WQD7007 Big Data Management

Introduction to the course

Academic Calendar

ACADEMIC CALENDAR
2024/2025 ACADEMIC SESSION
(MASTER'S AND DOCTORATE LEVEL)

SEMESTER I					
Orientation Week			29.09.2024	-	06.10.2024
Lectures	7	weeks*	07.10.2024	-	24.11.2024
Mid Semester I Break	1	week	25.11.2024	-	01.12.2024
Lectures	7	weeks*	02.12.2024	-	19.01.2025
Revision Week	1	week*	20.01.2025	-	26.01.2025
Semester I Final Examination	3	weeks*	27.01.2025	-	16.02.2025
Semester I Break	4	weeks	17.02.2025	-	16.03.2025
	23	weeks			
SEMESTER II					
Lectures	7	weeks*	17.03.2025	-	04.05.2025
Mid Semester II Break	1	week	05.05.2025	-	11.05.2025
Lectures	7	weeks*	12.05.2025	-	29.06.2025
Revision Week	1	week*	30.06.2025	-	06.07.2025
Semester II Final Examination	3	weeks*	07.07.2025	-	27.07.2025
Semester II Break	4	weeks	28.07.2025	-	24.08.2025
	23	weeks			
SPECIAL SEMESTER					
Lectures	7	weeks*	28.07.2025	-	14.09.2025
Special Semester Final Examination	1	week*	15.09.2025	-	21.09.2025
Break	1	week	22.09.2025		28.09.2025
	9	weeks			

Note:

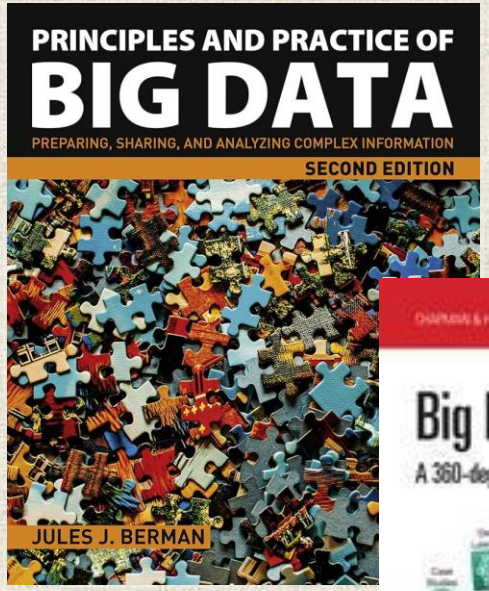
(*) The Academic Calendar has taken into account public and festive holidays and is subject to change:

Deepavali	01 November 2024 (Friday)
Christmas Day	25 December 2024 (Wednesday)
New Year	01 January 2025 (Wednesday)
Chinese New Year	29 & 30 January 2025 (Wednesday & Thursday)
Federal Territory Day	01 February 2025 ((Saturday)
Thaipusam	11 February 2025 (Tuesday)
Nuzul Al-Quran	17 March 2025 (Monday)
Eidul Fitri	31 March & 01 April 2025 (Monday & Tuesday)
Wesak Day	12 May 2025 (Monday)
His Majesty the King's Birthday	02 June 2025 (Monday)
Eidul Adha	06 June 2025 (Friday)
Awal Muharam	27 June 2025 (Friday)

Senate Approval Date: 24.01.2024

WQD7007 Big Data Management

- Main textbook
- Most of the resources is available in the book (case study, etc)
- Relying only on lecture slides provided is NOT enough!



WQD7007 Big Data Management

- Course learning outcome (CLO):
  1. Explain the processes in data pipeline
  2. Discuss database concepts and technologies for big data storage and retrieval
  3. Apply appropriate models, tools, and technologies to implement storage, search and retrieval systems for large-scale structured and unstructured system
  4. Analyze data provenance and data trustworthiness, and its role in sharing and reuse of data

WQD7007 Big Data Management

- Method of assessment:
  - Continuous assessment (60%)
    - Tugasan (Minggu 4-5) / Assignment (Week 4-5) : 15%
    - Ujian pertengahan (Minggu 8) / Mid-term Test (Week 8) : 10%
    - Ujian Makmal (Minggu 11) / Lab test (Week 11) : 15%
    - Projek (Minggu 13-14) / Project (Week 13-14) : 20%
  - Alternative assessments (40%)
    - Penilaian Alternatif 1 (Minggu 14) / Alternative assessment 1 (week 14) = MCQ : 15%
    - Penilaian Alternatif 2 (Minggu 14) / Alternative assessment 2 (week 14) = Case study report : 25%

WQD7007 Big Data Management

- Topics to be covered:
  - Big data pipeline using Hadoop
  - Big data concepts
  - Big data technologies
  - Distributed computing for big data
  - Data provenance and data trustworthiness

WQD7007 Big Data Management

- Things to take note:
  - Online learning platform:
    - https://spectrum.um.edu.my/
    - Microsoft Teams
  - Software to be used:
    - OS: Ubuntu or CentOS, or
    - Virtualbox (if your system have 8 GB RAM or more)
    - Hadoop
  - Assignment 1!

WQD7007 Big Data Management

- Assignment & Project: Topics in Big Data
  1. Education
  2. Business
  3. Marketing
  4. Banking
  5. Agriculture and Crops
  6. Tourism and Hospitality
  7. Medical
  8. Defense and Security
  9. Veterinary
  10. Logistics and Transportation

WQD7007 Big Data Management

Introduction to big data

Agenda

- What is big data?
- Characteristic of big data
- Big data and small data
- Six phases of big data
- When is big data needed?
- Key obstacles of the big data applications
- Application

What is data?

- Collection of information
  - can be measured quantitatively or qualitatively
  - Example:
    - The room temperature now is 25 degree Celsius. ($25^{o}C$) quantitative
    - The room is cold qualitative

What is data?

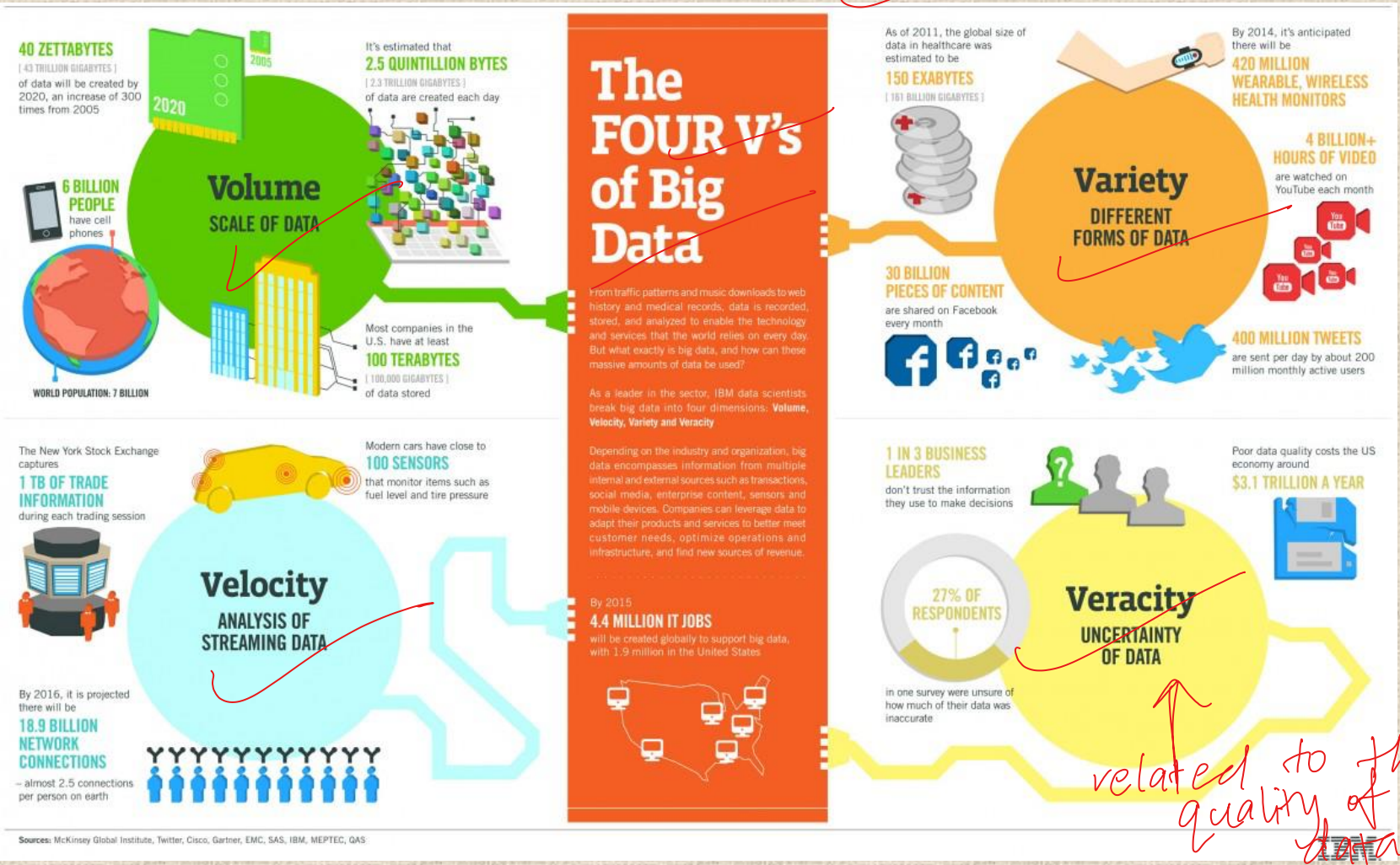
- Collection of Information
  - can be measured quantitatively or qualitatively
  - Example:
    - The room temperature now is 25 degree Celsius. ($25^{o}C$)
    - The room is cold
- How data is used?
- How database is setup?
- Why big data is needed?

Example: Internet users per 100 people

Increasing every year

	A	N	O	P	Q	R	S	T	U	V	W
1	Internet users (per 100 people)	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
125	Latvia	22.0499147	27.0830658	38.6869248	46.1002977	53.7502952	59.3153969	63.5528205	67.0340429	68.8188453	72.4272156
126	Lebanon	7	8	9	10.14	15	18.74	22.53	30.14	43.68	52
127	Lesotho	1.08394314	1.53249659	2.17552434	2.58024548	2.97970819	3.44543126	3.58	3.72	3.86	4.2248
128	Liberia	0.03271331	0.03186893	0.03101118			0.55137658	0.53	0.51	2.3	3
129	Libya	2.24441822	2.81451799	3.53283816	3.91778798	4.30105179	4.72199938	9	10.8	14	16.9981
130	Liechtenstein	59.4707107	58.8096918	64.0074481	63.3713561	64.2141614	65.0802184	70	75	80	85
131	Lithuania	17.6559975	25.8572304	31.1979027	36.2353184	43.9493184	49.95063	55.2476913	59.7893659	62.8153398	67.1719135
132	Luxembourg	39.5459604	53.9110489	64.84354	68.801977	71.4282649	78.2164508	81.9154848	87.2843919	90.7084834	90.7038249
133	Macao, China	25.1718802	25.742124	31.4840974	34.8629272	46.4	47.327	49.24	54	53.8	58
134	Macedonia, FYR	17.33	19.07	24.44	26.45	28.62	36.3	46.04	51.77	51.9	56.7
135	Madagascar	0.33972015	0.42325242	0.52535365	0.5677218	0.60755224	0.65	1.65	1.63	1.7	1.9
136	Malawi	0.21509473	0.27881512	0.34750533	0.38448933	0.42513749	0.96586474	0.7	1.07	2.26	3.33
137	Malaysia	32.3382043	34.9711523	42.2522656	48.6291702	51.637989	55.7	55.8	55.9	56.3	61
138	Maldives	5.34776517	5.97659285	6.58825488	6.86960507	11.0363528	16.3	23.2	24.8	28.3	34
139	Mali	0.22704558	0.31036444	0.43281964	0.50706314	0.72962728	0.81	1.57	1.8	1.9	2
140	Malta	29.378301	32.1325616	35.1303255	41.7973732	40.8749445	47.3079457	50.32265	59.0347355	63.0787509	69.0308427
141	Marshall Islands	2.33544458	2.56975037	3.59977681	3.8787024	3.79559021	3.71082549	3.62677217	3.54604213		
142	Martinique										
143	Mauritania	0.36322941	0.4240049	0.48147044	0.66996647	0.97966125	1.4336132	1.87	2.28	4	4.5

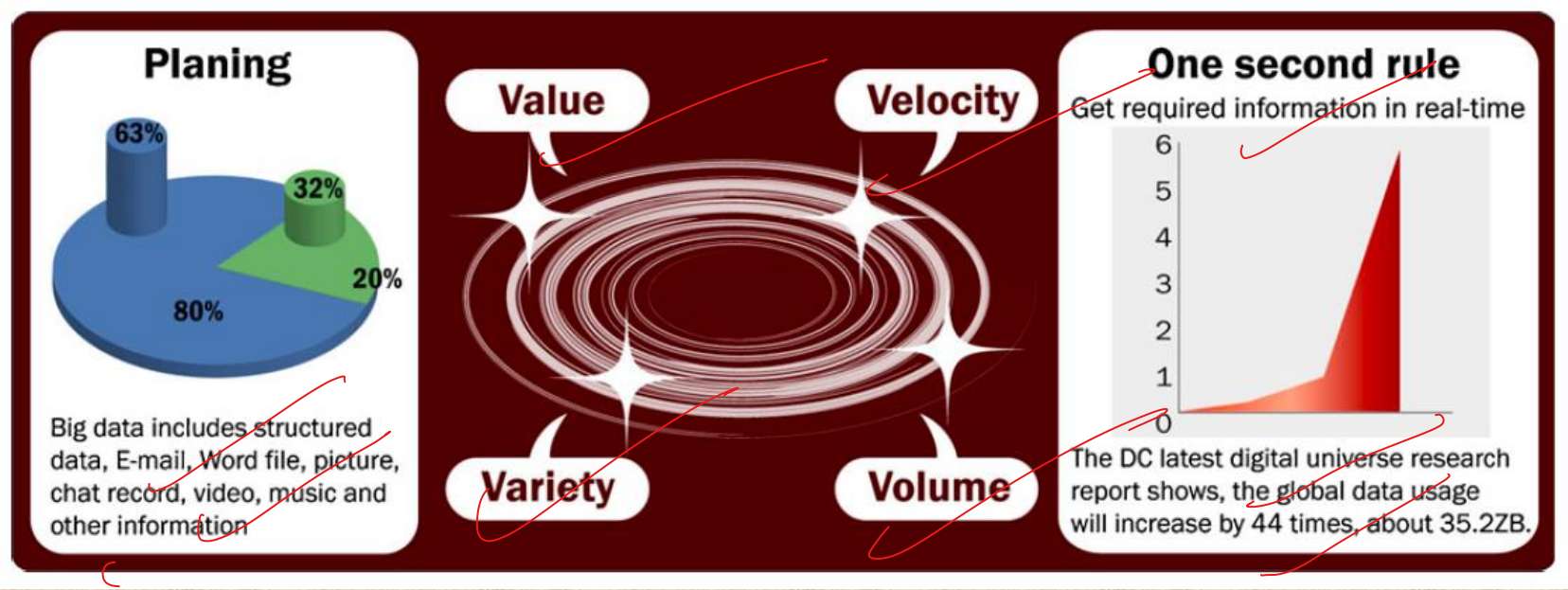
Characteristics of Big Data



related to the quality of the data

(Source: http://www.ibmbigdatahub.com/infographic/four-vs-big-data)

Characteristics of Big Data



[1] M. Chen, S. Mao and Y. Liu, "Big Data: A Survey", *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171-209, 2014.

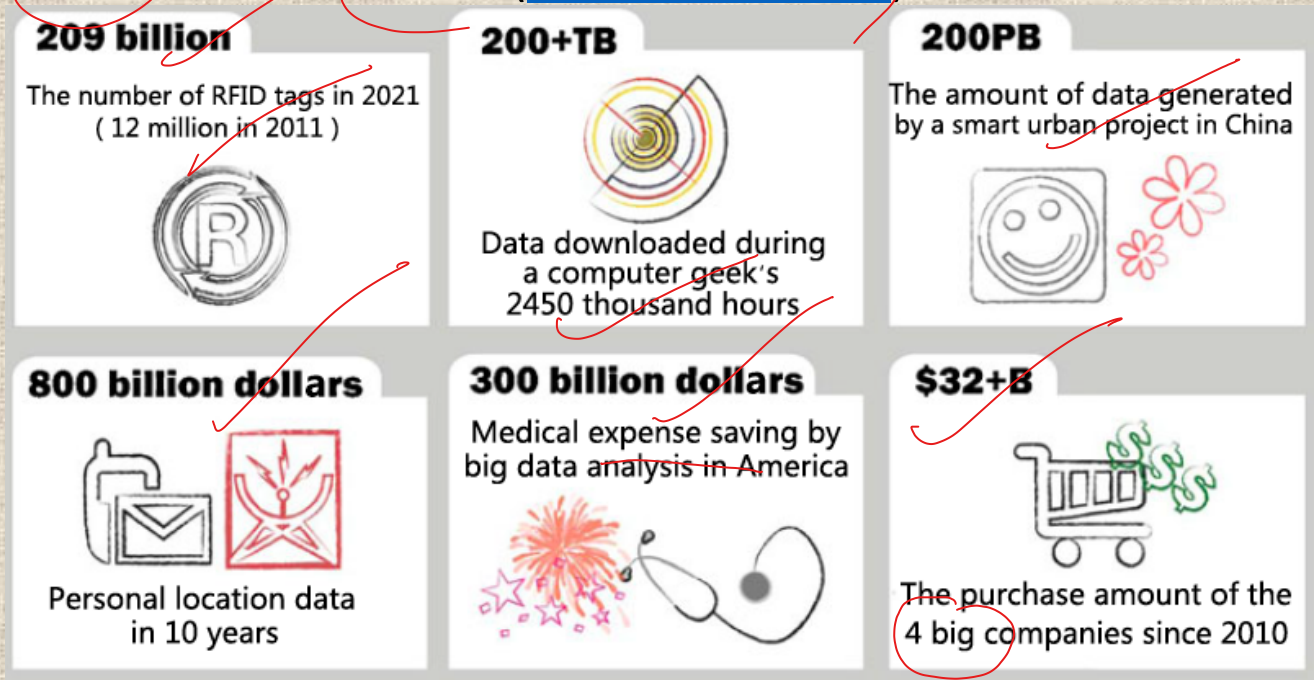
Characteristics of Big Data

- Volume
- Velocity
- Variety
- Veracity
- Variability
- Visualization
- Value

added later on

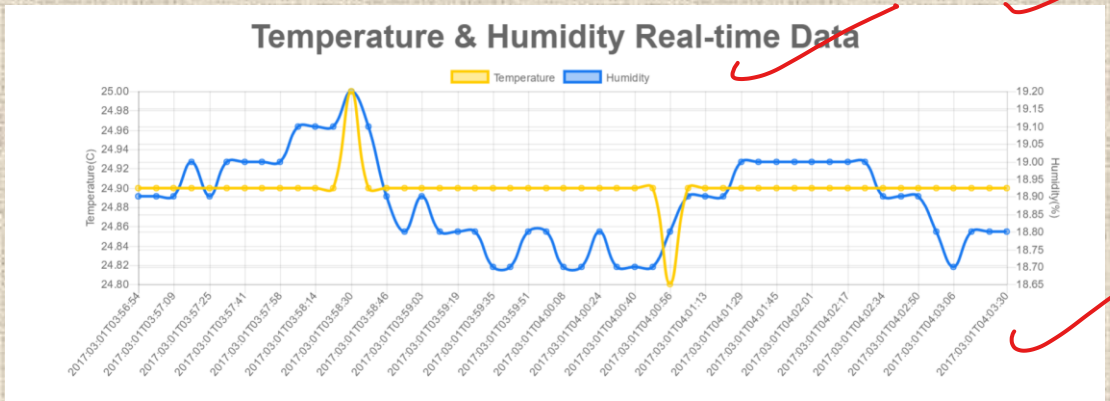
Characteristics of Big Data

- Volume
  - Large **amount** of data, big scale of data
  - Example:
    - 40 Zettabytes of data will be created by 2020, an increase of 300 times from 2015 (size of data units)



Characteristics of Big Data

- Velocity
  - Analysis of **streaming** data
    - Trade data, Internet-of-Things (IoT) sensing, network data
  - the content of the data is constantly changing, through
    - the absorption of complementary data collections
    - previously archived data or legacy collections
    - from streamed data arriving from multiple sources



Source: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-live-data-visualization-in-web-apps

Characteristics of Big Data

- Variety
  - Different **forms** of data
  - Example: traditional databases, text, images, documents, and complex records

Source: https://makeawebsitehub.com/social-media-sites/

Characteristics of Big Data

- Veracity
  - How to make sure the data is **accurate**?
  - Uncertainty of data exist, due to the **poor data quality**
  - Example: marketing automation system with false names and inaccurate contact information
  - One of three business leaders don't trust the information they use to make decisions

Characteristics of Big Data

- Variability
  - data whose **meaning is constantly changing**.
  - Example: language processing.
- Visualization
  - presenting data using graphs and charts
- Value
  - Get business insights
  - Know your customer

Source: http://dataconomy.com/2014/05/seven-vs-big-data/, https://www.impactradius.com/blog/7-vs-big-data/

Difference between big data and massive data

- Big Data
  - Most of the V's should apply
- Massive data
  - enormous collections of simple-format records

- Big data resources are **not equivalent** to a large spreadsheet, and a big data resource is not analyzed in its totality.
- Big Data analysis is a **multistep** process whereby data is extracted, filtered, and transformed, with analysis often proceeding in a piecemeal, sometimes recursive, fashion.

Big data and small data

- Big data is **NOT**
  - small data that has become bloated to the point that it can no longer fit on a spreadsheet, or
  - a database that happens to be very large
- **False impression** from professionals who customarily work with relatively small data:
  - they can apply their spreadsheet and database skills directly to big data resources without mastering new skills and without adjusting to new analytic paradigms
  - Only the computer need to be improved

Big data and Small data - Difference

1. Goal - main thing (must achieve)
2. Location
3. Data structure and content
4. Data preparation
5. Longevity
6. Measurement
7. Reproducibility
8. Stakes/cost
9. Introspection/observation
10. Analysis

set a goal to

All depends on these factors.

Big data and Small data - Difference

1. Goal:

- small data—Usually designed to answer a **specific** question or serve a particular goal.
- Big Data—Usually designed with a **goal in mind**, but the goal is **flexible** and the questions posed are protean.

more than one goal and very dynamic

Big data and Small data - Difference

1. Goal:
   - Example: "to combine high-quality data from fisheries, Coast Guard, commercial shipping, and coastal management agencies for a growing data collection that can be used to support a variety of governmental and commercial management studies in the lower peninsula."
     - a vague goal
     - there really is no way to completely specify
       - what the big data resource will contain
       - how the various types of data held in the resource will be organized, connected to other data resources, usefully analyzed.
     - Nobody can specify, with any degree of confidence, the ultimate destiny of any big data project. It usually comes as a surprise.

Big data and Small data - Difference

2. Location:

- small data—Typically, small data is contained **within one institution**, often on one computer, sometimes in one file.
- Big Data—Typically spread throughout electronic space, typically parceled onto **multiple Internet servers**, located anywhere on earth.

Big data and Small data - Difference

3. Data structure and content:

- small data—Ordinarily contains **highly structured data**.
  - The data domain is restricted to a single discipline or subdiscipline. The data often comes in the form of uniform records in an ordered spreadsheet.
- Big Data —**Must be capable of absorbing unstructured data** (e.g., such as free-text documents, images, motion pictures, sound recordings, physical objects).
  - The subject matter of the resource may cross multiple disciplines, and the individual data objects in the resource may link to data contained in other big data resources.

Big data and Small data - Difference

4. Data preparation

- small data —In many cases, the data user prepares her **own data**, for her **own purposes**.
- Big Data —The data comes from **many diverse sources**, and it is prepared by many people. People who use the data are seldom the people who have prepared the data.

Big data and Small data - Difference

5. Longevity

- small data —When the data project ends, the data is kept for a limited time (seldom longer than 7 years, the traditional academic life span for research data) and then discarded.
- Big Data —Big Data projects typically contain data that **must be stored in perpetuity**.
  - Ideally, data stored in a Big Data resource will be absorbed into another resource when the original resource terminates. Many Big Data projects extend into the future and the past (e.g., legacy data), accruing data prospectively and retrospectively.

Big data and Small data - Difference

6. Measurements

- small data —Typically, the data is measured using one experimental protocol, and the data can be represented using **one set of standard units**
- Big Data —Many different types of data are delivered in **many different electronic formats**.
  - Measurements, when present, may be obtained by **many different protocols**. Verifying the quality of big data is one of the most difficult tasks for data managers.

Big data and Small data - Difference

7. Reproducibility

- small data —Projects are typically **repeatable**.
  - If there is some question about the quality of the data, reproducibility of the data, or validity of the conclusions drawn from the data, the entire project can be repeated, yielding a new data set.
- Big Data —Replication of a Big Data project is **seldom feasible**.
  - In most instances, all that anyone can hope for is that bad data in a big data resource will be found and flagged as such.

Big data and Small data - Difference

8. Stakes/cost

- small data —Project costs are **limited**.
  - Laboratories and institutions can usually recover from the occasional small data failure.
- Big Data —Big Data projects can be obscenely **expensive**.

Big data and Small data - Difference

9. Introspection/observation

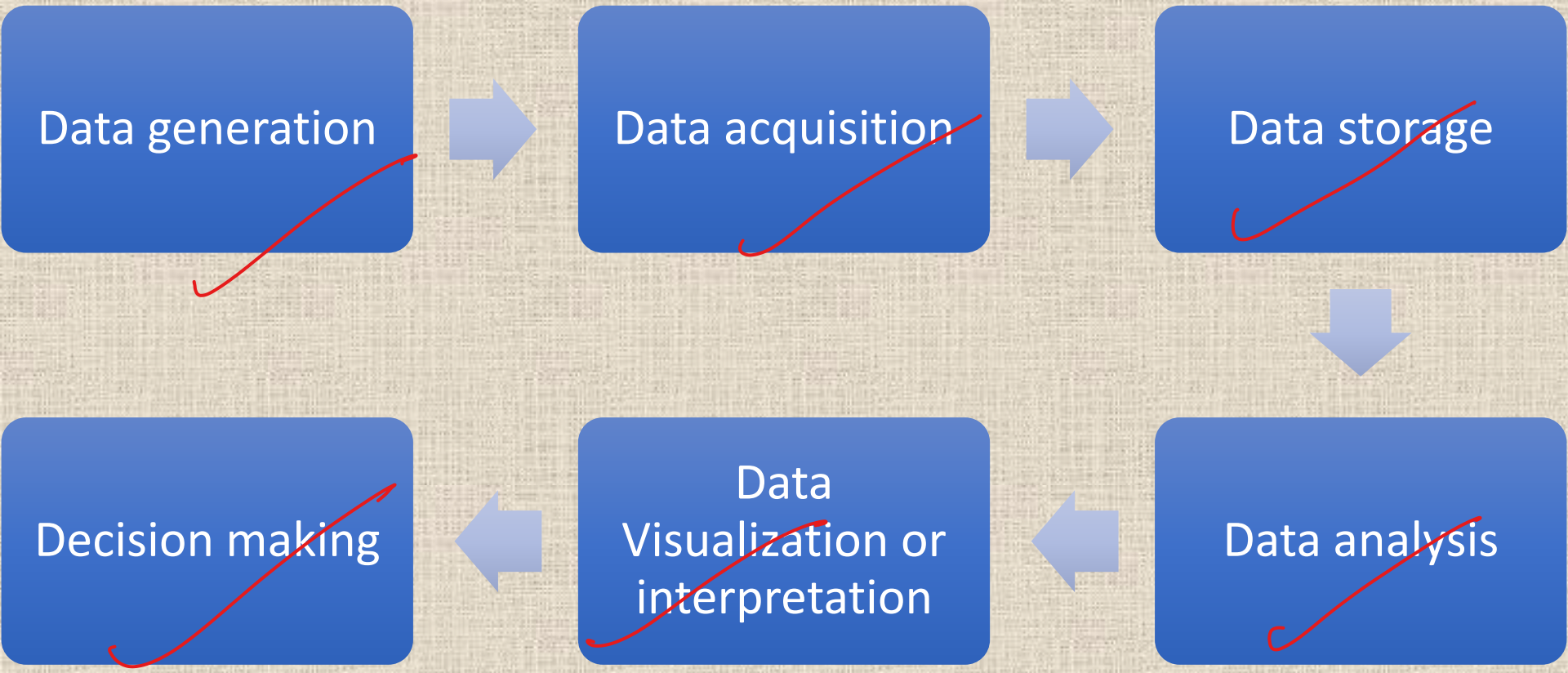
- small data —Individual data points are identified by their row and column location within a spreadsheet or database table.
  - If you know the row and column headers, you can find and specify all of the data points contained within.
- Big Data —Unless the Big Data resource is exceptionally well designed, the contents and organization of the resource can be inscrutable, even to the data managers

Big data and Small data - Difference

10. Analysis

- small data —In most instances, all of the data contained in the data project can be **analyzed together, and all at once**.
- Big Data —With few exceptions, such as those conducted on supercomputers or in parallel on multiple computers, Big Data is ordinarily analyzed **in incremental steps**.
  - The data are extracted, reviewed, reduced, normalized, transformed, visualized, interpreted, and reanalyzed with different methods.

