk1.8.0_231\lib\tools.jar

Refer to “Running the example.txt” file to execute

Explanation: Next slide

EXPLANATION

The WordCount sample uses MapReduce to count the number of word occurrences within a set of input documents.

The code has three parts—Mapper, Reducer, and the main program.

The Mapper extends from the org.apache.hadoop.mapreduce.Mapper interface.

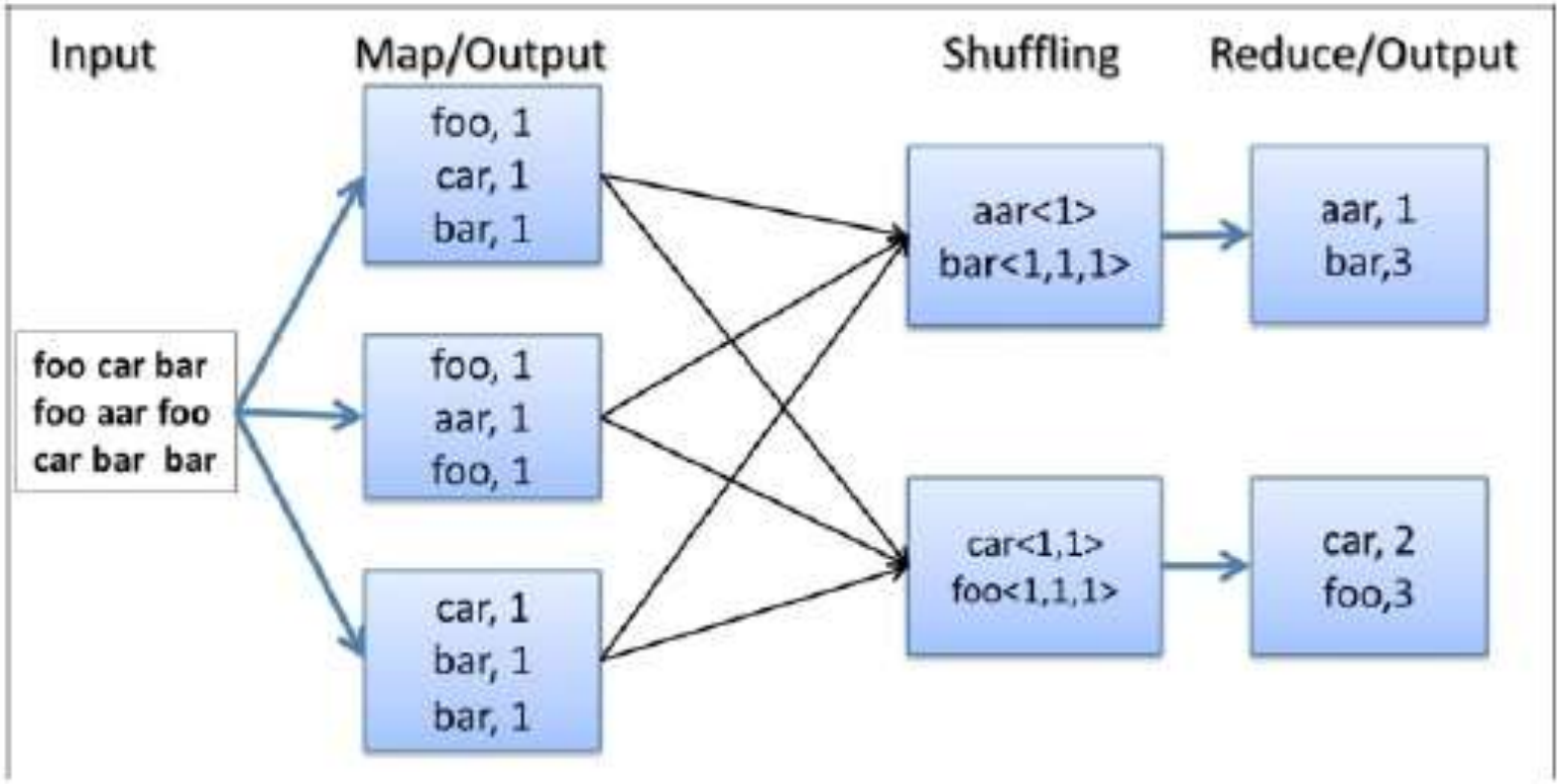
- Hadoop InputFormat provides each line in the input files as an input key-value pair to the map function.
- The map function breaks each line into substrings using whitespace characters such as the separator, and for each token (word) emits (word,1) as the output.

