



Data Science

Ubai Sandouk, PhD

2021

Note!



- *(ADS) Algorithms for Data Science*, by B. C. H. Steele, John Chandler, and Swarna Reddy. (Springer, 2016) [Link](#)
- *(IDS1) Introduction to Data Science (in R)*, by R. A. Irizarry. (CRC Press, 2019) [Link](#)
- *(IDS2) Introduction to Data Science (A Python Approach)*, by L. Igual, and S. Seguí. (Springer, 2017) [Link](#)
- *(4P) The Fourth Paradigm*, by T. Hey (Microsoft Research, 2009) [Link](#)
- *(KDD) The KDD process for extracting useful knowledge from volumes of data*, by U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. (Communications of the ACM, 1996) [Link](#)
- *(DWT) The data warehouse toolkit: the complete guide to dimensional modelling (3rd Ed.)*, by R. Kimball, and M. Ross. (John Wiley & Sons, 2013). [Link](#)
- *(BDATA) Big Data: Concepts, Technology, and Architecture*, by B. Balusamy, N. Abirami, S. Kadry, and A. Gandomi (John Wiley & Sons, 2021). [Link](#)



Agenda

- Big DATA (Challenges and Opportunity)
- SQL / NoSQL / NOSQL
- Architecture
 - HDFS
 - MapReduce
- Stack: Hadoop, Tools and Languages
- Big Data Cloud
- References

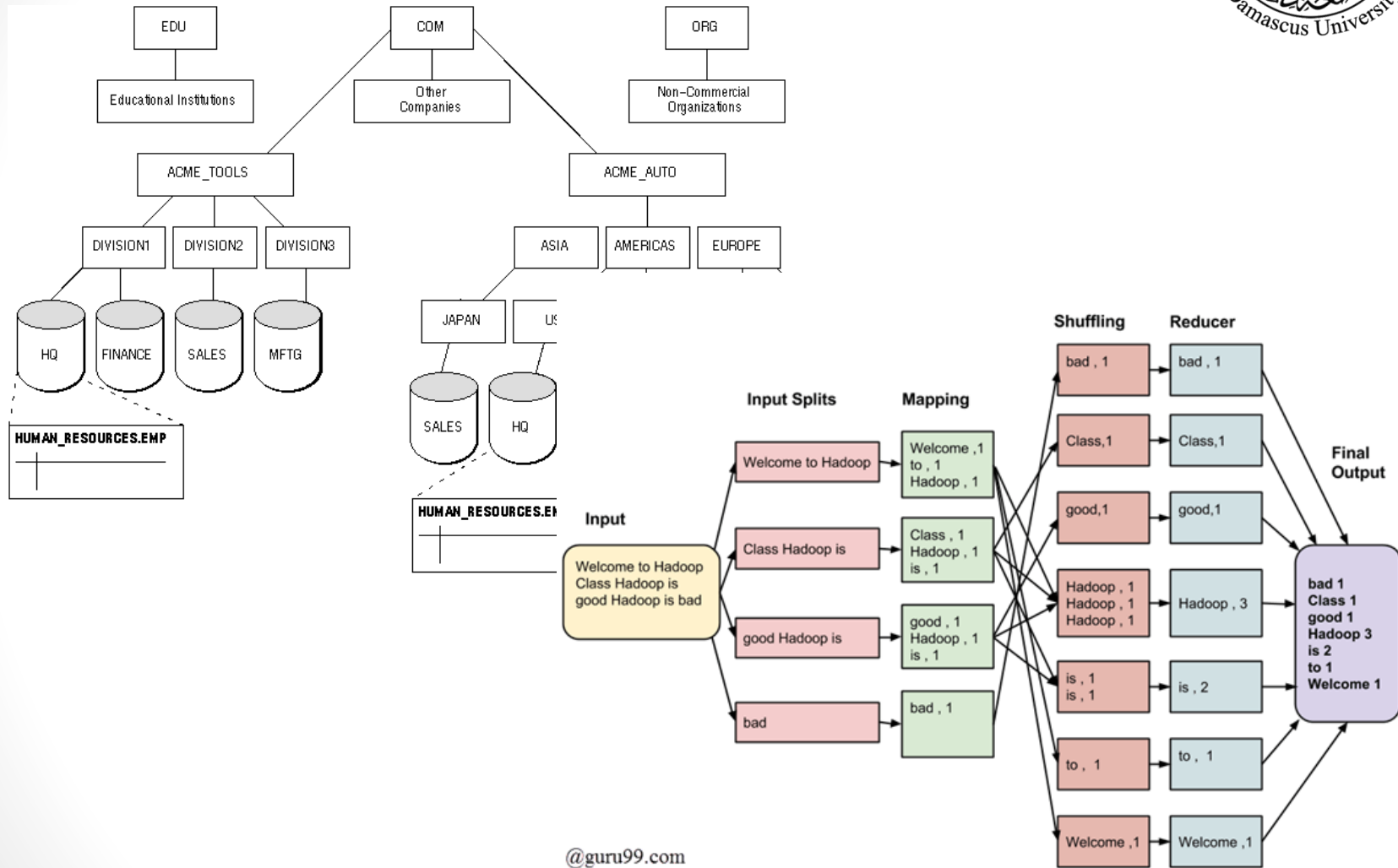
Recap – DataOps



Data Storage + Management + Analysis

Big Data + Data Quality + Data Mining

Big DATA





Big DATA Applications

- Banking and Securities – Credit/debit card fraud, credit risk reporting, customer data analytics.
- Healthcare – Storing patient data, early detection of ailments.
- Marketing – Analyzing customer history, recommendation.
- Web analysis – Social media data, data from search engines.
- Agriculture – Sensors are used in biotech to optimize crop efficiency.
- Smartphones — Facial recognition, retrieve information.

Big DATA Challenges

- Heterogeneity and incompleteness
- Volume and velocity of the data
- Data storage
- Data privacy

Big DATA Challenges

- Amortized Opportunity
 - Use of idle machine
 - Time to upgrade
 - Tool stack
 - Privacy skills

Big DATA – 3V's

- **Volume**
 - size in this case is measured in **Bytes**.
- **Velocity**
 - rapidly increasing speed at which new data is being created (**per second**), and the corresponding need for that data to be digested.
- **Variety**
 - diversity of **data types**
- **Variability**
- **Value**

SQL



- Traditional RDBMSs are **limited**:
 - Normalisation needed, many records represent the same thing.
 - One type of semantic: the relation.
 - Changes are difficult.
- Requires specialized staff
- Single point of failure
- Costly to change
- *Think of Data Warehouses!*



<https://www.stambia.com/en/solutions/by-technology/integration-to-database/relational-database-rdbms>