Six phases of Big Data



Data Generation

- Enterprise Data
  - online trading data and online analysis data, most of which are historically static data
  - Internal data e.g. production data, inventory data, sales data, and financial data which aims to capture informationized and data-driven activities in enterprises, so as to record all activities of enterprises
- IoT (Internet-of-Things) data
  - smart cities, industry, agriculture, traffic, transportation, medical care, public departments, and families
- Bio-medical data
  - Medical images and electronic medical records

Data Acquisition

- Data collection
  - Log files, sensing, network data
- Data transmission/transportation
  - Inter-DCN transmissions: Inter-DCN transmissions are from data source to data center, which is generally achieved with the existing physical network infrastructure.
  - Intra-DCN Transmissions: Intra-DCN transmissions are the data communication flows within data centers.
- Data pre-processing
  - Integration, cleaning and redundancy elimination

(DCN – Dynamic circuit network)

imbalance
missing values
data redundancy

Data Storage

- Storage system for massive data
  - Direct Attached Storage (DAS)
  - Network Attached Storage (NAS)
  - Storage Area Network (SAN)
  - Distributed storage system

HDFS
(our lab)

- Storage mechanism:
  - Traditional relational databases
    - cannot meet the challenges on categories and scales brought about by big data.
  - NoSQL databases (i.e., non traditional relational databases) feature flexible modes, support for simple and easy copy, simple API, eventual consistency, and support of large volume data.
    - Example: Key-value databases, column-oriented databases, and document-oriented databases, each based on certain data models.

Data Analysis

- Traditional data analysis
  - Cluster Analysis
  - Factor Analysis
  - Correlation Analysis
  - Regression Analysis
  - A/B testing
  - Statistical analysis
  - Data mining algorithms

- Big data analytics method:
  - Bloom Filter
  - Hashing
  - Index
  - Parallel Computing
  - ML, DL also

Data Visualization and Interpretation

- To represent knowledge more intuitively and effectively by using different graphs
  - e.g. Tableau

Graph, Charts Google lockore, power BI

Decision Making

- Decision is made based on the analytical results
  - Is Big Data really helpful in improving decision making process?

When is Big Data needed?

Case 1:

- An entity has collected a lot of data, in the course of its **normal activities**, and seeks to organize the data so that materials can be retrieved, as needed.
- The Big Data effort is intended to streamline the regular activities of the entity. In this case, **the data is just waiting to be used.** The entity is **not looking to discover anything or to do anything new**.
  - Example: **Medical center** as an "accidental" Big Data resource. The administrative staff anticipated that the collected data can be used to achieve mandated goals: e.g. improving quality of service, increasing staff efficiency, and reducing operational costs.

When is Big Data needed?

Case 2:

- An entity has collected a lot of data in the course of its normal activities and **decides that there are many new activities that could be supported** by their data.
  - Example: Consider modern corporations—these entities **do not restrict themselves to one manufacturing process or one target audience**. They are constantly looking for **new opportunities**.
  - Their collected data may enable them to develop **new products** based on the preferences of their loyal customers, to **reach new markets**, or to market and distribute items via the Web. These entities will become hybrid Big Data/manufacturing enterprises.

When is Big Data needed?

Case 3:

- An entity **plans a business model** based on a Big Data resource. Unlike the previous entities, this entity starts with Big Data and adds a physical component secondarily.
  - Amazon and FedEx may fall into this category, as they began with a plan for **providing a data-intense service** (e.g., the Amazon Web catalog and the FedEx package-tracking system). The traditional tasks of warehousing, inventory, pickup, and delivery had been available all along, but lacked the novelty and efficiency afforded by Big Data.

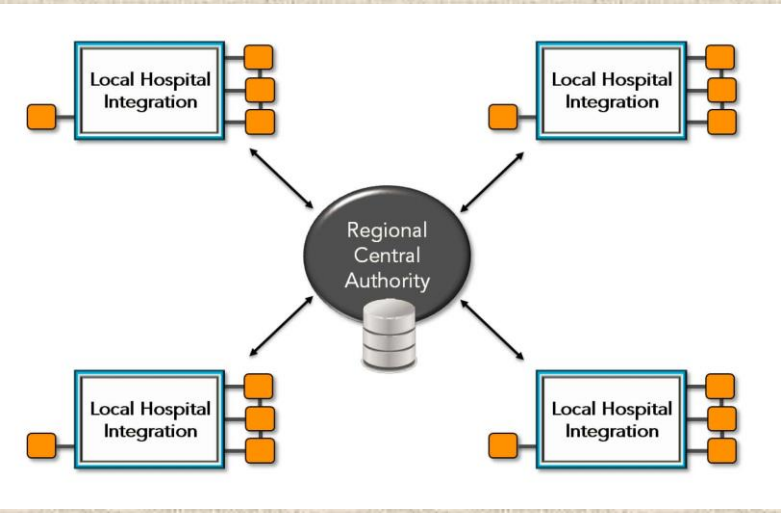
AWS
Cloud
services

make money for the Amazon group of companies

When is Big Data needed?

Case 4:

- An entity is part of a group of entities that have large data resources, all of whom understand that it would be to their mutual advantage to **federate** (centralize) their data resources.
  - Example: hospital databases that share electronic medical health records



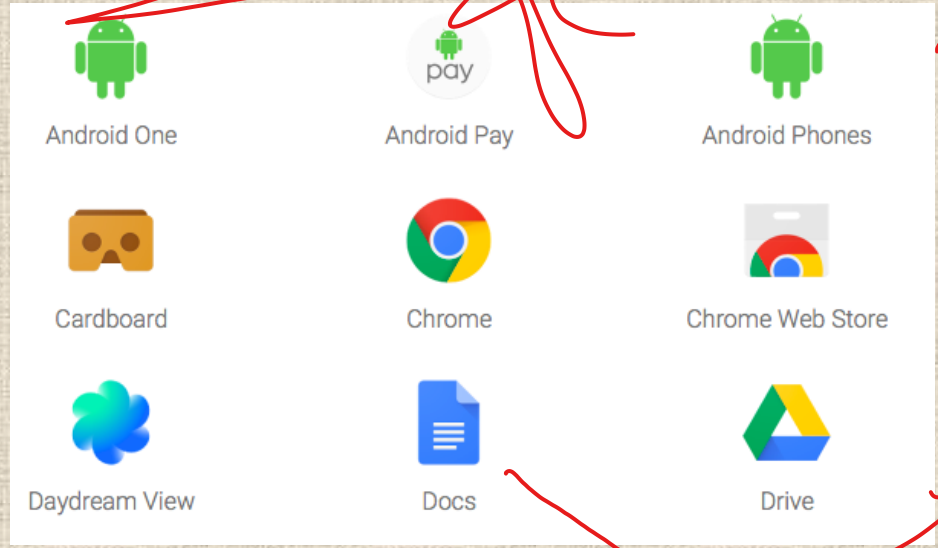
Source: https://corepointhealth.com/health-information-exchange-architecture-types

When is Big Data needed?

Example of "google"

Case 5:

- An entity with skills and vision **develops a project wherein large amounts of data are collected and organized** to the **benefit of themselves and their user-clients.**
  - Example: Google, and its many services



Big Data created

access document concurrently

Source: https://www.google.com/about/products/

When is Big Data needed?

Case 6: i.e "Government"

- An entity has **no data and has no particular expertise** in Big Data technologies, but it **has money and vision**.
- The entity seeks to **fund and coordinate** a group of data creators and data holders who will build a Big Data resource that can be used by others.
  - Example: Government agencies have been the major benefactors. These Big Data projects are justified if they lead to important discoveries that could not be attained at a lesser cost, with smaller data resources.

When is Big Data needed?

The most common purpose of Big Data is to produce SMALL DATA

- Imagine using a restaurant locater on your smartphone:
  1. With a few taps, it lists the Italian restaurants located within a 10 block radius of your current location.
  2. The database being queried is big and complex (a map database, a collection of all the restaurants in the world, their longitudes and latitudes, their street addresses, and a set of ratings provided by patrons, updated continuously)
  3. the data that it yields is small (e.g., five restaurants, marked on a street map, with pop-ups indicating their exact address, telephone number, and ratings).
  4. Your task comes down to selecting one restaurant from among the five and dining thereat.

When is Big Data needed?

The most common purpose of Big Data is to produce SMALL DATA

- your data selection was drawn from a large data set, but **your ultimate analysis was confined to a small dataset** (i.e., five restaurants meeting your search criteria).
  - The purpose of the Big Data resource was to proffer the small dataset. No analytic work was performed on the Big Data resource—just search and retrieval. **The real labor of the Big Data resource involved collecting and organizing complex data so that the resource would be ready for your query.** Along the way, the data creators had many decisions to make
  - e.g., Should bars be counted as restaurants? What about take-away only shops? What data should be collected? How should missing data be handled? How will data be kept current?

Industrial Application

- Big data VS cloud services
  - Oracle and IBM
- Big data VS Internet-of-Things
  - Smarthome, VYROX
- Big data VS data center
  - Digital realty trust, Equinix
- Big data VS Hadoop
  - Cloudera, Hortonworks

used to be an entity

Direct to the cloudera website

Source: https://www.smarthome.com , http://vyrox.com , http://www.datacenterknowledge.com/archives/2017/01/20/here-are-the-10-largest-data-center-providers-in-the-world

Key Obstacles of the Big Data applications

1. Data representation
   - many datasets have certain levels of heterogeneity in type, structure, semantics, organization, granularity, and accessibility.
   - aims to make data more meaningful for computer analysis and user interpretation.
   - an **improper data representation will reduce the value of the original data** and may even obstruct effective data analysis.
   - Efficient data representation shall reflect data structure, class, and type, as well as integrated technologies, so as to **enable efficient operations on different datasets**.

Key Obstacles of the Big Data applications

2. Redundancy reduction and data compression
   - Generally, the redundancy level in datasets are high.
   - Aim to **reduce the indirect cost of the entire system** on the premise that the potential values of the data are not affected.
     - Example: most data generated by sensor networks are highly redundant, which may be filtered and compressed at orders of magnitude.

Key Obstacles of the Big Data applications

3. Data life cycle management
   - compared with the relatively slow advances of storage systems, pervasive sensing and computing are generating data at unprecedented rates and scales.
   - current storage system could not support such massive data.
   - **In general**, values hidden in big data depend on **data freshness**. Therefore, a data importance principle related to the analytical value should be developed to decide **which data shall be stored and which data shall be discarded**.

Key Obstacles of the Big Data applications

4. Analytical mechanism
   - traditional RDBMSs are strictly designed with a lack of scalability and expandability
   - non-relational databases have shown their unique advantages in the processing of unstructured data. But still exist problems in their performance in particular applications.
   - **a compromising solution between RDBMSs and non-relational databases should be identified.**
     - Example: some enterprises have utilized a mixed database architecture that integrates the advantages of both types of database (e.g., Facebook and Taobao).

Mobile users
- major e-commerce companies doing that
They have "converter"

Key Obstacles of the Big Data applications

5. Data confidentiality
   - most big data service providers or owners at present could not effectively maintain and analyze such huge datasets because of their limited capacity.
   - They must **rely on professionals or tools** to analyze such data, which **increase the potential safety risks**.
     - Example: the transactional dataset generally includes a set of complete operating data to drive key business processes.
     - Such data contains details of the lowest granularity and **some sensitive information such as credit card numbers**.
     - Therefore, analysis of big data may be delivered to a **third party** for processing only when proper preventive measures are taken to protect such sensitive data, to ensure its safety.

Key Obstacles of the Big Data applications

6. Energy management
   - the energy consumption of mainframe computing systems has drawn much attention from both **economy and environment** perspectives.
   - With the increase of data volume and analytical demands, the processing, storage, and transmission of big data will inevitably consume more and more electric energy.
     - Therefore, system-level power consumption control and management mechanism shall be established for big data while the expandability and accessibility are ensured.

Key Obstacles of the Big Data applications

7. Expendability and scalability
   - the analytical system of big data must support **present and future datasets**.
     - The analytical algorithm must be able to process **increasingly expanding and more complex datasets**.
8. Cooperation
   - analysis of big data is an interdisciplinary research, which requires **experts in different fields cooperate to harvest the potential of big data**.
     - A comprehensive big data network architecture must be established to help scientists and engineers in various fields access different kinds of data and fully utilize their expertise, so as to cooperate to complete the analytical objectives.

Coming Next

- Big data pipeline using Hadoop

References

[1] M. Chen, S. Mao and Y. Liu, "Big Data: A Survey", *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171-209, 2014.

[2] C. Philip Chen and C. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data", *Information Sciences*, 2014.

[3] J. Berman, *Principles and Practices of big data,* 2018.

[4] Panimalar, A., Shree, V., & Kathrine, V. “The 17 V’s of big data.” *International Research Journal of Engineering and Technology*, 2017.

Size of data units

Sizes of data units.

Name	Equals to	Size in bytes
Bit	1 bit	1/8
Nibble	4 bits	1/2
Byte	8 bits	1
Kilobyte	1024 bytes	1024
Megabyte	1024 kilobytes	1,048,576
Gigabyte	1024 megabytes	1,073,741,824
Terrabyte	1024 gigabytes	1,099,511,627,776
Petabyte	1024 terrabytes	1,125,899,906,842,624
Exabyte	1024 petabytes	1,152,921,504,606,846,976
Zettabyte	1024 exabytes	1,180,591,620,717,411,303,424
Yottabyte	1024 zettabytes	1,208,925,819,614,629,174,706,176

Back

WQD7007 Big Data Management

Big Data Pipeline

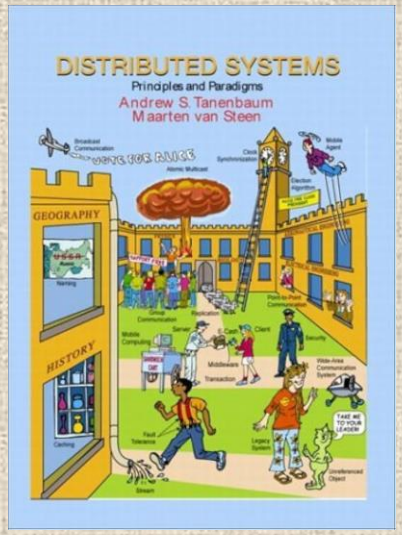
Using Hadoop

Agenda

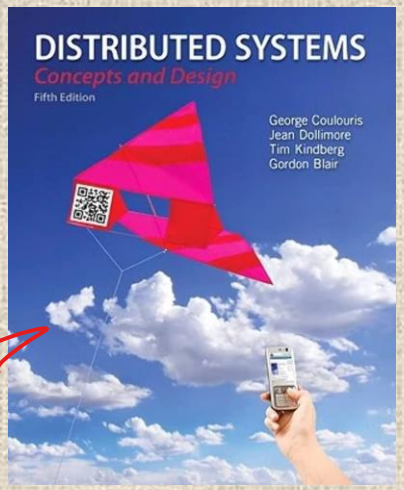
- What is Distributed Computing?
- What is Hadoop?
- Big Data Pipeline using Hadoop

Definition of Distributed Computing?

A collection of independent computers that appear to its users as a single coherent system (Tanenbaum book)



One in which components located at networked computers communicate and coordinate their actions only by passing messages (Coulouris book)



Why Distributed Computing?

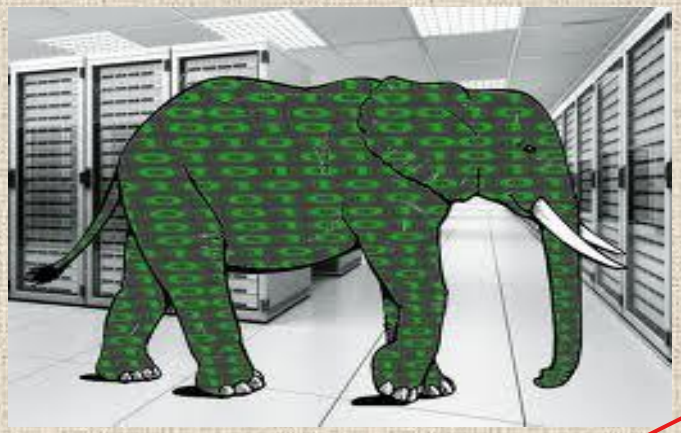
- Scale
  - Processing
  - Data
- Diversity in Application Domains
- Collaboration
- Cost
- Three sub-elements under DS
  - Storage
  - Computation
  - Messaging

Why Distributed Computing?

A. ***Big data*** continues to grow:

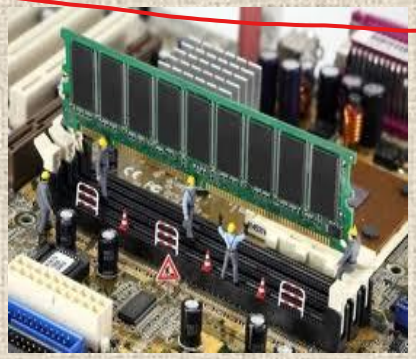
- In mid-2010, the information universe carried 1.2 zettabytes and 2020 predictions expect nearly 44 times more at 35 zettabytes coming our way.

B. Applications are becoming ***data-intensive***.

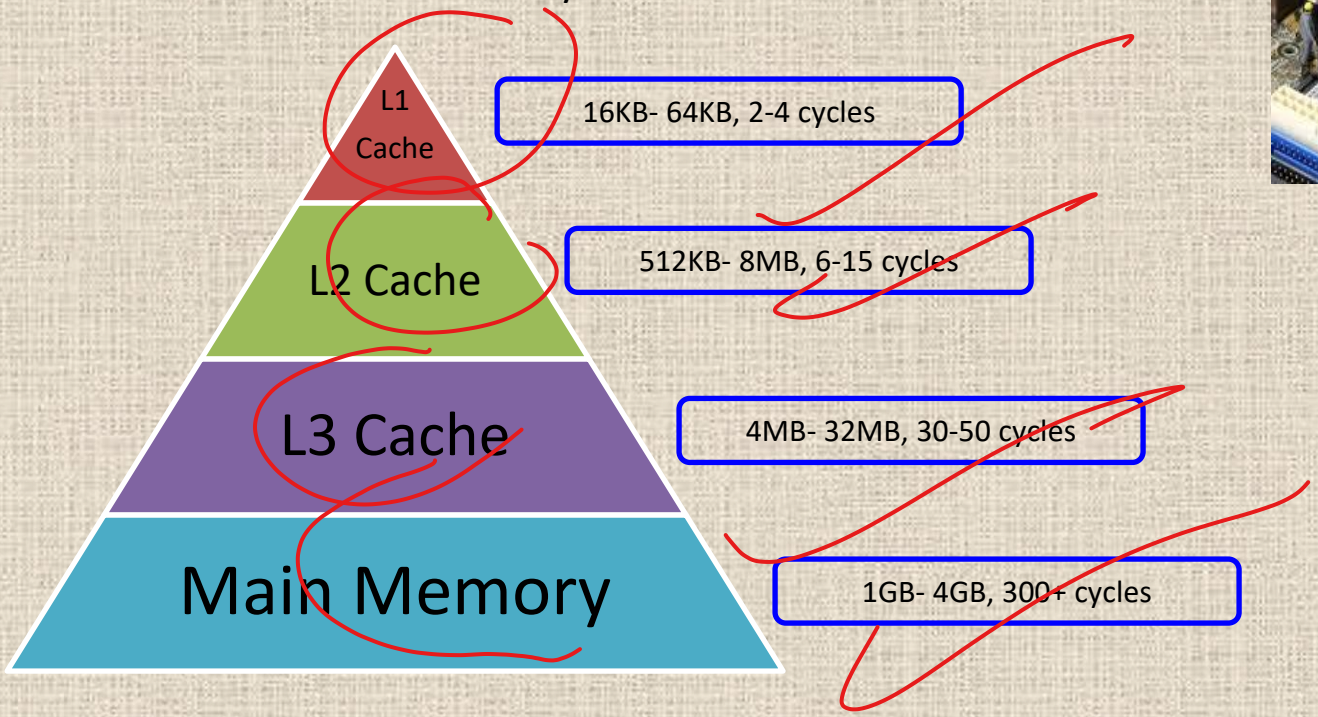


Why Distributed Computing?

C. Individual computers have limited resources compared to scale of current day problems & application domains:

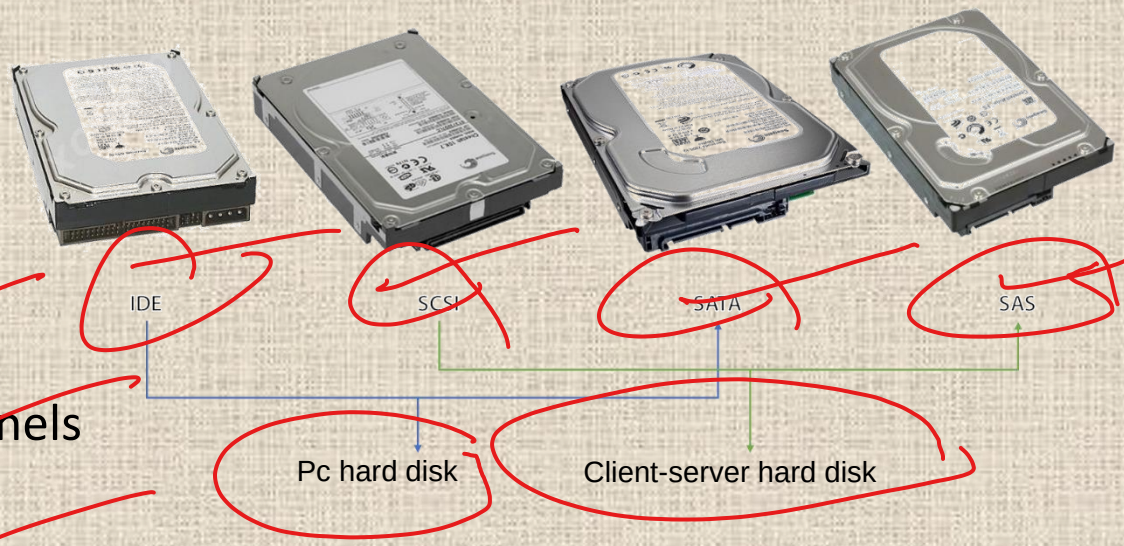


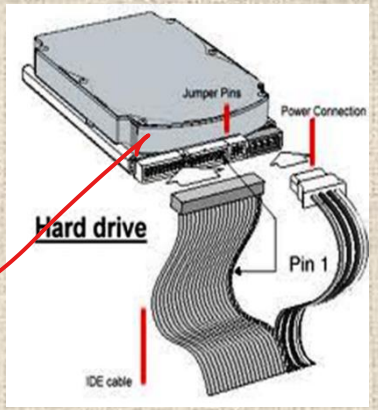
1. Caches and Memory:



2. Hard Disk Drive:

- Limited capacity
- Limited number of channels
- Limited bandwidth

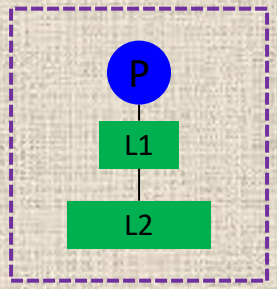




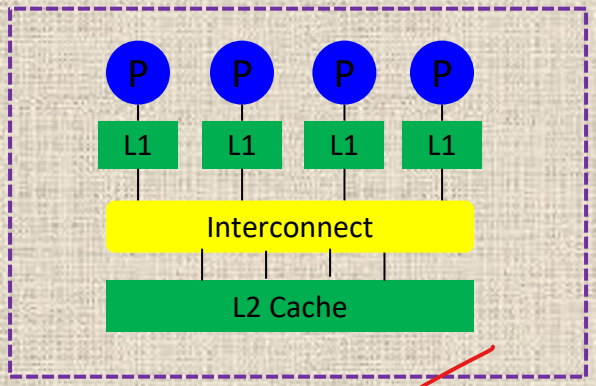
https://www.techgeekaura.com/what-is-ssd/

3. Processor:

- The number of transistors that can be integrated on a single die has continued to grow at Moore's pace.
- Chip Multiprocessors (*CMPs*) are now available