Each reduce function invocation receives a key and all the values of that key as the input.

- The reduce function outputs the key and the number of occurrences of the key as the output.

The main driver program configures the MapReduce job and submits it to the Hadoop YARN cluster.

INTERNAL WORKING



INTERNAL WORKING

1. Hadoop reads the input, breaks it using new line characters as the separator and then runs the map function passing each line as an argument with the line number as the key and the line contents as the value.

2. The map function tokenizes the line, and for each token (word), emits a key-value pair (word,1).

3. Hadoop collects all the (word,1) pairs, sorts them by the word, groups all the values emitted against each unique key, and invokes the reduce function once for each unique key passing the key and values for that key as an argument.

4. The reduce function counts the number of occurrences of each word using the values and emits it as a key-value pair.

5. Hadoop writes the final output to the output directory.

ADDING THE COMBINER STEP

A single Map task may output many key-value pairs with the same key causing Hadoop to **shuffle** (move) all those values over the network to the Reduce tasks, incurring a significant overhead. For example, in the previous WordCount MapReduce program, when a Mapper encounters multiple occurrences of the same word in a single Map task, the map function would output many <word,1> intermediate key-value pairs to be transmitted over the network.

However, we can optimize this scenario if we can sum all the instances of <word,1> pairs to a single <word, count> pair before sending the data across the network to the Reducers.

To optimize such scenarios, Hadoop supports a special function called **combiner**, which performs local aggregation of the Map task output key-value pairs. When provided, Hadoop calls the combiner function on the Map task outputs before persisting the data on the disk to shuffle the Reduce tasks.

This can significantly reduce the amount of data shuffled from the Map tasks to the Reduce tasks. It should be noted that the combiner is an optional step of the MapReduce flow. Even when you provide a combiner implementation, Hadoop may decide to invoke it only for a subset of the Map output data or may decide to not invoke it at all.