SQL

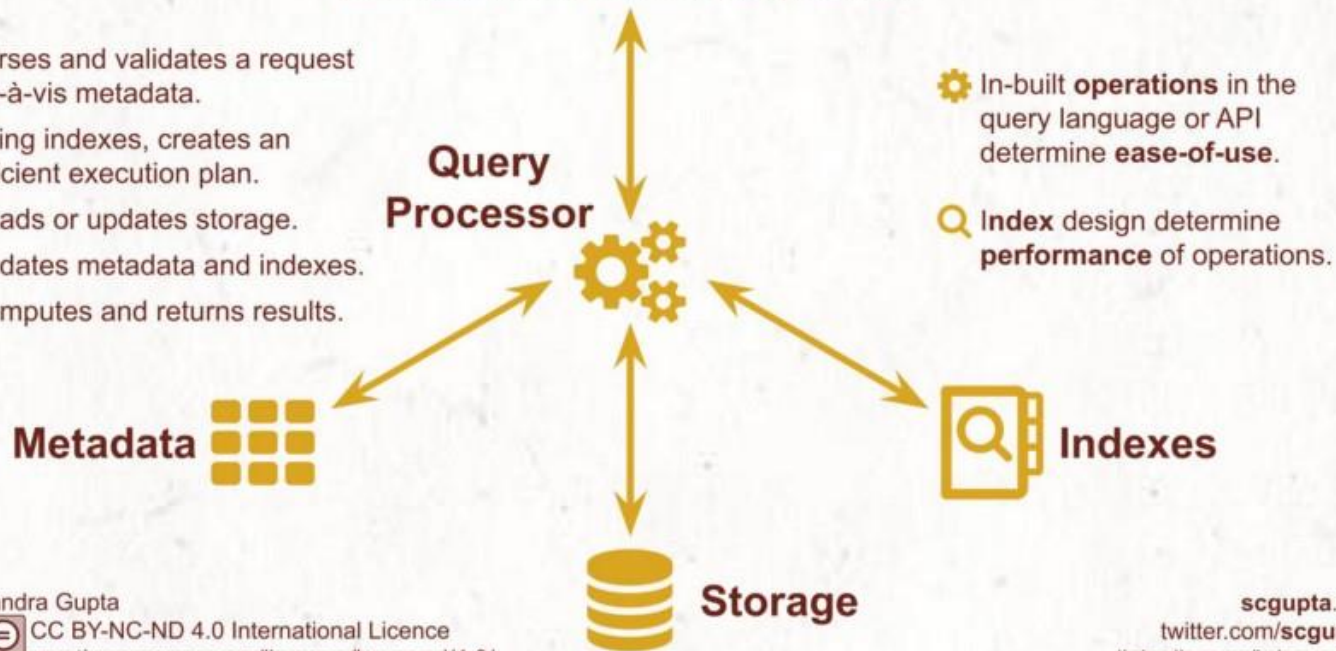


Database Components

scgupta.link/datastores

> __ Interface Language / API (definition, manipulation, query, control)

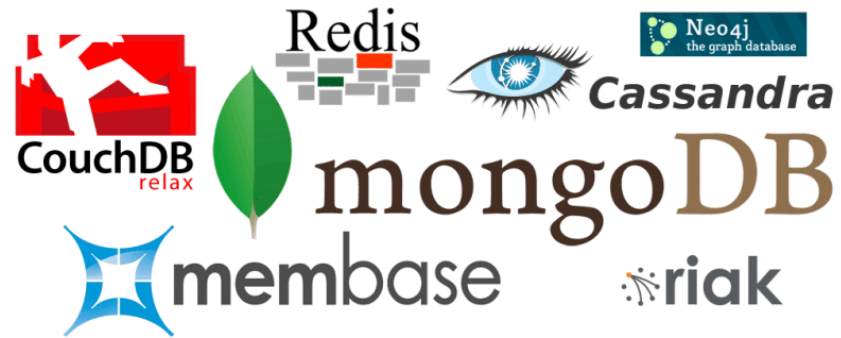
- ① Parses and validates a request vis-à-vis metadata.
- ② Using indexes, creates an efficient execution plan.
- ③ Reads or updates storage.
- ④ Updates metadata and indexes.
- ⑤ Computes and returns results.



NoSQL



- NonSQL (NOSQL) or Not only SQL (NoSQL):
 - De-normalized!
 - Multiple data models
 - Changes are possible



- Extremely easy to use
- Focus on
 - Scalability, Resilience, and Availability

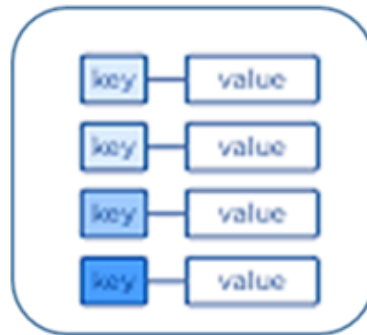
<https://www.ivanomalavolta.com/tag/nosql/>

SQL vs. NoSQL

- SQL
 - Atomic, Consistent, Isolated, Durable (ACID)
- NoSQL
 - Basically Available, Soft-state, Eventually Consistent (BASE)



Document
Store



Key-Value
Store

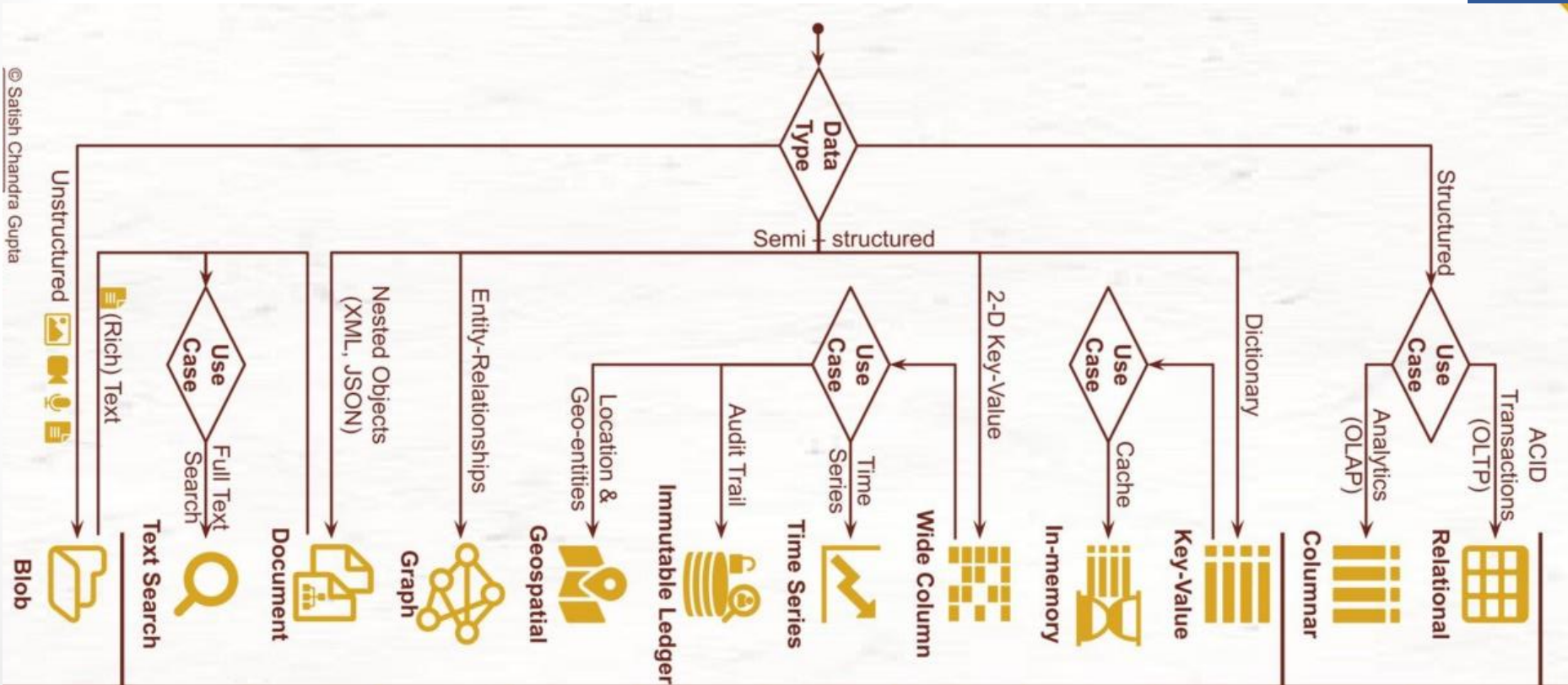


Wide-Column
Store



Graph
Store


SQL vs. NoSQL




SQL vs. NoSQL



SQL vs. NoSQL: Comparison

scgupta.link/datastores 

SQL	NoSQL
 Relational	Non-relational
Structured tables	Semi-structured
Strict schema	Dynamic schema
ACID	Mostly BASE, few ACID
Strong	Eventual to Strong
Consistency prioritized	Basic Availability
Vertically by upgrading hardware	Horizontally by data partitioning

