
DADS 6002 / CI 7301

Big Data Analytics

Data

Three types of data

1. Structured Data, e.g. record based data, relational tables.
2. Unstructured Data, e.g. text, log files, audio, voice, video, graph data (e.g. social media, webs).
3. Semi-structured Data, e.g. HTML file, XML, JSON.






```
1 {  
2   "string": "Hi",  
3   "number": 2.5,  
4   "boolean": true,  
5   "null": null,  
6   "object": { "name": "Kyle", "age": 24 },  
7   "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],  
8   "arrayOfObjects": [  
9     { "name": "Jerry", "age": 28 },  
10    { "name": "Sally", "age": 26 }  
11  ]  
12 }  
13
```

Big Data

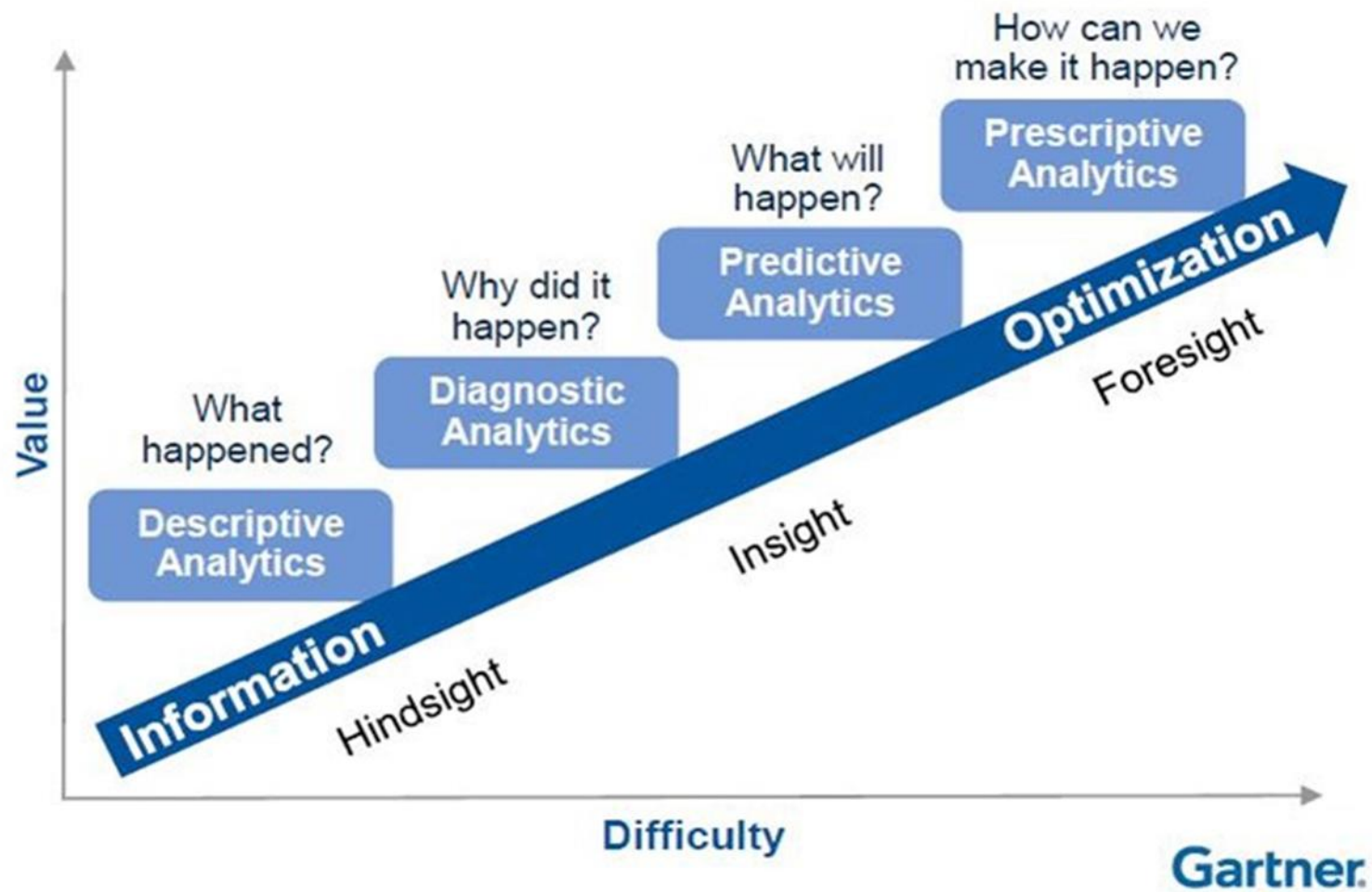
- Big Data Characteristics 5Vs
 1. Volume, Large enormous volume of data
 2. Variety, Many sources and types of data both structured or unstructured data
 3. Velocity, Flow of data from the source is massive and continuous e.g. real time data
 4. Veracity, Biases (opinions), noise and abnormality in data, quality of data
 5. Value, ability to turn data to value

The five Vs of big data

Big data is a collection of data from various sources, often characterized by what's become known as the 3Vs: *volume, variety and velocity*. Over time, other Vs have been added to descriptions of big data:

| VOLUME | VARIETY | VELOCITY | VERACITY | VALUE |
|---|---|--|---|---|
| The amount of data from myriad sources. | The types of data: structured, semi-structured, unstructured. | The speed at which big data is generated. | The degree to which big data can be trusted. | The business value of the data collected. |
|  |  |  |  |  |

Source : <https://searchcloudcomputing.techtarget.com/tip/Cost-implications-of-the-5-Vs-of-big-data>



Data product

- A data driven application that derives value from data and produce more data, more value in return.
- A system that learns from data, self-adapting and broadly applicable
- A data product is an application or tool that uses data to help businesses improve their decisions and processes.

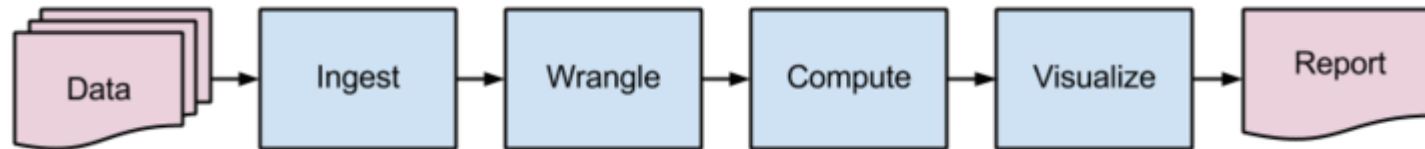
Examples :

- Amazon examines items customer purchased then makes recommendation based on similar purchase behaviors of other customers.
- Facebook uses social network data graph algorithms to infer of communities (based on mutual friends).
- Sensor data from autonomous vehicles can be used to improve driving platform.

Data Science

- Science that combines multi disciplines which aims at building efficient and effective data products.
- The multi disciplines include
 1. Machine Learning
 2. Software Engineering
 3. Statistics
 4. Database Management
 5. Distributed Computing