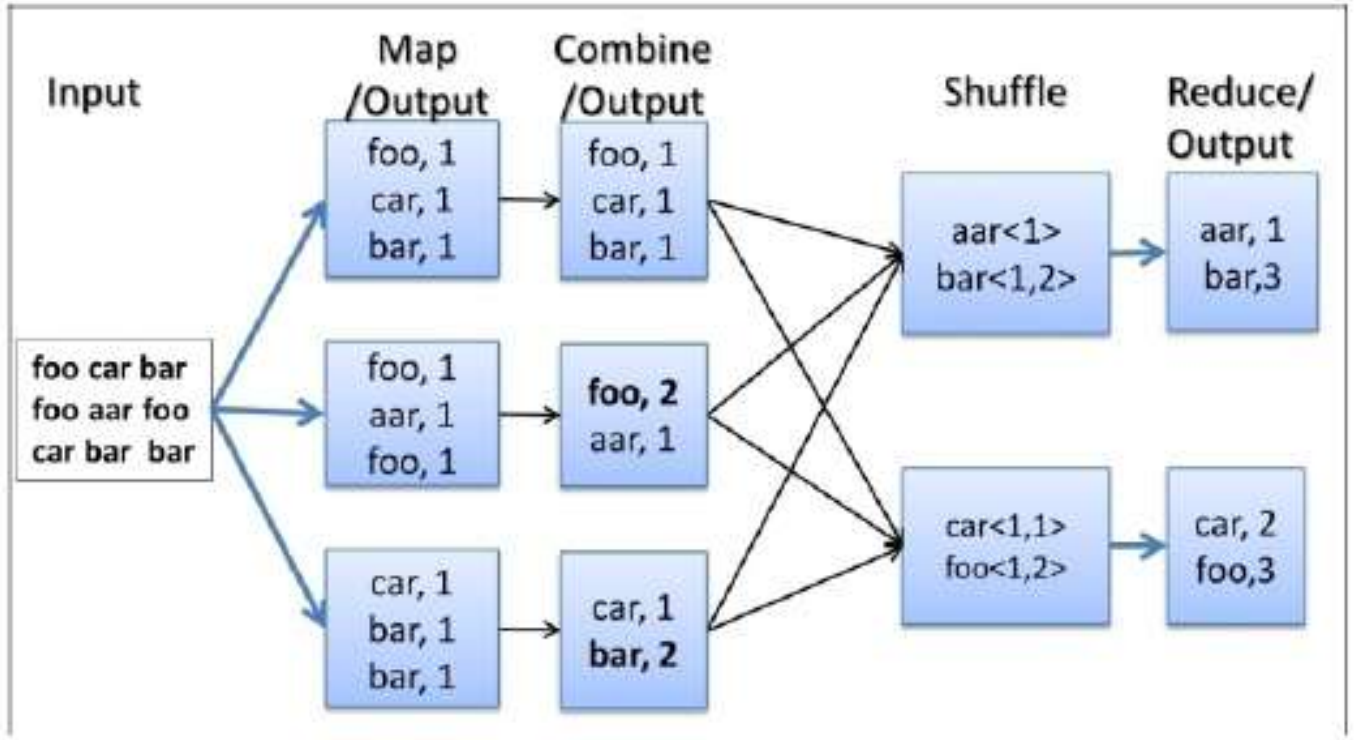
HOW IT WORKS

When provided, Hadoop calls the combiner function on the Map task outputs before persisting the data on the disk for shuffling to the Reduce tasks. The combiner can preprocess the data generated by the Mapper before sending it to the Reducer, thus reducing the amount of data that needs to be transferred.

In the WordCount application, combiner receives *N* number of (word,1) pairs as input and outputs a single (word, N) pair.

For example, if an input processed by a Map task had 1,000 occurrences of the word “the”, the Mapper will generate 1,000 (the,1) pairs, while the combiner will generate one (the,1000) pair, thus reducing the amount of data that needs to be transferred to the Reduce tasks.

HOW IT WORKS


Input
Map /Output
Combine /Output
Shuffle
Reduce/ Output
foo car bar
foo aar foo
car bar bar
foo, 1
car, 1
bar, 1
foo, 1
car, 1
bar, 1
foo, 1
aar, 1
foo, 1
foo, 2
aar, 1
car, 1
bar, 1
bar, 1
car, 1
bar, 2
aar<1>
bar<1,2>
car<1,1>
foo<1,2>
aar, 1
bar,3
car, 2
foo,3

CODE CHANGES

Uncomment the following line in the WordCount.java file to enable the combiner for the WordCount application:

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

Then recompile the code and execute.

SIMPLE ANALYTICS

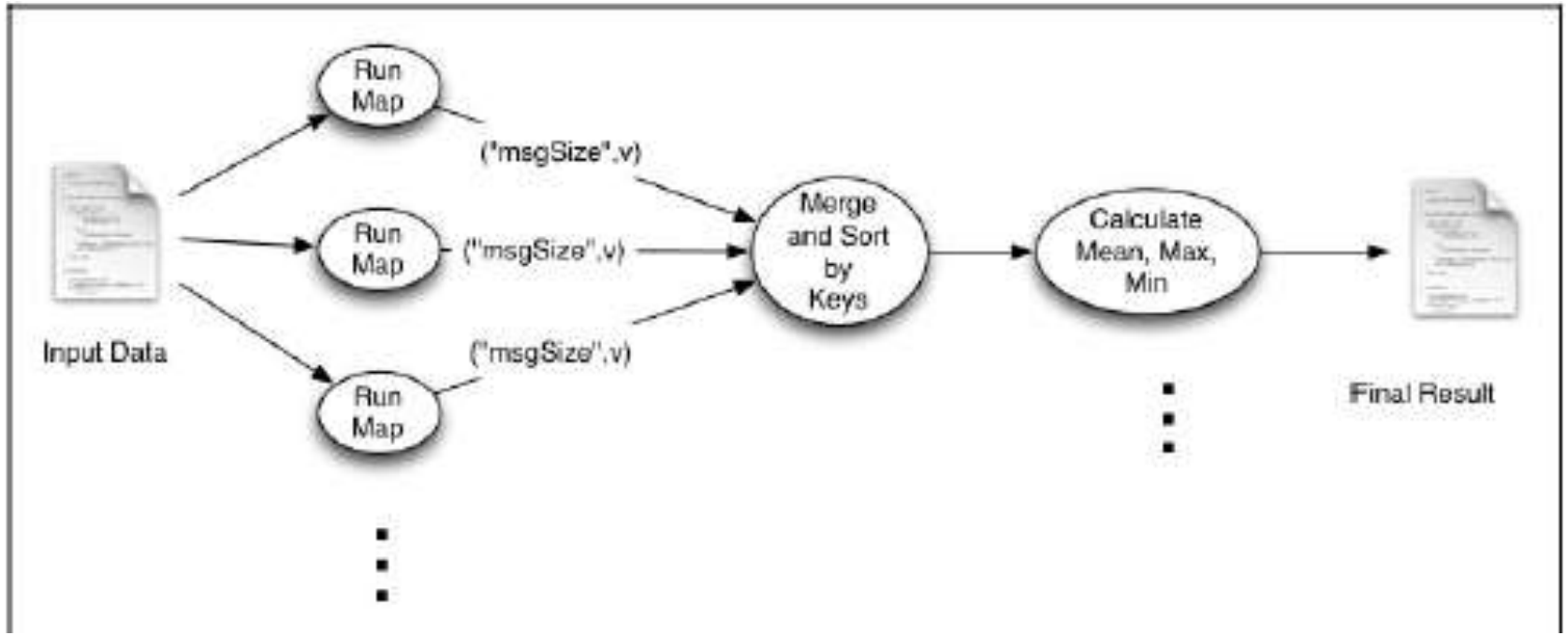
MsgSizeAggregateMapReduce in d:\code\hadoop\JavaBookEx\chapter_5