Key-Value



Dictionary or Hash Table

Wide Column



2-D Versioned Key-Value

Document



Nested Objects (XML, JSON, YAML)

Graph






Entity-Relationships

© Satish Chandra Gupta



CC BY-NC-ND 4.0 International Licence
creativecommons.org/licenses/by-nc-nd/4.0/

scgupta.me 
twitter.com/scgupta 
linkedin.com/in/scgupta 

SQL



RDBMS	NoSQL
Structured data with a rigid schema.	Structured, Unstructured, Semi-Structured data with a flexible schema.
Extract, Transform, Load (ETL) required.	ETL is not required.
Storage in rows and columns.	Data are stored in Key/Value pairs database, Columnar database, Document database, Graph Database.
RDBMS is based on ACID transactions. ACID stands for Atomic, Consistent, Isolated and Durable.	NoSQL is based on BASE transactions. BASE stands for Basically available, Soft state, Eventual consistency.
RDBMS Scale up when the data load increases, i.e., expensive servers are brought to handle the additional load.	NoSQL is highly scalable at low cost. NoSQL scales out to meet the extra load. i.e., low-cost commodity servers are distributed across the cluster.
SQL server, Oracle, and MySQL are some of the examples.	MongoDB, HBase, Cassandra are some of the examples.
Structured Query Language is used to query the data stored in the data warehouse.	Hive Query Language (HQL) is used to query the data stored in HDFS.
Matured and stable. Matured indicates that it is in existence for a number of years.	Flexible, Incubation. Incubation indicates that it is in existence from the recent past.

NoSQL – Tools

Next, in this article let us understand what is MongoDB?

What is MongoDB?

MongoDB is a non-relational database which stores the data in documents. This type of database stores the related information together for quick query processing.



mongoDB

The features of MongoDB are as follows:

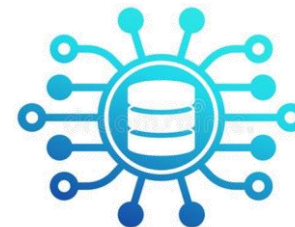
- **Indexing:** It indexes are created in order to improve the search performance.
- **Replication:** MongoDB distributes the data across different machines.
- **Ad-hoc Queries:** It supports ad-hoc queries by indexing the BSON documents & using a unique query language.
- **Schemaless:** It is very flexible because of its schema-less database that is written in C++.
- **Sharding:** MongoDB uses sharding to enable deployments with very large data sets and high throughput operations.

Alright, So, now that you know what is MySQL & MongoDB, let us now see, how these databases stand against each other.

Big DATA Architecture



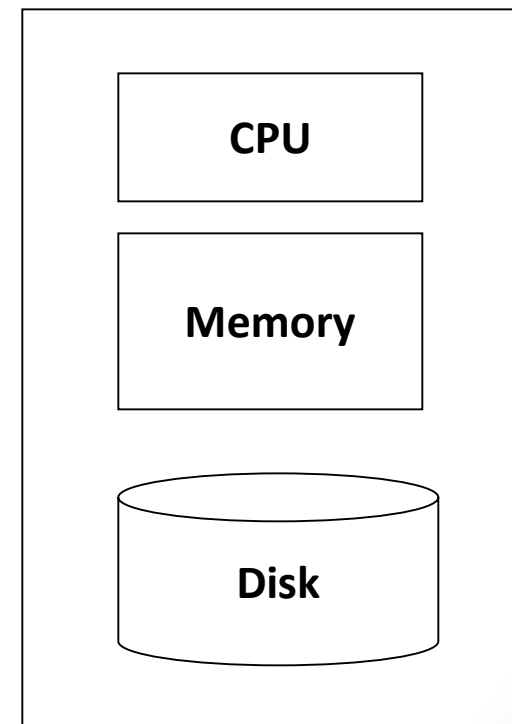
- SQL? NOSQL?
 - Still structured. Not raw data!
 - Operations over some OS and Network!
- Data...
 - Base
 - Warehouse
 - Lake
 - Mart
 - Mesh
 - ...



Architecture



- “Classical” Data model
- Large scale Machine Learning, Data Mining, Business Intelligence!!
 - How to distribute computation?
 - *Distributed programming* is hard!
- Solution: **MapReduce**
 - Google’s computational / data manipulation model



Architecture

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
 - ~4 months to read the web
 - ~1,000 hard drives to store the web
- Even more to do something useful with the data!
- A standard architecture for such problems is emerging:
 - Cluster of commodity Linux nodes
 - Commodity network (ethernet) to connect them

Single Machine



1 Machine

Distributed File System



100 Machines

Architecture



1- How can we make it easy to write distributed programs?

- One server may stay up 3 years (1,000 days)
- If you have 1,000 servers, expect to loose 1/day
- People estimated Google had ~1M machines in 2011
 - 1,000 machines fail every day!
- **Concept:** Store files multiple times for reliability

2- How do you distribute computation?

- **Concept:** Bring computation close to the data

Architecture – HDFS

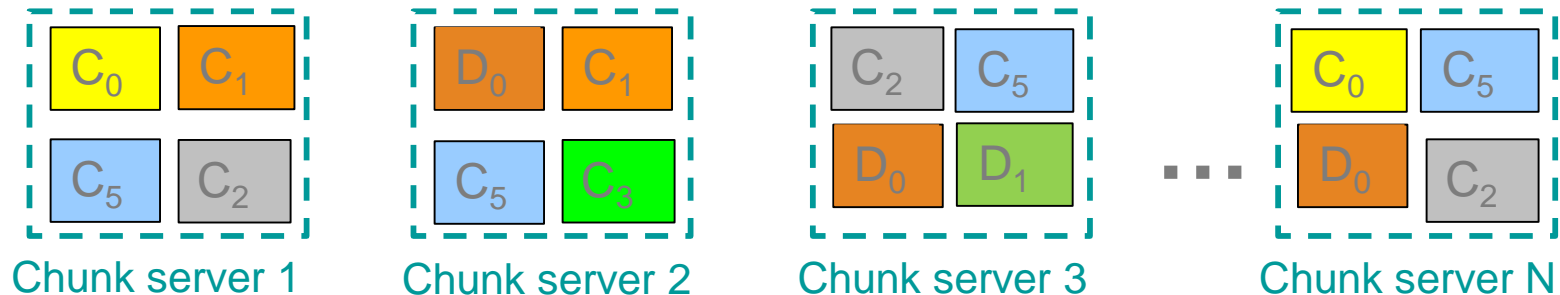
- Store files multiple times for reliability!
 - Distributed File System:
 - Provides global file namespace
 - Google GFS; Hadoop HDFS;
- Typical usage pattern
 - Huge files (100s of GB to TB)
 - Data is rarely updated in place
 - Reads and appends are common