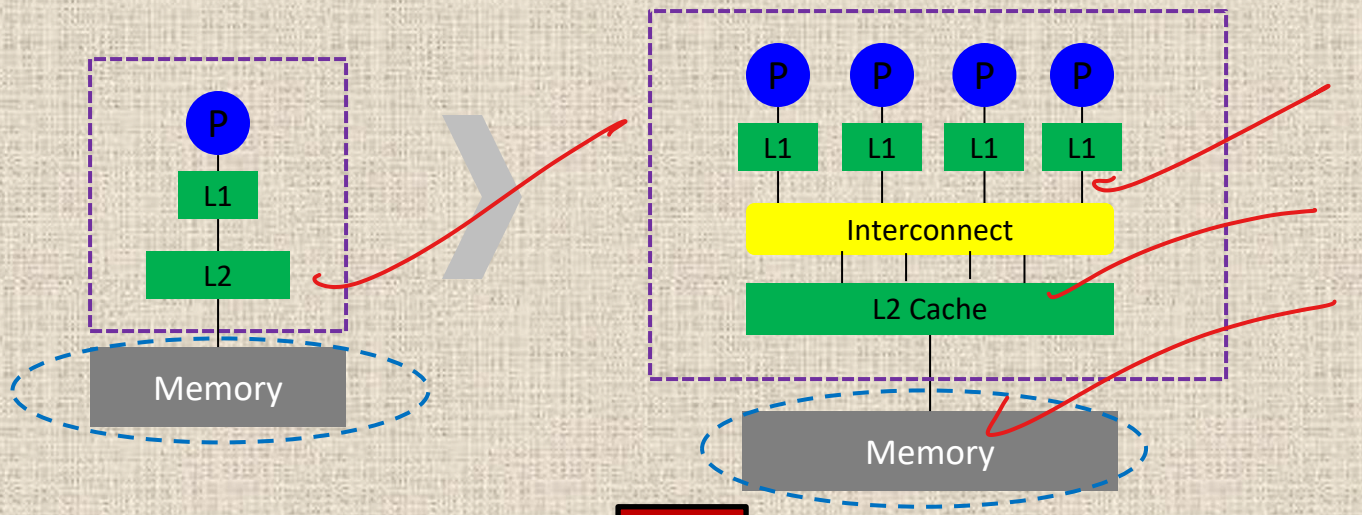
A single Processor Chip

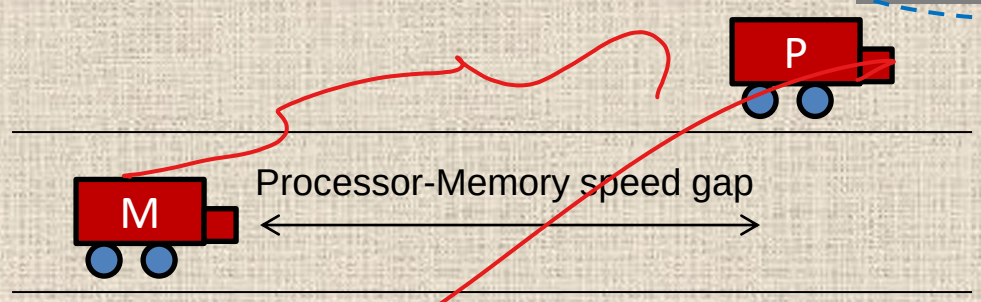


A CMP

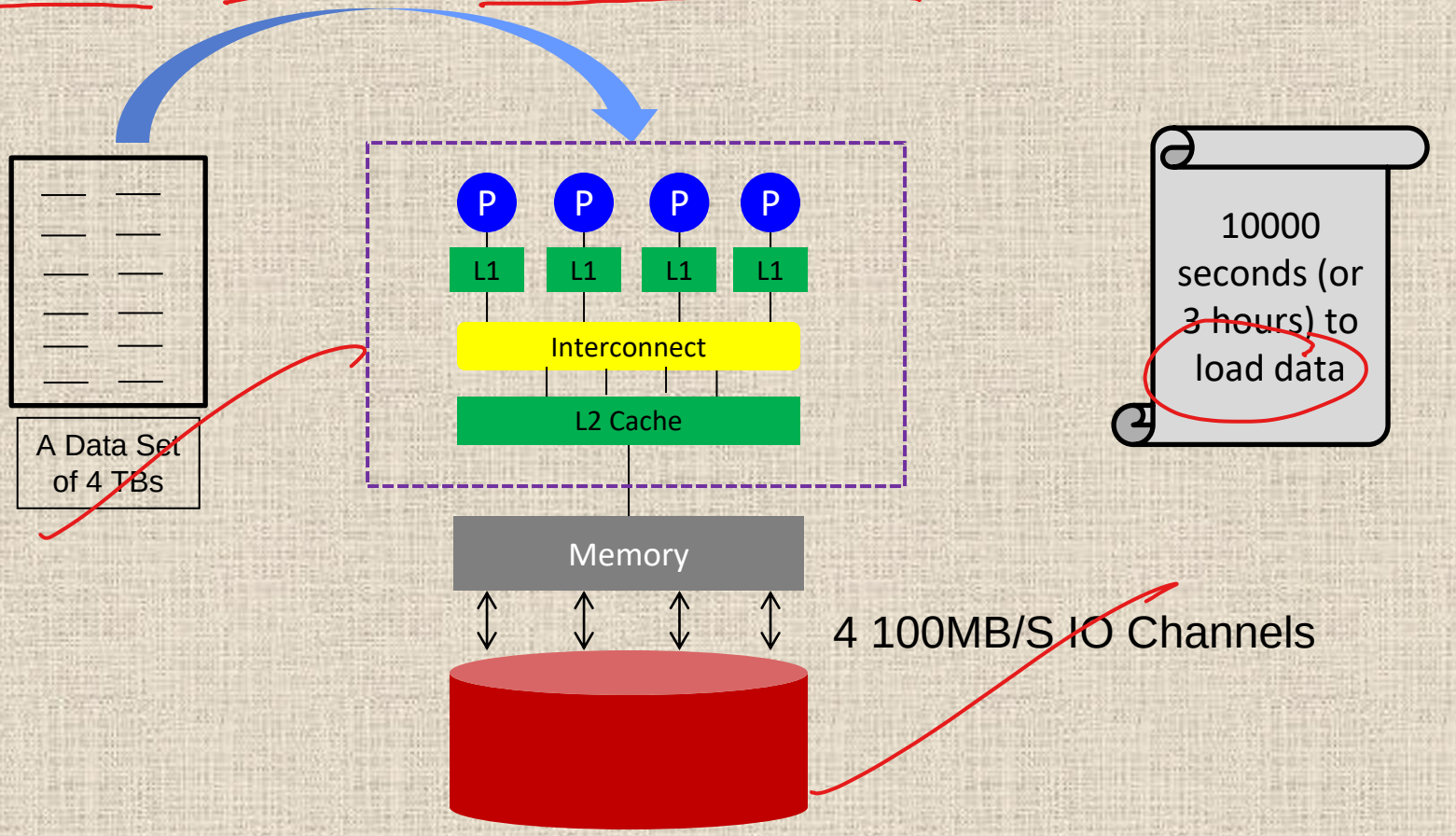
3. Processor (cont'd):

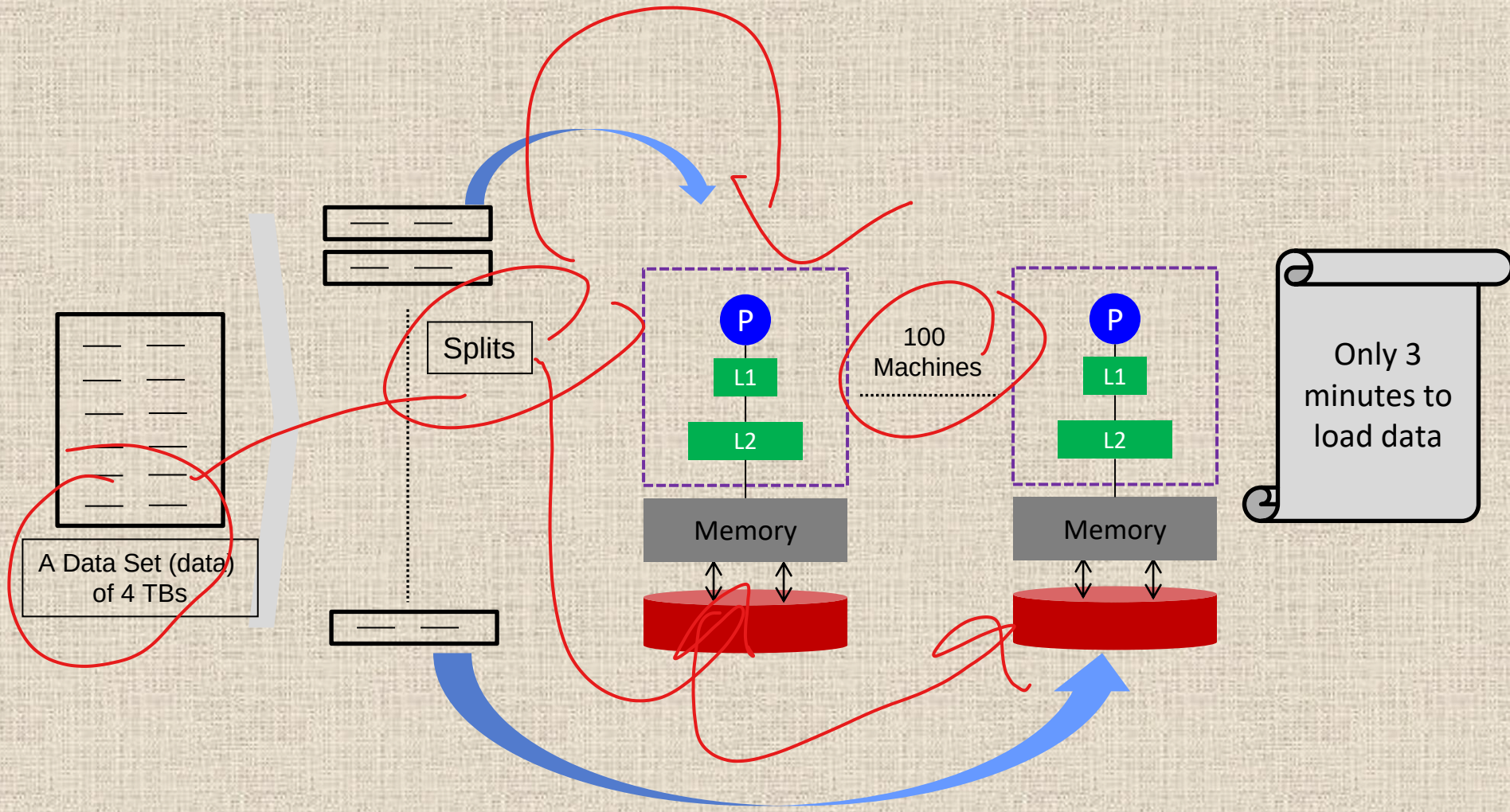
- Up until a few years ago, CPU speed grew at the rate of 55% annually, while the memory speed grew at the rate of only 7% [H & P].





- Even if 100s or 1000s of cores are placed on a CMP, it is a challenge to deliver input data to these cores fast enough for processing.




A Data Set (data) of 4 TBs
Splits
P
L1
L2
Memory
100 Machines
P
L1
L2
Memory
Only 3 minutes to load data

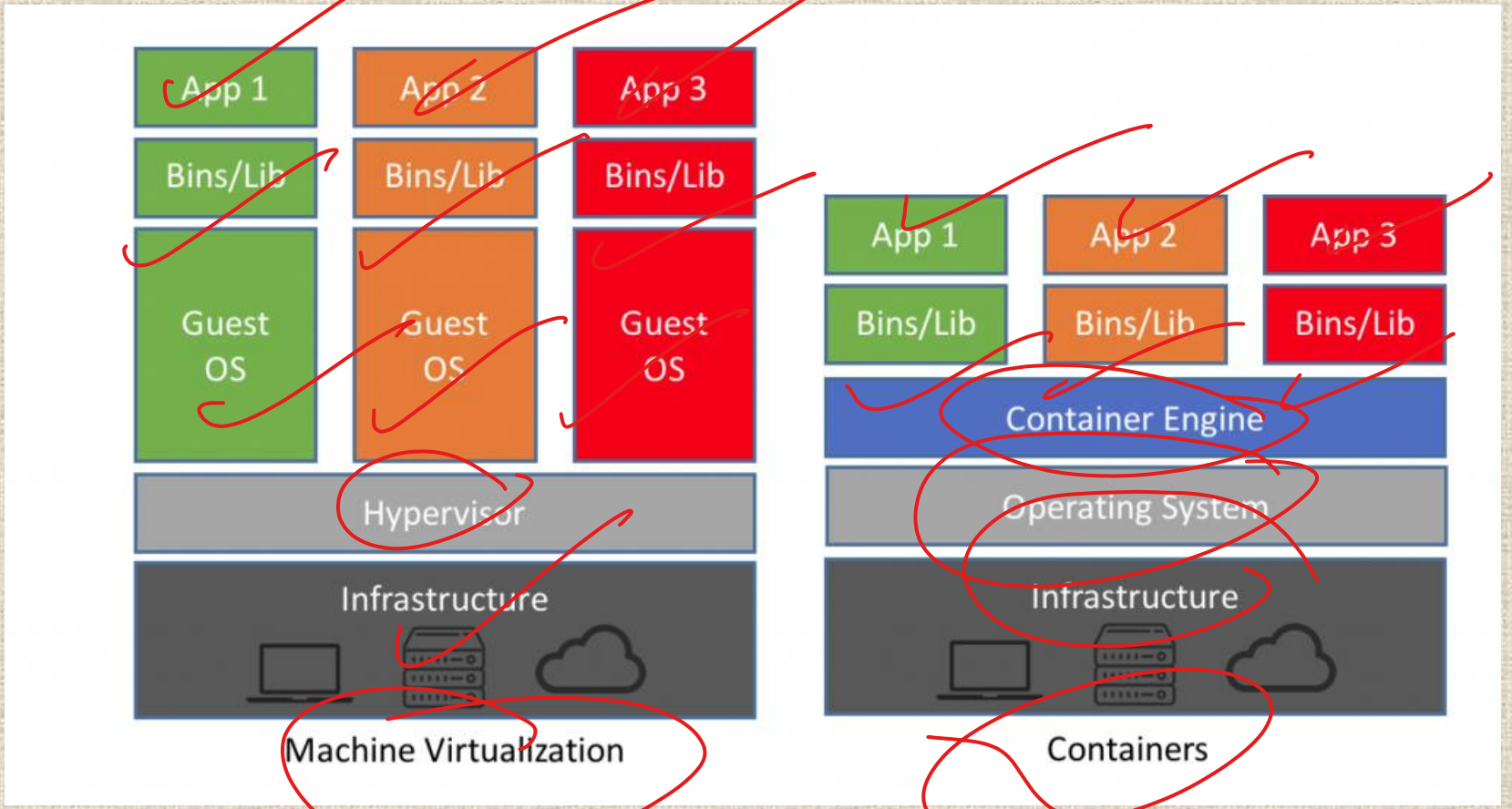
What is Distributed Computing?

But this requires:

- A way to express the problem as parallel processes and execute them on different machines (Programming Models and Concurrency).
- A way for processes on different machines to exchange information (Communication).
- A way for processes to cooperate, synchronize with one another and agree on shared values (Synchronization).
- A way to enhance reliability and improve performance (Consistency and Replication).
- A way to recover from partial failures (Fault Tolerance).
- A way to secure communication and ensure that a process gets only those access rights it is entitled to (Security).
- A way to extend interfaces to mimic the behavior of another system, reduce the diversity of platforms, and provide a high degree of portability and flexibility (Virtualization)

Virtual Machines vs Containers

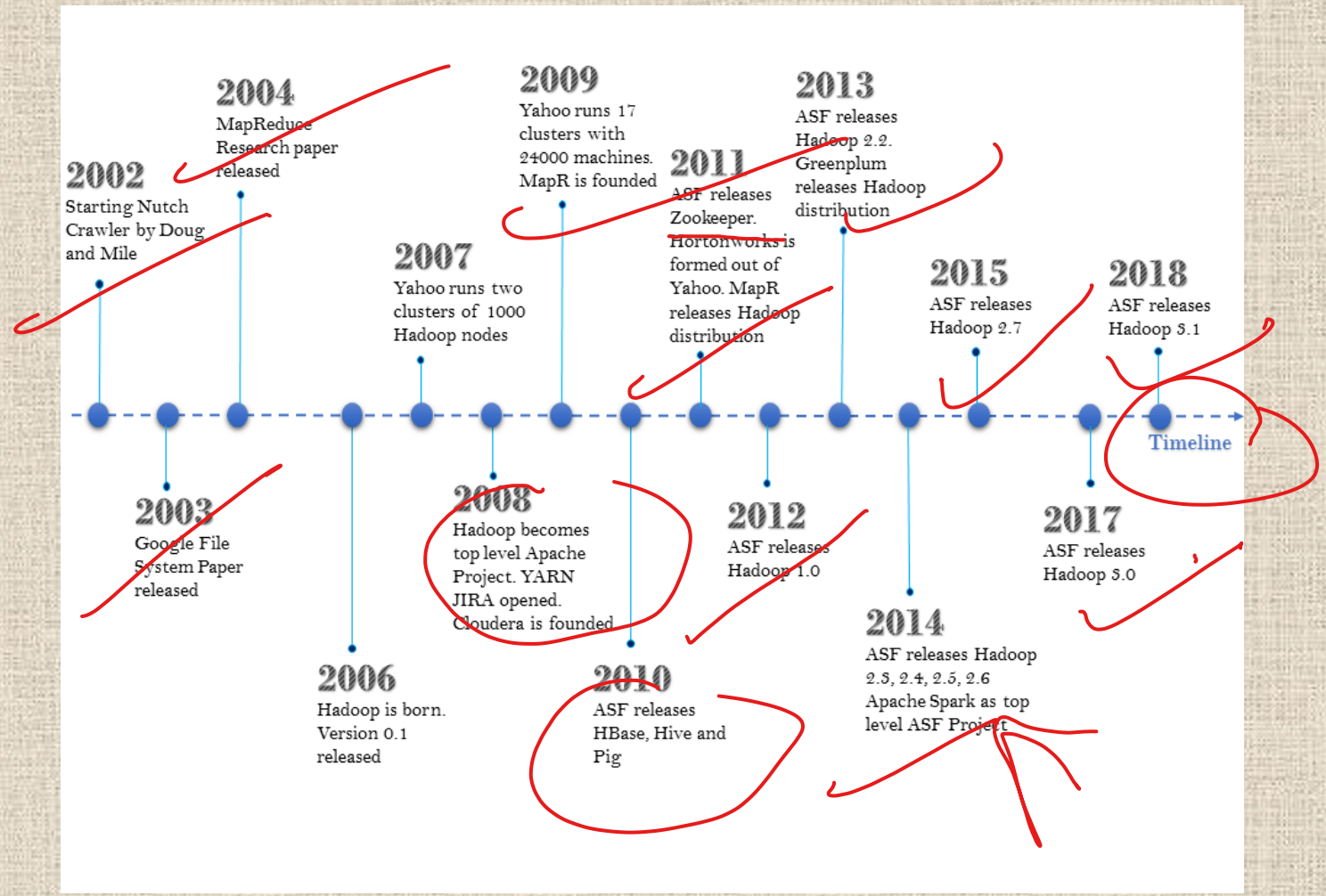
Hardware Utilization



What is Hadoop?

- Apache Hadoop is an **open source framework** for **distributed storage** and **processing** of **large sets** of data on **commodity** hardware.
  - Hadoop enables businesses to quickly gain insight from **massive amounts of structured and unstructured data**.
  - Numerous Apache Software Foundation projects make up the services required by an enterprise to deploy, integrate and work with Hadoop.

What is Hadoop?

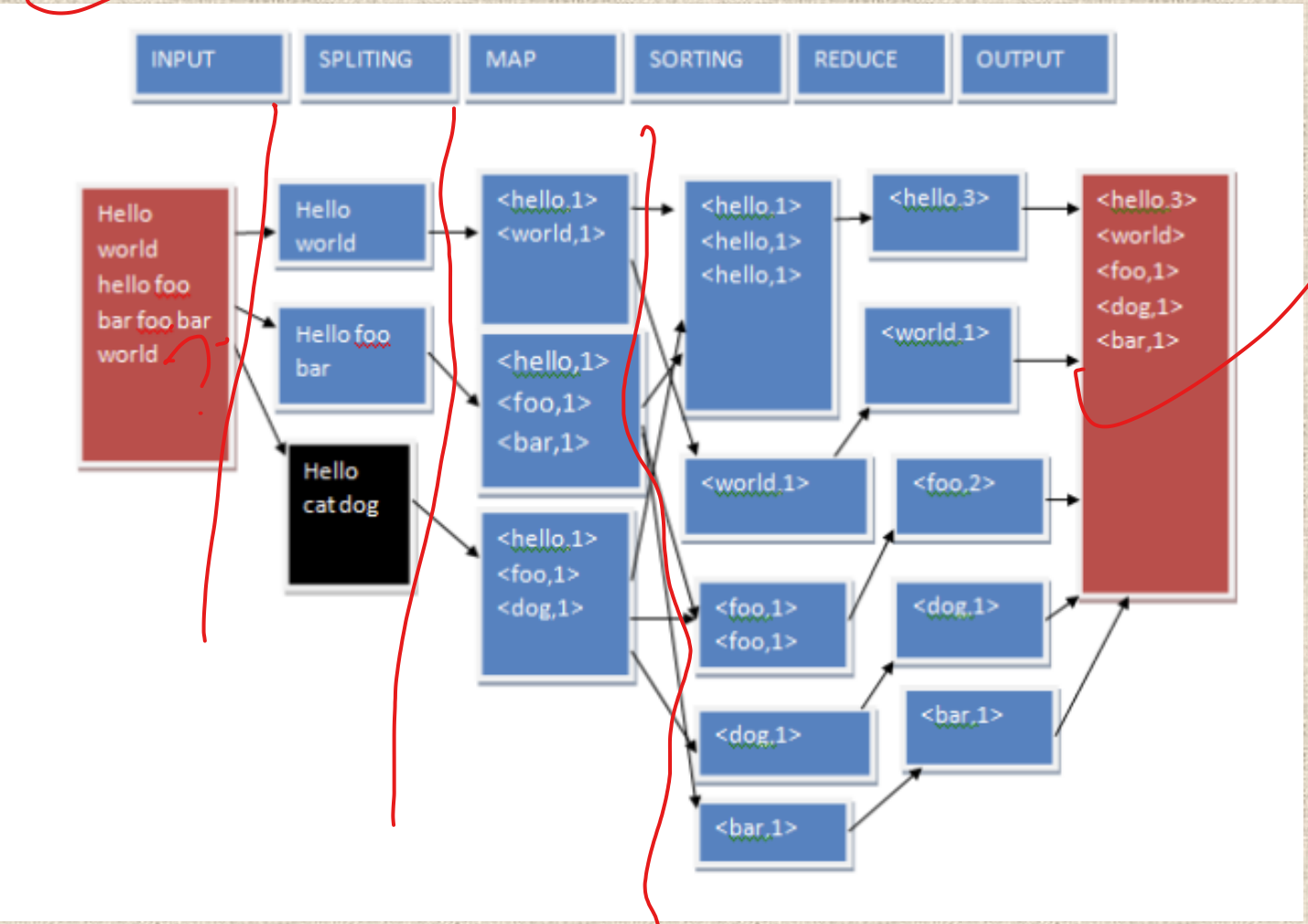


(Source: Vijay Karambelkar, H. (2018). Apache Hadoop 3 Quick Start Guide. United Kingdom: Packt Publishing.

What is Hadoop?

- The base Apache Hadoop framework are composed of following modules:
  - **Hadoop Common**: The common utilities that support the other Hadoop modules.
  - **Hadoop Distributed File System (HDFS)**: A distributed file system that provides high-throughput access to application data.
  - **Hadoop YARN**: A framework for job scheduling and cluster resource management.
  - **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

HDFS & Map Reduce Data Flow Diagram



https://www.e2matrix.com/blog/2017/10/27/hadoop-and-word-count-hadoop-distributed-file-system/

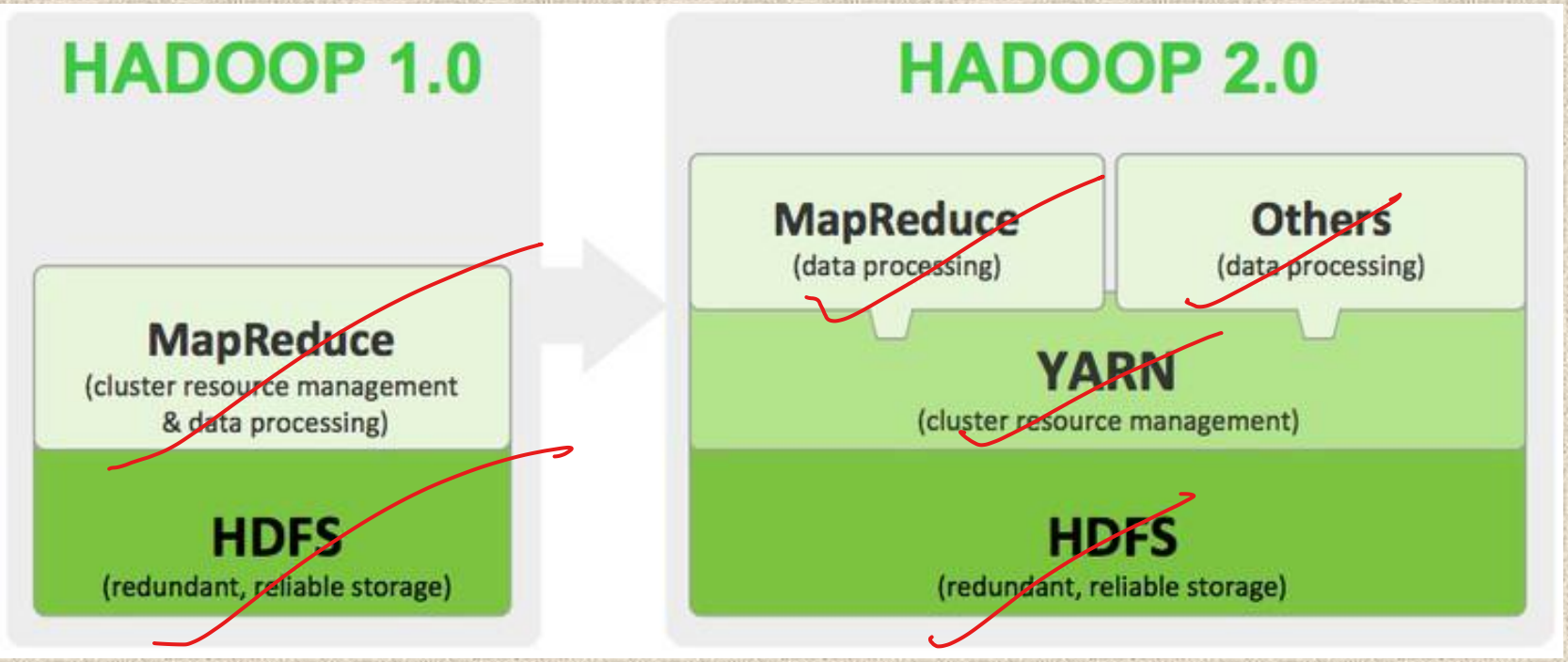
Hadoop 1.0

- Hadoop 1.0 is based on the Hadoop .20.205 branch (it went 0.18 -> 0.19 -> 0.20 -> 0.20.2 -> 0.20.205 -> 1.0).
- Not hard for an open source developer, but obscure for an enterprise product – so everyone agreed to call 0.20.205 '1.0', the project having matured to that point.

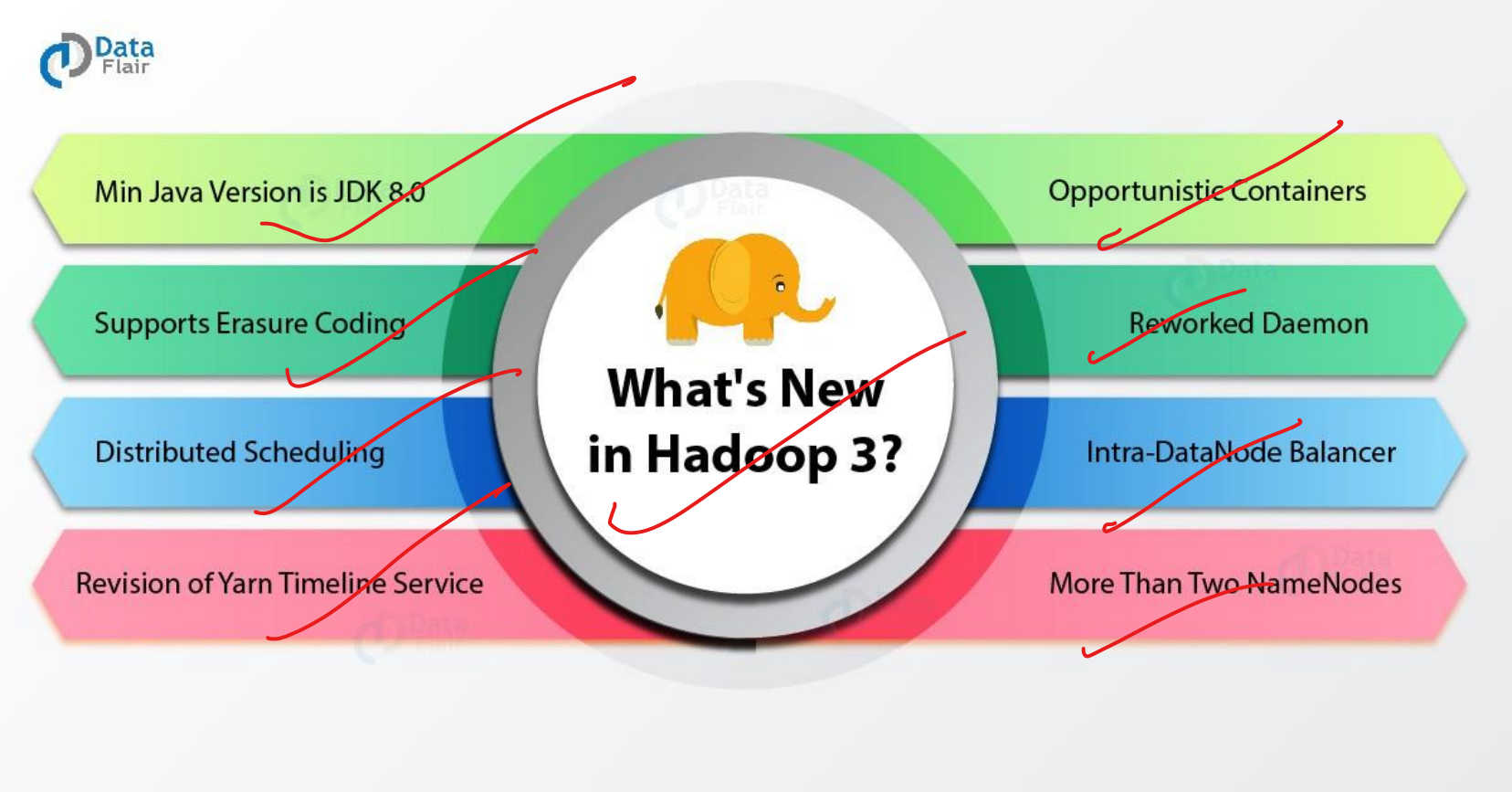
Hadoop 2.0

- Hadoop 2.0 is from the Hadoop 0.23 branch, with **major components re-written to enable support for features like High Availability, and MapReduce 2.0 (YARN)**, and to enable Hadoop to scale out past 4,000 machines per cluster.
- Specifically, Hadoop 2.0 adds:
  - HDFS Federation – multiple, redundant namenodes acting in congress
  - MapReduce NextGen aka YARN aka MRv2 – which transforms Hadoop into a full blown platform as a service.