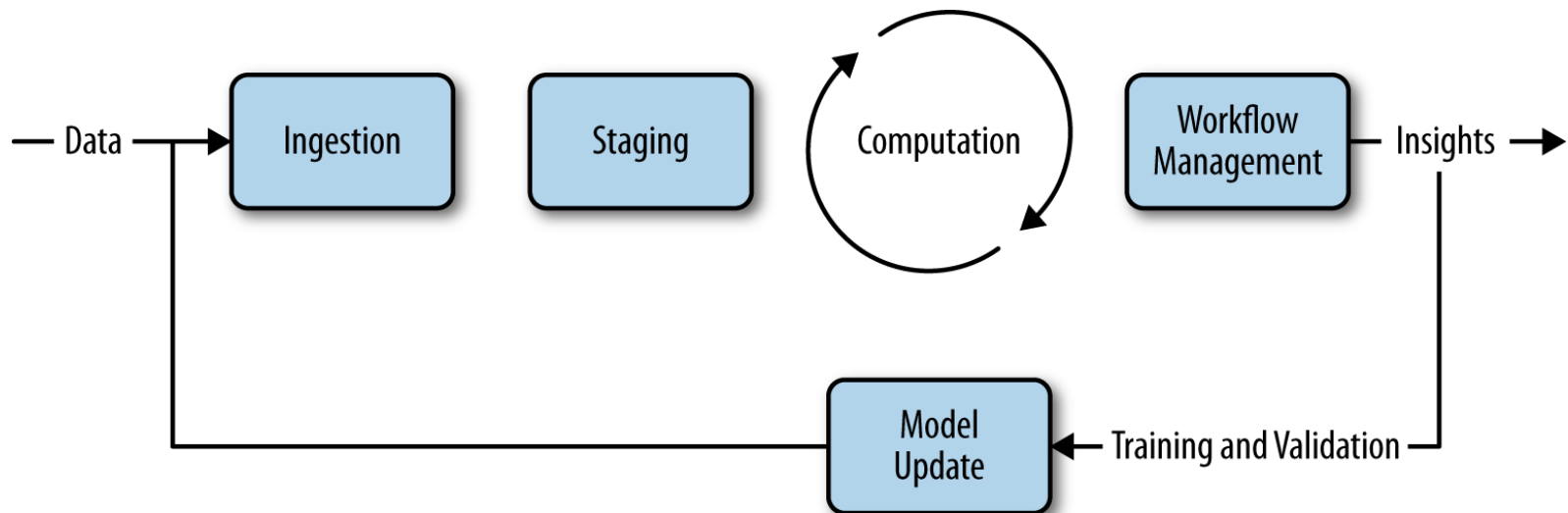
Data Science Pipeline



- A workflow Model for building a data product
- Human-powered augmented by used of scripting languages
- Not suitable for building a system that deals with larger, faster and more variable data
- May need tools that automatically derive insight or patterns without human intervention

Big Data Workflows

- For big data, human driven data science pipeline can be modified into an iterative model (Big Data Pipeline) with four primary phases.



Big Data Workflows

- Ingestion Phase includes the initialization of a model (Data sources are located and data is imported) and application interaction between users and the model (users consume the model and provide feedback that is used to reinforce or adjust the model)
- Staging Phase includes transformations of data to make it consumable and stored so that it can be available for processing (the transformations include normalization, standardization, data management)

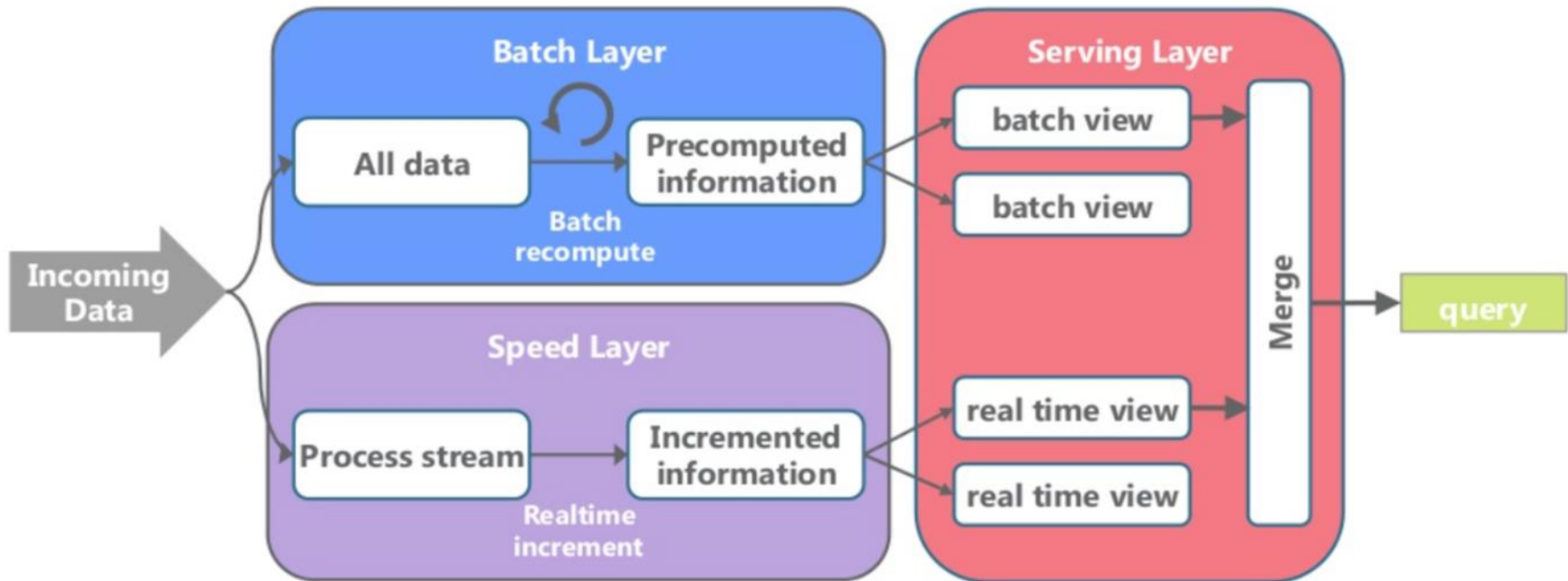
Big Data Workflows

- Computation Phase, Machine Learning is performed to get insights or Models
- Workflow Management Phase performs abstraction, orchestration and automation tasks that enable the workflow steps to be operationalized for production.

Lambda Architecture

- A Lambda architecture is a data-processing architecture combining both batch- and (real-time) stream-processing methods. The advantage of this dual architecture is the ability to handle massive amounts of data while maintaining real-time monitoring. As a result, it is generally used as a basis for [big data](#) architectures.

Lambda Architecture



Lambda architecture duplicates incoming data and processes them in parallel at different speeds

Source : <http://www.element61.be/nl/resource/lambda-architecture>

Lambda Architecture

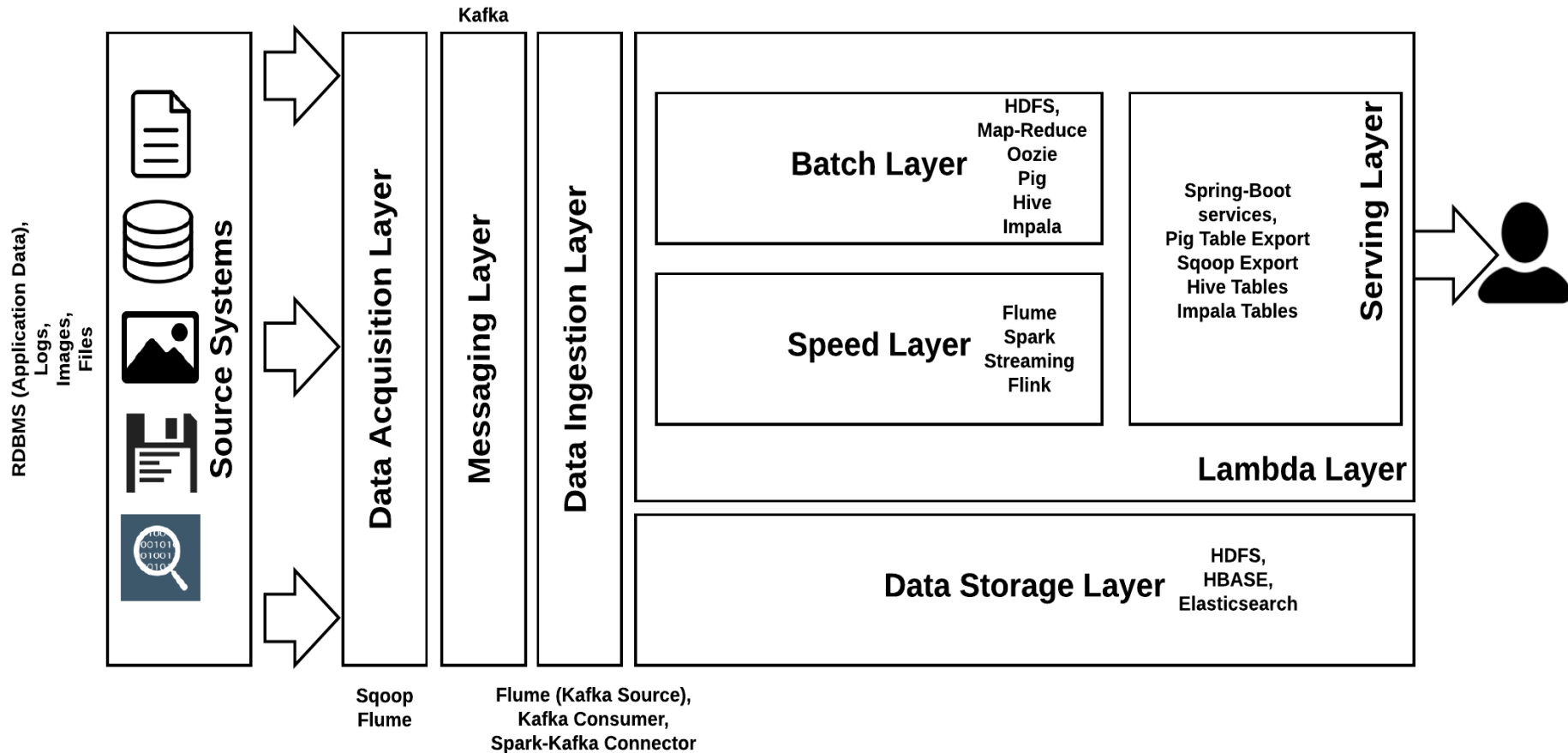
- A Lambda architecture generally has three major components:
- Batch layer aimed to
 - keep and secure the master raw dataset (historical and latest data)
 - provide pre-computed views (in batch) on business-relevant aggregations and metrics. (One can compare this layer to the conventional DWH layer currently available in BI)

Lambda Architecture

- Speed layer designed to deliver fast, real-time data streams which have low latency requirements
- Serving layer designed to interface with the end-user and consuming from both the batch and speed layer.