Architecture – HDFS

- Chunk servers
 - File is split into contiguous chunks
 - Typically each chunk is 16-64MB
 - Each chunk replicated (usually 2x or 3x)
 - Try to keep replicas in different racks
- Master node
 - a.k.a. Name Node in Hadoop's HDFS
 - Stores metadata about where files are stored
 - Might be replicated
- Client library for file access
 - Talks to master to find chunk servers
 - Connects directly to chunk servers to access data

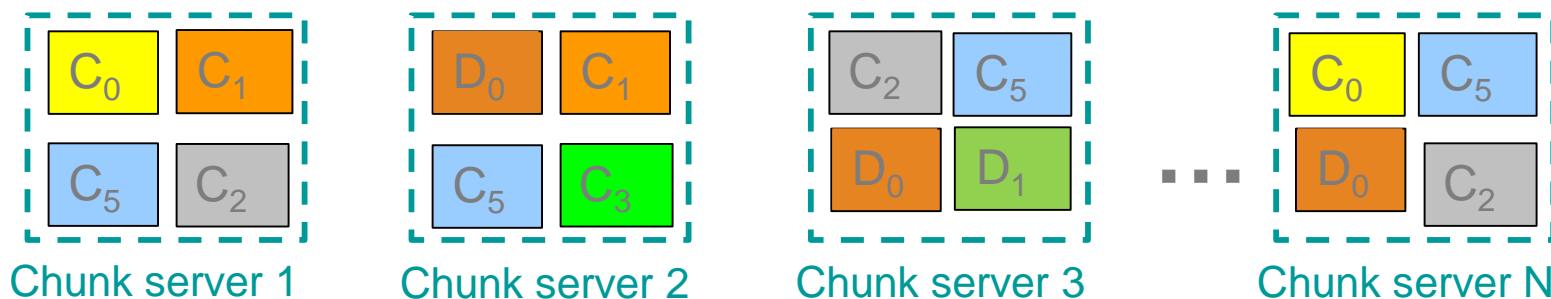
Architecture – HDFS

- Reliable distributed file system
 - Data is kept in “chunks” spread across machines
 - Each chunk replicated on different machines
 - Seamless recovery from disk or machine failure



Architecture – MapReduce

- Reliable distributed file system
 - Data is kept in “chunks” spread across machines
 - Each chunk replicated on different machines
 - Seamless recovery from disk or machine failure



Bring computation directly to the data!

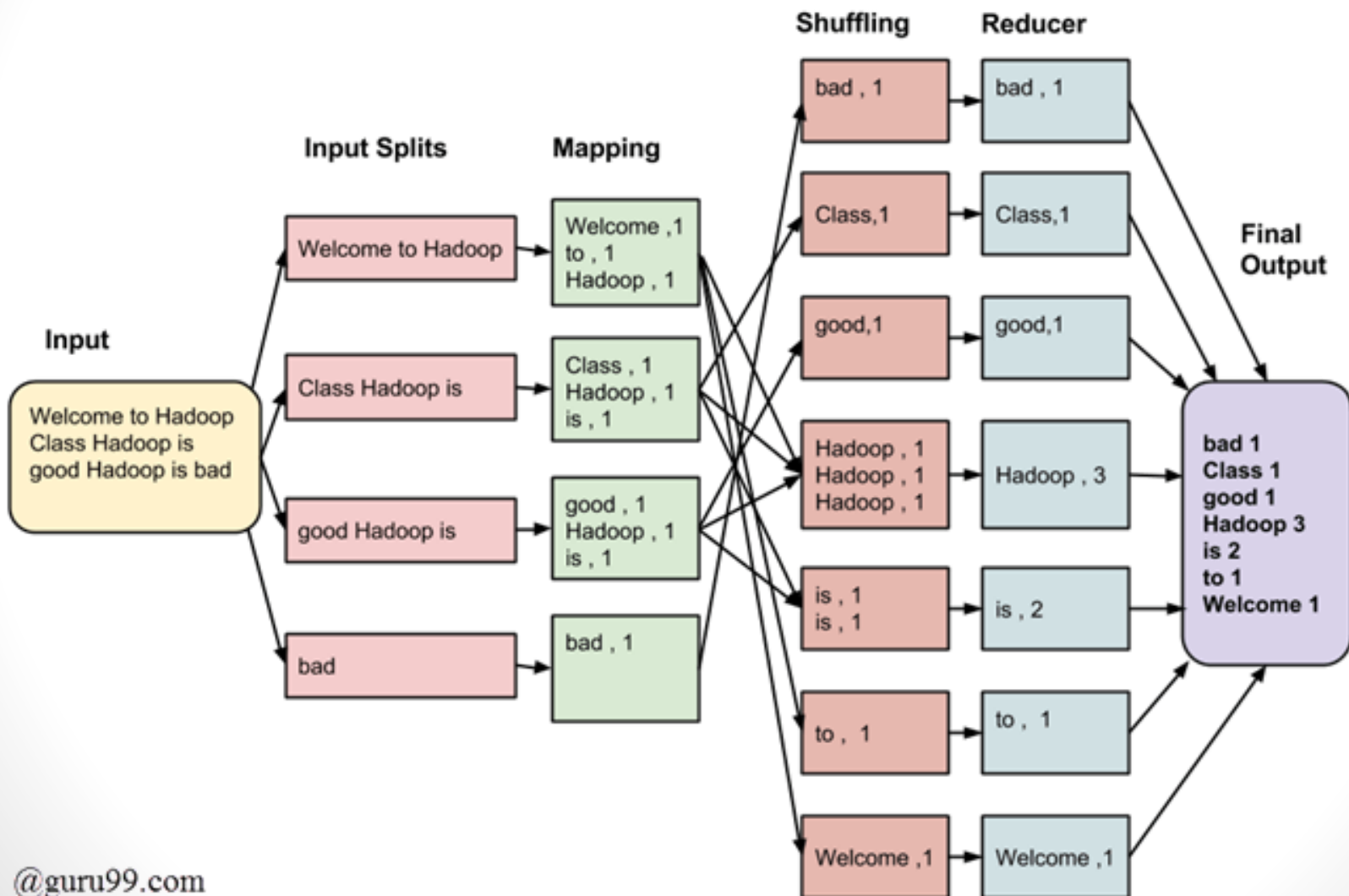
Chunk servers also serve as compute servers

Architecture – MapReduce



- Count the number of times each distinct word appears in a file
- Case 1:
 - File too large for memory, but all <word, count> pairs fit in memory
- Case 2:
 - Count occurrences of words:
 - `words(doc.txt) | sort | uniq -c`
 - where `words` takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of MapReduce
 - Great thing is that it is naturally parallelizable