Hadoop 1.0 to Hadoop 2.0

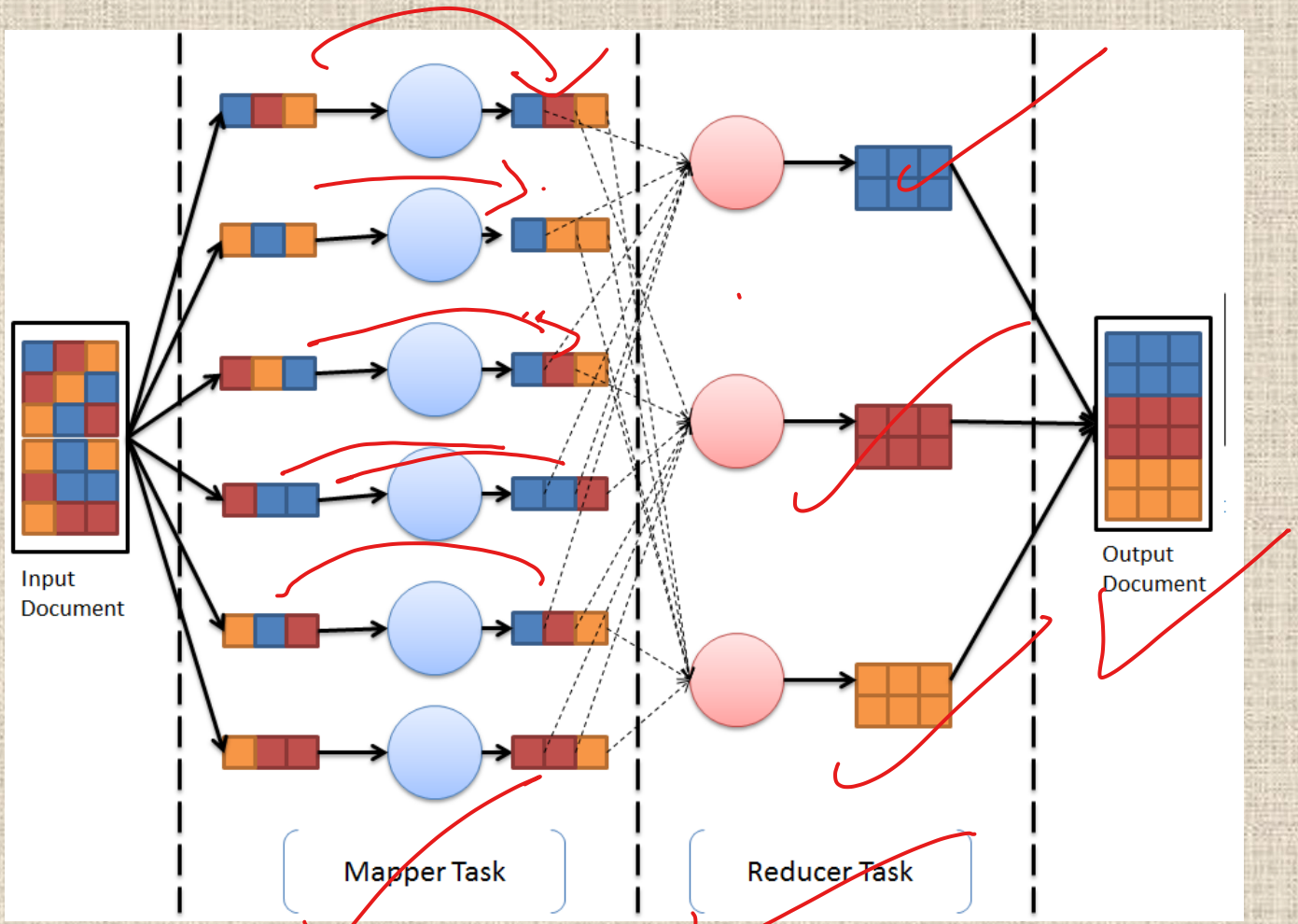


Hadoop 3.0



(Source: https://data-flair.training/blogs/what-is-new-in-hadoop-3/)

Hadoop 3.0



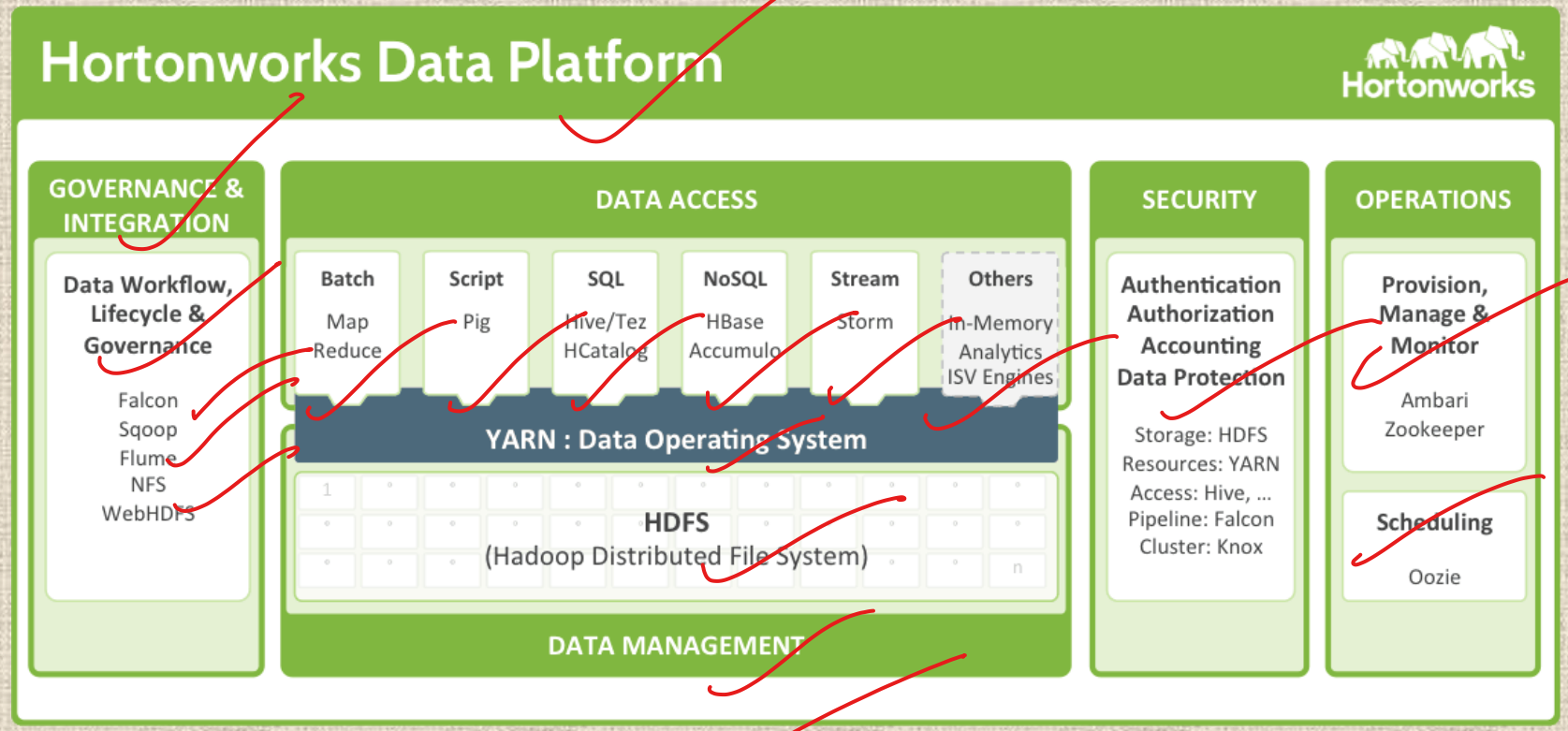
(Source: Vijay Karambelkar, H. (2018). Apache Hadoop 3 Quick Start Guide. United Kingdom: Packt Publishing.

This Photo by Unknown Author is licensed under CC BY

Five pillars to Hadoop

- Data Management
- Data Access
- Data Governance and Integration
- Security
- Operations

Five pillars to Hadoop



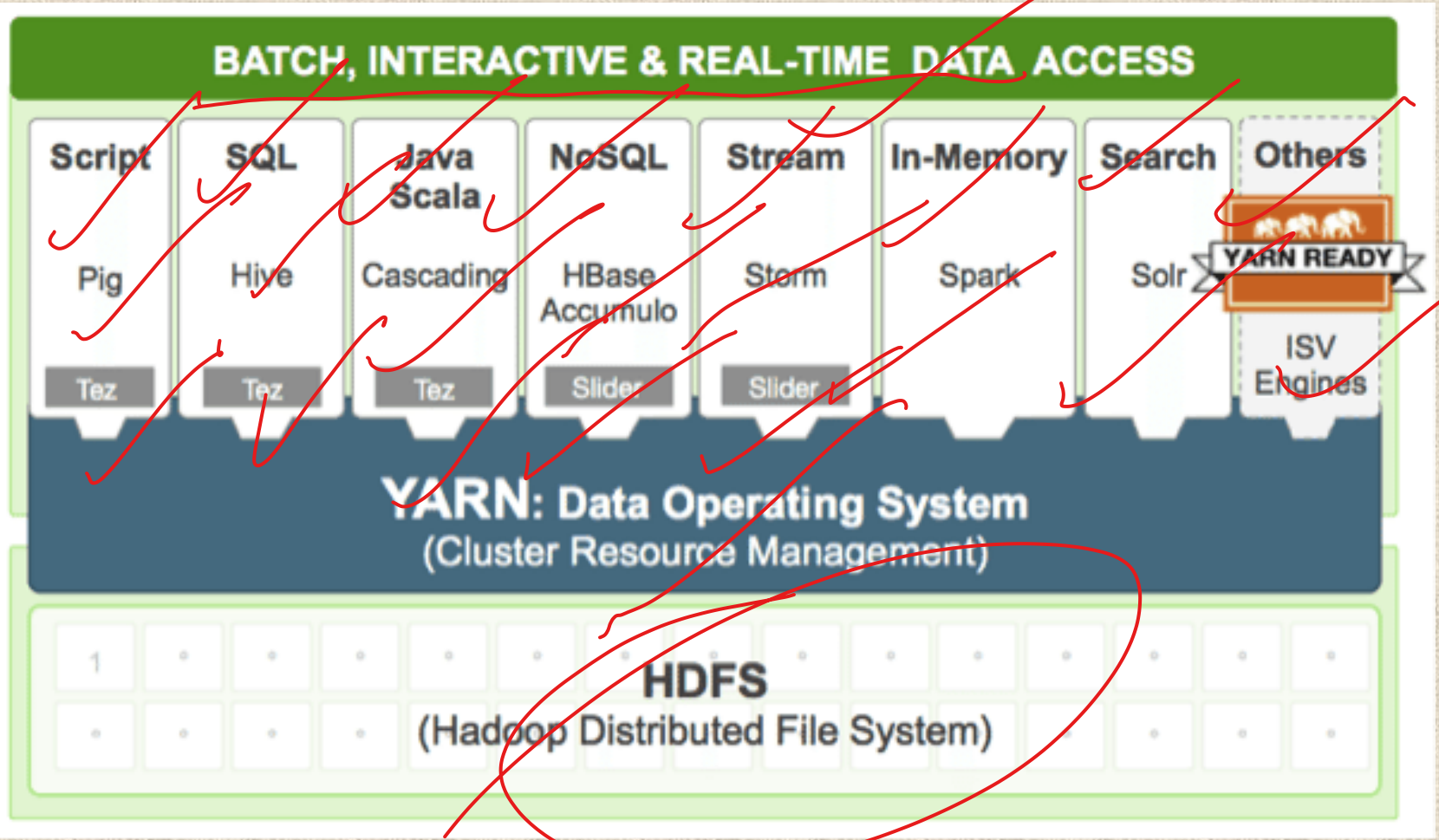
(Source: https://www.tableau.com/es-es/solutions/hortonworks)

READ MORE - https://en.wikipedia.org/wiki/Hortonworks

Data Management - YARN

- **Apache Hadoop YARN (Yet Another Resource Negotiator)** – Part of the core Hadoop project, YARN is a next-generation framework for Hadoop data processing extending MapReduce capabilities by supporting non-MapReduce workloads associated with other programming models.
  - allows **multiple data processing engines** such as interactive SQL, real-time streaming, data science and batch processing to **handle data stored in a single platform**, unlocking an entirely new approach to analytics.
  - is the pre-requisite for Enterprise Hadoop as it provides the **resource management** and pluggable architecture for enabling a wide variety of data access methods to operate on data stored in Hadoop with predictable performance and service levels.

YARN - The Architectural Center of Enterprise Hadoop



MapReduce and YARN: background

- A **computer cluster** is a set of loosely or tightly connected **computers** that work together so that, in many respects, they can be viewed as a single system.
  - Therefore, cluster computing can be viewed as the techniques used to connect multiple computers together as a single system.

- **Cluster computing** faces several challenges:
  - how to store data persistently and keep it available **if nodes fail**
  - how to deal with node failures during a **long running computation**.
  - there is **network bottleneck** which delays the time of processing data.

MapReduce and YARN: background

- MapReduce offers a solution by **bring computation close to data** → minimizing data **movement**.
  - It is a simple programming model designed to process large volumes of data in parallel by **dividing the job into a set of independent tasks.**
- The biggest limitation with MapReduce programming is that map and reduce jobs are **not stateless**.
  - This means that **Reduce jobs have to wait for Map jobs to be completed first**. This limits maximum parallelism
- Therefore **YARN** was born as a generic resource management and distributed application framework.

MapReduce

- A MapReduce job splits a large data set into **independent chunks** and organizes them into **key, value pairs** for parallel processing.
  - This parallel processing **improves the speed and reliability** of the cluster.
- The **Map** function divides the input into **ranges** by the **InputFormat** and creates a **map task** for each range in the input. The **JobTracker** distributes those tasks to the **worker nodes**. The **output** of each map task is partitioned into a group of **key-value pairs for each reduce.**
  - map(key1,value) ➔ list<key2,value2>

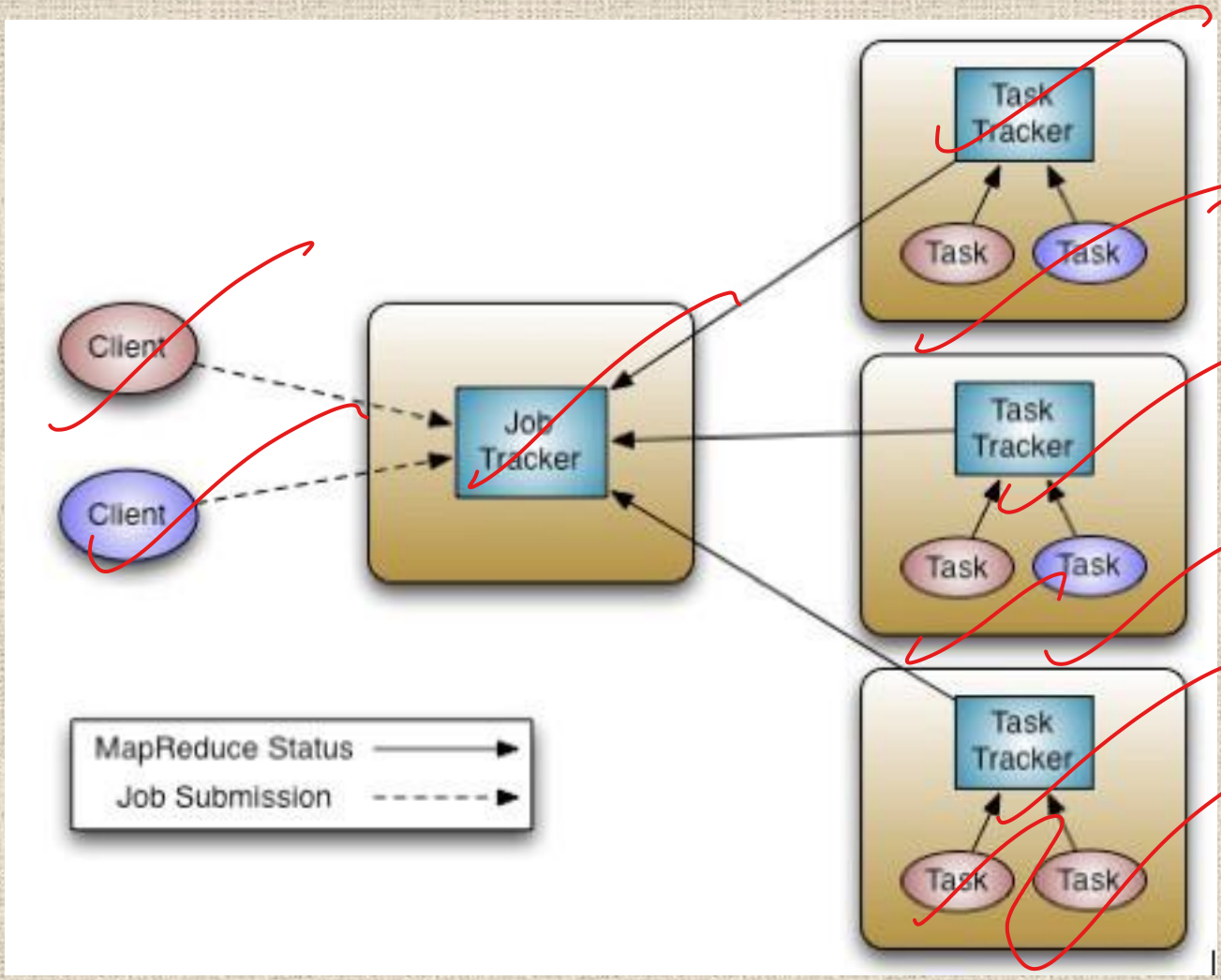
MapReduce

- The **Reduce** function then **collects the various results** and **combines** them to answer the **larger problem** that the master node needs to solve.
- Each reduce pulls the **relevant partition** from the machines where the maps executed, then writes its output back into HDFS.
  - Thus, the reduce is able to **collect the data from all of the maps** for the keys and **combine** them to solve the problem.
  - reduce(key2, list<value2>) -> list<value3>

MapReduce in Hadoop

- The current Apache Hadoop MapReduce System is composed of the **JobTracker (master)**, and **TaskTrackers (per-node slaves)**.
- The **JobTracker** is responsible for
  - resource management (managing the worker nodes i.e. TaskTrackers),
  - tracking resource consumption/availability
  - job life-cycle management (scheduling individual tasks of the job, tracking progress, providing fault-tolerance for tasks etc).
- The **TaskTracker** is responsible for
  - launch/teardown tasks on orders from the JobTracker
  - provide task-status information to the JobTracker periodically.

MapReduce in Hadoop



YARN – Yet Another Resource Negotiator

- To **split up** the functionalities of resource management and job scheduling/monitoring (initially by JobTracker) into separate daemons.
  - to have a global **ResourceManager** (*RM*), and

  - per-application **ApplicationMaster** (*AM*).

- An **application** is either a single job or a DAG (directed acyclic graph) of jobs.
- The ResourceManager and the NodeManager form the **data-computation framework**.

YARN

- The **ResourceManager** is the **ultimate authority** that arbitrates resources among all the applications in the system.

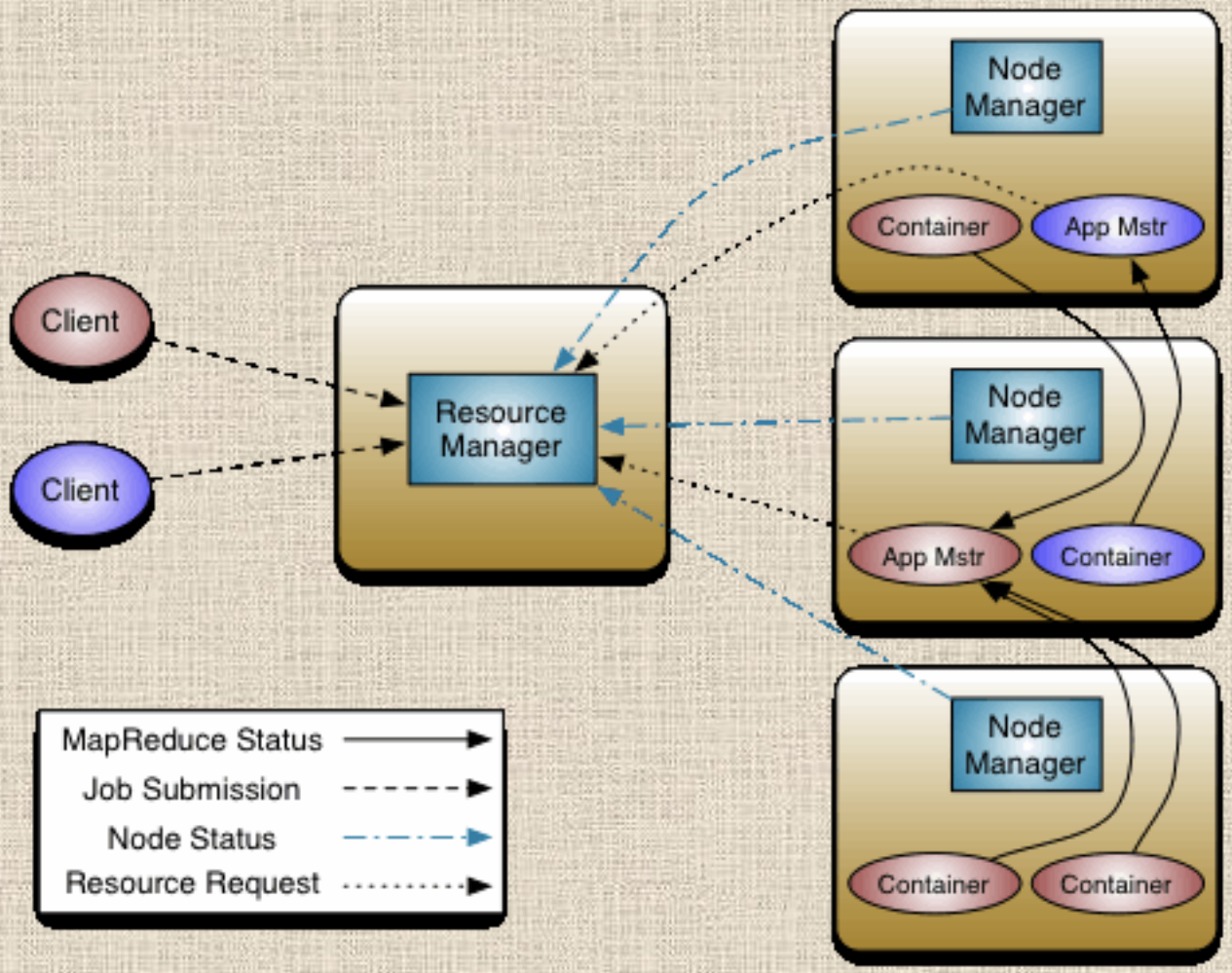
  - has a pluggable **Scheduler**, which is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc.
  - The Scheduler performs its scheduling function based on the *resource requirements* of the applications; it does so based on the abstract notion of a ***Resource Container*** which incorporates resource elements such as memory, CPU, disk, network etc.

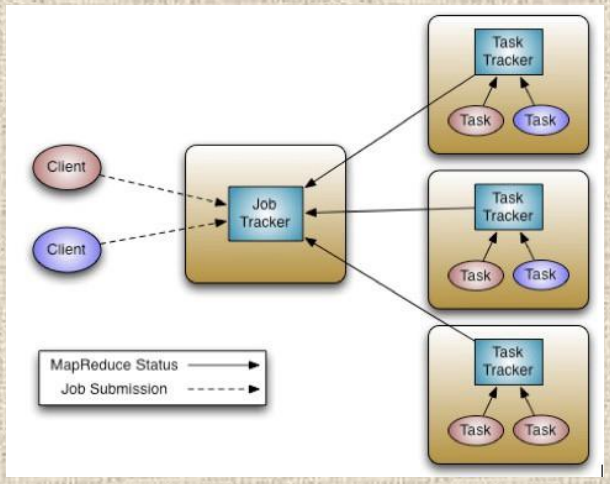
YARN

- The **NodeManager** is the **per-machine framework agent** who is responsible for **containers**, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

- The per-application **ApplicationMaster (AppMstr)** is, in effect, a **framework specific library** and is tasked with **negotiating resources** from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.
  - From the system perspective, the ApplicationMaster itself runs as a normal *container*.

YARN





MapReduce

Data Management -HDFS

- Store and process **vast quantities of data** in a storage layer that scales linearly.
- **Hadoop Distributed File System** (**HDFS**) is the core technology for the **efficient scale out storage layer**, and is designed to run across low-cost commodity hardware.
  - is a Java-based file system that provides scalable and reliable data storage that is designed to span large clusters of commodity servers.

HDFS – Hadoop Distributed File System

- HDFS:
  - is a scalable, fault-tolerant, distributed storage system that works closely with a wide variety of concurrent data access applications, **coordinated by YARN**.
  - will "just work" under a variety of physical and systemic circumstances.
- By distributing storage and computation **across many servers**, the combined storage resource **can grow linearly with demand** while **remaining economical** at every amount of storage.

HDFS

- An HDFS cluster is comprised of a **NameNode**, which manages the cluster metadata, and **DataNodes** that store the data.
- Files and directories are represented on the NameNode by **inodes**. Inodes record attributes like permissions, modification and access times, or namespace and disk space quotas.
- The file content is split into **large blocks (typically 128 megabytes)**, and each block of the file is independently replicated at multiple DataNodes.
- The blocks are stored on the local file system on the DataNodes.