A serving layer can be seen as a dashboarding/reporting layer aimed to handle both batch reporting as well as real-time reporting

Lambda Architecture



Data Lake

- A data lake is a method of storing data within a system or repository, in its natural format, that facilitates the collocation of data in various schemata and structural forms, usually object blobs or files. The idea of data lake is to have a single store of all data in the enterprise ranging from raw data (which implies exact copy of source system data) to transformed data which is used for various tasks.

Source : https://en.wikipedia.org/wiki/Data_lake

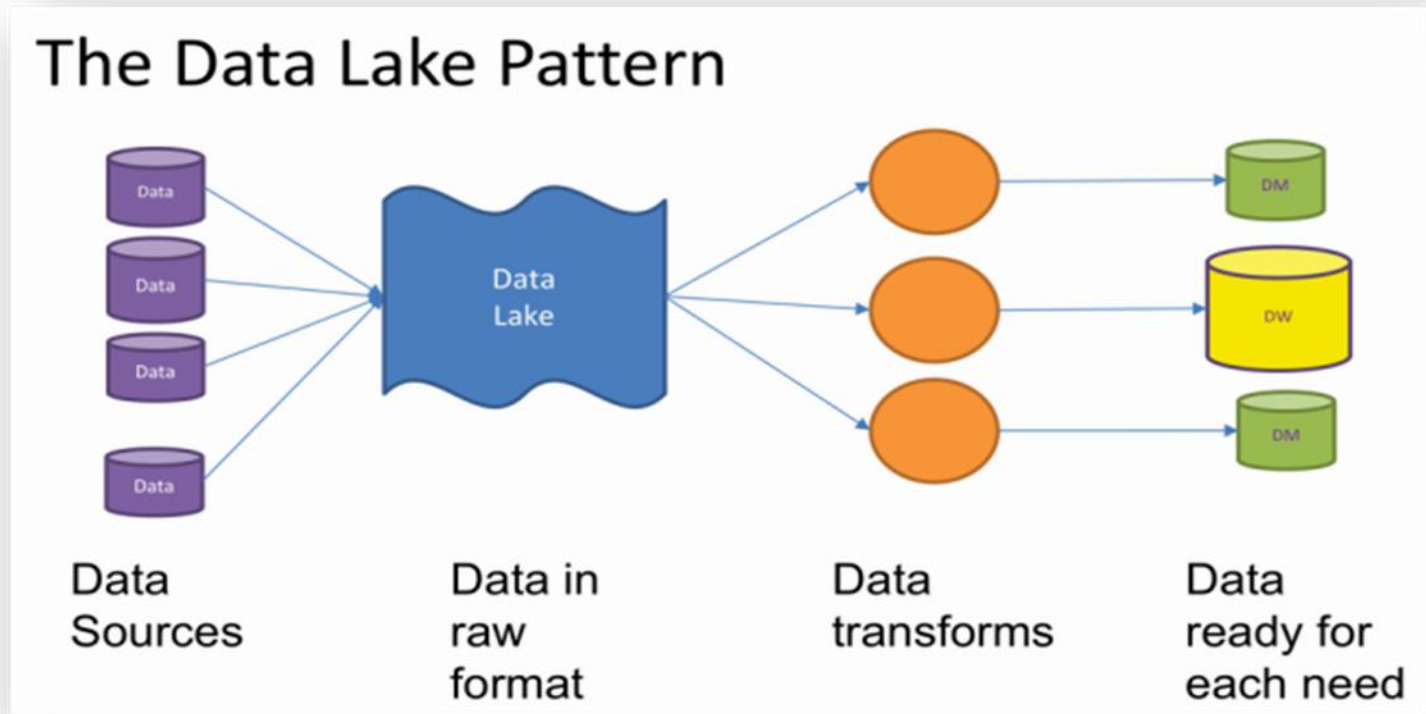
Data Lake

- The tasks include reporting, visualization, analytics and machine learning.
- The data lake includes structured data from relational databases (rows and columns), semi-structured data (CSV, logs, XML, JSON), unstructured data (emails, documents, PDFs) and even binary data (images, audio, video) thus creating a centralized data store accommodating all forms of data.

Data Lake

- Structured and unstructured data flow into a data lake which can be implemented by distributed data storages e.g. Hadoop File System, Amazon S3.
- The data stored in the data lake is then queried against using Extract-Transform-Load (ETL) processes to produce a data warehouse or local data marts (e.g. using Hbase or Hive) that can be analyzed by some data analytic tools or programs.

Data lake



Source : <https://www.dragon1.com/terms/data-lake-definition>

Data Lake

Data Warehouse vs. Data Lake – Data Loading

Data Warehouses use a traditional ETL Process



Data is transformed when it enters the data warehouse

Data Lakes make use of the ELT Process



Data is transformed when it is retrieved from the data lake

Hadoop

- Hadoop becomes an ecosystem of tools that operationalize some parts of Big Data Pipeline, for examples:
- Sqoop, Flume, and Kafka are designed for data ingestion which allow import of data into Hadoop file system or Database
- Hbase and Hive provide data management on Hadoop
- Spark's GraphX and MLlib or Mahout provide analytical packages for Big Data Analytic

Hadoop

- Operating System for Big Data, developed by Googles
- It distributes an analytical computation that involves a massive dataset to many machines, each simultaneously operates on their own individual chunk of data so the whole computation can be speeded up.
- It requires distributed algorithms, machines in clusters and networking services

Hadoop

- Hadoop addresses the following requirements for distributed systems :
- 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.

Hadoop

- Recoverability, in the event of failure, no data should be lost.
- Consistency, the failure of one job should not effect the final results.
- Scalability, Adding load (more data, more computation) leads to a decline in performance not failure. Increasing resources should result in a proportional increase in performance.

Hadoop

Hadoop implements the following concepts to achieve the distributed system requirements

1. Data is distributed immediately when added to the cluster and stored on multiple nodes
2. Nodes prefer to process data that is stored locally to minimize communication traffic across the network
3. Data is stored in blocks of a fixed size (usually 128 MB). Each block is duplicated multiple times across the system to provide redundancy and data safety.

Hadoop

4. 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
5. Jobs are written at a high level without concern for network programming and underlying communication.
6. The amount of network traffic between nodes should be minimized. Each task should be independent and nodes should not have to communicate with each other during processing (no inter process dependencies to avoid deadlock situations)