Architecture – MapReduce



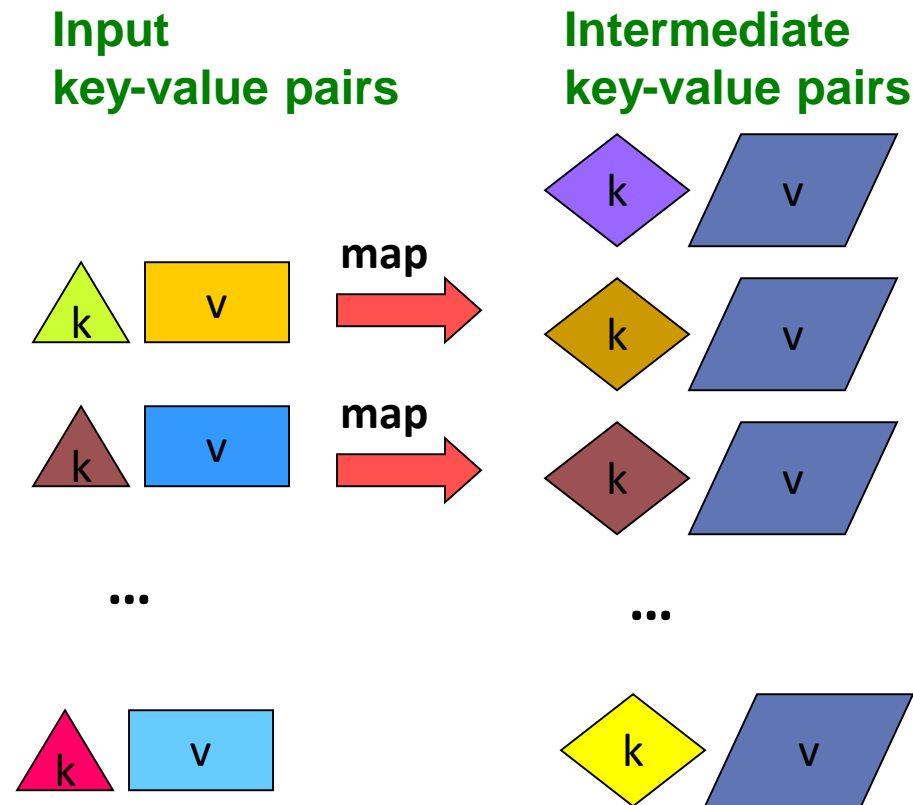


Architecture – MapReduce

- Map
 - Extract something you care about
- Shuffle:
 - Sort and group by key
- Reduce
 - Aggregate, summarize, filter or transform
- Write the result

Outline stays the same, **Map** and **Reduce**
change to fit the problem

MapReduce – Map



Map(k, v) \rightarrow $\langle k', v' \rangle^*$

Takes a key-value pair and outputs a set of key-value pairs

E.g., key is the filename, value is a single line in the file

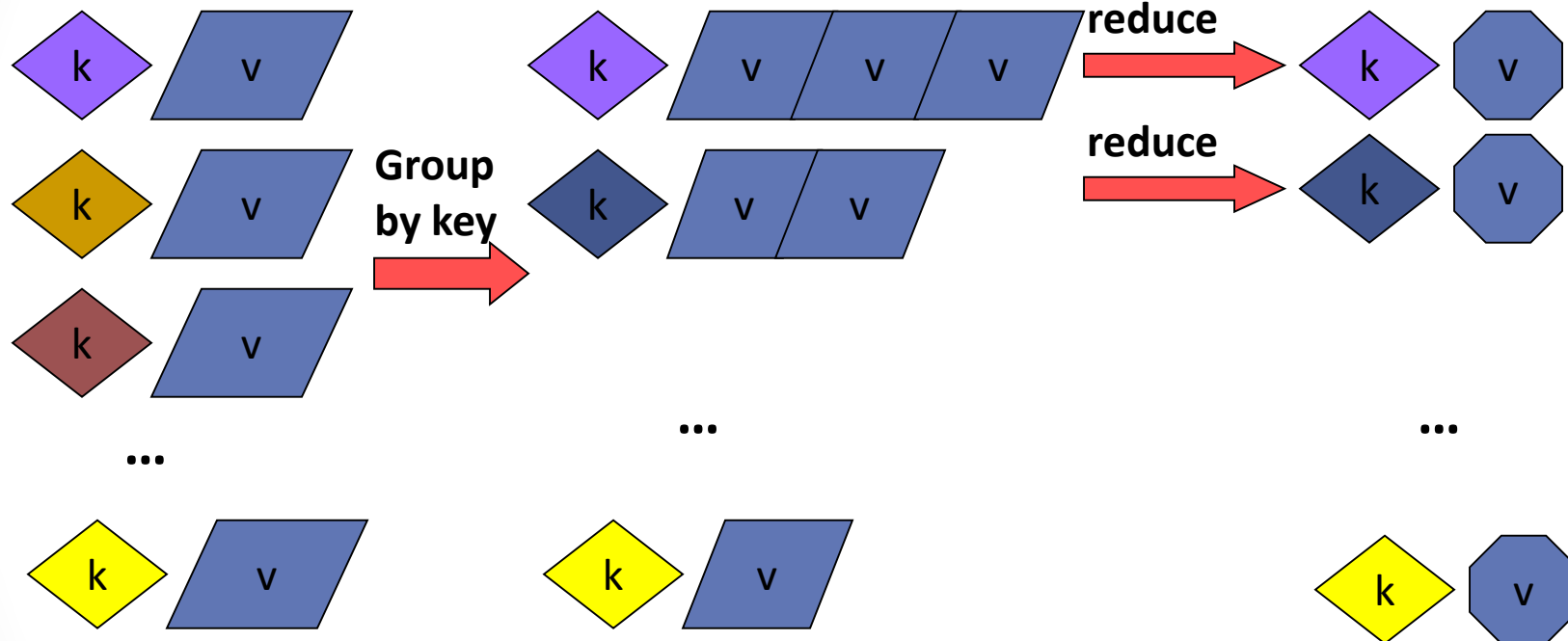
There is one Map call for every (k, v) pair

MapReduce – Reduce

Intermediate
key-value pairs

Key-value groups

Output
key-value pairs



Reduce(k' , $\langle v' \rangle^*$) \rightarrow $\langle k', v'' \rangle^*$

All values v' with same key k' are reduced together
and processed in v' order

There is one Reduce function call per unique key k'

MapReduce

Provided by the programmer

MAP:

Read input and produces a set of key-value pairs

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(key, value)

Group by key:

Collect all pairs with same key

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...

(key, value)

Provided by the programmer

Reduce:

Collect all values belonging to the key and output

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...

(key, value)

Only sequential reads

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long term space based man/machine partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need

Big document



MapReduce

map(key, value) :

```
// key: document name; value: text of the document
for each word w in value:
    emit(w, 1)
```

reduce(key, values) :

```
// key: a word; value: an iterator over counts
result = 0
for each count v in values:
    result += v
emit(key, result)
```

MapReduce

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

MapReduce



MAP:

Read input and produces a set of key-value pairs

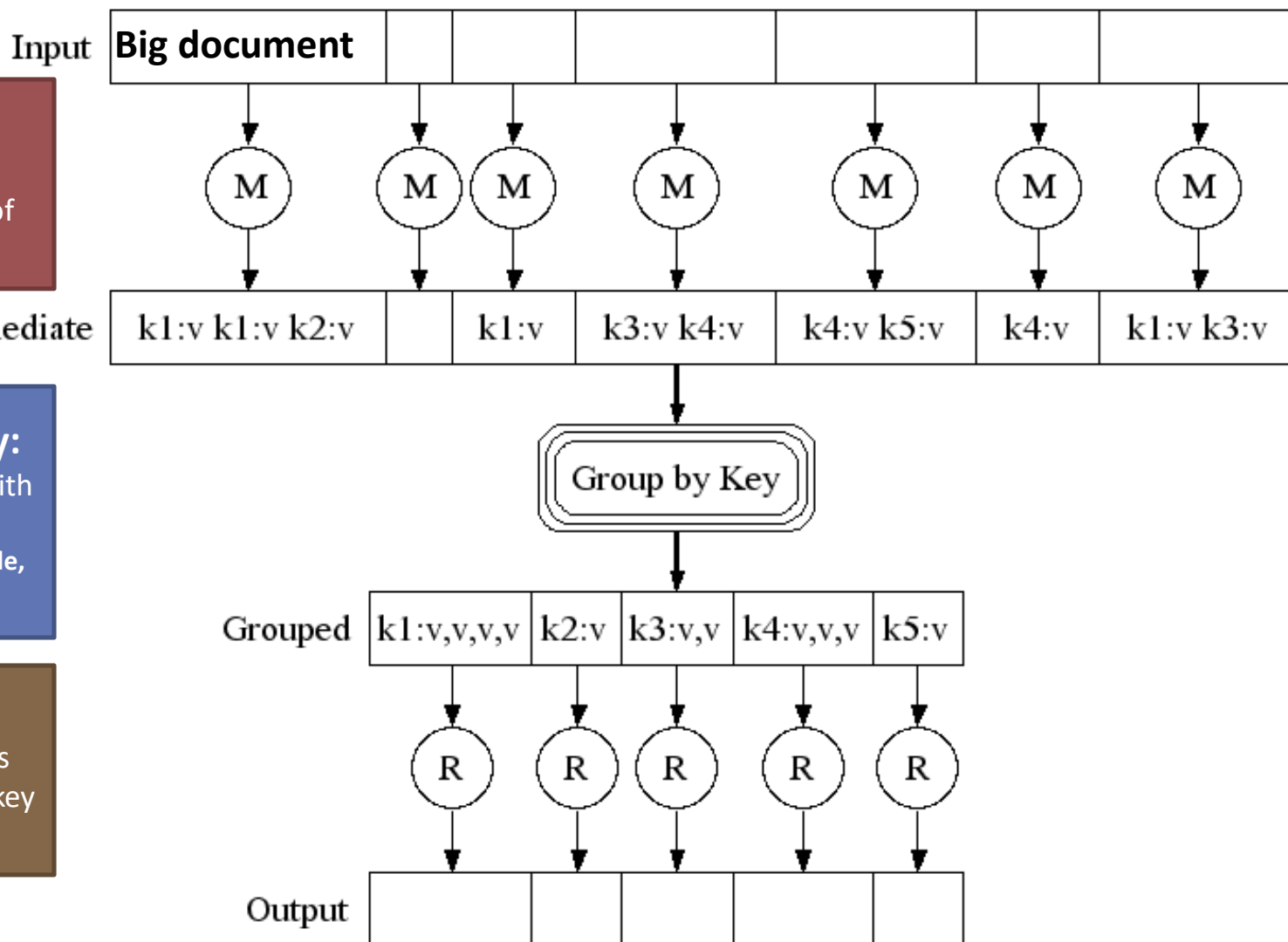
Intermediate

Group by key:

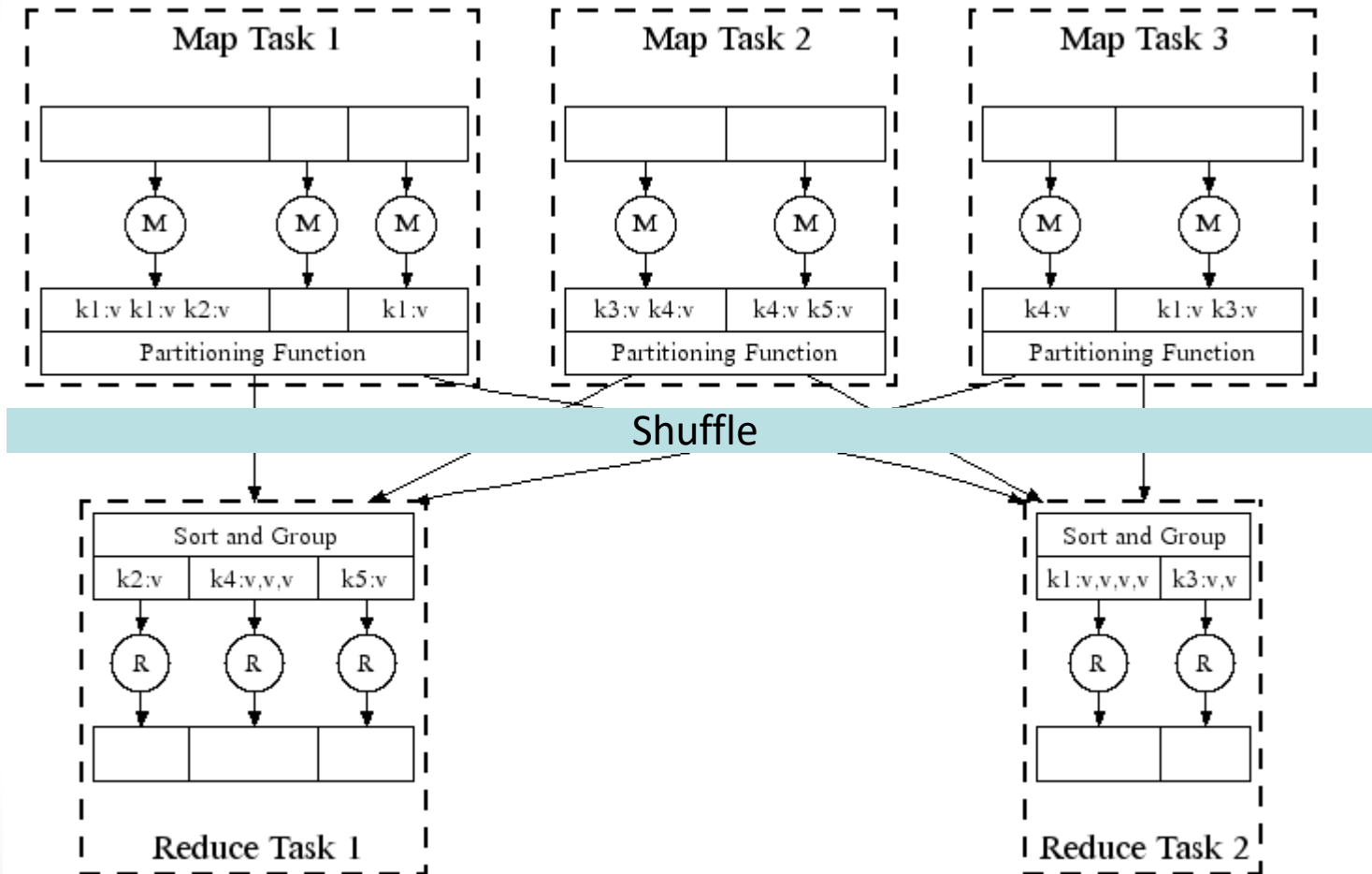
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:

Collect all values belonging to the key and output



MapReduce



All phases are distributed with many tasks doing the work



MapReduce – Data Flow

- Input and final output are stored on a distributed file system:
 - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- **Intermediate results** are stored on **local FS**
- Output is often input to another task

MapReduce – Data Flow

- **Master node takes care of coordination:**
 - **Task status:** (idle, in-progress, completed)
 - **Idle tasks** get scheduled as workers become available
 - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
 - Master pushes this info to reducers
- Master pings workers periodically to detect failures

MapReduce – Failure

- **Map worker failure**
 - Map tasks completed or in-progress at worker are reset to idle
 - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
 - Only in-progress tasks are reset to idle
- **Master failure**
 - MapReduce task is aborted and client is notified



MapReduce – Resources

- M map tasks, R reduce tasks
- Rule of a thumb:
 - Make M much larger than the number of nodes in the cluster
 - One DFS chunk per map is common
 - Improves dynamic load balancing and speeds up recovery from worker failures
 - Fine granularity tasks: map tasks \gg machines
- R is (smaller, equal to larger) than M
 - Because output is spread across R files



