Course Wrap-up and Review

COMP9313: Big Data Management

Big Data: What is it?

- **Wikipedia:**

  "Big data is a field that treats ways to analyze, systematically extract information from, or otherwise deal with data sets that are too large or complex to be dealt with by traditional data-processing application software"

- **Oxford English dictionary:**

  "Extremely large data sets that may be analysed computationally to reveal patterns, trends, and associations, especially relating to human behaviour and interactions"

- **Apache Hadoop[1]:**

  "Datasets which could not be captured, managed, and processed by general computers within an acceptable scope"

[1]Chen et al. 2014. Big data: A survey. Mobile networks and applications, 19(2), pp.171-209.

Examples of Big Data

source

source

e-commerce

source

Open Data

source

source

Social Media

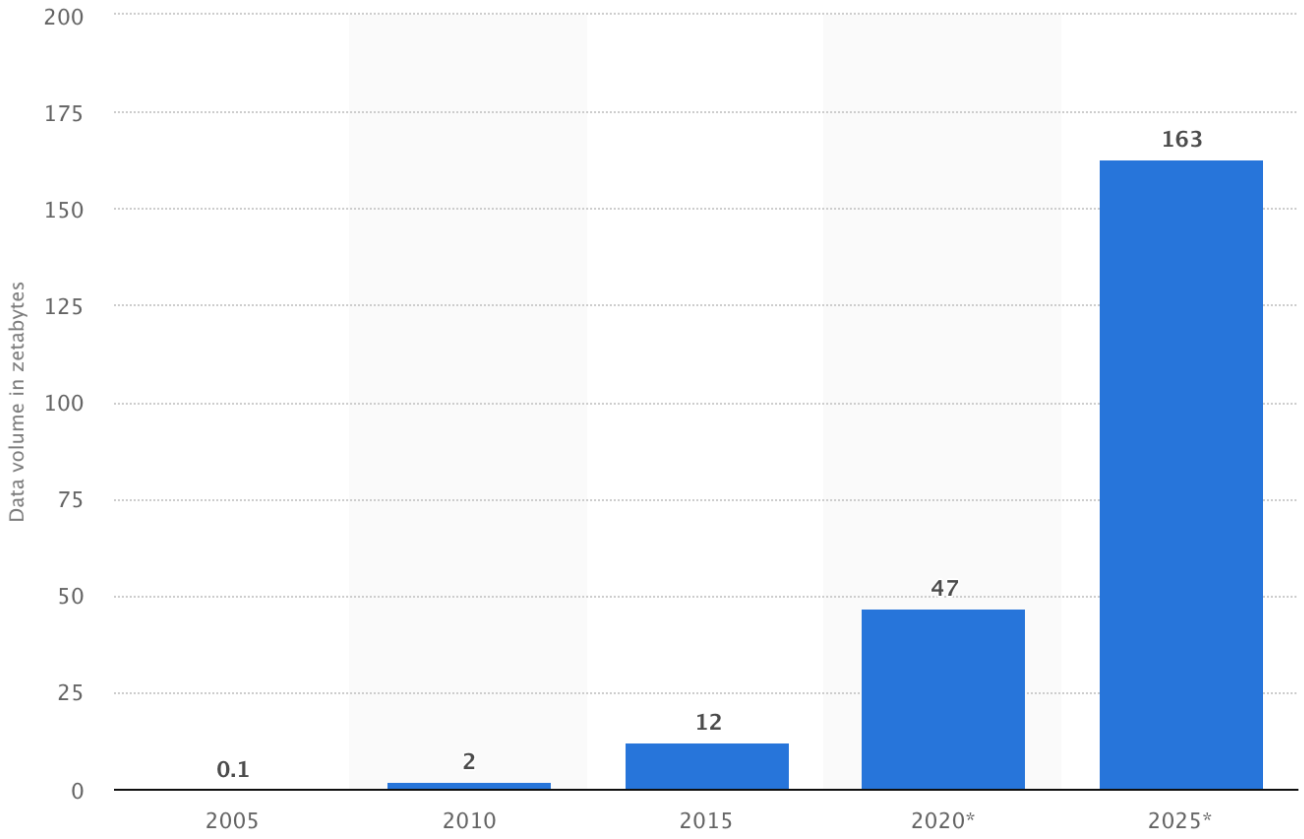
Healthcare

source

Internet of Things

Financial Sector

Volume of data/information created worldwide from 2005 to 2025