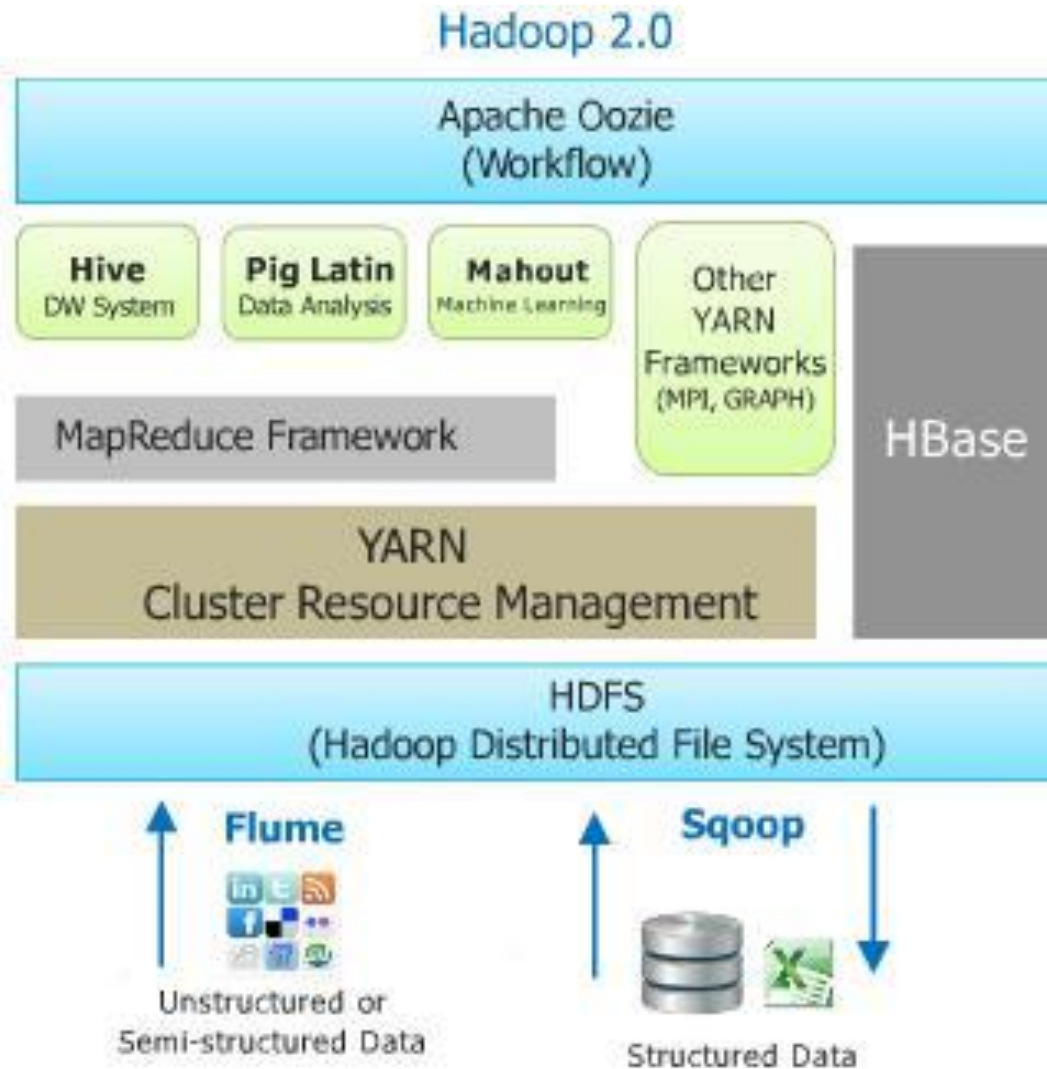
Hadoop

7. Jobs are fault tolerant usually through task redundancy
8. Master programs allocates work to worker nodes such that many worker nodes can operate in parallel, each on their own portion of the larger dataset.

Hadoop Architecture

- Hadoop is composed of two primary components
 1. HDFS (Hadoop Distributed File System) responsible for managing data stored on disk access across the cluster.
 2. YARN (Yet Another Resource Negotiator) acts as a cluster resource manager, allocating computational assets (e.g. processing availability and memory on worker nodes) to applications.

Hadoop Architecture



Hadoop Architecture

- HDFS and YARN work in concert to minimize network traffic (ensure data is local to required computation)
- Replicate data blocks to ensure fault tolerance, recoverability while maintain data consistency.
- Speculative Execution - launch duplicate tasks to run on some other nodes if original tasks show no sign of progress. Duplicate tasks are killed when one has completed.
- YARN and HDFS are implemented by several daemon processes that provides services (accept input and deliver output through the network, similar to HTTP server)

Hadoop Architecture

- A set of machines that is running HDFS and YARN is known as a cluster of machines called nodes.
- Each node in the cluster is identified by the type of process or processes that it runs.
- Master nodes run coordinating services for Hadoop workers (are usually entry points for user access to the cluster)
- Worker nodes run services that accept tasks from master nodes (store, retrieve data or run a particular application)

HDFS

- HDFS has multiple master and worker nodes to provide services :
1. NameNode (Master) stores the directory tree of the file system, file metadata and locations of each file in the cluster. Access HDFS file, client must request information about the storage nodes of the file from the NameNode. It also maintains the fsimage file (the snapshot of the file system info when the NameNode starts) and the editlogs file (the sequence of changes made to the file system info after NameNode started).

HDFS

2. Secondary NameNode (Master) performs housekeeping tasks and checkpointing on behalf of the NameNode (not a backup site of the NameNode), i.e. periodically reads the editlogs of the NameNode and applies the changes in the logs to its own copy of fsimage and copy the updated fsimage back to the NameNode (as well as clear the editlogs).
3. Data Nodes (Worker) Stores and manages HDFS blocks on the local disk, report health and status of individual data stores back to the NameNode.

YARN

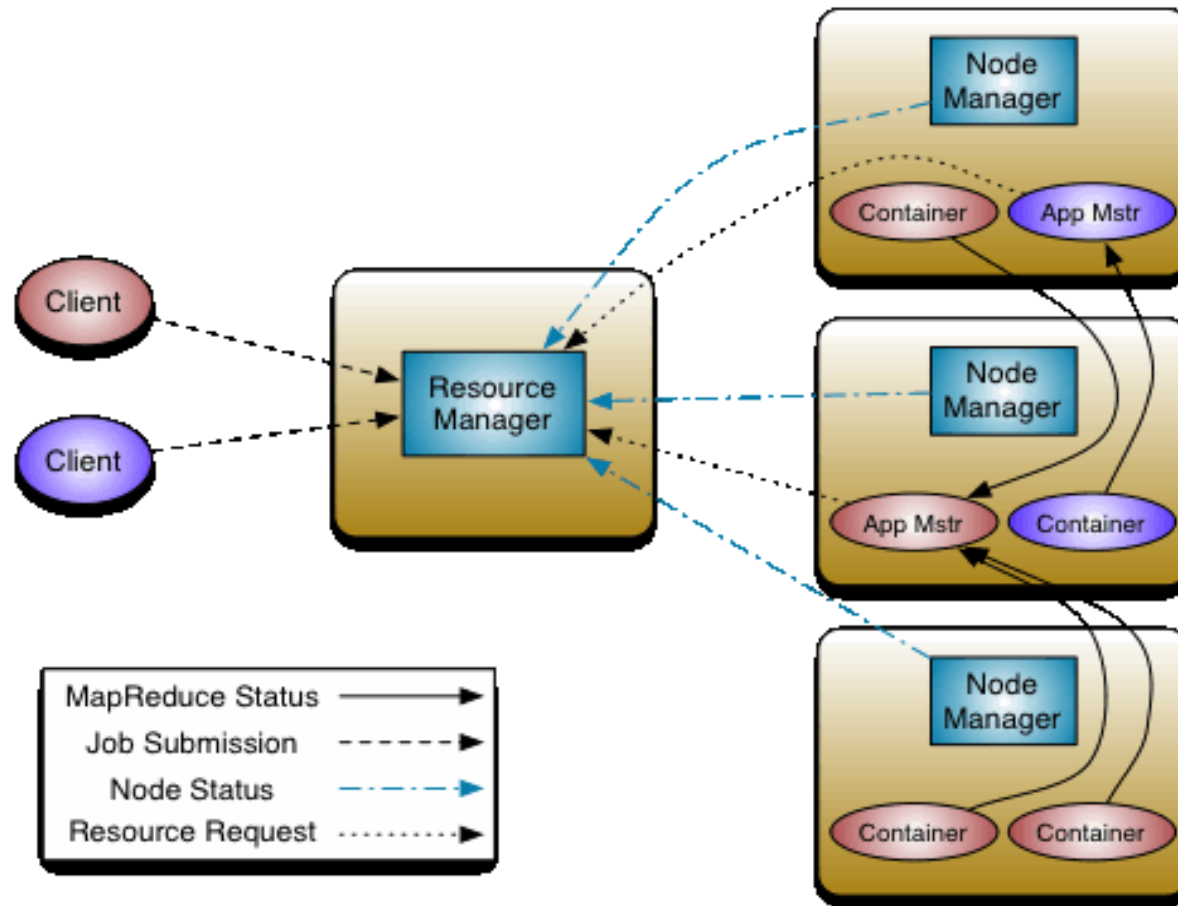
- Has multiple master services and a worker service as follows:
 1. Resource Manager (Master) allocates and monitors available cluster resources e.g. physical memory, processor cores to application as well as handling scheduling of jobs on clusters.
 2. Application Master – launched per application (Master) coordinates a particular application being run on the cluster as scheduled by the Resource Manager.