MapReduce – Partitioning

- Want to control how keys get partitioned
 - Inputs to map tasks are created by contiguous splits of input file
 - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function:
 - $\text{hash}(\text{key}) \bmod R$
- Sometimes useful to override the hash function:
 - $\text{hash}(\text{hostname}(\text{URL})) \bmod R$
 - ensures URLs from a host end up in the same output file

MapReduce – Example

- In a large web corpus:
- Look at the metadata file
 - Lines of the form: (URL, size, date, ...)
- For each host, find the total number of bytes
 - That is, the sum of the page sizes for all URLs from that particular host
- Other examples:
 - Link analysis and graph processing
 - Machine Learning

MapReduce – Example

- In machine translation:
- Count number of times every 5-word sequence occurs in a large corpus of documents
- Very easy with MapReduce:
 - Map:
 - Extract (5-word sequence, count) from document
 - Reduce:
 - Combine the counts

MapReduce – Example (Join)

- Compute the natural join $R(A,B) \bowtie S(B,C)$
- R and S are each stored in files
- Tuples are pairs (a,b) or (b,c)

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R



B	C
b_2	c_1
b_2	c_2
b_3	c_3

S



A	B	C
a_3	b_2	c_1
a_3	b_2	c_2
a_4	b_3	c_3



MapReduce – Example (Join)

- A Map process turns:
 - Each input tuple $R(a,b)$ into key-value pair $(b,(a,R))$
 - Each input tuple $S(b,c)$ into $(b,(c,S))$
- Map processes send each key-value pair with key b to Reduce process $h(b)$
 - Use a hash function h from B -values to $1...k$
 - Hadoop does this automatically; just tell it what k is.
- Each Reduce process matches all the pairs $(b,(a,R))$ with all $(b,(c,S))$ and outputs (a,b,c) .



MapReduce – Costs

- Cost is quantified using:
 - **Communication cost** = total I/O of all processes
 - **Elapsed communication cost** = max of I/O along any path
 - (Elapsed) **computation cost** analogous, but count only running time of processes
- Generally, Either communication cost or computation cost dominates.
- Note that traditional big-O notation is not the very useful.

MapReduce – Costs

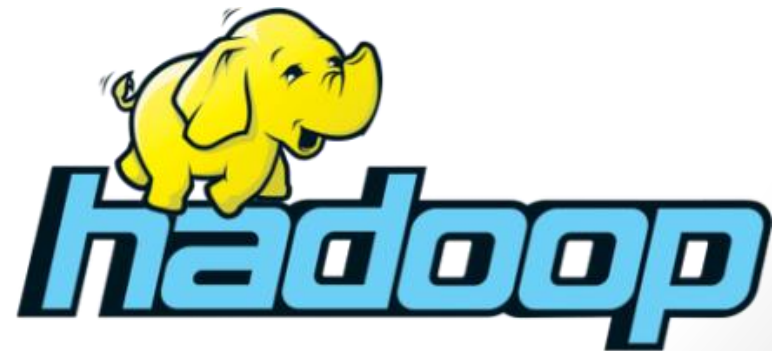
- Communication cost =
 - input file size
 - + 2 x (sum of the sizes of all files passed from Map processes to Reduce processes)
 - + the sum of the output sizes of the Reduce processes.
- Elapsed communication cost =
 - the sum of the largest input
 - + output for any map process
 - + the same for any reduce process

MapReduce – Costs (Join)

- **Total communication cost** = $O(|R| + |S| + |R \bowtie S|)$
- **Elapsed communication cost** = $O(s)$
 - We're going to pick k and the number of Map processes so that the I/O limit s is respected
 - We put a limit s on the amount of input or output that any one process can have. s could be:
 - What fits in main memory
 - What fits on local disk
- **With proper indexes**, computation cost is linear in the input + output size
 - So computation cost is like communication cost

MapReduce – Implementation

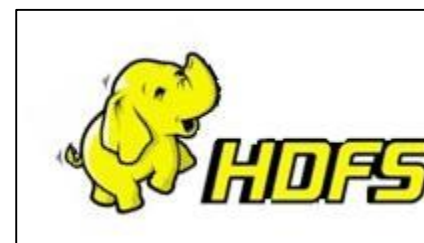
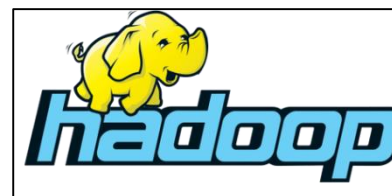
- Google (GFS)
 - Not available outside Google
- Hadoop (HDFS)
 - An open-source implementation in Java



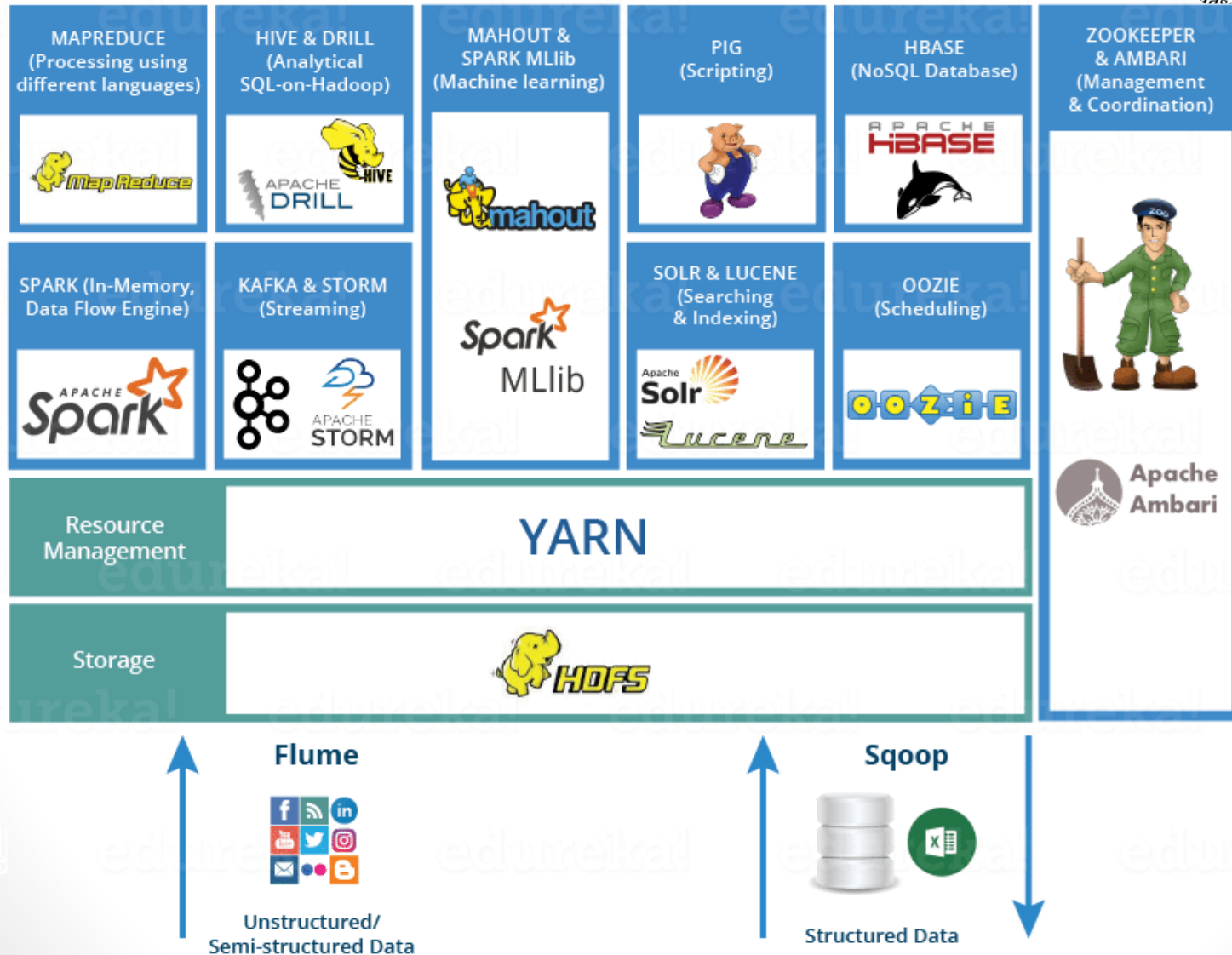
Technology Stack – Hadoop



- Google Cloud Platform
- Amazon AWS AMR
- Microsoft Azure HDInsight
- IBM BigInsights
- Oracle Big Data Appliance (Legacy)