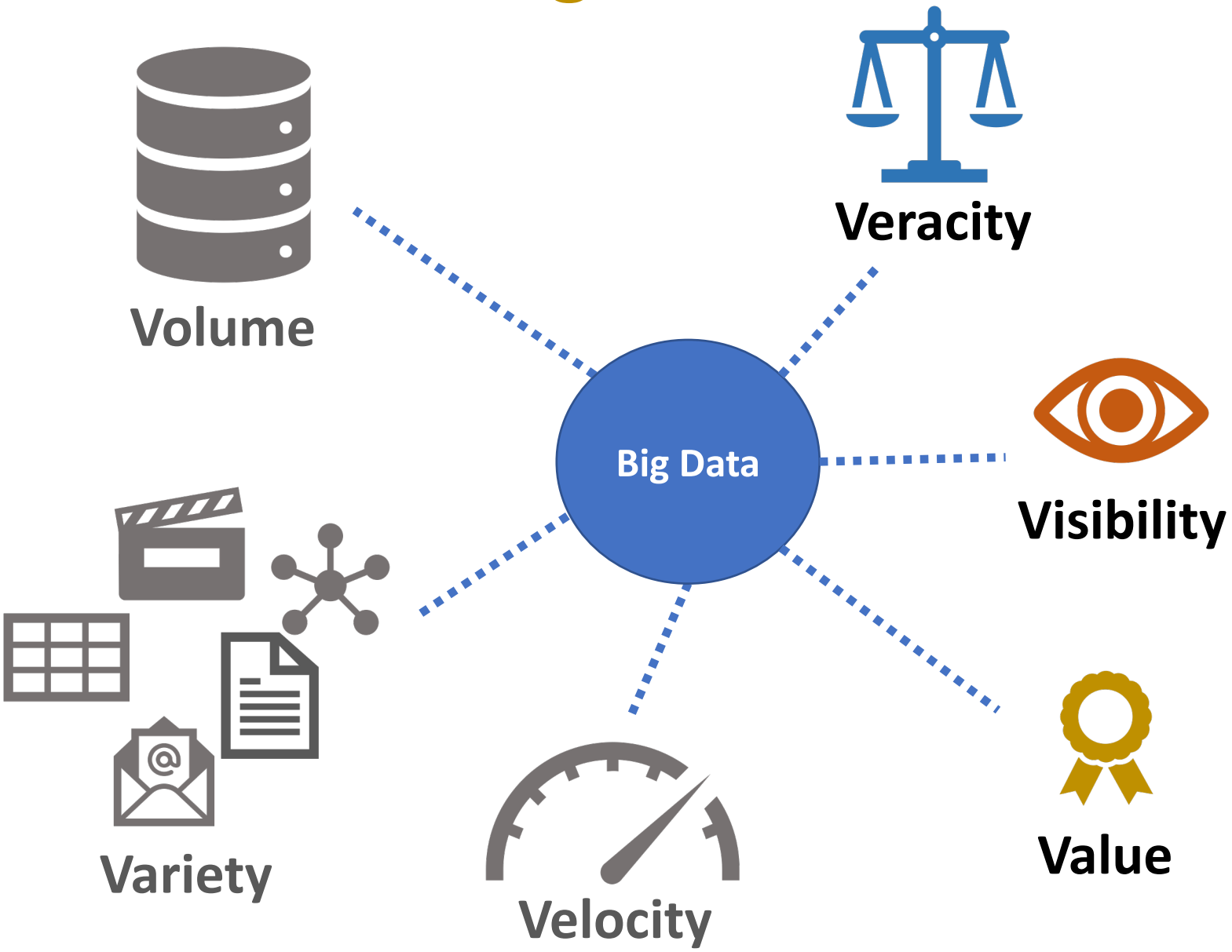


1 zettabyte = 1 million petabytes = 1 billion terabytes

https://www.statista.com/statistics/871513/worldwide-data-created/

The 3 Vs of Big Data

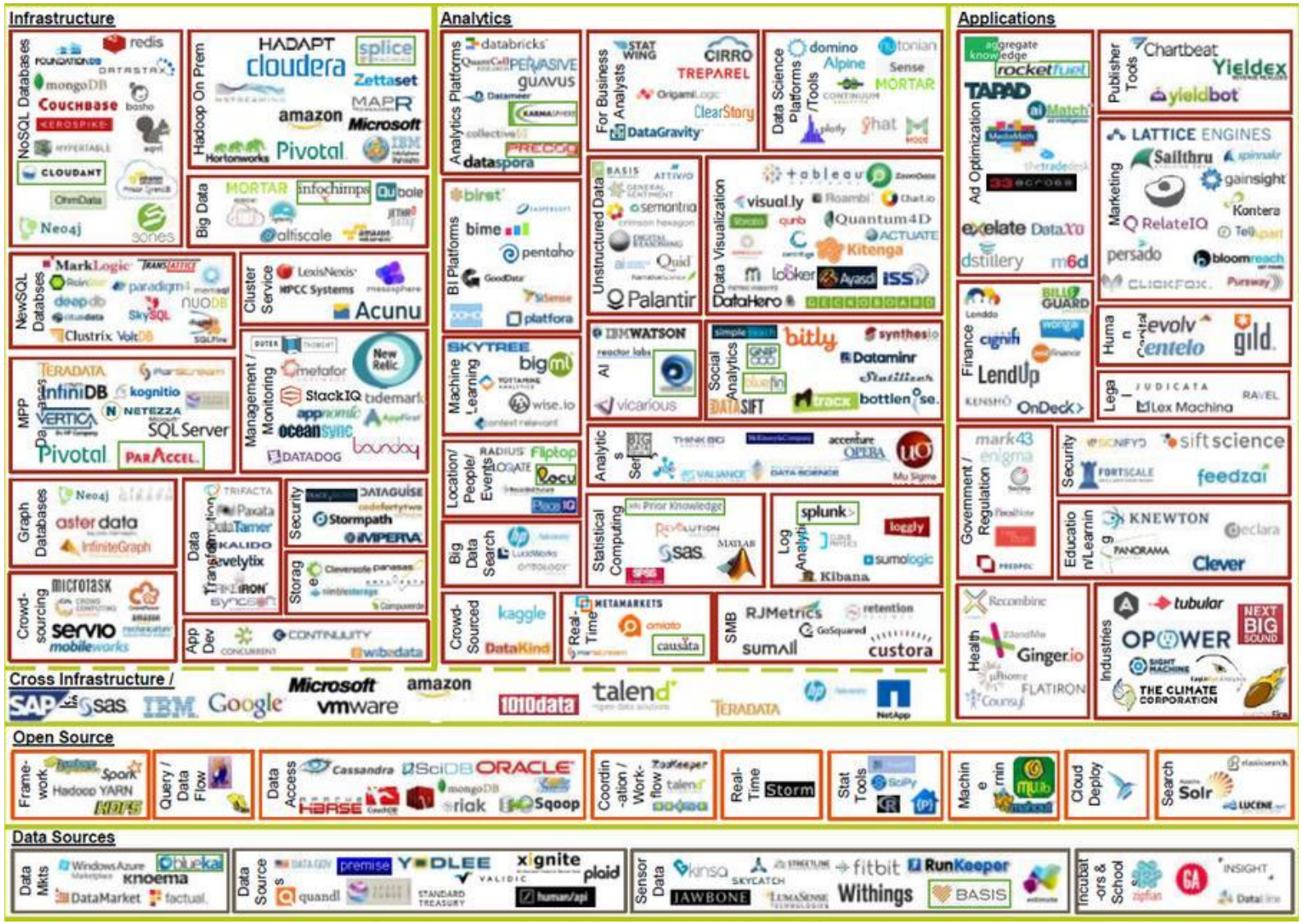
Other Vs for Big Data



Expansion of the original Vs

- Vagueness
- Validity
- **Value**
- Variability
- **Variety**
- **Velocity**
- Venue
- **Veracity**
- Viability
- Vincularity
- Viscosity
- **Visibility**
- Visible
- Visualization
- Vitality
- Vocabulary
- Volatility
- **Volume**