YARN

3. Node Manager (Worker) runs and manages processing tasks on an individual node as well as reports the health and status of tasks.

Master processes are important. They should run on their own nodes so they don't compete for resources and present bottleneck. However, for a small cluster, two master nodes may be sufficient.

YARN



HDFS concepts

- Provides redundant storage for big data by storing the data across a cluster.
- A software layer on top of a native file system
- It performs best with a modest number of very large files
- It implements the WORM pattern (write once, read many). No random writes or appends to files are allowed (not fit for application that require updates in real-time interactive data analysis or record-based transactions).
- HDFS files are split into blocks, usually of either 64 MB or 128 MB

HDFS concepts

- The block size is the minimum amount of data that can be read or written to the file.
- Blocks of a file are distributed to many Data Nodes. They are also replicated for fault tolerance (by default, each block is replicated to three machines)
- Master NameNode keeps track of what blocks make up a file and where those blocks are located. These meta data are loaded into memory for quick lookups when accessing the file.
- Secondary NameNode performs housekeeping tasks on behalf of NameNode e.g. editing log records, checkpointing meta data present on NameNode.

Working with HDFS

- Interaction with HDFS can be performed through
 1. Command Line Interface (CLI)
 2. HTTP interface (communicate with HDFS via web browser)
 3. API (via programming interface written in JAVA)

Examples of CLI with HDFS

```
$ hadoop fs -put x.txt /user/cloudera/y.txt
```

```
$ hadoop fs -get /user/cloudera/y.txt x.txt
```

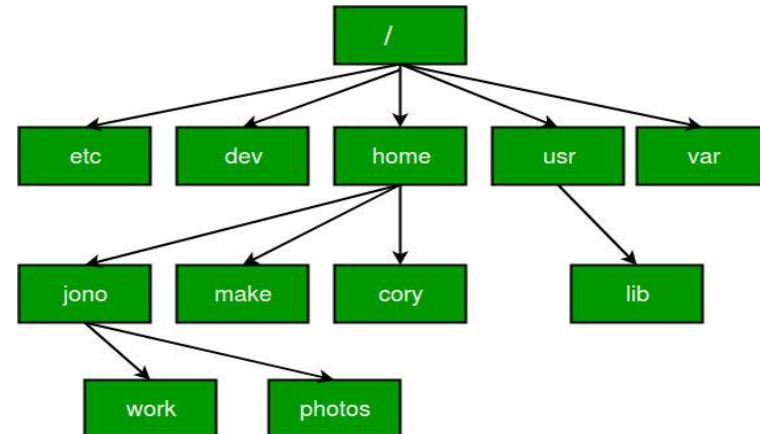
```
$ hadoop fs -mkdir -p /user/cloudera/corpora
```

```
$ hadoop fs -ls /user/cloudera
```

```
$ hadoop fs -mv /user/cloudera/x.txt /user/cloudera/rawdata
```

```
$ hadoop fs -cp /user/cloudera/x.txt y.txt
```

```
$ hadoop fs -rm /user/cloudera/x.txt
```



Examples of CLI with HDFS

```
$ hadoop fs -chmod 664 /user/cloudera/x.txt
```

(-rw-rw-r— permission is set for the given file)

MapReduce

- A simple but very powerful computational framework designed to enable fault-tolerant distributed computation across a cluster.
- It employs a functional programming style that allow multiple independent tasks to execute a function on local chunks of data.
- Data is transferred between functions by sending the output of one function as the input to another.
- Functions are stateless and depend solely on their input.

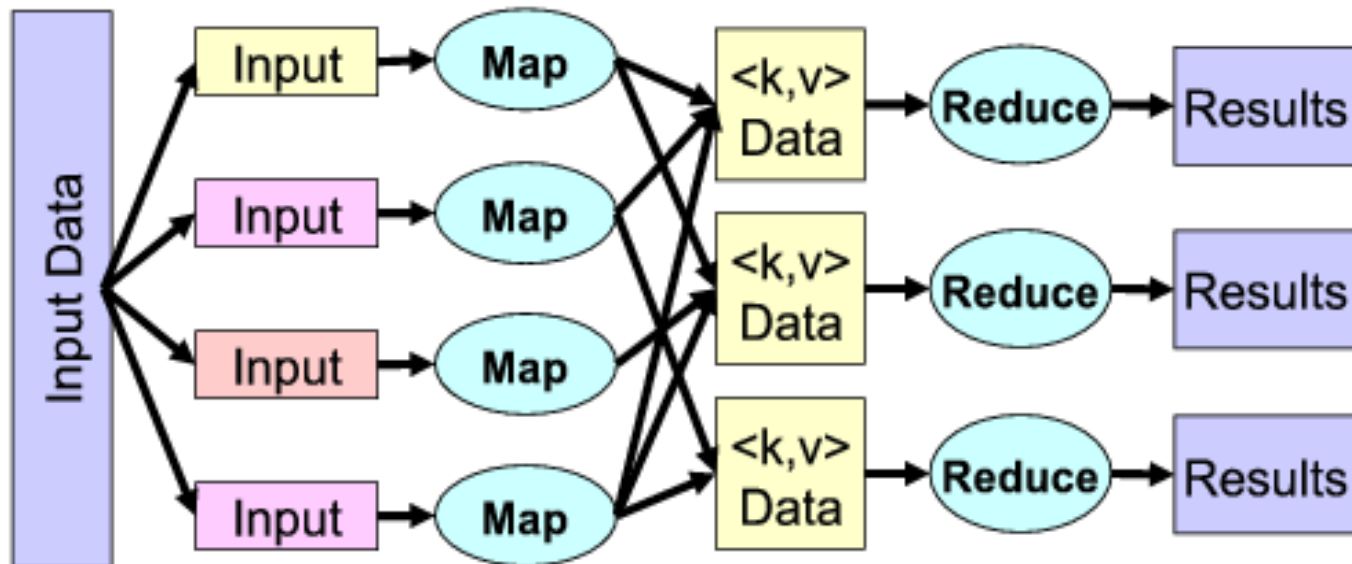
MapReduce

- Two functions that distribute work and aggregate results are *map* and *reduce*.
- MapReduce utilizes “Key/Value” pairs to coordinate computation.
- Map function takes as input a series of key/value pairs (input list) and operates on each individual pair to produce zero or more intermediate key/value pairs (output list)