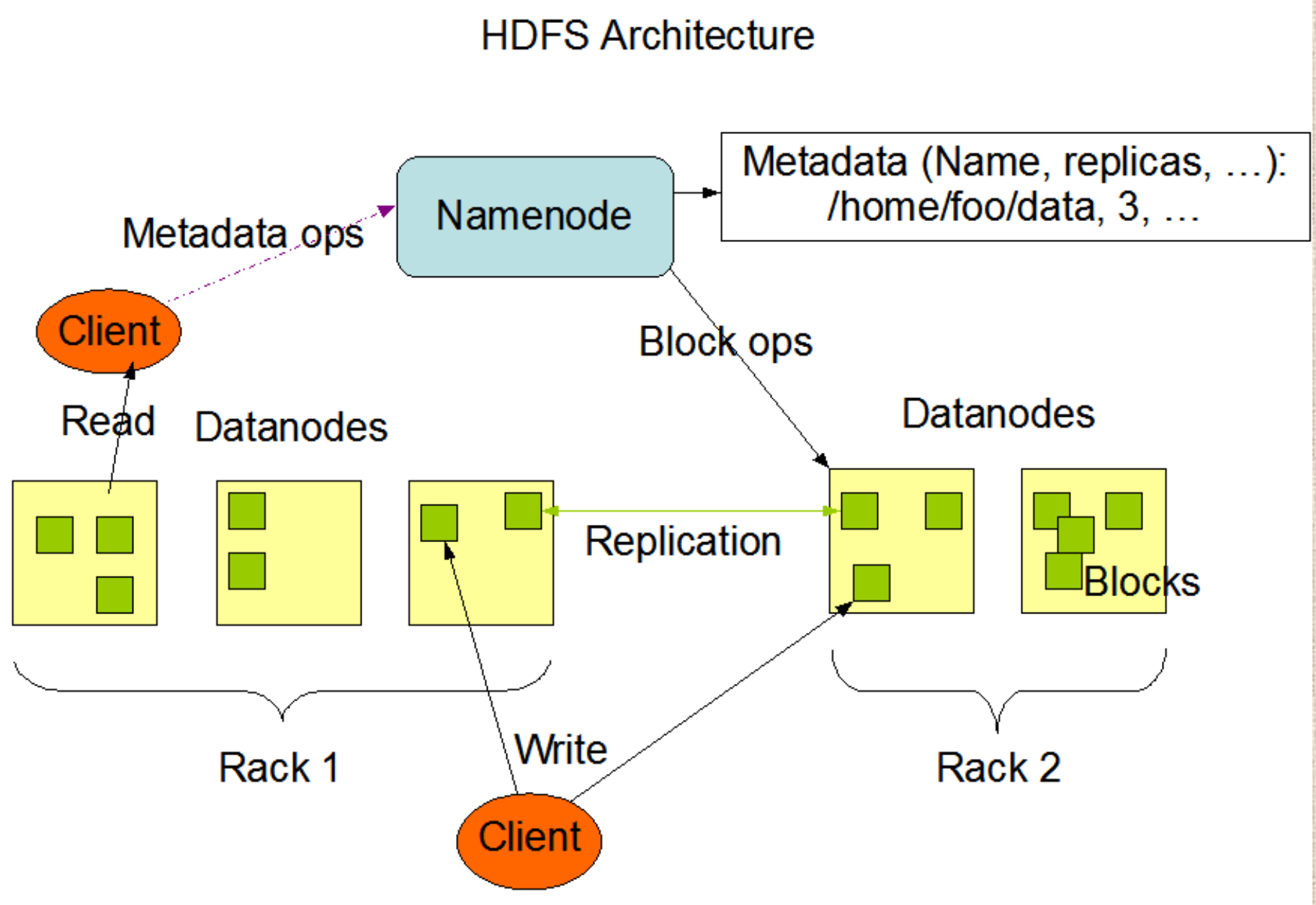
HDFS

- The **Namenode** actively monitors the number of replicas of a block.
  - When a replica of a block is lost due to a DataNode failure or disk failure, the NameNode creates another replica of the block.
  - The NameNode maintains the namespace tree and the mapping of blocks to DataNodes, holding the entire namespace image in RAM.
  - The NameNode **does not directly send requests to DataNodes**. It sends instructions to the DataNodes by **replying to heartbeats** sent by those DataNodes. The instructions include commands to:
    - replicate blocks to other nodes
    - remove local block replicas
    - re-register and send an immediate block report, or
    - shut down the node

HDFS cartoon

- https://hadoopknowledgebasebyabhi.blogspot.com/2015/03/hdfs-through-cartoon.html

- What can you summarize from the cartoon?
  - Writing data in HDFS cluster
  - Reading data in HDFS cluster
  - Fault tolerance in HDFS


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

HDFS architecture

- The file content is **split into large blocks** (typically 128 megabytes), and each block of the file is **independently replicated at multiple DataNodes**. The blocks are stored on the local file system on the DataNodes.
- The Namenode **actively monitors the number of replicas of a block**. When a **replica** of a block **is lost** due to a DataNode failure or disk failure, the NameNode **creates another replica of the block**. The **NameNode maintains the namespace tree and the mapping of blocks to DataNodes**, holding the entire namespace image in RAM.
- The NameNode does not directly send requests to DataNodes. It **sends instructions to the DataNodes by replying to heartbeats** sent by those DataNodes. The instructions include commands to:
  - replicate blocks to other nodes,
  - remove local block replicas,
  - re-register and send an immediate block report, or
  - shut down the node.

Data Access

- Interact with your data in a wide variety of ways – from **batch to real-time**.
  - **Apache Hive** – Built on the MapReduce framework, Hive is a data warehouse that **enables easy data summarization and ad-hoc queries via an SQL-like interface** for large datasets stored in HDFS.
  - **Apache Pig** – A platform for **processing and analyzing large data set**s. Pig consists of a high-level language (Pig Latin) for expressing data analysis programs paired with the MapReduce framework for processing these programs.

Data Access

- Interact with your data in a wide variety of ways – from **batch to real-time**.
  - **Apache Spark** – Spark is ideal for **in-memory data processing**. It allows data scientists to implement fast, iterative algorithms for advanced analytics such as clustering and classification of datasets.
  - **Apache HBase** – A **column-oriented NoSQL data storage** system that provides **random** real-time read/write access to big data for user applications.

Data Access

- **Others:**
  - Apache Storm
  - Apache Kafka
  - Apache HCatalog
  - Apache Slider
  - Apache Tez
  - Apache Solr
  - Apache Mahout
  - Apache Accumulo

Data Governance and Integration

- Quickly and easily load data, and manage according to policy.
  - Workflow Management
  - Apache Flume
  - Apache Sqoop

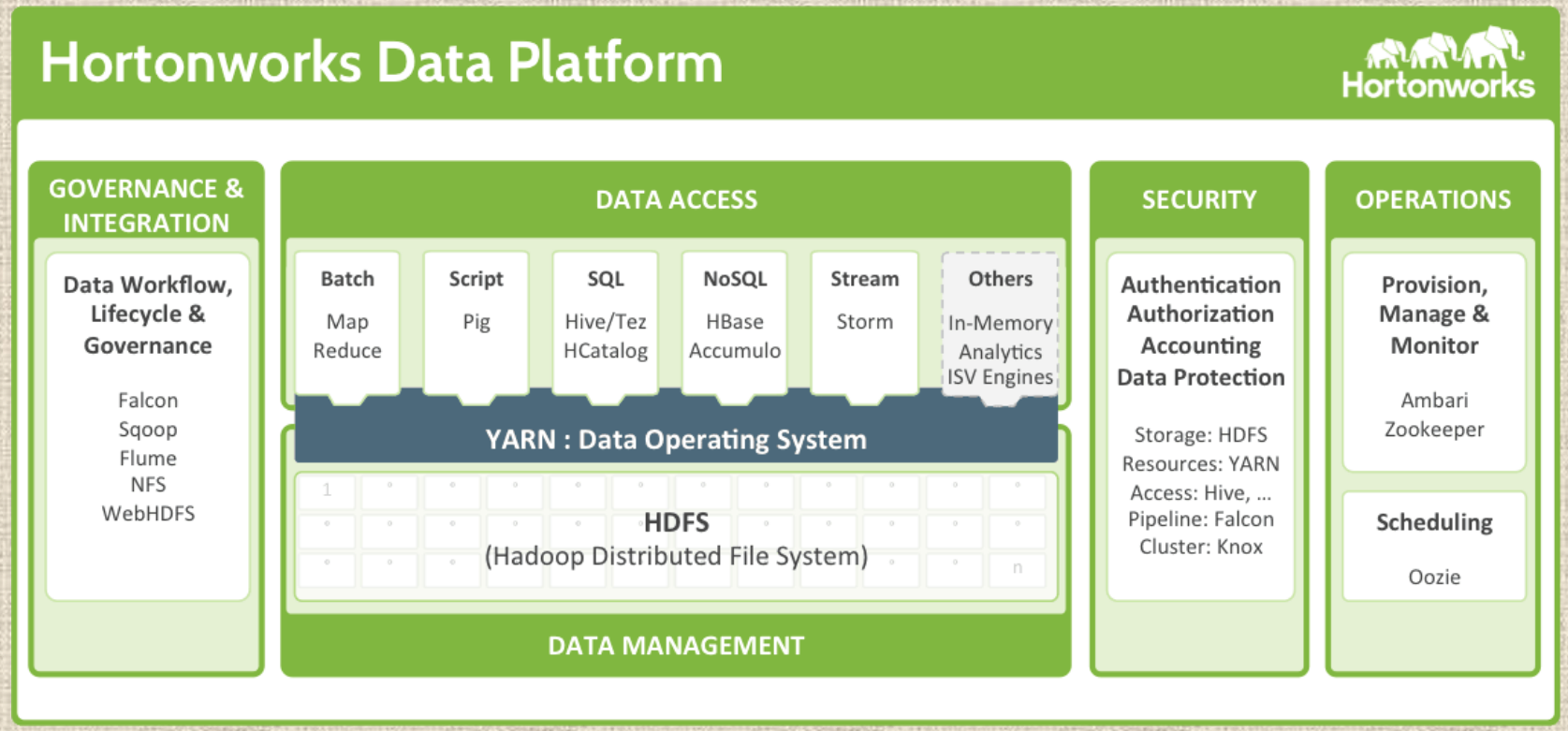
Security

- Address requirements of **Authentication, Authorization, Accounting and Data Protection.**
  - Apache Knox
  - Apache Ranger
- Security is provided at every layer of the Hadoop stack from HDFS and YARN to Hive and the other Data Access components on up through the entire perimeter of the cluster via Apache Knox.

Operations

- Provision, manage, monitor and operate Hadoop clusters at scale.
  - Apache Ambari
  - Apache Oozie
  - Apache ZooKeeper

Five pillars to Hadoop - Recap



(Source: https://www.tableau.com/es-es/solutions/hortonworks)

This Photo by Unknown Author is licensed under CC BY

Applications

- Apache Hadoop can be useful across a range of use cases spanning virtually every industry.
- It is becoming popular anywhere that you **need to store, process, and analyze large volumes of data**. Examples:
  - digital marketing automation,
  - fraud detection and prevention,
  - social network and relationship analysis,
  - predictive modeling for new drugs,
  - retail in-store behavior analysis, and
  - mobile device location-based marketing.

Data Pipeline

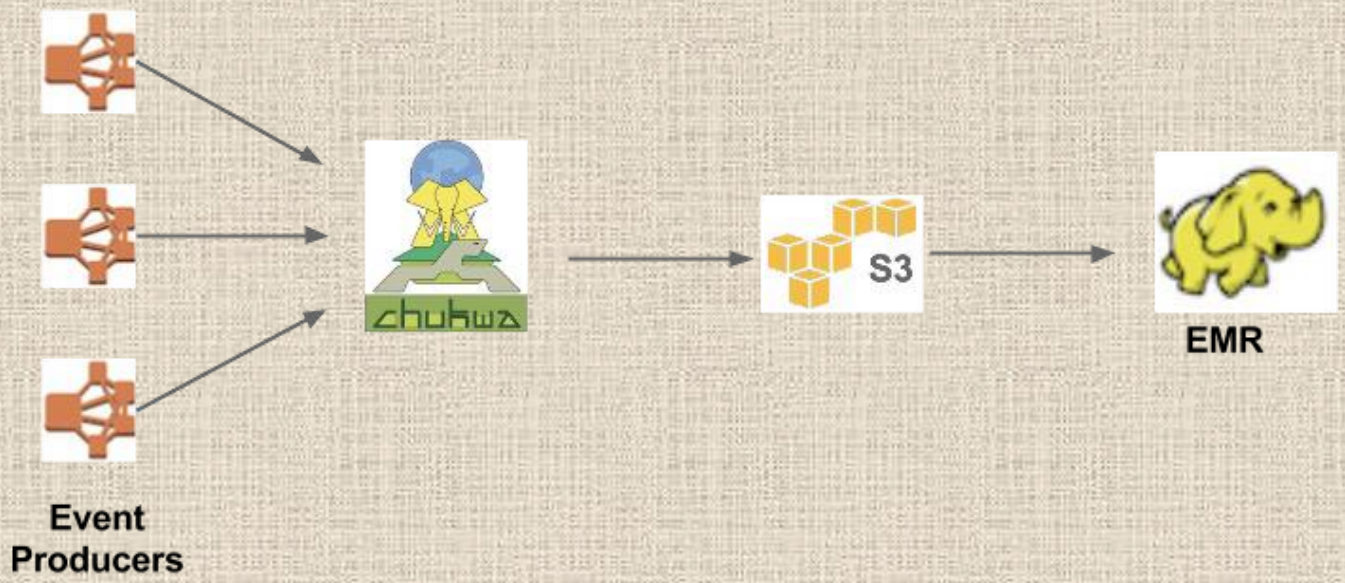
- Is the process of **structuring, processing, and transforming data in stages** regardless of what the source data form may be.
- to **collect, aggregate, process and move data at cloud scale**.

Example: Netflix Data Pipeline

- There are several hundred event streams flowing through the pipeline. For example:
  - Video viewing activities
  - UI activities
  - Error logs
  - Performance events
  - Troubleshooting & diagnostic events

Example: Netflix – V1.0 Chukwa pipeline

- To aggregate and upload events to **Hadoop/Hive** for batch processing.
- Chukwa collects events and writes them to S3 in Hadoop sequence file format.
- The Big Data Platform team further processes those S3 files and writes to Hive in Parquet format.
- End-to-end latency is up to 10 minutes. That is sufficient for batch jobs which usually scan data at daily or hourly frequency.

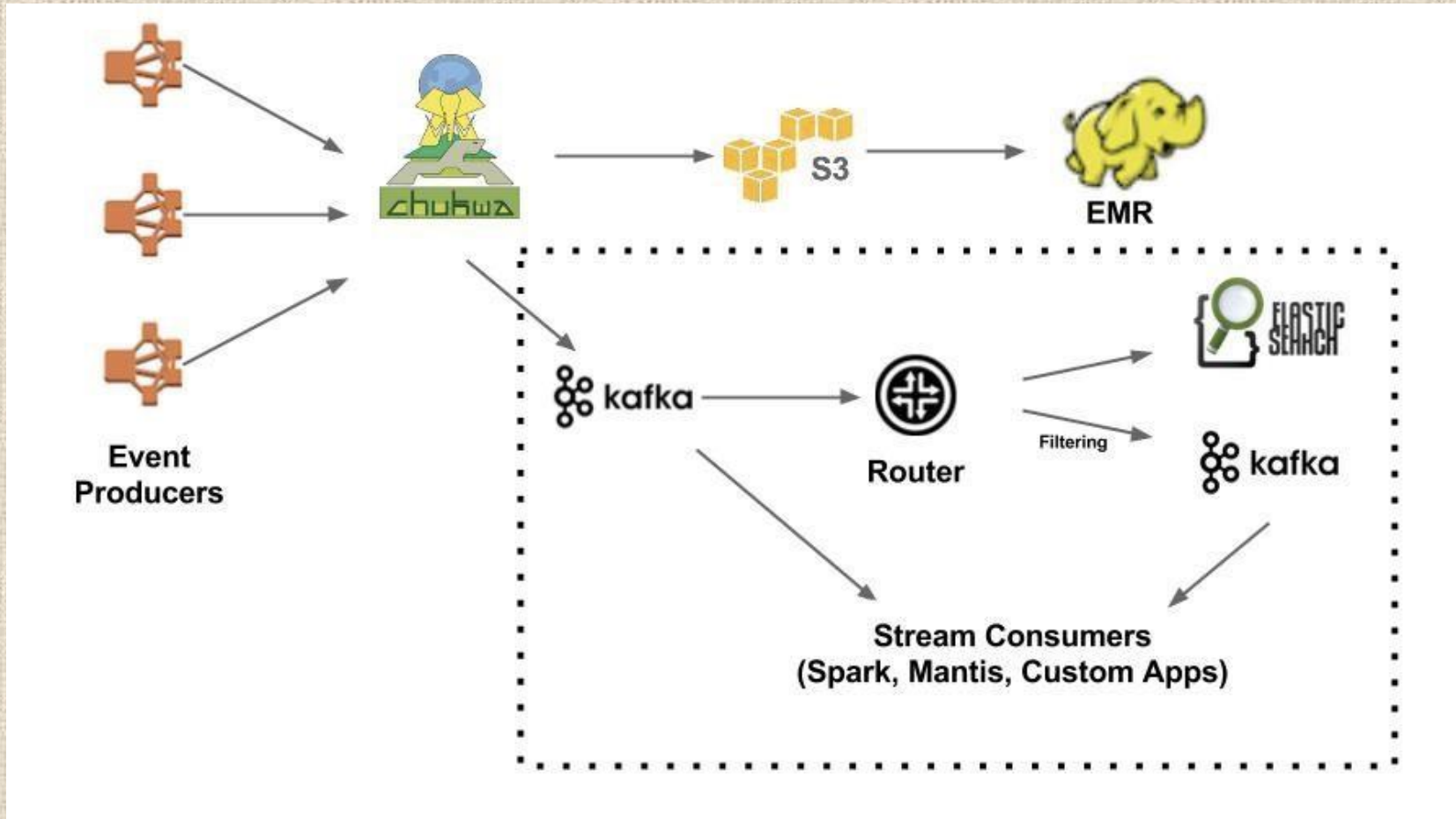


Source: https://medium.com/netflix-techblog/evolution-of-the-netflix-data-pipeline-da246ca36905

Example: Netflix – V1.5 Chukwa pipeline with real-time branch

- With the emergence of **Kafka and Elasticsearch** over the last couple of years, there has been a **growing demand** for **real-time analytics** in Netflix.
  - In addition to uploading events to S3/EMR, Chukwa can also **tee traffic to Kafka** (the front gate of real-time branch). In V1.5, approximately 30% of the events are branched to the real-time pipeline. The centerpiece of the real-time branch is the router. It is responsible for routing data from Kafka to the various sinks: Elasticsearch or secondary Kafka.
    - When Chukwa tees traffic to Kafka, it can deliver full or filtered streams. Sometimes, further filtering is applied on the Kafka streams written from Chukwa. That is why the router is needed to consume from one Kafka topic and produce to a different Kafka topic.
  - Once data is delivered to Kafka, it empowers users with real-time stream processing: **Mantis, Spark**, or custom applications.

Example: Netflix – V1.5 Chukwa pipeline with real-time branch

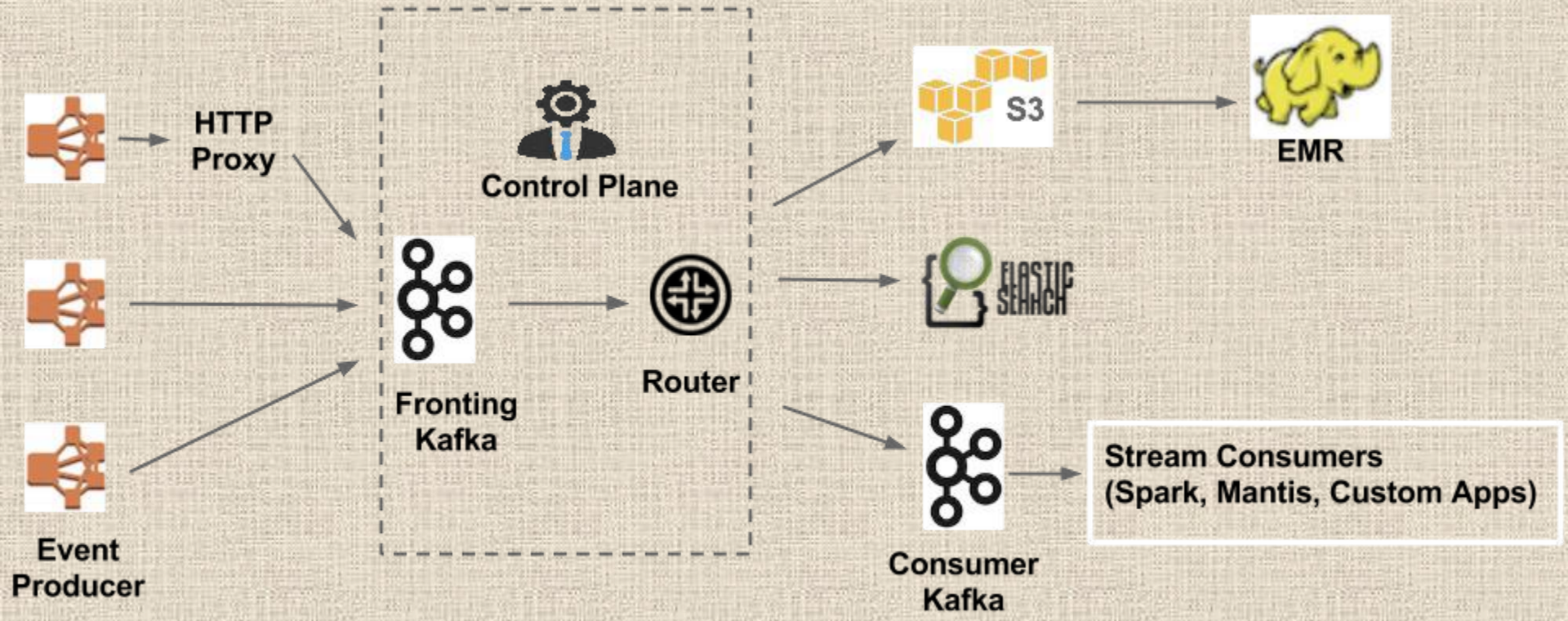


Source: https://medium.com/netflix-techblog/evolution-of-the-netflix-data-pipeline-da246ca36905

Example: Netflix – V2.0 Keystone pipeline (Kafka fronted)

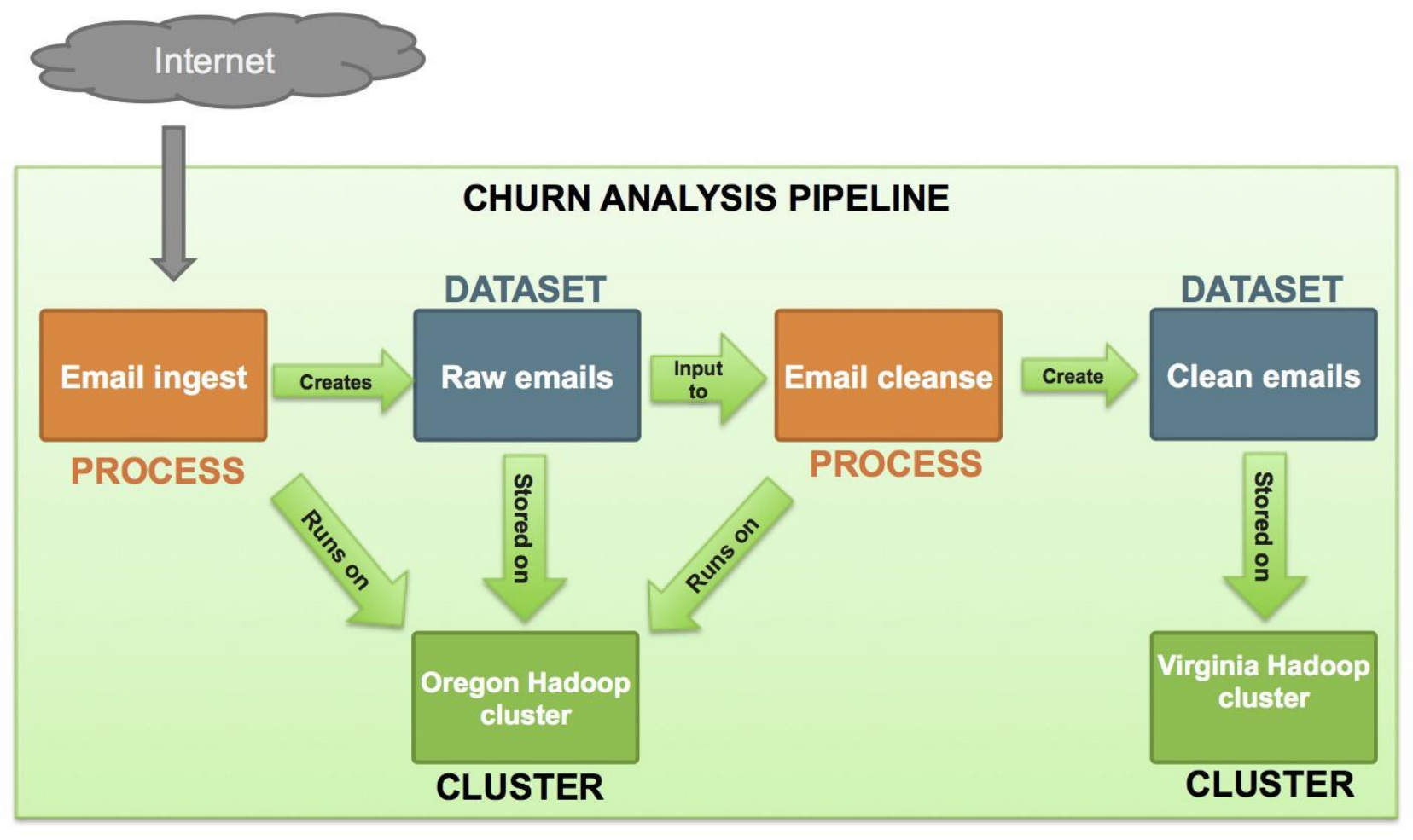
- In addition to the issues related to routing service, there are other motivations to revamp the data pipeline:
  - **Simplify the architecture**.
  - Kafka implements replication that **improves durability**, while Chukwa doesn't support replication.
  - Kafka has a vibrant community with strong momentum.
- There are three major components:
  - **Data Ingestion** — There are two ways for applications to ingest data: use our Java library and write to Kafka directly. send to an HTTP proxy which then writes to Kafka.
  - **Data Buffering** — Kafka serves as the replicated persistent message queue. It also helps absorb temporary outages from downstream sinks.
  - **Data Routing**—The routing service is responsible for moving data from fronting Kafka to various sinks: S3, Elasticsearch, and secondary Kafka.

Example: Netflix – V2.0 Keystone pipeline (Kafka fronted)



Source: https://medium.com/netflix-techblog/evolution-of-the-netflix-data-pipeline-da246ca36905

Another example: Customer Churn Analysis



WQD7007 Big Data Management

Big Data Concepts

Providing Structure to Unstructured Data

Previous lecture

- What is Hadoop
- Big data pipeline using Hadoop

Agenda

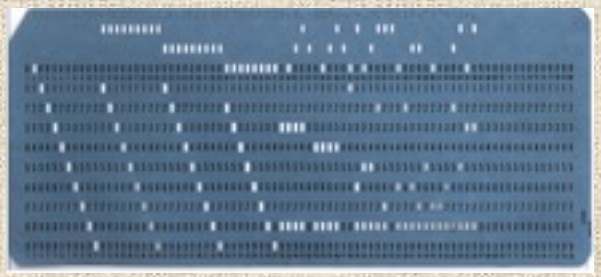
- Providing structure to unstructured data
  - Machine translation
  - Autocoding
  - Indexing
  - Term Extraction

Providing structure to unstructured data

- In the early days of computing, data was always highly structured.
  - E.g. **All data was divided into fields**
    - the fields had a fixed length, and the data entered into each field was constrained to a predetermined set of allowed values.
  - Data was entered into **punch cards**, with preconfigured rows and columns.
    - Depending on the intended use of the cards, **various entry and read-out methods were chosen to express binary data, numeric data, fixed-size text, or programming instructions**
    - For many **analytic purposes**, card-encoded data sets were analyzed without the assistance of a computer; all that was needed was a punch card sorter.

Providing structure to unstructured data

- Punch cards:



- Punch card sorters:

- In short, all data are expected to be structured data in early days of computing.

Providing structure to unstructured data

- Today, **most data entered by humans is unstructured**, in the form of free text.
  - E.g. e-mail messages, tweets, documents, and so on.