Aggregate metrics such as mean, max, min, standard deviation, and so on, provide the basic overview of a dataset.

You may perform these calculations, either for the whole dataset or to a subset or a sample of the dataset.

In this example, we will use Hadoop MapReduce to calculate the minimum, maximum, and average size of files served from a web server, by processing logs of the web server.

UNDERSTANDING THE PROCESSING



EXPLANATION

The Map function emits the size of the file as the value and the string msgSize as the key.

We use a single Reduce task, and all the intermediate key-value pairs will be sent to that Reduce task.

Then, the Reduce function calculates the aggregate values using the information emitted by the Map tasks.

HTTP logs follow a standard pattern as follows. The last token is the size of the web page

served:

205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET

/shuttle/countdown/countdown.html HTTP/1.0" 200 3985

We will use the Java regular expressions to parse the log lines, and the Pattern.compile() method at the top of the class defines the regular expression. Regular expressions are a very useful tool while writing text-processing Hadoop computations.

EXPLANATION

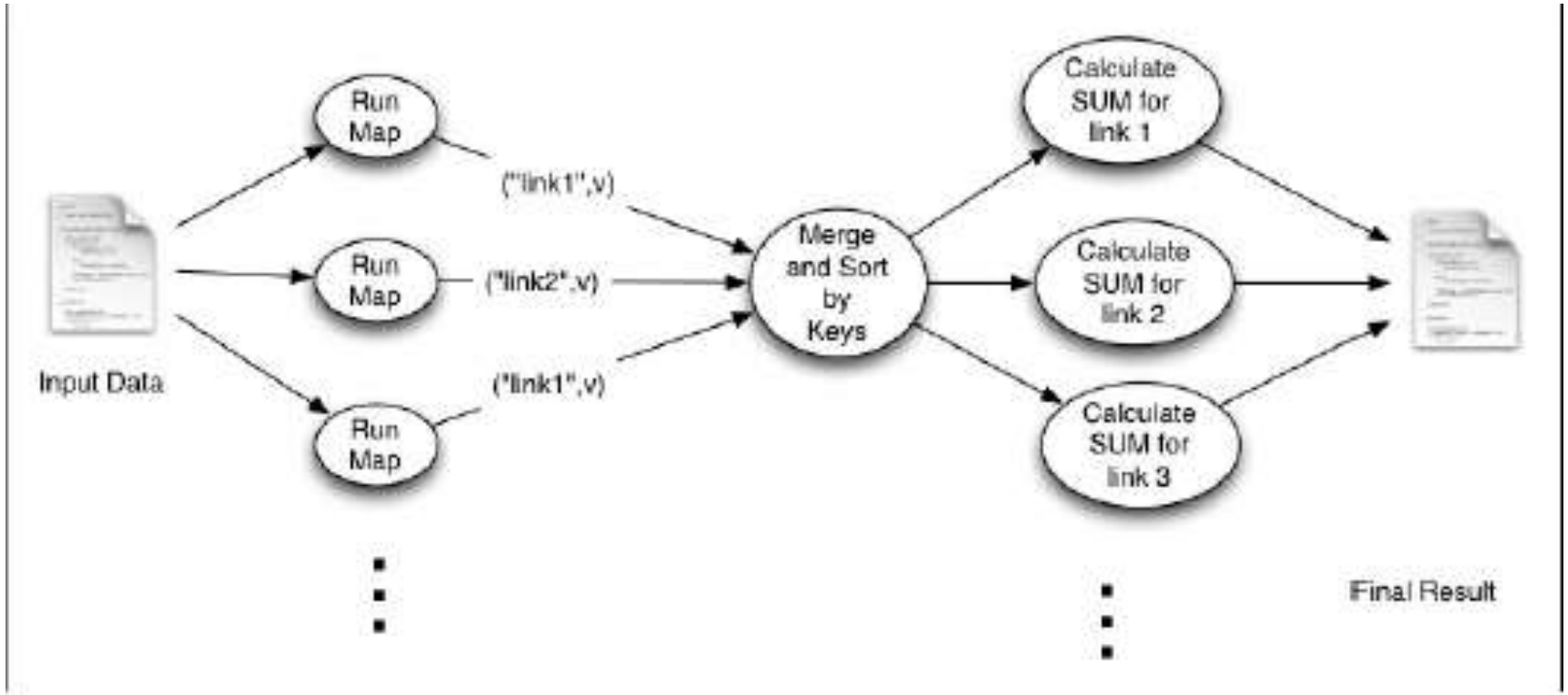
Map tasks receive each line in the log file as a different key-value pair. It parses the lines using regular expressions and emits the file size as the value with msgSize as the key.