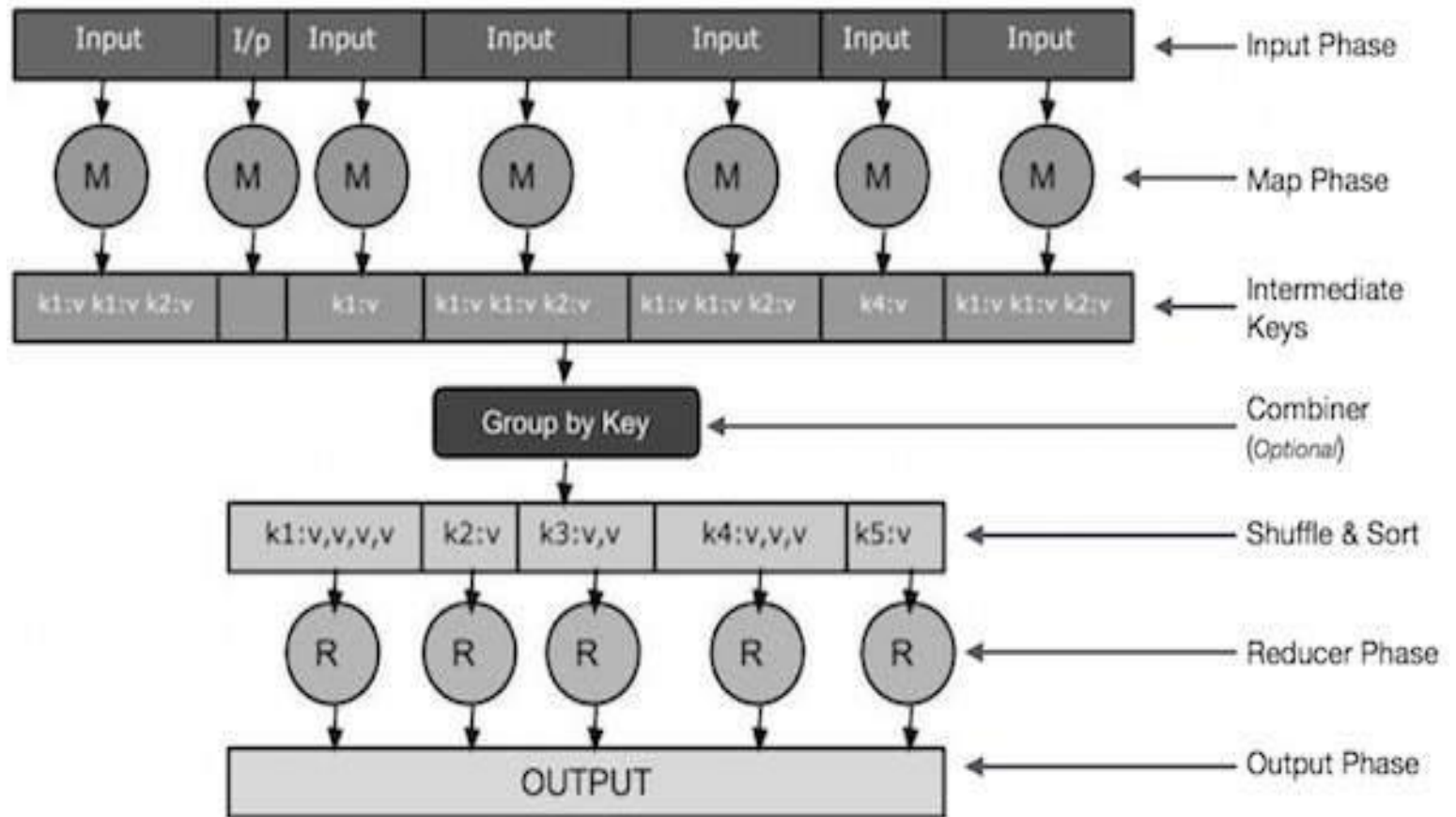
MapReduce

- The key/value pairs in the output list are then sorted and shuffled based on the key by a partitioner (Hadoop system function) and then passed to the reduce function.
- The reduce function performs final processing on the list usually combination or aggregation, and then output zero or more final key/value pairs.

MapReduce



MapReduce



MapReduce Example

- Word Counting problem (Psuedo code)

```
def map ( dockey, line );
```

```
    for word in line.split();
```

```
        emit( word, 1 )
```

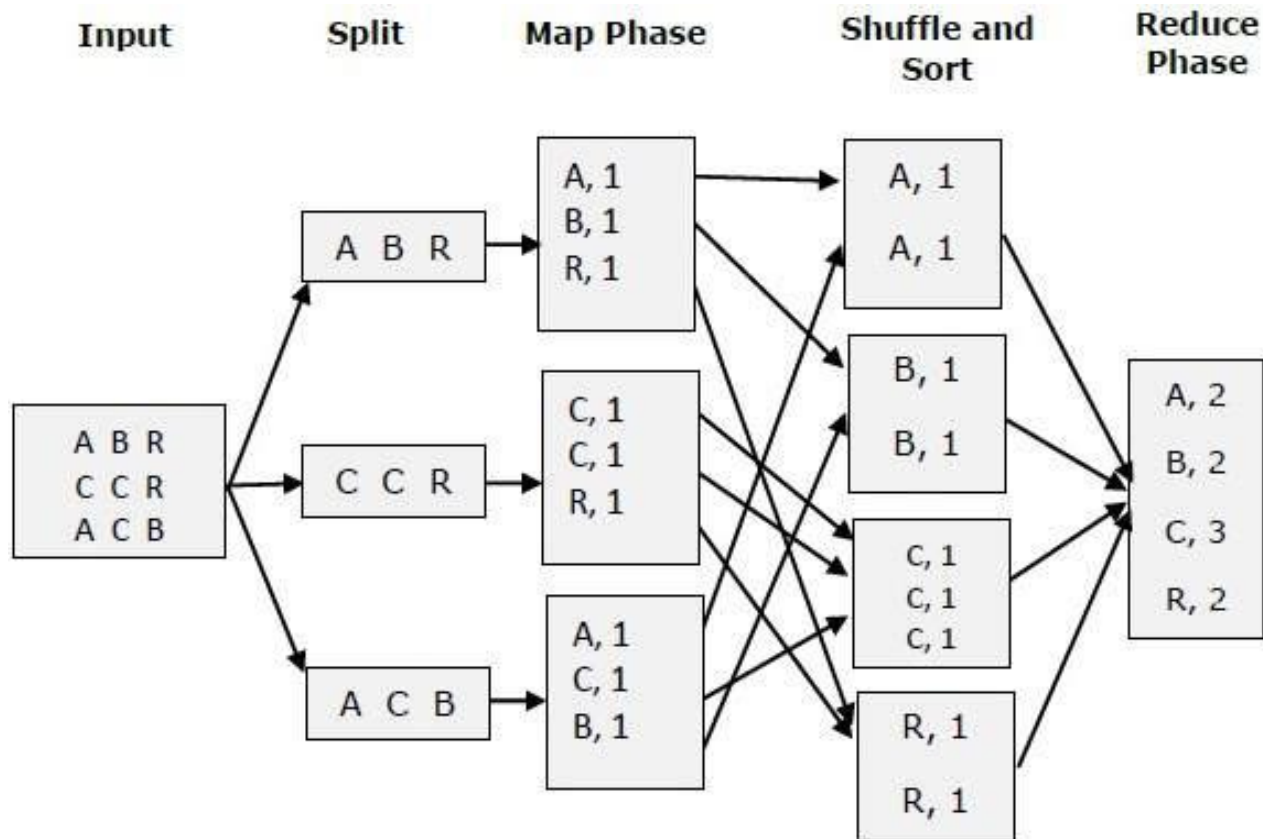
```
def reduce ( word, values );
```

```
    count = sum ( value for value in values )
```

```
    emit( word, count )
```

(note: emit – perform Hadoop I/O which sends its arguments to the next phase of MapReduce pipeline)

MapReduce Example



MapReduce example

- Mappers receive key/value pairs
Key = line id (block id)
Value = string
- Each mapper can work on a single document or if the documents are very large, mappers can work on chunks of single document.
- The output key is a word and the output value is an integer.

MapReduce Example

Block 1

(27183, "The fast cat wears no hat.")



Mapper 1



("The",1), ("fast",1), ("cat",1), ("wear",1), ("no",1),
("hat",1), (".", 1)

MapReduce Example

Block 2

(31416, "The cat in the hat ran fast.")



Mapper 2



("The",1), ("cat",1), ("in",1), ("the",1), ("hat",1),
("ran",1), ("fast",1), (".", 1)

MapReduce Example

- Outputs from all mappers are grouped together and sorted by key in Shuffle and Sort phase which is performed by Hadoop itself (by default).
- The output after Shuffle and Sort phase contains a list of keys and their lists of associated values.
- The output is then partitioned into sets.
- Each set will be sent to each reducer for final computation.

MapReduce Example

- The output from shuffle and sort phase

(“.”, [1, 1])

(“cat”, [1, 1])

(“fast”, [1, 1])

(“hat”, [1, 1])

(“in”, [1])

.....

MapReduce Example

- A reducer receives a partition of the output (from shuffle and sort) then sums the ones for each key and emit the word as the key and the count as the value.

(".", 2)

("cat", 2)

("fast", 2)

("hat", 2)

("in", 1)

.....