Technology Stack – Hadoop



Technology Stack – Hadoop



- *Hadoop Common* : libraries and utilities needed by other Hadoop modules
- *Hadoop Distributed File System (HDFS)* : distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
- *Hadoop YARN* : platform responsible for managing computing resources in clusters and using them for scheduling users' applications;
- *Hadoop MapReduce* : implementation of the MapReduce programming model for large-scale data processing;
- *Hadoop Ozone* : (introduced in 2020) An object store for Hadoop.

Technology Stack – Ecosystem



- collection of additional software packages that can be installed on top of or alongside Hadoop, such as:
 - Apache Spark: Data flow tool
 - Apache Pig: Scripting tool
 - Apache Sqoop: Data loading tool
 - Apache Phoenix: SQL query tool
 - Apache Hbase: NoSQL query tool
 - Apache Hive: Data warehousing query tool
 - Apache Flume: Logging tool
 - Apache ZooKeeper: Cloud orchestration
 - Apache Storm: distributed streaming tool
 - Apache Oozie: Workflow scheduling tool
 - Apache Airflow: Distributed workflow scheduling tool



Technology Stack – Tools

- *Big Data Tools*: Other packages that sits on top of Hadoop and provide extended functionality to uses, such as:
 - Solr: Indexing and load balancing tool
 - Kafka: event streaming platform
 - HIPI: Hadoop Image processing interface
 - Mahout: Machine learning tool

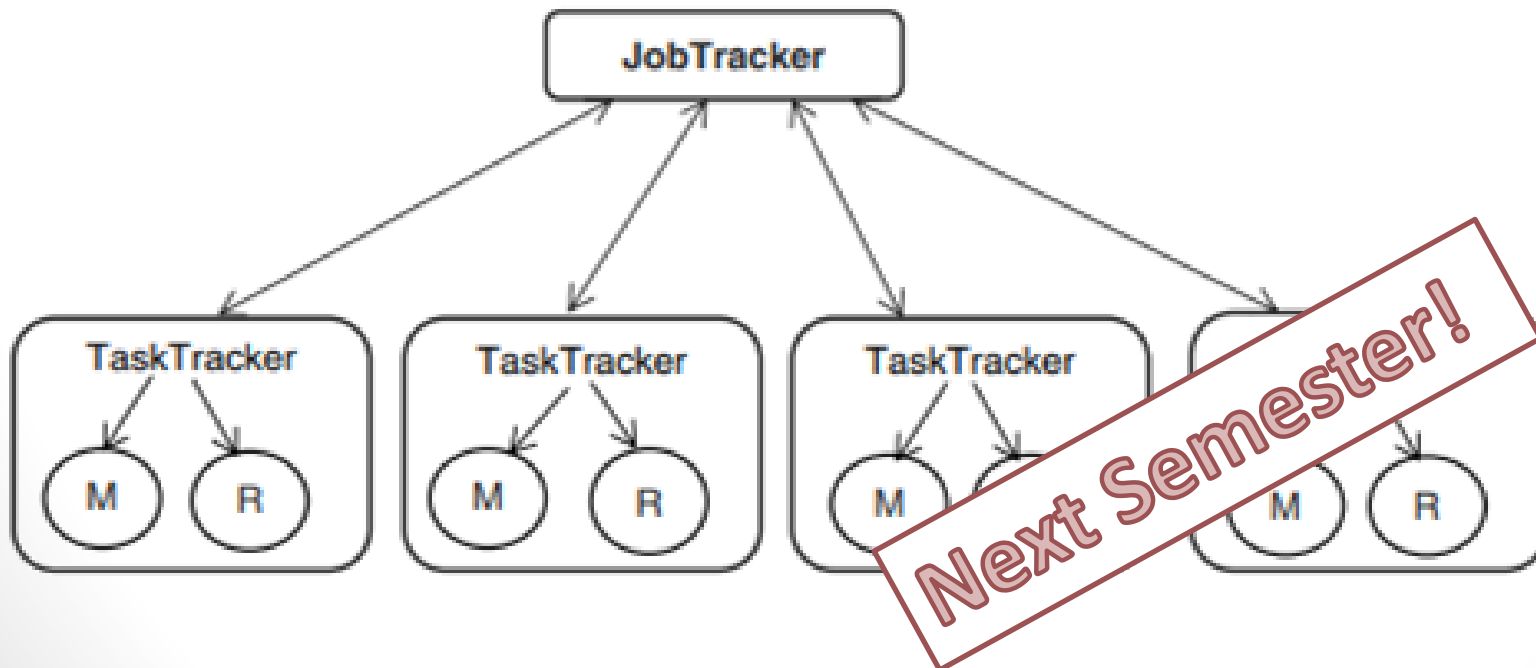
Technology Stack – Languages



- *Programming Languages:*
 - Python
 - Java
 - Scala
 - R

Hadoop – Cloud

- Ability to rent computing by the hour
 - Additional services e.g., persistent storage
- Amazon’s “Elastic Compute Cloud” (EC2)



References



Assessed Reading:

- A. Jain. The 5 V's of big data. 2016.09. <https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/>
- B. Anderson, B. Nicholson. SQL vs. NoSQL Databases: What's the Difference? 2021.06. <https://www.ibm.com/cloud/blog/sql-vs-nosql>
- S. C. Gupta, SQL vs. NoSQL Database: When to Use, How to Choose. 2021.06. <https://towardsdatascience.com/datastore-choices-sql-vs-nosql-database-ebec24d56106>
- Z. Dehghani, How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh. 2019. <https://martinfowler.com/articles/data-monolith-to-mesh.html>
- J. Dean, and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. 2004. <https://research.google.com/archive/mapreduce-osdi04.pdf>
- S. Ghemawat, H. Gobioff, and S. Leung: The Google File System. 2003. <https://research.google.com/archive/gfs-sosp2003.pdf>

Resources

- Hadoop Wiki
 - Introduction
 - <http://wiki.apache.org/lucene-hadoop/>
 - Getting Started
 - <http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop>
 - Map/Reduce Overview
 - <http://wiki.apache.org/lucene-hadoop/HadoopMapReduce>
 - <http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses>
 - Eclipse Environment
 - <http://wiki.apache.org/lucene-hadoop/EclipseEnvironment>
- Releases from Apache download mirrors
 - <http://www.apache.org/dyn/closer.cgi/lucene/hadoop/>
- Nightly builds of source
 - <http://people.apache.org/dist/lucene/hadoop/nightly/>
- Source code from subversion
 - http://lucene.apache.org/hadoop/version_control.html

Up next

- Data Quality Attributes