Next, Hadoop collects all the output key-value pairs from the Map tasks and invokes the Reduce task. Reducer iterates all the values and calculates the minimum, maximum, and mean size of the files served from the web server.

It is worth noting that by making the values available as an iterator, Hadoop allows us to process the data without storing them in memory, allowing the Reducers to scale to large datasets.

Whenever possible, you should process the reduce function input values without storing them in memory.

GROUP BY EXAMPLE



EXPLANATION

The **Map** tasks emit the requested URL path as the key.

Then, Hadoop sorts and groups the intermediate data by the key.

All values for a given key will be provided into a single Reduce function invocation, which will count the number of occurrences of that URL path.

HitCountMapReduce.java

FIND MAXIMUM TEMPERATURE FOR EVERY YEAR

D:\code\hadoop\JavaBookEx\maxtemp

Mapper parses each line in input file to retrieve only the year and temperature fields and adds them to the output line to the Reducer

Reducer finds the maximum for each year group and finally prints year and max temperature for that year

Same example in Python

D:\code\hadoop\pythonMaxTemp

BATCH ANALYSIS OF SENSOR DATA

For this example, we will assume that we have a data collector, which retrieves the sensor data collected in the cloud database and creates a raw data file in a form suitable for processing by Hadoop.

The raw data file consists of the raw sensor readings along with the timestamps as shown below:

"2015-04-29 10:15:32",38,42,34,5

:

"2015-04-30 10:15:32",87,48,21,4

SENSOR DATA EXAMPLE

d:\code\hadoop\pythonSensorData\Mapper.py, Reducer.py and sensor.txt

D:\code\hadoop\pythonSensorData>type sensor.txt | python Mapper.py | python Reducer.py

"2020-04-29 10:15 [37.0, 43.6, 32.2, 4.0]

"2020-04-29 10:16 [35.4, 43.2, 31.8, 3.2]

COUNTRY-WISE SALES

C:\Windows\System32\cmd.exe

D:\code\hadoop\pythonCountrySales>type data.csv | python mapper.py | python reducer.py

Note: Data size is quite large (File is here: https://www.kaggle.com/carrie1/ecommerce-data/data)

FIND AVERAGE SALARY

D:\code\hadoop\JavaBookEx\avgsal

AverageSalary.java

EVEN/ODD NUMBERS

Find total and count of even and odd numbers

D:\code\hadoop\JavaBookEx\evenodd

WHERE TO SEE SYSTEM.OUT.PRINTLN OUTPUT

Example:

D:\Hadoop-2.8.0\logs\userlogs\application_1617885109459_0012\container_1617885109459_0012_01_000003