MapReduce Example

Shared Friendship problem

- The goal is to analyze a social network to find friends someone might know (you might know recommendation)

Input : Key/Value where Key is the name of a user
and the value is the list of friends of the user

Output : A list of common friends of two users

MapReduce Example

```
def map ( person, friends );  
    for friend in friends.split( “,” );  
        pair = sort( [ person, friend] )  
        emit ( pair, friends )  
  
def reduce ( pair, friends );  
    shared = set( friends[0] )  
    shared = shared.intersection(friend[1])  
    emit ( pair, shared )
```

MapReduce Example

- Input to Mapper 1
Allen -> (Betty, Chris, David)
- Mapper 1 Output
(Allen, Betty) -> (Betty, Chris, David)
(Allen, Chris) -> (Betty, Chris, David)
(Allen, David) -> (Betty, Chris, David)

MapReduce Example

Input to Mapper 2

Betty -> (Allen, Chris, David , Ellen)

Mapper 2 Output

(Allen, Betty) -> (Allen, Chris, David , Ellen)

(Betty, Chris) -> (Allen, Chris, David , Ellen)

(Betty, David) -> (Allen, Chris, David , Ellen)

(Betty, Ellen) -> (Allen, Chris, David , Ellen)

MapReduce Example

Input to Mapper 3

Chris -> (Allen, Betty, David , Ellen)

Mapper 3 Output

(Allen, Chris) -> (Allen, Betty, David , Ellen)

(Betty, Chris) -> (Allen, Betty, David , Ellen)

(Chris, David) -> (Allen, Betty, David , Ellen)

(Chris, Ellen) -> (Allen, Betty, David , Ellen)

MapReduce Example

Input to Mapper 4

David -> (Allen, Betty, Chris , Ellen)

Mapper 4 Output

(Allen, David) -> (Allen, Betty, Chris, Ellen)

(Betty, David) -> (Allen, Betty, Chris, Ellen)

(Chris, David) -> (Allen, Betty, Chris, Ellen)

(David, Ellen) -> (Allen, Betty, Chris, Ellen)

MapReduce Example

Input to Mapper 5

Ellen -> (Betty, Chris , David)

Mapper 5 Output

(Betty, Ellen) -> (Betty, Chris, David)

(Chris, Ellen) -> (Betty, Chris, David)

(David, Ellen) -> (Betty, Chris, David)

MapReduce Example

- After Shuffle/Sort, Reducer input is
(Allen, Betty) -> ((Allen,Chris,David,Ellen),
(Betty,Chris,David))
(Allen,Chris) -> ((Allen,Betty,David,Ellen)
(Betty,Chris,David))
(Allen,David) -> ((Allen,Betty,Chris,Ellen),
(Betty,Chris,David))

■ ■ ■ ■ ■ ■ ■ ■ ■ ■

MapReduce Example

- After Reduction

(Allen, Betty) -> (Chris, David)

(Allen, Chris) -> (Betty, David)

(Allen, David) -> (Betty, Chris)

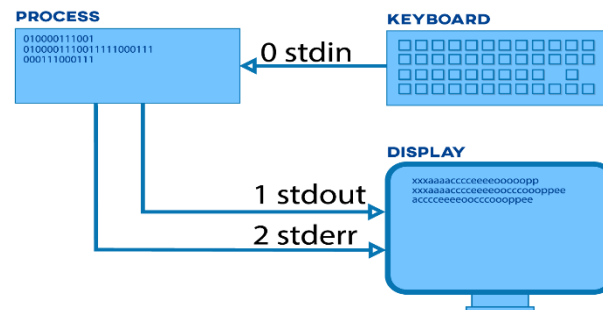
.....

MapReduce Jobs

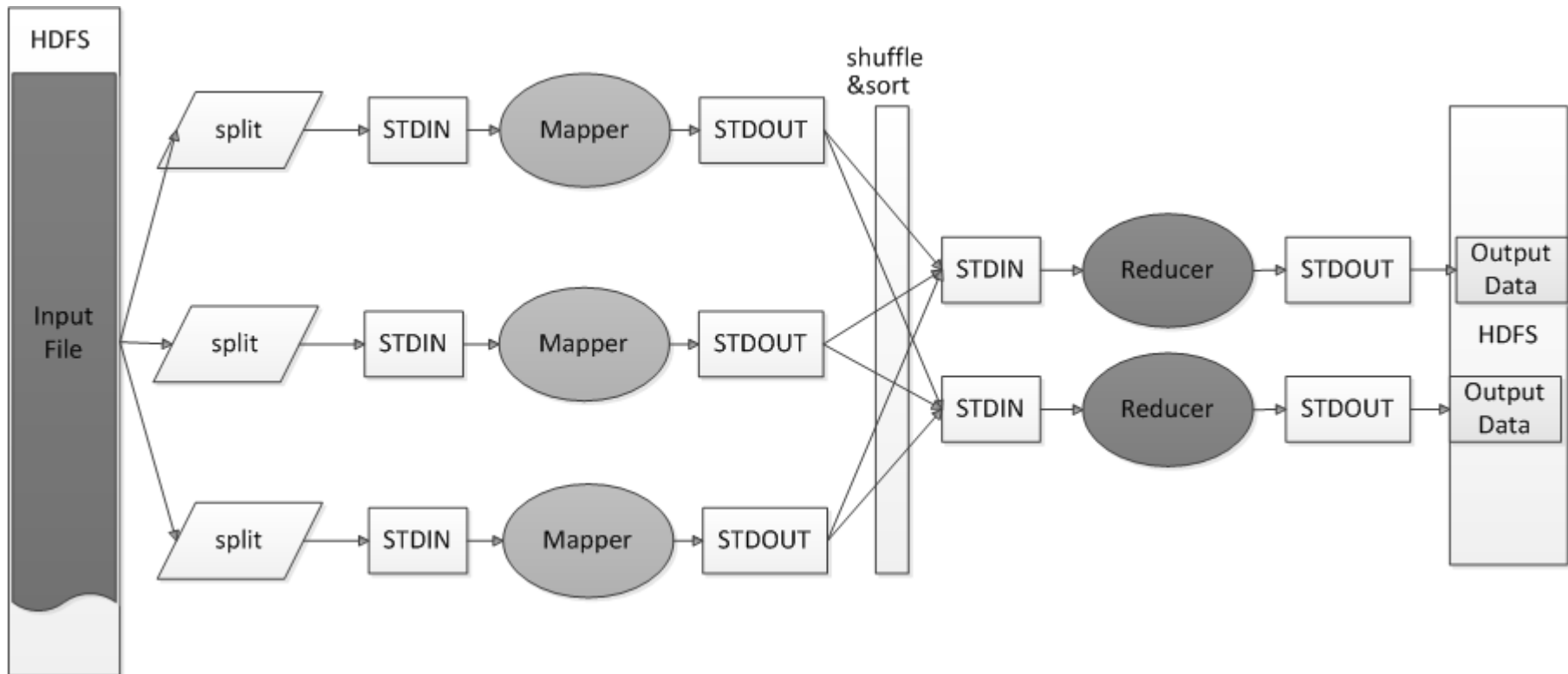
- MapReduce jobs can be written in Java (or other languages) so they must be compiled into Java Archive (JAR) files.
- When submitting the jobs to Hadoop, it will transmit the JAR files to each node that will run a task (a Mapper or a Reducer)

Hadoop Streaming

- With Hadoop streaming, Shell utilities, R, Python, scripts can all be used to compose MapReduce jobs.
- Hadoop Streaming utilizes the standard Unix Streams for input and Output (stdin and stdout).
- Input to both Mappers and Reducers (written in R, Python or others) is read from stdin file.
- Hadoop expects the Mappers and Reducers to write their output key/value pairs (separated by \t) to stdout file



Hadoop Streaming



Source :

<http://www.zhaizhouwei.cn/hadoop/290.html>

Hadoop Streaming

- When streaming executes a job, each mapper task will launch the supplied executable inside of its own process.
- The mapper task then converts the input data into lines of text and pipes it to the stdin of the launched process while simultaneously collecting output from stdout of the launched process.
- The reducer task is launched as its own executable after the output from the mappers is shuffled and sorted.

Hadoop Streaming

- The key/value strings are streamed to the reducer as input via stdin. The reducers perform reduction operation and emit the output via stdout.

Hadoop Streaming

Example : Word count problem

Mapper.py

```
#!/usr/bin/env python
```

```
Import sys
```

```
If __name__ == "__main__":  
    for line in sys.stdin;  
        for word in line.split();  
            sys.stdout.write("{}\t1\n", format(word))
```

Hadoop Streaming

Reducer.py

```
#!/usr/bin/env python
```

```
Import sys
```

```
If __name__ == "__main__":
```

```
    curkey = None
```

```
    total = 0
```

```
    for line in sys.stdin;
```

```
        key,val = line.split("\t")
```

```
        val = int(val)
```

Hadoop Streaming

```
if key == curkey;
    total +=val
else:
    if curkey is not none
        sys.stdout.write("{}\t{}\n", format(curkey,total))
    curkey = key
    total = val
sys.stdout.write("{}\t{}\n", format(curkey,total))
```


Combiners

- Mappers can produce a lot of intermediate data that must be sent over the network to be shuffled, sorted and then reduced.
- This can lead to communication delays and memory bottlenecks (too much data for the reducer to hold into memory).
- Combiners reduce network traffic by performing mapper-local reduction of the data before forwarding it to be shuffled/sorted.