- Structured data has not disappeared, but it sits in the shadows cast by mountains of unstructured text.
  - Free text may be more interesting to read than punch cards, but punch cards was much easier to analyze.

Providing structure to unstructured data

- To get much informational value from free text, it is necessary **to impose some structure**. This may involve:
  - translating the text to a preferred language
  - parsing the text into sentences
  - extracting and normalizing the conceptual terms contained in the sentences
  - mapping terms to a standard nomenclature annotating the terms with codes from one or more standard nomenclatures
- All of these activities are difficult to do on a small scale and virtually impossible to do on a large scale
  - Every big data project that uses unstructured data must deal with these tasks to yield the best possible results with the resources available.

Machine Translation

- Unstructured data: data objects whose contents are **NOT** organized into arrays of attributes or values
  - Example: paragraph in a book
- Structured data?
  - Example: Spreadsheets, with data distributed in cells, marked by a row and column position
- **Without structure**, the contents of the data cannot be **sensibly collected and analyzed**.
  - Because Big Data is immense, the tasks of imposing structure on text must be **automated and fast**.

Machine Translation

- What is machine translation?
  - Translate text **from one language into another language**.
  - is one of the better known areas in which computational methods have been applied to free text.
- **Process**:
  1. extracting sentences from text
  2. parsing the words of the sentence into grammatical parts
  3. arranging the grammatical parts into an order that imposes logical sense on the sentence.

Machine Translation

- Next, each of the parts **can be translated by a dictionary** that finds **equivalent terms in a foreign language** to be reassembled by applying **grammatical positioning rules** appropriate for the **target language**.
  - Because this process uses the natural rules for sentence constructions in a foreign language, the process is often referred to as natural language machine translation.

- Example?

Machine Translation

- **Pros**: It all seems simple and straightforward if **proper look-up tables are developed**.

- **Cons**: Languages do not make much sense for computers.
  - Computers cannot find meaning in sentences **that have no meaning**.
  - humans find meaning in the English language because we **impose our own cultural background** onto the sentences we read, to create meaning where none exists.

Autocoding

- Coding, as used in the context of unstructured textual data, is the process of **tagging terms with an identifier code** that corresponds to a **synonymous term** listed in a standard nomenclature
  - Nomenclature: the devising or choosing of names for things, especially in a science or other discipline
  - Example: a medical nomenclature might contain the term *renal cell carcinoma*, a type of kidney cancer, attaching a unique identifier code for the term, such as "C9385000."

Autocoding

- There are about **50 recognized synonyms** for "*renal cell carcinoma*."
  - A few of these synonyms and near-synonyms are listed here to show that a single concept can be expressed many different ways, including
    - adenocarcinoma arising from kidney,
    - adenocarcinoma involving kidney,
    - cancer arising from kidney,
    - carcinoma of kidney,
    - Grawitz tumor,
    - Grawitz tumour,
    - hypernephroid tumor,
    - hypernephroma,
    - kidney adenocarcinoma,
    - renal adenocarcinoma,
    - renal cell carcinoma.
- All of these terms could be assigned the same identifier code, "C9385000."

Autocoding

- The process of coding a text document involves **finding all the terms** that belong to a **specific nomenclature** and **tagging** the term with the corresponding **identifier code**.

- Example, there may be a nomenclature of diseases, or makes and models of automobiles.
  - Some nomenclatures are ordered alphabetically.
  - Others are ordered by synonymy, wherein all synonyms and plesionyms are collected under a canonical (i.e., best or preferred) term.

Autocoding

- Purpose of Nomenclatures includes:
  - to enhance interoperability and integration,
  - to allow synonymous terms to be retrieved regardless of which specific synonym is entered as a query,
  - to support comprehensive analyses of textual data

- Traditionally, nomenclature coding has been **considered a specialized and highly detailed task** that is best accomplished by human beings.
  - Tagging documents with nomenclature codes is serious business. If the coding is flawed, the consequences can be dire.

Autocoding

- Example: In 2009, the Department of Veterans Affairs (VA) sent out hundreds of letters to veterans with the devastating news that they had contracted amyotrophic lateral sclerosis, also known as Lou Gehrig's disease, a fatal degenerative neurologic condition.
  - About 600 of the recipients did not, in fact, have the disease. The VA retracted these letters, attributing the confusion to a coding error.
  - Coding text is difficult. **Human coders are inconsistent, idiosyncratic, and prone to error.**

Autocoding

- When dealing with text in gigabyte and greater quantities, human coding is simply out of the question.
  - **There is not enough time, or money, or talent to manually code the textual data contained in** Big Data resources.
  - Computerized coding (i.e., autocoding) is the only practical solution.

- **Cons**: The best natural language autocoders are **pitifully slow.** The reason for the slowness relates to their algorithm
  - As a result, a good natural language autocoder parses text at about 1 kilobyte per second.

Autocoding

- The algorithm steps:
  1. parsing text into sentences;
  2. parsing sentences into grammatical units,
  3. rearranging the units of the sentence into grammatically permissible combinations,
  4. expanding the combinations based on stem forms of words,
  5. allowing for singularities and pluralities of words,
  6. matching the allowable variations against the terms listed in the nomenclature.
- This means that if an autocoder must parse and code **a terabyte of textual material**, it would require 1000 million seconds to execute, or about **30 years**.
- Big Data resources typically contain many terabytes of data; thus, natural language autocoding software is unsuitable for translating Big Data resources.

Autocoding

- This being the case, what good are they? (**Pros**)
- Natural language autocoders have value when they are employed at the time of **data entry**.
  - **Humans type sentences at a rate far less than 1 kilobyte per second**, and natural language autocoders can keep up with typists, inserting codes for terms, as they are typed.
    - They can operate much the same way as auto**correct, autospelling, look-ahead**, and other commonly available crutches intended to **improve or augment the output** of plodding human typists.
    - In cases where a variant term evades capture by the natural language algorithm (e.g. unrecognizable grammatical structure), the typist can supply the application with an equivalent (i.e., renal cell carcinoma = rcc) that can be stored by the application and applied against future inclusions of alternate forms.

Indexing

- A well-designed book index is a creative, literary work that **captures the content and intent of the book** and transforms it into a listing wherein related concepts, found scattered throughout the text, are **collected under common terms and keyed to their locations**.
  - to create a keyed encapsulation of concepts, subconcepts, and term relationships.

Indexing

- Big Data resources that lack a proper index **CANNOT** be utilized to their full potential. **Without an index, you never know what your queries are missing.**
  - it is the **relationship among data objects that are the keys to knowledge.** Data by itself, even in large quantities, tells only part of a story.
  - electronic indexes map concepts, classes, and terms to specific locations in the resource where data items are stored
  - An index imposes **order and simplicity** on the big data resource.

Indexing

- Indexing? Why not just use the ”find” function?

1. An index can be read, like a book, to acquire a **quick understanding of the contents and general organization** of the data resource.

2. When you do a “find” search in a query box, your search may come up empty if there is nothing in the text that matches your query.
   - This can be very frustrating if you know that the text covers the topic entered into the query box. **Indexes avoid the problem of fruitless searches.**
   - By browsing the index you can find the term you need, without foreknowledge of its exact wording within the text.

Indexing

3. Index searches are instantaneous, even when the Big Data resource is enormous.
   - Indexes are constructed to contain the results of the search of every included term, **obviating the need to repeat the computational task of searching** on indexed entries.

4. Indexes can be tied to a classification.
   - This permits the analyst to know the **relationships among different topics within the index** and within the text.

5. Many indexes are **cross-indexed**, providing relationships among index terms that might be extremely helpful to the data analyst.

Term Extraction

- Consider the following: "The diagnosis is chronic viral hepatitis."
  - This sentence contains two very **specific medical concepts**: "diagnosis" and "chronic viral hepatitis."
  - These two concepts are connected to form a meaningful statement with the words "the" and "is," and the sentence delimiter, "." "The," "diagnosis," "is," "chronic viral hepatitis," "."
  - **A term** can be defined as a **sequence** of one or more uncommon words that are demarcated (i.e., bounded on one side or another) by the occurrence of one or more common words, such as "is," "and," "with," "the."

Term Extraction

- If we had **a list of all the words that were considered common**, we could write a program that extracts all the concepts found in any text of any length.
  - The concept terms would consist of all **sequences of uncommon words that are uninterrupted by common words.**

Term Extraction

- An algorithm for extracting terms from a sentence follows:
  1. Read the first word of the sentence. If it is a common word, delete it. If it is an uncommon word, save it.
  2. Read the next word.
     1. If it is a common word, delete it and place the saved word (from the prior step, if the prior step saved a word) into our list of terms found in the text.
     2. If it is an uncommon word, append it to the word we saved in step one and save the two-word term.
     3. If it is a sentence delimiter, place any saved term into our list of terms and stop the program.
  3. Repeat step two.

"diagnosis" "chronic viral hepatitis"

Term Extraction

- This simple algorithm, is a fast and efficient method to build a collection of **index terms**.
  - To use the algorithm, you must prepare or find a list of common words appropriate to the information domain of your big data resource.
  - Example: To extract terms from the National Library of Medicine's citation resource (about 20 million collected journal articles), the following list of common words is used:
    - "about, again, all, almost, also, although, always, among, an, and, another, any, are, as, at, be, because, been, before, being, between, both, but, by, can, could, did, do, does, done, due, during, each, either, enough, especially, etc, for, found, from, further, had, has, have, having, here, how, however, i, if, in, into, is, it, its, itself, just, kg, km, made, mainly, make, may, mg, might, ml, mm, most, mostly, must, nearly, neither, no, nor, obtained, of, often, on, our, overall, perhaps, pmid, quite, rather, really, regarding, seem, seen, several, should, show, showed, shown, shows, significantly, since, so, some, such, than, that, the, their, theirs, them, then, there, therefore, these, they, this, those, through, thus, to, upon, use, used, using, various, very, was, we, were, what, when, which, while, with, within, without, would."
  - Such lists of common words are sometimes referred to as **"stop word lists" or "barrier word lists,"** as they demarcate the beginnings and endings of extraction terms.

Term Extraction

- Notice that the algorithm parses through text **sentence by sentence.**
  - This is a somewhat awkward method for a computer to follow, as **most programming languages automatically cut text from a file line by line** (i.e., breaking text at the newline terminator).
  - A computer program has no way of knowing where a sentence begins or ends, **unless the programmer finds sentences, as a program subroutine.**

Term Extraction

- There are many **strategies** for determining where one sentence stops and another begins.
  - The easiest method looks for the occurrence of a **sentence delimiter** immediately following a lowercase alphabetic letter, that precedes one or two space characters, that precede an uppercase alphabetic character.

Term Extraction

- Here is an example: **"I like pizza. Pizza likes me."**
  - Between the two sentences is the sequence **"a. P,"** which consists of a lowercase "a" followed by a period, followed by a space, followed by an uppercase "P".
  - This general pattern (lowercase, period, one or two spaces, uppercase) usually signifies a **sentence break**.
  - The routine fails with sentences that break at the end of a line or at the last sentence of a paragraph.

Concluding remarks

How to build a good index?

1. Should the index be devoted to a **particular knowledge domain?**

   - You may want to create an index of names of persons, an index of geographic locations, or an index of types of transactions.

   - Your choice depends on the intended uses of the Big Data resource.

Concluding remarks

How to build a good index?

1. Should the index be devoted to a **particular knowledge domain?**
2. Should the index be devoted to a **particular nomenclature?**

- A coded nomenclature might facilitate the construction of an index if synonymous index terms are attached to their shared nomenclature code.

Concluding remarks

How to build a good index?

1. Should the index be devoted to a **particular knowledge domain?**
2. Should the index be devoted to a **particular nomenclature?**
3. Should the index be **built upon a scaffold that consists of a classification?**

- For example, an index prepared for biologists might be keyed to the classification of living organisms.
- Gene data has been indexed to a gene ontology and used as a research tool.

Concluding remarks

How to build a good index?

1. Should the index be devoted to a **particular knowledge domain?**
2. Should the index be devoted to a **particular nomenclature?**
3. Should the index be **built upon a scaffold that consists of a classification?**
4. In the absence of a classification, might **proximity among terms** be included in the index?
   - Term associations leading to useful discoveries can sometimes be found by collecting the distances between indexed terms.
   - Terms that are proximate to one another (i.e., co-occurring terms) tend to have a relational correspondence.
   - For example, if "aniline dye industry" co-occurs often with the seemingly unrelated term "bladder cancer," then you might start to ask whether aniline dyes can cause bladder cancer.

Concluding remarks

How to build a good index?

1. Should the index be devoted to a **particular knowledge domain?**
2. Should the index be devoted to a **particular nomenclature?**
3. Should the index be **built upon a scaffold that consists of a classification?**
4. In the absence of a classification, might **proximity among terms** be included in the index?
5. Should **multiple indexes** be created?
   - Specialized indexes might be created for data analysts who have varied research agendas.

Concluding remarks

How to build a good index?

1. Should the index be devoted to a **particular knowledge domain?**
2. Should the index be devoted to a **particular nomenclature?**
3. Should the index be **built upon a scaffold that consists of a classification?**
4. In the absence of a classification, might **proximity among terms** be included in the index?
5. Should **multiple indexes** be created?
6. Should the index be **merged into another index**?
   - It is far easier to merge indexes than to merge Big Data resources.
   - It is worthwhile to note that the greatest value of Big Data comes from finding relationships among disparate collections of data.

WQD7007 Big Data Management

Big Data Concepts

Identification, de-identification, and re-identification

Ontologies and semantics

Identification, de-identification, and re-identification

Agenda

- Identification
  - Feature of an identifier system
  - Object identifier
    - Really bad identifier methods
    - Embedding Information in an Identifier: Not Recommended
  - One-way hashes
- De-identification
- Re-identification

Identification, de-identification and re-identification

- Data identification is certainly **the most underappreciated and least understood** Big Data issue.

- Measurements, annotations, properties, and classes of information have no informational meaning unless they are attached to an **identifier** that distinguishes one data object from all other data objects
  - It links together all of the information that has been or will be associated with the identified data object.
  - Example?

Identification, de-identification and re-identification

- Properties of identified information, including
  - uniqueness, exclusivity, completeness, authenticity, and harmonization

- Once data objects have been properly identified, they can be **de-identified** and, under some circumstances, **re-identified**
  - The ability to **de-identify** data objects confers enormous advantages when issues of **confidentiality, privacy, and intellectual property emerge**.
  - The ability to **re-identify** deidentified data objects is required for **error detection, error correction, and data validation**.

Identification, de-identification and re-identification

- A good information system is an identification system where:
  - Provide a way of **naming data objects** so that they can be **retrieved by their name**
  - Provide a way of **distinguishing each object from every other object** in the system.

- If data managers **properly** identified their data, they would be producing a collection of data objects with more informational value than many existing big data resources.

Features of an identifier system

- An **object identifier** is an **alphanumeric string** associated with the **object**.

- For many big data resources, the objects that are of greatest concern to data managers are **human beings**.
  - many Big Data resources are built to **store and retrieve** information about **individual humans**.
  - the data manager's pre-occupation with human identifiers relates to the paramount importance of establishing **human identity, with absolute certainty** (e.g., banking transactions, blood transfusions).

Features of an identifier system

One of the most important tasks for data managers is the **creation of a dependable identifier system**. The properties of a good identifier system are the following:

1. **Completeness**. Every unique object in the Big Data resource **must be assigned an identifier**.
2. **Uniqueness**. Each identifier is a **unique sequence**.
3. **Exclusivity**. Each identifier is assigned to a unique object, **and to no other object**.
4. **Authenticity**. The objects that receive identification must be **verified** as the objects that they are intended to be.
   - For example, if a young man walks into a bank and claims to be Richie, then the bank must ensure that he is, in fact, who he says he is.

Features of an Identifier System

5. **Aggregation**. The big data resource must have a mechanism to aggregate all of the data that is **properly associated with the identifier**
6. **Permanence**. The identifiers and the associated data must be **permanent**.
   - In the case of a hospital system, when the patient returns to the hospital after 30 years of absence, **the record system must be able to access his identifier and aggregate his data.**
   - Even when a patient dies, the patient's identifier must not perish.
7. **Reconciliation**. There should be a mechanism whereby the data associated with a unique, identified object in one big data resource **can be merged** with the data held in another resource, for the same unique object.

Features of an Identifier System

8. **Immutability**. In addition to being permanent (i.e., never destroyed or lost), the identifier **must never change**.
9. **Security**. The identifier system is vulnerable to malicious attack.
   - A Big Data resource with an identifier **system can be irreversibly corrupted** if the identifiers are modified.
10. **Documentation and quality assurance**. A system should be in place **to find and correct errors in the patient identifier system.**
    - Every problem and every corrective action taken must be documented and reviewed.

Object identifiers

- For computer scientists, a data object **is a holder for data values** (the so-called encapsulated data), and become **descriptors of the data** based on properties of the holder (i.e., the class of objects to which the instance belongs).
- Uniqueness is achieved when the **data object is permanently bound to its own identifier sequence**. Unique objects have three properties:
  1. An unique object can be distinguished from all other unique objects.
  2. An unique object cannot be distinguished from itself.
  3. Uniqueness may apply to collections of objects (i.e., a class of instances can be unique).

Object identifiers

- **Object identifier:**
  - **Individual objects within the resource are provided with a registry number and a suffix sequence, appended locally.**
  - **Life Science Identifiers** serve as a typical example of a registered identifier. Every LSID is composed of the following five parts: Network Identifier, root DNS name of the issuing authority, name chosen by the issuing authority, a unique object identifier assigned locally, and an optional revision identifier for versioning information.
    - Example 1: **urn:lsid:pdb.org:1AFT:1**. This identifies the first version of the 1AFT protein in the Protein Data Bank
    - Example 2: **urn:lsid:ncbi.nlm.nig.gov:GenBank:T48601:2**. This refers to the second version of an entry in GenBank.

Object identifiers

- In some cases, the registry **does not provide the full identifier** for data objects. The registry may provide a **general identifier sequence** that will apply to every data object in the resource.

- An **object identifier (OID)** is a hierarchy of identifier prefixes. Successive numbers in the prefix identify the **descending order of the hierarchy.**

Object identifiers

- Example: Hospitals use an OID system for identifying images—part of the DICOM (Digital Imaging and Communications in Medicine) image standard.
  - There is a **prefix** consisting of a permanent, registered code for the institution and the department and a **suffix** consisting of a number generated for an image, as it is created.
- A hospital may assign consecutive numbers to its images, appending these numbers to an OID **that is unique for the institution and the department within the institution**.
  - For example, the first image created with a computed tomography (CT) scanner might be assigned an identifier consisting of the OID (the assigned code for institution and department) followed by a separator such as a hyphen, followed by “1”.

Object identifiers

- In a worst-case scenario, different instruments may assign consecutive numbers to images, independently of one another.
  - This means that **the CT scanner in room A may be creating the same identifier (OID + image number) as the CT scanner in room B for images on different patients.** This problem could be remedied by constraining each CT scanner **to avoid using numbers assigned by any other CT scanner.** This remedy can be defeated if there is a glitch anywhere in the system that accounts for image assignments (e.g., if the counters are reset, broken, replaced, or simply ignored).
- Other problem:
  - the image service is assigned to another department in the institution, when departments merge, or when institutions merge. The old records in both of the merging institutions **will be assigned the same prefix and will contain replicate.**

Really bad identifier methods

- <u>Names are poor identifiers.</u>
  - Aside from the obvious fact that they are **not unique** (e.g., surnames such as Smith, Zhang, Garcia, Lo, and given names such as John and Susan), **a single name can have many different representations.**
  - The sources for these variations are many. Here is a partial listing:
    - Modifiers to the surname, accents that may or may not be transcribed onto records, special typographic characters, multiple "middle names" for an individual that may not be transcribed onto records, latinized and other versions of a single name, hyphenated names that are confused with first and middle names and others

Embedding Information in an Identifier: Not Recommended

- Most identifiers are not purely random numbers—they usually contain some embedded information that can be **interpreted by anyone familiar with the identification system**.
  - For example, they may embed the **first three letters of the individual's family name in the identifier**. Likewise, the last two digits of the birth year are commonly embedded in many types of identifiers. Such information is usually included as a crude "honesty" check by people "in the know."
  - For instance, the nine digits of a social security number are divided into an area code (first three digits), a group number (the next two digits), followed by a serial number (last four digits). People with expertise in the social security numbering system can pry considerable information from a social security number and can determine whether certain numbers are bogus based on the presence of excluded subsequences.

Embedding Information in an Identifier: Not Recommended

- Unimpressed? Consider this scenario:
  - You know that a prominent member of the President's staff had visited a Washington, DC, hospital on February 15, 2005, for the purpose of having a liver biopsy. You would like to know the results of that biopsy. You go to a Web site that lists the **de-identified** pathology records for the hospital for the years 2000 to 2010. Though **no personal identifiers** are included in these public records, the individual records are sorted by **accession** numbers. Using the aforementioned strategy, you collect all of the surgical biopsies performed **on or about February 15, 2005**. Of these biopsies, only **three are liver biopsies**. Of these three biopsies, only one was performed on a person whose **gender and age** matched the President's staff member. The report provides the diagnosis. You managed to discover some very private information without access to any personal identifiers.
- The alphanumeric character string composing the identifier **should not expose the patient's identity**.

One-way Hashes

- A one-way hash is an algorithm that **transforms a string into another string** in such a way that the **original string cannot be calculated by operations on the hash value** (hence the term "one-way" hash). Popular one-way hash algorithms are MD5 and Standard Hash Algorithm.
  - A one-way hash value can be calculated for any character string, including a person's name, a document, or even another one-way hash. For a given input string, the resultant one-way hash will always be the same.
  - MD5 example:
    - Jules Berman → Ri0oaVTIAilwnS8tnvKhfA
    - "Whatever" → n2YtKKG6E4MyEZvUKyGWrw
    - Whatever → OkXaDVQFYjwkQtMOC8dpOQ
    - jules berman → SlnuYpmyn8VXLsxBWwO57Q
    - Jules J. Berman → i74wZ/CsIbxt3goH2aCStA
    - Jules J Berman → yZQfJmAf4dIYO6Bd0qGZ7g
    - Jules Berman → Ri0oaVTIAilwnS8tnvKhfA

One-way Hashes

- One-way hash values can substitute for identifiers in individual data records. This permits Big Data resources to accumulate data over time to a specific record, even when the record is **de-identified**.
  - A record identifier serves as the input value for a one way hash.
  - The primary identifier for the record is now a **one-way hash sequence**.
  - The data manager of the resource, looking at such a record, **cannot determine** the individual associated with the record because the original identifier has been replaced with an unfamiliar sequence.

One-way Hashes

- Implementation of one-way hashes carries certain practical problems:
  - If anyone happens to have a **complete listing** of all of the original identifiers, then it would be a simple matter to perform one-way hashes on every listed identifier. This would produce a **look-up table** that can match de-identified records back to the original identifier, a strategy known as a **dictionary attack**.
  - For de-identification to work, the original identifier sequences **must be kept secret**.

Brute Force

TIME IT TAKES A HACKER TO BRUTE FORCE YOUR PASSWORD

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 tn years	7qd years

HIVE SYSTEMS

-Data sourced from HowSecureismyPassword.net

De-identification

- De-identification is the process of **stripping information from a data record that might link the record to the public name of the record's subject.**
  - In the case of a patient record, this would involve stripping any information from the record that would enable someone to connect the record to the name of the patient.
    - For example: **patient's name,** patient's address (which could be linked to the name), the patient's date of birth (which narrows down the set of individuals to whom the data record might pertain), and the patient's social security number.
  - It is important to clarify that deidentification is **not achieved by removing an identifier from a data object**.
    - identifiers can be **substituted** with a one-way hash value, thus preserving the uniqueness of the record.

De-identification

- De-identification involves removing information contained in the data object that reveals something about the **publicly known name** of the data object.
  - This kind of information is often referred to as **identifying information**, or "**name-linking information**."
  - It may seem counterintuitive, but there **is very little difference between an identifier and a de-identifier**; under certain conditions the two concepts are equivalent.

De-identification

- Here is how a dual identification/de-identification system might work:
  - Collect data on unique object: “Joe Ferguson’s bank account contains $100.”
  - Assign a unique identifier: “Joe Ferguson’s bank account is 754003894713.”
  - Substitute name of object with its assigned unique identifier: “754003894713 contains $100.”
  - Consistently use the identifier with data.
  - Do not let anyone know that Joe Ferguson owns account “754003894713.”

De-identification

- The dual use of an identifier/de-identifier is a tried-and-true technique. Swiss bank accounts are essentially unique numbers (**identifiers**) assigned to a person. You access the bank account by producing the identifier number. The identifier number does not provide information about the identity of the bank account holder (**de-identifier**).

- The purpose of an identifier is to tell you that whenever the **identifier** is encountered, it **refers to the same unique object**, and whenever two different identifiers are encountered, they refer to different objects. The identifier, **by itself, contains no information that links the data object to its public name**.

De-identification

- At this point, you might be asking yourself the following question: "**What gives the data manager the right to distribute parts of a confidential record, even if it happens to be de-identified**?"
  - The act of de-identification renders the data harmless by transforming information about a person or data object into information about **nothing in particular**. Because the use of de-identified data poses no harm to human subjects, U.S. regulations allow the **unrestricted use of such data for research purposes**. Other countries have similar provisions.

Re-identification

- For scientists, deidentification serves two purposes:

1. **To protect the confidentiality and the privacy of the individual** (when the data concerns a particular human subject)

2. **To remove information that might bias the experiment** (e.g., to blind the experimentalist to patient identities)

- Because confidentiality and privacy concerns always apply to human subject data and because issues of experimental bias always apply when analyzing data, it would seem imperative that de-identification should be an irreversible process (i.e., the names of the subjects and samples should be held a secret, forever).

Re-identification

- Scientific integrity does not always accommodate **irreversible de-identification.**
  - On occasion, experimental samples are mixed up; samples thought to come from a certain individual, tissue, record, or account may come from another source.
  - Sometimes major findings in science need to be **retracted** when a sample mix-up has been shown to occur. When samples are submitted, without mix-up, the data is sometimes collected improperly.
    - For example, reversing electrodes on an electrocardiogram may yield spurious and misleading results. Sometimes data is purposefully fabricated and otherwise corrupted to suit the personal agendas of dishonest scientists.
    - When data errors occur, regardless of reason, it is important to retract the publications.

Re-identification

- To preserve scientific integrity, it is sometimes necessary to discover the identity of de-identified records. In some cases, de-identification stops the data analyst from helping individuals whose confidentiality is being protected.
  - Imagine you are conducting an analysis on a collection of deidentified data and **you find patients with a genetic marker for a disease that is curable**, if treated at an early stage.
  - De-identified records can, **under strictly controlled circumstances**, be re-identified. Re-identification is typically achieved by entrusting a third party with a confidential list that maps individuals to their deidentified records.

Concluding remarks

- Identification issues are often ignored by Big Data managers who are accustomed to working on small data projects.
  - It is worthwhile to repeat the most important ideas described in this chapter, many of which are counterintuitive and strange to those whose lives are spent outside the confusing realm of Big Data.

1. All big data resources can be imagined as an **identifier system for data objects and data-related events** (i.e., timed transactions). The data in a big data resource can be imagined as character sequences that are attached to identifiers.

2. Without an adequate identification system, a big data resource has **no value**. The data within the resource **cannot be trusted**.

Concluding remarks

3. An identifier is a **unique alphanumeric sequence** assigned to a data object.

4. A data object is a collection of data that contains **self-describing information**, and one or more data values. Data objects should be associated with a unique identifier.

5. De-identification is the process of **stripping information from a data record** that might link the record to the public name of the record's subject.

6. De-identification should not be confused with the act of stripping a record of an identifier. A de-identified record must have an associated identifier, just as an identified data record must have an identifier.

Concluding remarks

7. Where there is no identification, there can be no de-identification and no re-identification.

8. Re-identification is the **assignment of the public name associated with a data record to the de-identified record**.
   - Re-identification is sometimes necessary to verify the contents of a record or to provide information that is necessary for the well-being of the subject of a deidentified data record.
   - Reidentification always requires approval and oversight.