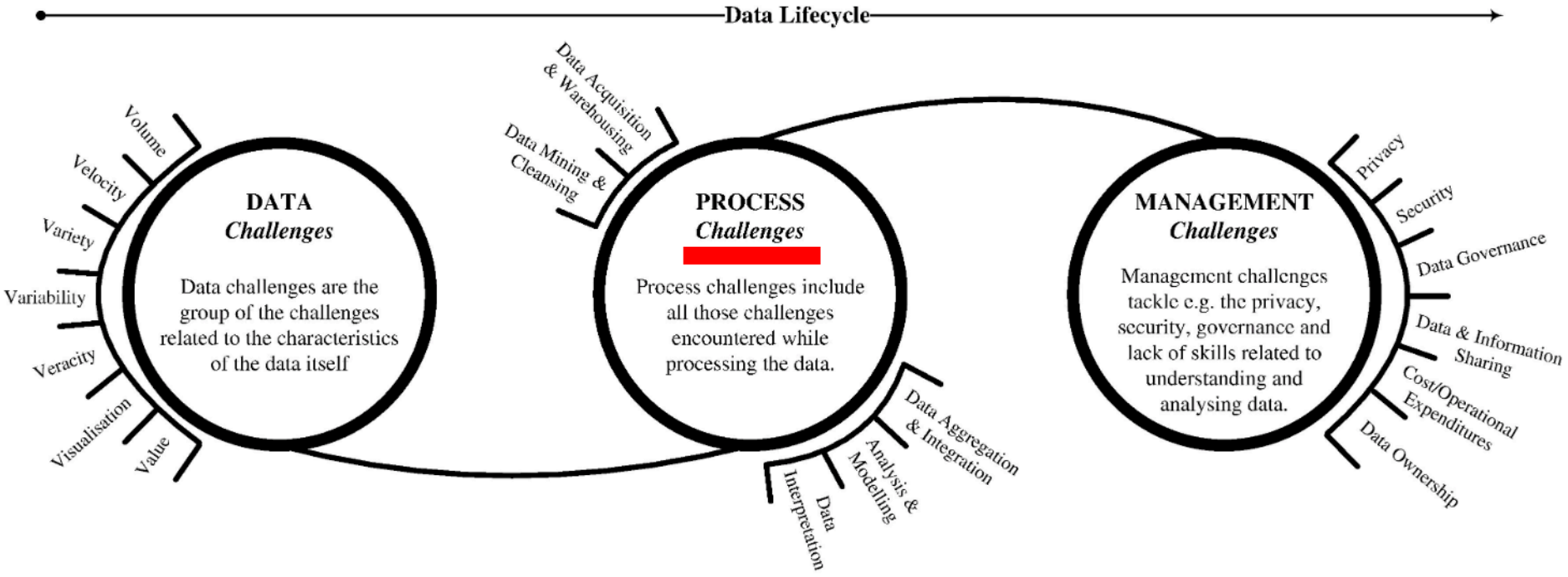
Examples of Big Data Technologies



source

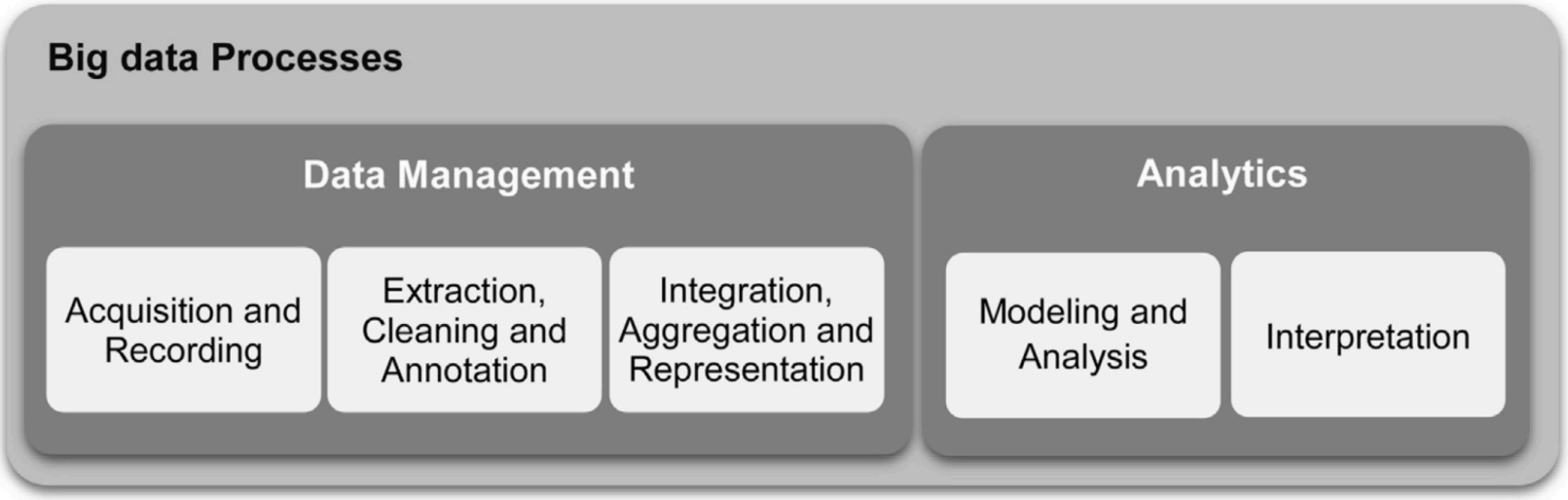
Big Data Processes and Management