Combiners

Example :

Mapper 1 Output :

(IAD, 14.4), (SFO, 3.9), (JFK, 3.9), (IAD, 12.2),
(JFK, 5.8)

Mapper 2 Output :

(SFO, 4.7), (IAD, 2.3), (SFO, 4.4), (IAD, 1.2)

Intended Sum Reduce Output :

(IAD, 29.1), (JFK, 9.7), (SFO, 13.0)

Combiners

- Each Mapper is emitting extra work for the reducers and combiners can precompute the sums for each key so the number of key/value pairs generated can be reduced and therefore the amount of network traffic.
- With combiners
 - Mapper 1 Output to be reduced
(IAD, 26.6), (SFO, 3.9), (JFK, 9.7)
 - Mapper 2 Output to be reduced
(SFO, 9.1), (IAD, 3.5)

Hadoop Streaming

To submit a job with streaming

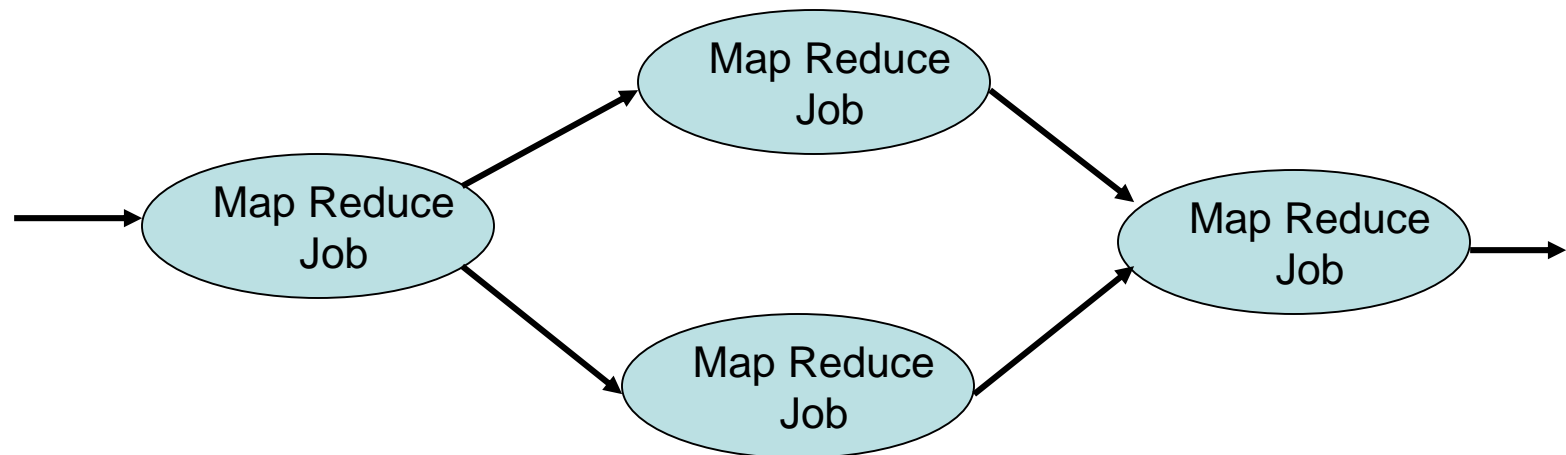
```
$ hadoop jar  
$HADOOP_HOME/share/Hadoop/tools/lib/hadoop_  
streaming-*.jar -input input.data  
-output output.data -mapper mapper.py  
-combiner combiner.py  
-reducer reducer.py  
-file mapper.py -file reducer.py -file combiner.py
```

Partitioners

- Partitioners control how keys and their values get sent to individual reducers by dividing up the keyspace.
- The default partitioner is HashPartitioner. It uniformly distribute keys based on the number of reducers. Each reducer will get a relatively equal number of keys.
- The issue arises when there is key imbalance such that a large number of values are associated with one key and other keys are less likely.
- A custom partitioner can ease the problem by dividing the keyspace according to domain specific structure besides hashing.

Job Chaining

- If a complex algorithm can be decomposed into several smaller MapReduce tasks, then these tasks can be chained together to form a workflow that produces the final output.
- The workflow can be in a simple form of linear chaining jobs or more complex workflow represented by a directed acyclic graph (DAG)



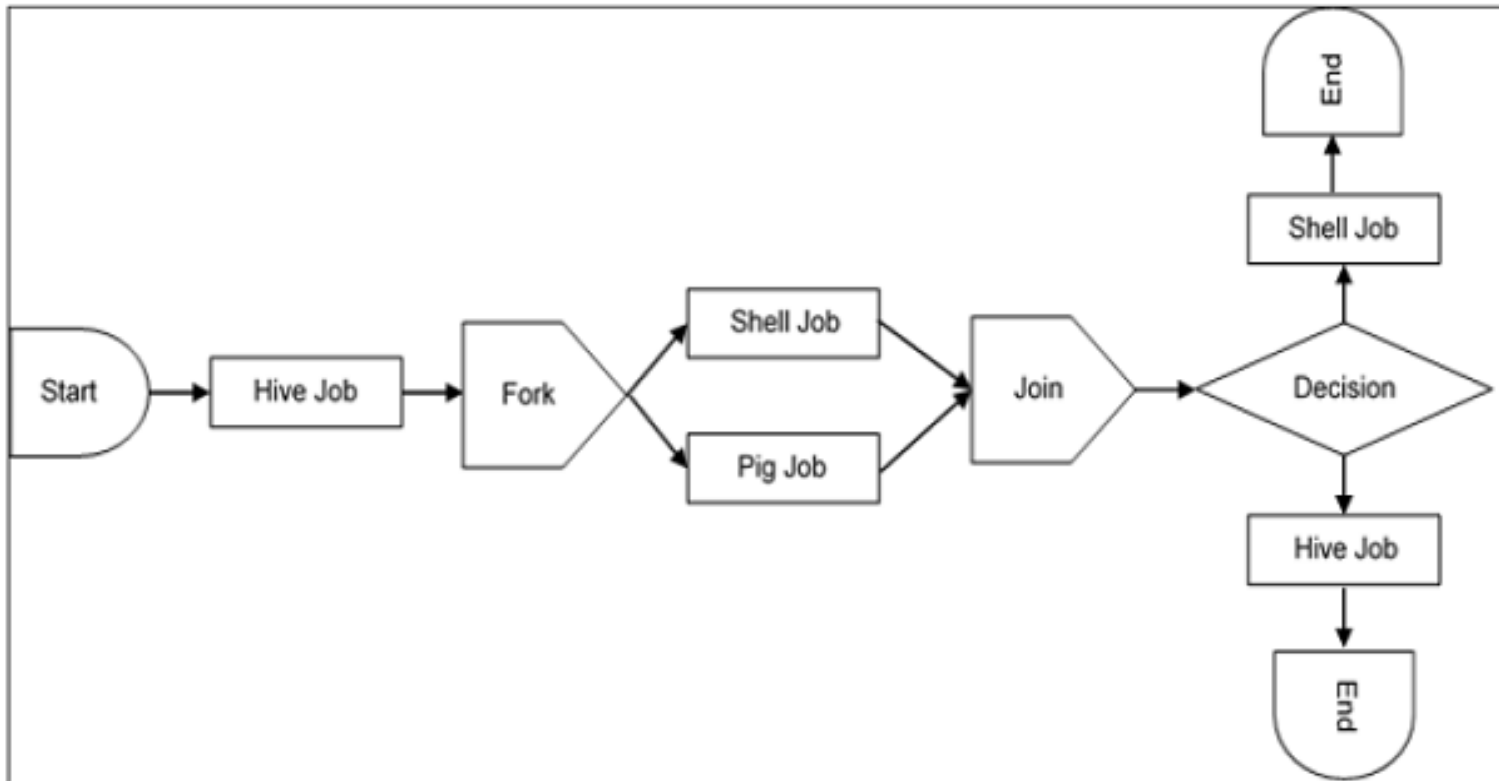
Job Chaining

- Apache Oozie is a scheduler system to run and manage Hadoop jobs in a distributed environment.
- It allows to combine multiple complex jobs to be run in a sequential order or in parallel using fork-join construct.
- Oozie supports various Hadoop jobs like Hive, Pig, Sqoop as well as system-specific jobs like Java and Shell Script.

Job Chaining

- Nodes in a workflow can be control flow nodes (e.g. start, end, kill, fork , join, decision) or action nodes (e.g. Map-Reduce jobs, Pig jobs, Shell commands)
- Oozie provides a GUI editor to create a workflow which can be output in a form of xml file.
- The file can then submitted to Oozie engine for running.

Job Chaining



Oozie Workflow Example