9. When a de-identified data set contains **no unique records** (i.e., every record has one or more additional records from which it cannot be distinguished, aside from its assigned identifier sequence), then it becomes impossible to uncover a de-identified record's public name.

Concluding remarks

Aspect	Identification	De-identification	Re-identification
Purpose	Recognize unique individuals/objects	Anonymize individuals/ objects for privacy	Link anonymized data back to individuals/ objects
Data Features Used	Unique features (e.g., face, voice, biometrics)	Masked or removed identifiable details	Residual patterns or re-linking unique characteristics
Application	Personalization, security access	Privacy-preserving data sharing	Testing privacy robustness and risk assessment
Privacy Concern	High risk if misused	Lower risk if effective	High risk, may lead to privacy breach
Example Techniques	Face/voice recognition models	Data masking, pseudonymization, aggregation	Cross-referencing residual data for re-identification

WQD7007 Big Data Management

Big Data Concepts

Identification, de-identification, and re-identification

Ontologies and semantics

PART 2

Ontologies and Semantics

Agenda

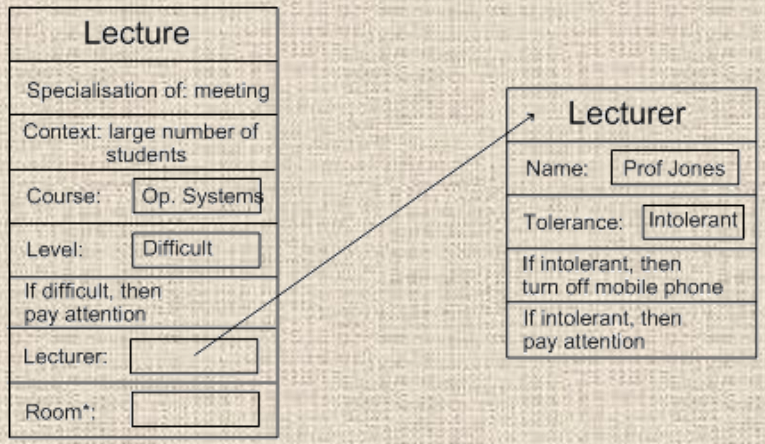
- Classifications, The Simplest Of Ontologies
- Ontologies, classes with multiple parents
- Choosing a Class Model
- Introduction to Resource Description Framework Schema
- Common Pitfalls in Ontology Development

Ontologies and Semantics

- Information has **limited value** unless it can take its place within our **general understanding** of the world.
  - When a financial analyst learns that the **price of a stock has suddenly dropped**, he cannot help but wonder if the drop of a single stock reflects **conditions in other stocks** in the **same** industry.
  - If so, the analyst may check to ensure that **other industries** are following a downward trend.
  - He may wonder whether the downward trend represents **a shift in the national or global economies**.
- In every case, the analyst is asking a **variation** on a single question: **"How does this thing relate to that thing?"**

Ontologies and Semantics

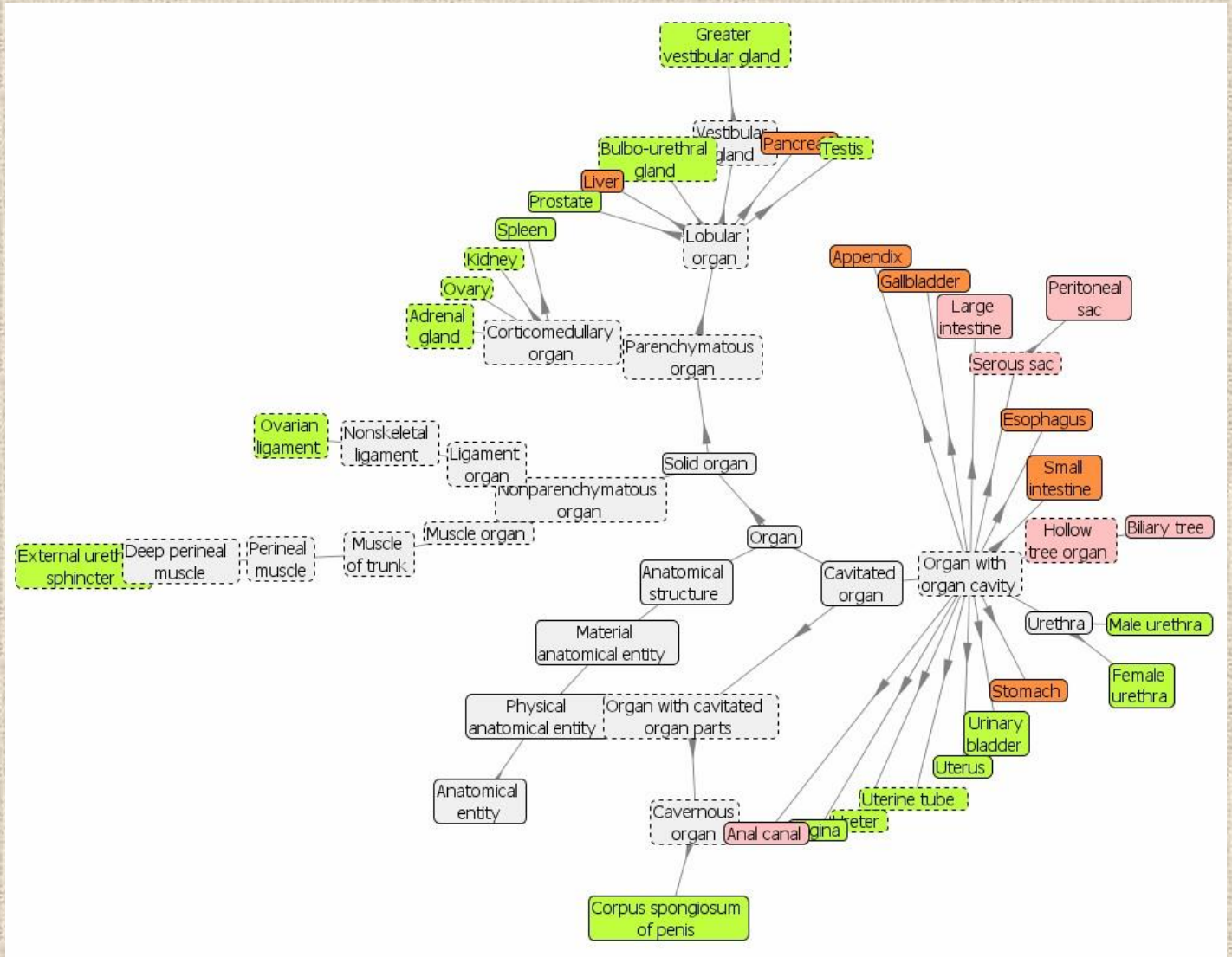
- Big data resources are **complex**.
  - When data is simply stored in a database, without any general principles of organization, it is impossible to discover the **relationships** among the data objects.
  - To be useful, the information in a big data resource must be **divided into classes of data**. Each data object within a class **shares a set of properties** chosen to enhance our ability to **relate one piece of data with another**.



Ontologies

- **Ontologies** are formal systems that **assign data objects to classes** and **that relate classes to other classes.**
  - When the data within a big data resource is classified **within an ontology**, data analysts can determine whether **observations on a single object will apply to other objects in the same class**.
  - Similarly, data analysts can begin to ask whether observations that **hold true for a class of objects** will **relate to other classes of objects**.
- Ontologies help to determine how things relate to other things.

Ontology example



Classifications, The Simplest Of Ontologies

- The human brain is **constantly processing visual and other sensory information collected from the environment**.
  - When we walk down the street, we see images of concrete and asphalt and millions of blades of grass, birds, dogs, other persons, and so on.
  - Every step we take conveys a new world of sensory input.

- **How can we process it all?**

Classifications, The Simplest Of Ontologies

- The mathematician and philosopher Karl Pearson (1857–1936) has likened the human mind to a "**sorting machine**."
  - We take a stream of sensory information, **sort it into a set of objects**, and then assign the individual objects to **general classes**.
  - The green stuff on the ground is classified as "grass," and the grass is sub-classified under some larger grouping, such as "plants."
  - A flat stretch of asphalt and concrete may be classified as a "road," and the road might be sub-classified under "man-made constructions."

Classifications, The Simplest Of Ontologies

- If we lacked a culturally determined classification of objects for our world, we would be overwhelmed by sensory input, and we would have **no way to remember what we see and no way to draw general inferences about anything**.
  - Simply put, without our ability to classify, we would not be human.

Classifications, The Simplest Of Ontologies

- Every culture has some particular way to impose a uniform way of perceiving the environment.
  - In English-speaking cultures, the term "**hat**" denotes a universally recognized object. Hats may be composed of many **different types of materials** and they may vary greatly in **size, weight, and shape**.
  - Nonetheless, we can almost always identify a hat when we see one, and we can distinguish a hat from all other types of objects. A hat is classified as a hat because it has a **class relationship; all hats are items of clothing that fit over the head.**

- Likewise, all **biological** classifications are built **by relationships, not by similarities**.

Classifications, The Simplest Of Ontologies

- Aristotle was one of the first experts in classification. **His greatest insight came when he correctly identified a dolphin as a mammal.**
  - Through observation, he knew that a large group of animals was distinguished by a **gestational period** in which a developing embryo is nourished by a placenta, and the offspring are delivered into the world as formed but small versions of the adult animals (i.e., not as eggs or larvae), and the newborn animals feed from milk excreted from nipples, overlying specialized glandular organs (mammae).
  - Aristotle knew that **these features, characteristic of mammals, were absent in all other types of animals.**

Classifications, The Simplest Of Ontologies

- He also knew that **dolphins had all these features; fish did not**. He correctly reasoned that dolphins were a type of mammal, not a type of fish.

- Aristotle was ridiculed by his contemporaries for whom it was obvious that dolphins were a type of fish.

- Unlike Aristotle, they based their classification on **similarities**, not on **relationships**.

Classifications, The Simplest Of Ontologies

- A classification is a very simple form of ontology, in which **each class is limited to one parent class**.
- To build a classification, the ontologist must do the following:
  1. define classes (i.e., **find the properties that define a class** and extend to the subclasses of the class),
  2. **assign instances** to classes,
  3. position classes within the **hierarchy**, and
  4. **test and validate** all of the above.

Classifications, The Simplest Of Ontologies

The constructed classification becomes a **hierarchy of data objects** conforming to a set of principles:

1. The classes (groups with members) of the hierarchy have **a set of properties or rules that extend to every member of the class and to all of the subclasses of the class**, to the exclusion of unrelated classes.
   - A subclass is itself a type of class wherein the members have the defining class properties of the parent class plus some additional property(ies) specific to the subclass.
2. In a **hierarchical classification**, **each subclass may have no more than one parent class**. The root (top) class has no parent class.
   - The biological classification of living organisms is a hierarchical classification.

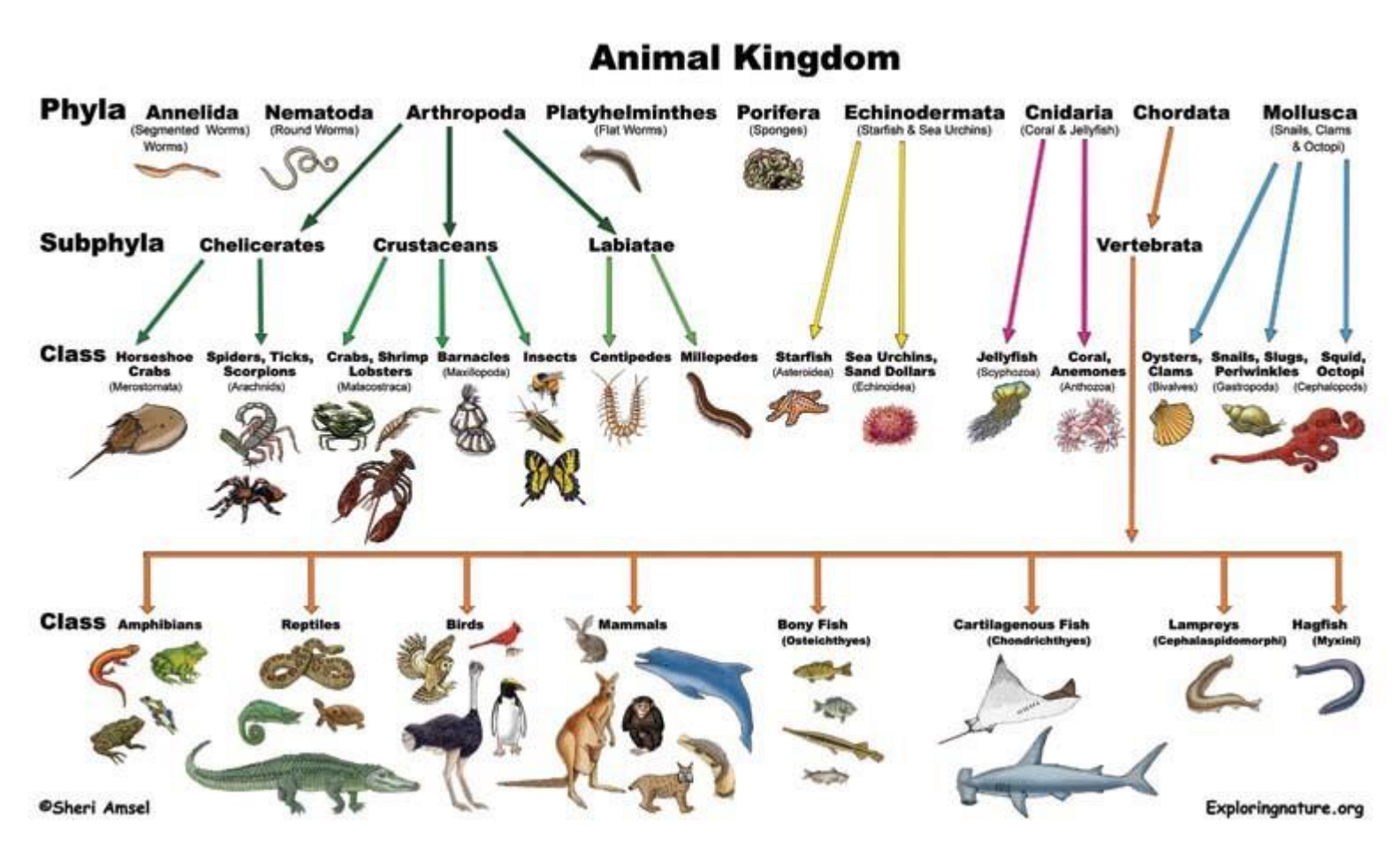
Classifications, The Simplest Of Ontologies

3. At the **bottom** of the hierarchy is the **class instance**.
   - For example, the book ”Principles of Big Data” is an instance of the class of **objects** known as “books.”
4. Every **instance** belongs to **exactly one class**.
5. Instances and classes **do not change their positions** in the classification.
   - For example, a horse never transforms into a sheep and a book never transforms into a harpsichord.
6. The **members** of classes **may be highly similar** to one another, but their similarities **result from their membership** in the same class (i.e., conforming to **class properties**), and not the other way around (i.e., **similarity alone cannot define class inclusion**).

Classifications, The Simplest Of Ontologies

- A **taxonomy** is a classification with the instances "filled in".
  - This means that for each class in a taxonomy, all the **known instances** (i.e., member objects) are **explicitly listed**. For the taxonomy of living organisms, the instances are named species.
  - Currently, there are several million named species of living organisms, and each of these several million species is listed under the name of some class included in the full classification.
  - Classifications drive down the complexity of their data domain because every instance in the domain is assigned to a single class and every class is related to the other classes through a **simple hierarchy**.

Classifications, The Simplest Of Ontologies



Classification system VS identification system?

- **An identification system puts a data object into its correct slot within the classification.**
  - For example, a **fingerprint-matching system** may look for a set of features that puts a fingerprint into a special subclass of all fingerprints, but the primary goal of fingerprint matching is to establish the identity of an instance (i.e., to show that two sets of fingerprints belong to the same person).
  - In the realm of medicine, when a doctor renders a **diagnosis on a patient's diseases**, she is not classifying the disease—she is finding the correct slot within the pre-existing classification of diseases that holds her patient's diagnosis.

Ontologies, classes with multiple parents

- Ontologies are constructions that permit an object to be **a direct subclass of more than one class**.
  - In an ontology, the class "**horse**" might be a subclass of Equus, a zoologic term, as well as a subclass of "racing animals," "farm animals," and "four-legged animals."
  - The class "**book**" might be a subclass of "works of literature," as well as a subclass of "wood-pulp materials" and "inked products."
- Ontologies are unrestrained classifications.
  - Ontologies are predicated on the belief that a **single object or class of objects might have multiple different fundamental identities** and that these different identities will often place one class of objects directly under more than one superclass.

Ontologies, classes with multiple parents

- Data analysts sometimes **prefer ontologies over classification**s because they permit the analyst **to find relationships** among classes of objects that would have been **impossible to find under a classification**.
  - For example, a data analyst might be interested in determining the relationships among groups of flying animals, such as butterflies, birds, bats, and so on.
    - In the classification of living organisms, these animals occupy classes that are not closely related to one another—no two of the different types of flying animals share a single parent class. **Because classifications follow relationships through a lineage**, they cannot connect instances of classes that fall outside the line of descent.

Ontologies, classes with multiple parents

- In an ontology, a data object can be an instance of **many different kinds of classes**; thus, **the class does not define the essence of the object** as it does in a classification.
  - the **assignment** of an object to a class and the **behavior** of the members of the objects of a class are determined by **rules**.
    - An object belongs to a class when it behaves like the other members of the class, according to a rule created by the ontologist.
    - Every class, subclass, and superclass is defined by rules, and rules can be programmed into software.

Ontologies, classes with multiple parents

- Classifications were created and implemented at a time when scientists **did not have powerful computers** that were capable of handling the complexities of ontologies.
  - For example, the classification of all living organisms on earth was created over a period of two millennia. Several million species have been assigned to the classification. It is currently estimated that we will need to add another 10 to 50 million species before we come close to completing the **taxonomy** of living organisms.
  - Prior generations of scientists could cope with a **simple classification**, wherein each class of organisms falls under a single superclass; they could not cope with a complex ontology of organisms.

Ontologies, classes with multiple parents

- The question that should confront every big data manager is

  1. "Should I model my data as a **classification**, wherein every class has one direct parent class", or

  2. "should I model the resource as an **ontology**, wherein classes may have multi-parental inheritance?"

Choosing a Class Model

- The simple and fundamental question "Can a class of objects have more than one parent class?" lies at the heart of several related fields:
  - **database management,**
  - **computational informatics,**
  - **object-oriented programming,**
  - **semantics, and**
  - **artificial intelligence**

Choosing a Class Model

- Computerscientists are choosing sides, often without acknowledging the problem or fully understanding the stakes.
  - For example, when a programmer builds object libraries in the Python or the Perl programming languages, he is choosing to program in a permissive environment that supports multiclass object inheritance
  - In Python and Perl, any object can have as many parent classes as the programmer prefers.

Choosing a Class Model

- When a programmer chooses to program in the Ruby programming language, he shuts the door on **multiclass inheritance.**
  - A Ruby object can have only one direct parent class. Most programmers are totally unaware of the liberties and restrictions imposed by their choice of programming language until they start to construct their own object libraries or until they begin to use class libraries prepared by another programmer.

Choosing a Class Model

- **Without stating a preference** for single-class inheritance (classifications) or multiclass inheritance (ontologies), one should always strive to design a model that is **as simple as possible**.
  - The wise ontologist will settle for a **simplified approximation of the truth**. Regardless of your personal preference, you should learn to recognize when an ontology has become too complex. Here are the danger signs of an overly complex ontology:

Choosing a Class Model

1. **Nobody**, even the designers, **fully understands** the ontology model.
2. You realize that the ontology **makes no sense**.
3. For a given problem, **no two data analysts seem able to formulate the query the same way, and no two query results are ever equivalent**.
4. The **time** spent on **ontology design** and improvement **exceeds** the time spent on **collecting the data** that populates the ontology.
5. The ontology cannot be fitted into a **higher** level ontology or a **lower** level ontology.
6. The ontology **cannot be debugged when errors are detected.**
7. Errors occur **without anyone knowing** that the error has occurred.

Choosing a Class Model

Simple classifications are not flawless. Here are a few danger signs of an overly simple classification:

1. The classification is **too granular** to be of much value in associating observations with particular instances within a class or with particular classes within the classification.
2. The classification **excludes important relationships among data objects**.
3. The classes in the classification lack **inferential competence.**
4. The classification **contains a "miscellaneous" class.**
5. The classification may be **unstable**.

Choosing a Class Model

- It seems obvious that in the case of big data, a computational approach to data classification is imperative, but **a computational approach that consistently leads to failure is not beneficial**.

- Most of the ontologies that have been created for data collected in many of the fields of science have been ignored or abandoned by their intended beneficiaries.
  - **They are simply too difficult to understand and too difficult to implement.**

Common Pitfalls in Ontology Development

- Do ontologies serve a **necessary role** in the design and development of big data resources?
  - **Yes**. Because every big data resource is composed of many different types of information, it becomes important to assign types of data into groups that have similar properties:
    - images, music, movies, documents, and so forth.

Common Pitfalls in Ontology Development

- The data manager needs to **distinguish one type of data object from another and must have a way of knowing the set of properties** that apply to the members of each class.
  - When a query comes in asking for a list of songs written by a certain composer or performed by a particular musician, the data manager will need to have a software implementation wherein **the features of the query are matched to the data objects for which those features apply**.

Common Pitfalls in Ontology Development

Common Pitfalls:

1. Don't build **transitive** classes
   - Puppy ➔ dog?

2. Don't build **miscellaneous** classes.

3. Don't **invent** classes and properties if they have already been invented.

4. Do not **confuse properties** with your **classes**.
   - Is a leg a subclass of the human body?

WQD7007 Big Data Management

Traditional Database

Database Management System (DBMS)

- DBMS contains information about a particular enterprise
  - Collection of interrelated data
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use
- Database Applications:
  - Banking: transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
- Databases can be very large.
- Databases touch all aspects of our lives

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

University Database Example

- Application program examples
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts

- In the early days, database applications were built directly on top of file systems

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Drawbacks of using file systems to store data

- Data redundancy and inconsistency
  - Multiple file formats, duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation
  - Multiple files and formats
- Integrity problems
  - Integrity constraints (e.g., account balance > 0) become "buried" in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Drawbacks of using file systems to store data

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Data Models

- A collection of tools for describing
  - Data; Data relationships; Data semantics; Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Relational model

- All the data is stored in various tables.
- Example of tabular data in the relational model

Attributes (Columns)

Tuples (Rows)

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

A Sample Relational Database

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

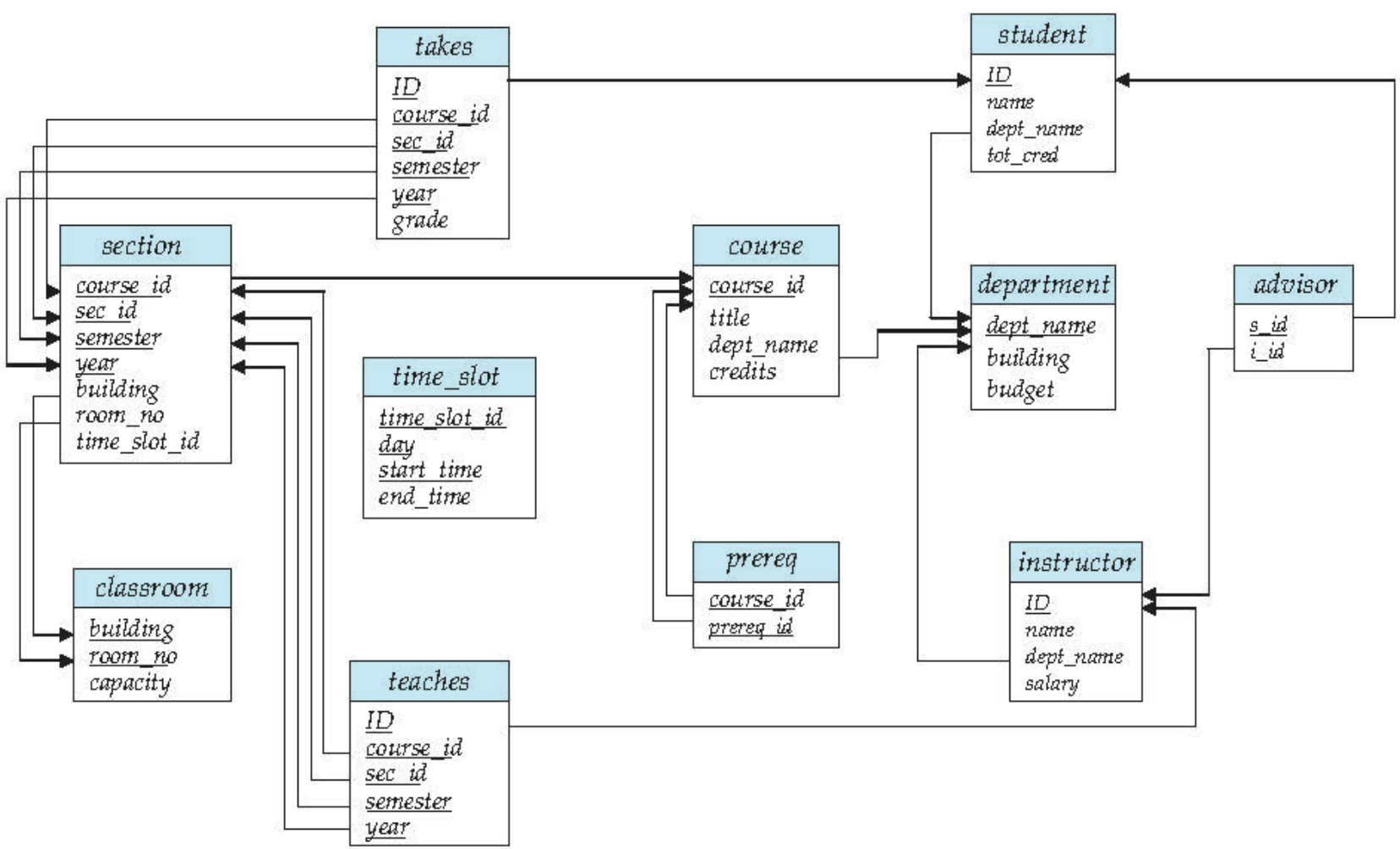
Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Keys

- Let K $\subseteq$ R
- *K* is a superkey of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a candidate key if *K* is minimal
  - Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the primary key.
  - which one?
- Foreign key constraint: Value in one relation must appear in another
  - Referencing relation
  - Referenced relation
  - Example – *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Schema Diagram for University Database



Slide credit: Silberschatz, Korth, Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

SQL: Create Table Construct

- An SQL relation is defined using the create table command:

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - r is the name of the relation
  - each A_i is an attribute name in the schema of relation r
  - D_i is the data type of values in the domain of attribute A_i
- Example:

```
create table instructor (
    ID          char(5),
    name        varchar(20),
    dept_name   varchar(20),
    salary      numeric(8,2))
```


Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Updates to tables

- Insert
  - insert into *instructor* values ('10211', 'Smith', 'Biology', 66000);
- Delete
  - Remove all tuples from the *student* relation
    - delete from *student*
- Drop Table
  - drop table *r*
- Alter
  - alter table *r* add *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute.
  - alter table *r* drop *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Basic Query Structure

- A typical SQL query has the form:

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate.
- The result of an SQL query is a relation.

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

The *select* clause

- The select clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*

  from *instructor*
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

The *where* clause

- The where clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

  select *name*
  from *instructor*
  where *dept_name* = *'*Comp. Sci.'

- Comparison results can be combined using the logical connectives and, or, and not
  - To find all instructors in Comp. Sci. dept with salary > 80000

  select *name*
  from *instructor*
  where *dept_name* = *'*Comp. Sci.' and *salary* > 80000

- Comparisons can be applied to results of arithmetic expressions.