Big Data Lifecyle



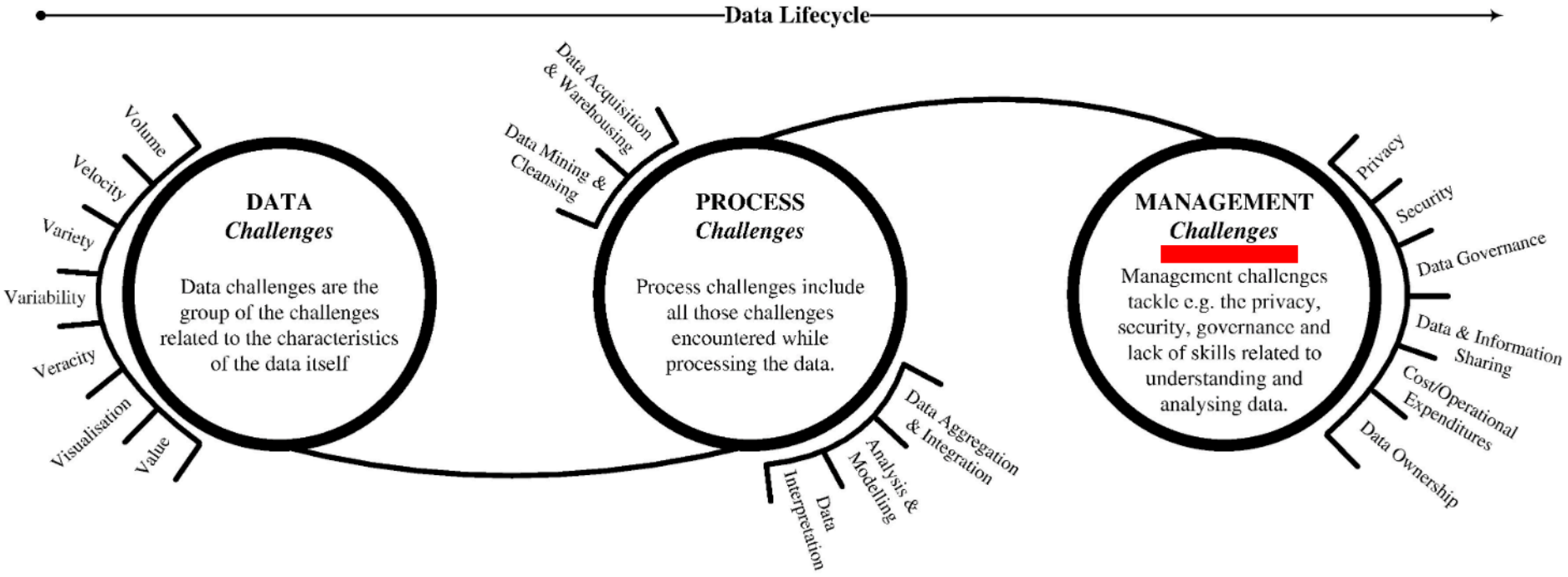
https://www.sciencedirect.com/science/article/pii/S014829631630488X#bb9030