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

The *from* clause

- The from clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  select ∗
  from *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Example Table: Cartesian Product

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Examples

- Find the names of all instructors who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*

- Find the names of all instructors in the Art department who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID* ***and*** *instructor. dept_name =* 'Art'

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Modification in the database

- **Deletion** of tuples from a given relation.
- **Insertion** of new tuples into a given relation
- **Updating** of values in some tuples in a given relation

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Deletion

- Delete all instructors

  delete from *instructor*

- Delete all instructors from the Finance department

  delete from *instructor*
  where *dept_name*= ’Finance’;

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

  delete from *instructor*
  where *dept name* **in** (**select** *dept name*
  from *department*
  where *building* = ’Watson’);

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Insertion

- Add a new tuple to *course*

  insert into *course*
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- or equivalently

  insert into *course* (*course_id*, *title*, *dept_name*, *credits*)
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- Add a new tuple to *student* with *tot_creds* set to null

  insert into *student*
  values ('3003', 'Green', 'Finance', *null*);

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Update

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - The order is important

Slide credit: Silberschatz, Korth. Sudarshan, 2010 http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html

Advantages of Traditional File Access

- Data Security
  - Safely kept and **not easy to access** by unauthorized personnel.

- Complexity
  - Traditional filing systems are **less complex** than electronic systems, which can make it easier for untrained people to access and manipulate data.
    - Anyone can look through alphabetized filing cabinets to find a file. Locating and manipulating an electronic database information may require technical training, and user error can result in unintended alterations or data loss.

Source: http://smallbusiness.chron.com/advantages-disadvantages-traditional-file-organization-41400.html

Disadvantages of Traditional File Access

- Access Time
  - It can **take minutes if not hours to locate a few files** in a large paper filing system.
    - Electronic databases allow for almost instantaneous access to information.
- Editing and Communication
  - Traditional file systems are cumbersome in that they **do not allow users to easily edit files** or send information to others.
    - Paper files often cannot be edited directly, forcing users to make new copies to update old files.
- Order of Data
  - Data can **get out of order** in traditional filing systems.
    - If someone **accidentally puts a file in the wrong place**, or takes a file out of a cabinet and forgets to put it back, it can lead to lost data or the creation of additional copies of files.

Source: http://smallbusiness.chron.com/advantages-disadvantages-traditional-file-organization-41400.html

Characteristics of existing electronic database

- Clearly defined fields organized in records. Records are usually stored in **tables**. **Fields** have names, and relationships are defined between different fields.
  - Schema-on-write that requires data be validated against a schema before it can be written to disk.
    - A significant amount of requirements analysis, design, and effort up front can be involved in putting the data in clearly defined structured formats.
    - **This can increase the time before business value can be realized from the data**.
  - A design to get data from the disk and load the data into memory to be processed by applications.
    - This is an **extremely inefficient architecture when processing large volumes of data** this way.
    - The data is extremely large and the programs are small. The big component must move to the small component for processing.

Source: http://www.pearsonitcertification.com/articles/article.aspx?p=2427073&seqNum=2

Question:

- What is the major difference between traditional electronic database system (e.g. RDBMS) and Big Data database system (e.g. HDFS in Hadoop)?

Concluding Remarks

- This lecture is to give a feel how traditional database looks like, and how to retrieve and update information inside a traditional electronic database, using SQL language.

- Comparison between traditional and modern (electronic) data access, and between traditional electronic database and big data database is discussed.

WQD7007 Big Data Management

Distributed Computing

Agenda

- Distributed computing compare to other computing concepts
- Distributed computing in Hadoop

Distributed Computing

- Distributed computing is a field of computer science that studies distributed systems.
  - A distributed system is a model in which components located on **networked computers**, communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.
  - Three characteristics:
    - the computers operate concurrently
    - fail independently
    - do not share global clock

Source: Tanenbaum, Andrew S.; Steen, Maarten van (2002). Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall.

Distributed Computing

- A model consists of **multiple software components** that are run on multiple computers to improve efficiency and performance.
  - something that shared among multiple systems which may also be in different locations to make such a network work as a single computer.

- **Two types** of distributed systems:
  - Computers are physically close together (connected by a local network)
  - Computers are geographically distant (connected by a wide area network)

Distributed Computing

- A distributed system can consist of **many different possible configurations**, such as mainframes, personal computers, workstations, minicomputers, and so on.
  - For example, in a 3-tier distributed model each tier will do different task:
    1. User interface processing (performed in the **PC** at the user's location)
    2. Business processing (done in a **remote computer**)
    3. Database access and processing (performed in another computer that provides **centralized access**)

Distributed Computing VS Parallel computing

- Parallel computing: There are many processing steps and each processing step is **completed at the same specified time**.
  - For example: It is important to have steps work in parallel when dealing with videos.
- Distributed Computing: There are many processing steps but processing is **divided into several computers or nodes**
  - each computer/node works concurrently and then forwarding their individual outputs of some process to another **master** or responsible node that aggregates the outputs together.
  - The sequence of such processing is not guaranteed.
  - Example: Hadoop and MapReduce
- <u>How to choose?</u>

Distributed Computing VS Grid Computing

- Grid Computing: It is a kind of **more secured and location-specified distributed system** such that in a specific organization or firm.
  - a service for sharing computer power and data storage capacity over the Internet
  - is more concerned to efficient **utilization of a pool of heterogeneous systems** with optimal workload management - utilizing an enterprise's entire computational resources (servers, networks, storage, and information), acting together to create one or more large pools of computing resources.
- Distributed Computing normally refers to managing or pooling the hundreds or thousands of computer systems which **individually are more limited in their memory and processing power**.

Source: http://www.jatit.org/distributed-computing/grid-vs-distributed.htm

Distributed Computing VS Cloud Computing

- Cloud Computing: Creating a **distributed system/ architecture at a remote location or over a virtual facility**.
  - Example: Amazon Cloud Services
  - massively scalable and flexible IT-related capabilities are delivered as a service to the users using Internet technologies
  - services may include: infrastructure, platform, applications, and storage space.
  - The users pay for these services, resources they actually use. They do not need to build infrastructure of their own.
  - minimal flexibility: The application and services run on a remote server. Due to this, enterprises using cloud computing have **minimal control over the functions** of the software as well as hardware.

Distributed Computing VS Cluster Computing

- Cluster Computing - A cluster is a system comprising two or more computers or systems (called nodes) which work together to execute applications or perform other tasks, so that users who use them, **have the impression that only a single system responds to them**, thus creating an illusion of a single resource (virtual machine).
  - Types of cluster: High Availability (HA) and fail-over clusters
    - These models are built to provide an availability of services and resources in an uninterrupted manner through the use of implicit redundancy to the system.
  - The general idea is that if a cluster node fail (fail-over), applications or services may be available in another node

Summary

- **Grid** Computing
  - Loosely coupled (Decentralization)
  - Diversity and Dynamism
  - Distributed Job Management & scheduling
- **Cloud** computing
  - Dynamic computing infrastructure
  - IT service-centric approach
  - Self service based usage model
  - Minimally or self managed platform

Source: https://www.quora.com/What-is-the-difference-between-grid-cloud-cluster-and-distributed-computing

Summary

- **Cluster** computing
  - Tightly coupled systems
  - Single system image
  - Centralized Job management & scheduling system
- **Distributed** Computing
  - Is to solve a single large problem by breaking it down into several tasks where each task is computed in the individual computers of the distributed system.

Source: https://www.quora.com/What-is-the-difference-between-grid-cloud-cluster-and-distributed-computing

Advantages

- Distributed systems are **inherently scalable** because they work across a variety of different machines.
  - The system can easily be expanded **by adding more machines** as needed.
- They can **adjust how many system resources** it is making use of in light of what kind of demand the system is under.
  - If a system is under high demand, then it can have every machine running to capacity.
  - However, if the load on the system is relatively low, it can take different components of the distributed system offline to save power and wear on the system.
  - When demand on the system goes up again, these components can come back online.

Advantages

- **Redundancy**:
  - several machines can provide the same services, so if one is unavailable, work does not stop.
  - fault tolerant and able to sustain computer failures without crashing since it makes the whole network work as a single computer.
- distributed computing **encourages parallel processing** of jobs which can offer enormous performance gains.
  - For example, a network of processors can execute graphics computations much faster than a single highly-clocked core.
- **Better pricing versus performance ratio**
  - adding microprocessors in distributed computing systems are more cost effective compared to adding mainframes in centralized computer.

Drawbacks

- Analysis as a whole:
  - When the project is **divided into each personal computing calculating power**, it cannot solve some big problem which **require to analyse in a whole**.
- Trusted source:
  - With so many computers involving, we need to make sure they are **from trusted source**, and not malpractice participating in it
- Data synchronization:
  - a problem in distributed systems because **different distributed system components could handle different tasks and data**.
  - At any given point in time, there will be small periods of time in which data exists on one component, but not on others.
- Difficulties in troubleshooting and diagnosing problems

Distributed Computing in Hadoop

- Tools that are using distributed computing concepts:
  - HDFS → distributed file system
  - MapReduce → distributed computation
    - These include all other tools that uses MapReduce as backbone
  - ZooKeeper → distributed coordination

Source: https://www.slideshare.net/KonstantinVShvachko/distributed-computing-with-apache-hadoop-technology-overview

ZooKeeper

- A distributed coordination service for distributed apps
  - Event coordination and notification
  - Leader election
  - Distributed locking
- ZooKeeper can help in building High Availability (HA) system
- Installing Zookeeper:
  - https://medium.com/@ryannel/installing-zookeeper-on-ubuntu-9f1f70f22e25

Source: https://www.slideshare.net/KonstantinVShvachko/distributed-computing-with-apache-hadoop-technology-overview

Examples:

- SETI:
  - There are **massive amounts of data are collected around the world from the stars**, searching for any intelligent life, recorded via many observatories.
  - Search for Extraterrestrial Intelligence (**SETI**) takes these massively large information stores and slices them up into smaller pieces of data for easy analysis via distributed computing applications running as screen savers on individual user PC's, all around the world.
  - Tens of thousands of PC's running the SETI screen saver will download a small file, and while a PC is unused, it's screen saver downloads a data slice from SETI, runs the analysis application while the PC is idle, and when the analysis is complete, the analyzed data slice is uploaded back to SETI.

Source: https://setiathome.berkeley.edu/

Examples:

- World Community Grid - IBM
  - By using **the idle time of computers around the world**, this research project can analyse human genome, HIV, etc.
  - When our computer connected to internet, the users install WCG client software onto computers. This will work in background when your computer is idle, sing spare system resources.
  - Major findings in 2014: they manage to discover **7 compounds that destroy a particular nerve type cancer cells without side effects**

Examples:

- "Folding at home" project is a real example of distributed computing where you install their application and when your computer goes to screen saver they use your computer processing resource.
- *"While you keep going with your everyday activities, your computer will be working to help us find cures for diseases like cancer, ALS, Parkinson's, Huntington's, Influenza and many others."*

Source: https://foldingathome.org/start-folding/ /

WQD7007 Big Data Management

Big Data Tools
HBase

HBase

- An **open source NoSQL database** that provides random and real-time read/write (CRUD operation) access to those large datasets that runs on top of HDFS.
- HBase **scales linearly to handle huge data sets with billions of rows and millions of columns**, and it easily combines data sources that use a wide variety of different structures and schemas.
  - making it a great choice to store multi-structured or sparse data.

HBase

- These following characteristics make HBase a great choice for **storing semi-structured data like log data** and then providing that data very quickly to users or applications integrated with HBase.
- Linear scaling of Hbase:
  - Require all tables to have a primary key.
  - The key space is divided into sequential blocks that are then allotted to a region.
  - RegionServers own one or more regions, so the load is spread uniformly across the cluster.

HBase

Characteristics	Benefit
Fault tolerant	•**Replication** across the data center •**Atomic and strongly consistent** row-level operations •**High availability** through automatic failover •**Automatic sharding and load balancing**s of tables
Fast	•**Near real time lookups** •**In-memory caching** via block cache and bloom filters •**Server side processing** via filters and co-processors
Usable	•**Data model** accommodates wide range of use cases •**Metrics exports** via File and Ganglia plugins •**Easy Java API** as well as Thrift and REST gateway APIs

HBase

- Enterprises use Apache HBase's **low latency storage** for scenarios that require real-time analysis and tabular data for end user applications.
  - Example 1: One company that provides **web security services** maintains a system accepting billions of event traces and activity logs from its customer' desktops every day.
    - The company's programmers can tightly integrate their **security solutions** with HBase (to assure that the protection they provide keeps pace with **real-time changes** in the threat landscape.)

HBase

- Example 2: One company provides **stock market ticker plant data** that its **users query more than thirty thousand times per second**, with an SLA of only a few milliseconds.
  - Apache HBase provides that **super low-latency** access over an enormous, rapidly changing data store.

HBase

- Apache HBase provides high availability in several ways:
  - **Highly available cluster topology information** through production deployments with multiple HMaster and ZooKeeper instances
  - **Data distribution across many nodes** means that loss of a single node only affects data stored on that node

HBase

- Apache HBase provides high availability in several ways:
  - **HBase HA** allows data storage, ensuring that loss of a single node does not result in loss of data availability
  - **HFile format stores data directly in HDFS.** HFile can be read or written to by Apache Hive, Apache Pig, MapReduce, and Apache Tez, permitting deep analytics on HBase without data movement

Version command

```
hbase(main):001:0> version
```

```
hbase(main):001:0> version
1.1.2.2.5.0.0-1245, r53538b8ab6749cbb6fdc0fe448b89aa82495fb3f, Fri Aug 26 01:32:27 UTC 2016
```

`list` command

```
hbase(main):002:0> list
```

```
hbase(main):002:0> list
TABLE
ATLAS_ENTITY_AUDIT_EVENTS
Contacts
atlas_titan
iemployee
test
5 row(s) in 0.6480 seconds

=> ["ATLAS_ENTITY_AUDIT_EVENTS", "Contacts", "atlas_titan", "iemployee", "test"]
```

Create table

```
hbase(main):003:0> create 'customer','address','order’
hbase(main):004:0> list
```

```
hbase(main):003:0> create 'customer','address','order'
0 row(s) in 5.5740 seconds

=> Hbase::Table - customer
hbase(main):004:0> list
TABLE
ATLAS_ENTITY_AUDIT_EVENTS
Contacts
atlas_titan
customer
iemployee
test
6 row(s) in 0.0270 seconds

=> ["ATLAS_ENTITY_AUDIT_EVENTS", "Contacts", "atlas_titan", "customer", "iemployee", "test"]
```

Insert entry

- hbase(main):026:0> put 'customer','john','address:city','Boston’
  - customer is the table name
  - John is the row key
  - address is the column family
  - Boston is its value.

```
hbase(main):005:0> put 'customer','john','address:city','Boston'
0 row(s) in 0.4760 seconds

hbase(main):006:0> put 'customer','john','address:state','Massachusetts'
0 row(s) in 0.0320 seconds

hbase(main):007:0> put 'customer','john','address:street','street1'
0 row(s) in 0.0470 seconds

hbase(main):008:0> put 'customer','john','order:number','ORD-15'
0 row(s) in 0.0140 seconds

hbase(main):009:0> put 'customer','john','order:amount','15'
0 row(s) in 0.0860 seconds
```

Put another record

```
hbase(main):010:0> put 'customer','Finch','address:city','Newyork'
0 row(s) in 0.0200 seconds

hbase(main):011:0> put 'customer','Finch','address:state','Newyork'
0 row(s) in 0.0430 seconds

hbase(main):012:0> put 'customer','Finch','order:number','ORD-16'
0 row(s) in 0.0320 seconds

hbase(main):013:0> put 'customer','Finch','order:amount','15'
0 row(s) in 0.0190 seconds
```

Get record

```
hbase(main):026:0> Get ‘customer’, ‘john’
```

```
hbase(main):014:0> get 'customer','john'
COLUMN                                 CELL
 address:city                          timestamp=1523930451244, value=Boston
 address:state                         timestamp=1523930461470, value=Massachusetts
 address:street                        timestamp=1523930468611, value=street1
 order:amount                          timestamp=1523930484954, value=15
 order:number                          timestamp=1523930476471, value=ORD-15
5 row(s) in 0.1710 seconds
```

Get record (2)

- Using get command to retrieve the address of john
  - `hbase(main):044:0> get 'customer','john','address'`

```
hbase(main):015:0> get 'customer','john','address'
COLUMN                                CELL
 address:city                         timestamp=1523930451244, value=Boston
 address:state                        timestamp=1523930461470, value=Massachusetts
 address:street                       timestamp=1523930468611, value=street1
3 row(s) in 0.0420 seconds
```

Get record (3)

- Using get command to retrieve city of john
  - `hbase(main):045:0> get 'customer','john','address:city'`

```
hbase(main):016:0> get 'customer','john','address:city'
COLUMN                                  CELL
 address:city                           timestamp=1523930451244, value=Boston
1 row(s) in 0.0930 seconds
```

Scan record

```
hbase(main):017:0> scan customer
```

```
hbase(main):018:0> scan 'customer'
ROW                                    COLUMN+CELL
 Finch                                 column=address:city, timestamp=1523930561402, value=Newyork
 Finch                                 column=address:state, timestamp=1523930568997, value=Newyork
 Finch                                 column=order:amount, timestamp=1523930587114, value=15
 Finch                                 column=order:number, timestamp=1523930576473, value=ORD-16
 john                                  column=address:city, timestamp=1523930451244, value=Boston
 john                                  column=address:state, timestamp=1523930461470, value=Massachusetts
 john                                  column=address:street, timestamp=1523930468611, value=street1
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
2 row(s) in 0.1890 seconds
```

Update record

```
hbase(main):011:0>
put 'customer', 'john', 'address:street', 'street2'
```

```
hbase(main):010:0> scan 'customer'
ROW                                    COLUMN+CELL
 john                                  column=address:street, timestamp=1523930468611, value=street1
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
1 row(s) in 0.0520 seconds

hbase(main):011:0> put 'customer', 'john', 'address:street', 'street2'
0 row(s) in 0.2130 seconds

hbase(main):012:0> scan 'customer'
ROW                                    COLUMN+CELL
 john                                  column=address:street, timestamp=1523932537860, value=street2
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
1 row(s) in 0.0570 seconds
```

Delete entire record

```
hbase(main):019:0> deleteall 'customer','Finch'
```

```
hbase(main):003:0> scan 'customer'
ROW                                    COLUMN+CELL
 Finch                                 column=address:state, timestamp=1523930568997, value=Newyork
 Finch                                 column=order:amount, timestamp=1523930587114, value=15
 Finch                                 column=order:number, timestamp=1523930576473, value=ORD-16
 john                                  column=address:state, timestamp=1523930461470, value=Massachusetts
 john                                  column=address:street, timestamp=1523930468611, value=street1
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
2 row(s) in 0.2870 seconds

hbase(main):004:0> deleteall 'customer', 'Finch'
0 row(s) in 0.0860 seconds

hbase(main):005:0> scan 'customer'
ROW                                    COLUMN+CELL
 john                                  column=address:state, timestamp=1523930461470, value=Massachusetts
 john                                  column=address:street, timestamp=1523930468611, value=street1
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
1 row(s) in 0.0340 seconds
```

Delete specific column

- hbase(main):046:0> **delete** 'customer',john,'address:state'

```
hbase(main):008:0> scan 'customer'
ROW                                    COLUMN+CELL
 john                                  column=address:state, timestamp=1523930461470, value=Massachusetts
 john                                  column=address:street, timestamp=1523930468611, value=street1
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
1 row(s) in 0.1670 seconds

hbase(main):009:0> delete 'customer', 'john', 'address:state'
0 row(s) in 0.0320 seconds

hbase(main):010:0> scan 'customer'
ROW                                    COLUMN+CELL
 john                                  column=address:street, timestamp=1523930468611, value=street1
 john                                  column=order:amount, timestamp=1523930484954, value=15
 john                                  column=order:number, timestamp=1523930476471, value=ORD-15
1 row(s) in 0.0520 seconds
```

Other commands

- `count ‘customer’`
- `disable ‘customer’`
- `drop ‘customer’`

WQD7007 Big Data Management

NoSQL and MongoDB

NoSQL

- NoSQL → Not Only SQL
  - being **non-relational, distributed,** and **open source**

- Characteristics:
  - schema-free
  - easy replication support
  - simple API
  - Big data
  - Commodity
  - Flexible data model

Source: http://nosql-database.org/

MongoDB

- A document-based (e.g. JSON or BSON) database.
- Characteristics:
  - Expressive query languages and secondary indexes
  - Strong consistency
  - Enterprise Management and Integration
  - Flexibility
  - Scalability and performance
  - Always on, global deployments

Source: https://www.mongodb.com/

JSON and BSON

- Java Script Object Notation (JSON) is the lightweight data-interchange format to exchange data.
- Binary JSON → BSON: JSON that is serialized as a binary document
- JSON build on two structures:
  - A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

Source: https://www.json.org/

Example of JSON:

```
{
    "Rail Booking": {
        "reservation": {
            "ref_no": 1234567,
            "time_stamp": "2016-06-24T14:26:59.125",
            "confirmed": true
        },
        "train": {
            "date": "07/04/2016",
            "time": "09:30",
            "from": "New York",
            "to": "Chicago",
            "seat": "57B"
        },
        "passenger": {
            "name": "John Smith"
        },
        "price": 1234.25,
        "comments": ["Lunch & dinner incl.", "\"Have a nice day!\""]
    }
}
```

Why JSON and not XML?

- XML is not well suited to data-interchange. It carries a lot of baggage, and it doesn't match the data model of most programming languages.

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo"  :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                        ]
      }
   }
```

MongoDB VS RDBMS

MongoDB	RDBMS
Database	Database
Collection	Table
Document	Tuple/Row
Field	Column
Embedded Documents	Table Join

From RDBMS to MongoDB

Relational

Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

no relation

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB Document

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, ... },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, ... }
  ]
}
```

Benefits of using JSON (document model)

- Document maps nicely to programming languages data type
- Embedded documents and arrays reduces the need for joins

MongoDB vs HBase

MongoDB	HBase
In MongoDB data is stored in JSON format. We have to insert/update/fetch the document as a whole	In HBase we can work on partial data.
MongoDB relies on the OS to buffer its data and maps the whole data to memory with mmap.	HBase use the log-structured merge tree, which probably is more efficient for heavy load and big data.
MongoDB supports secondary index and the index should be keep in memory.	HBase requires a third party module to support secondary index.
MongoDB has good feature of searching a document data using find/findOne by its inner attribute. The attribute can be indexed to search faster.	HBase do not have this feature.
Examples: Product/catalog data, geospatial data, Aadhar biometric database, ebay search suggestion	Examples: storing genome sequences, sports match histories, time-series data, log files

Source: https://www.oodlestechnologies.com/blogs/Comparison-Between-MongoDB-and-Hbase

Installation

- Download MongoDB in https://www.mongodb.org/downloads#production
  - If you're using Mac, you can use Homebrew
  - For Windows:
    - This lab assumes that you install it to C:\Program Files\MongoDB. Add the bin directory to your System Environment Variables
  - If you're using any other operating systems, please refer to this https://docs.mongodb.org/manual/installation/ for more info
- During installation, you'll be required to install it to a PATH. Remember the path that you installed MongoDB to.

Running MongoDB Server

- Before you proceed, you will need to create a directory which stores your MongoDB data.
  - It is always /data/db by default. (For Windows, it's C://data/db/.) Create that data directory.
  - If you can't create the directory in Mac or Linux, use "sudo" privileges
- Finally, head to your terminal and type mongod --smallfiles --noprealloc to start your MongoDB Server.
  - If it states "no such command is recognized", that means you have not added the bin directory into your PATH
  - If you use "sudo" privileges to create path, please use "sudo" privileges for this command too

Run MongoDB

- open a new terminal (or a tab), and type mongo.
- Basic commands:
  - **show dbs**: Show the names of all databases
  - **show collections**: Show the names of all collections in the database
  - **use dbname**: Change to the database named dbname
  - **db**: Returns the name of the current database that you are connected to

Run MongoDB

```
Wais-MacBook-Air:~ wailamhoo$ mongo
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2018-05-15T13:49:07.234+0800 I CONTROL  [initandlisten]
2018-05-15T13:49:07.234+0800 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-05-15T13:49:07.234+0800 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2018-05-15T13:49:07.234+0800 I CONTROL  [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten]
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this server.
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specify which IP
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with --bind_ip_all to
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desired, start the
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this warning.
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten]
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten]
2018-05-15T13:49:07.235+0800 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> show collections
> use admin
switched to db admin
> use wlhoo
switched to db wlhoo
```

Run MongoDB

- There will not be any commands to **create a database**. If the database does not exist, say when you do **use random1234**, Mongo will **create a database called random1234** for you. So, be careful in your syntax!
- Mongo by default connects you to the **test database** upon opening a connection.

Create database and Insert Data

- We'll now be using a new database called **firstdb** , and we'll be inserting some entries into the collection named **users**.
  - **use firstdb**
  - **db.users.insert({ name: 'Enter Your Name Here', age: 25 })**
  - **db.users.find().pretty())**

```
> use firstdb
switched to db firstdb
> db.users.insert( { name: 'Enter Your Name Here', age: 25 } )
WriteResult({ "nInserted" : 1 })
> db.users.find().pretty()
{
        "_id" : ObjectId("5afa78457d617295fea6c1a7"),
        "name" : "Enter Your Name Here",
        "age" : 25
}
```

Import JSON file

- Assume you download the JSON data, stored it in Desktop and wish to import to MongoDB:
- **mongoimport –d school –c students < ~/Desktop/student.json**

```
Wais-MacBook-Air:~ wailamhoo$ mongoimport -d school -c students < ~/Desktop/students.json
2018-05-15T14:11:53.352+0800    connected to: localhost
2018-05-15T14:11:53.422+0800    imported 200 documents
Wais-MacBook-Air:~ wailamhoo$
```

- Enter mongoDB by typing **mongo**. Then, switch to *school* database by typing **use school**

Inserting data

Try insert these documents!

Id	Name	Scores
200	Drake	[]
201	Logic	[{“type”: “homework”, ”score”: 89}, {“type”: “quiz”, ”score”: 56}]
202	Kraimir	[{“type”: ”homework”, “score”: 11}]

Finding documents

Example: Display the details of the student named "Brain Lachapelle"

```
> db.students.find({"name":"Brain Lachapelle"}).pretty()
{
        "_id" : 86,
        "name" : "Brain Lachapelle",
        "scores" : [
                {
                        "type" : "exam",
                        "score" : 2.013473187690951
                },
                {
                        "type" : "quiz",
                        "score" : 45.01802394825918
                },
                {
                        "type" : "homework",
                        "score" : 63.74658824265818
                },
                {
                        "type" : "homework",
                        "score" : 88.04712649447521
                }
        ]
}
```

Finding documents

1. Obtain the number of students who achieved a a score greater than 95 of any type.
2. Display the student who achieved the highest score on any type in the whole school
3. Display the _id and name of the first 10 students (arranged in alphabetical order by name) who achieved a score between 10 and 35 of any type
4. What is the number of students whose names are between C and E lexically? Also, display only the name of 10 of these students, in descending order

Updating and removing documents

- Update from one value to another
  - **db.users.update({“username”:”admin123”}, {$set: {“password”: “secret”}})**
- Removes “profile ID” attribute
  - **db.users.update({“username”:”admin123”}, {$unset: {“profile_id”: 1}})**
- Increments “login_count” value by 1
  - **db.users.update({“username”:”admin123”}, {$inc: {“login_count”: 1}})**
- Insert a new permission in **“permission” array**
  - **db.users.update({“username”:”admin123”}, {$push: {“permissions”: “add_blog”}})**

Updating and removing documents

- Removes the first element from “permission” array:
  - **db.users.update({“username”:”admin123”}, {$pop: {“permissions”: -1}})**
- Removes the specified element from “permission” array
  - **db.users.update({“username”:”admin123”}, {$pull: {“permissions”: “edit_bg”}})**
- Insert the specified element to “permission” array
  - **db.users.update({“username”:”admin123”}, {$addToSet: {“permissions”: “edit_blog”}})**
- Remove user
  - **db.users.remove({“username”:”admin123”})**