Big Data Processes



https://www.sciencedirect.com/science/article/pii/S0268401214001066

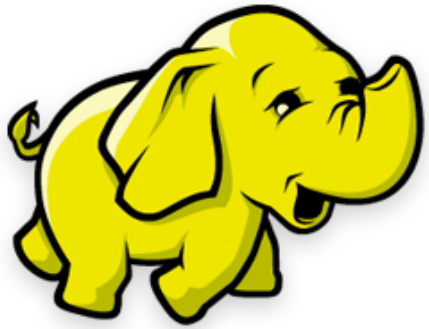
Data Lifecyle



https://www.sciencedirect.com/science/article/pii/S014829631630488X#bb9030

Apache Hadoop and MapReduce

Hadoop



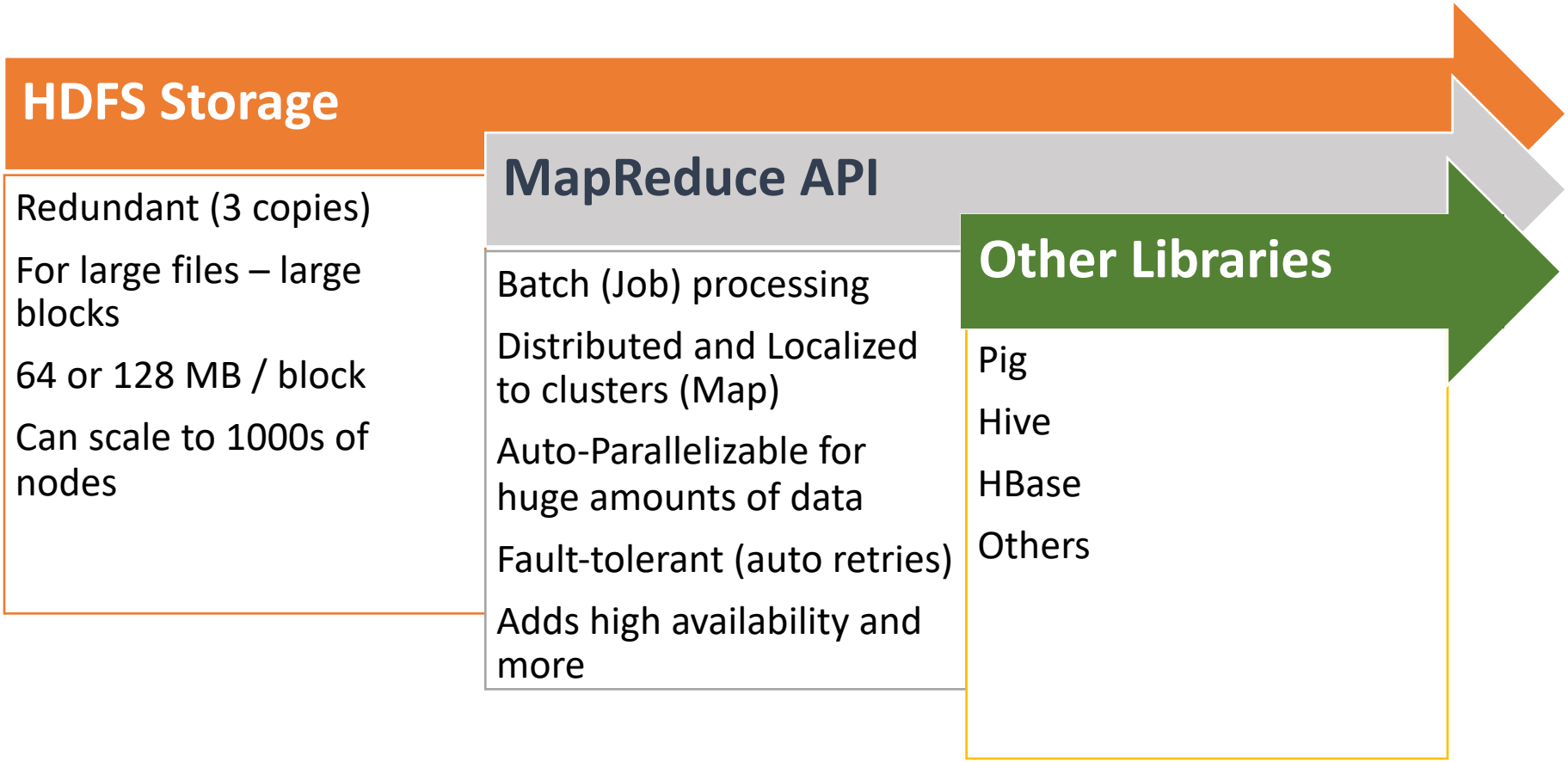
source

- Open-source data storage and processing platform
- Before the advent of Hadoop, storage and processing of big data was a big challenge
- Massively scalable, automatically parallelizable
  - Based on work from Google
    - Google: GFS + MapReduce + BigTable (Not open)
    - Hadoop: HDFS + Hadoop MapReduce + Hbase (opensource)
- Named by Doug Cutting in 2006 (worked at Yahoo! at that time), after his son's toy elephant

What's offered by Hadoop?

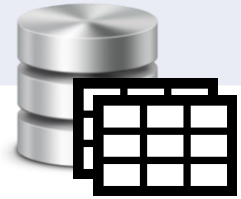
- Redundant, fault-tolerant data storage
- Parallel computation framework
- Job coordination
- Programmers do not need to worry about:
  - Where are files located?
  - How to handle failures and data loss?
  - How to distribute computation?
  - How to program for scaling?

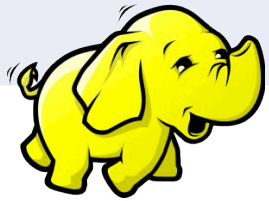
Core Parts of a Hadoop Distribution



RDMS vs Hadoop

Feature	RDBMS	Hadoop
Data variety	Mainly structured data	Structured, semi-structured and unstructured data
Data storage	Average size data (GBs)	Used for large datasets (TBs, PTs)
Querying	SQL	HQL (Hive Query Language)
Schema	Required on write (static schema)	Required on read (dynamic schema)
Speed	Read are fast	Both read and write are fast
Use case	OLTP (Online Transaction Processing)	Analytics (audio, video, logs), data discovery
Data objects	Works on relational tables	Works on key/value pairs
Scalability	Vertical	Horizontal
Hardware profile	High-end servers	Commodity / Utility hardware
Integrity	ACID	Low





What is MapReduce?

- Originated from Google, [OSDI'04]
  - MapReduce: Simplified Data Processing on Large Clusters
  - Jeffrey Dean and Sanjay Ghemawat (paper)
- Programming model for parallel data processing
- Hadoop can run MapReduce programs written in various languages:
  e.g. Java, Ruby, Python, C++
- For large-scale data processing
  - Exploits large set of commodity computers
  - Executes process in distributed manner
  - Offers high availability

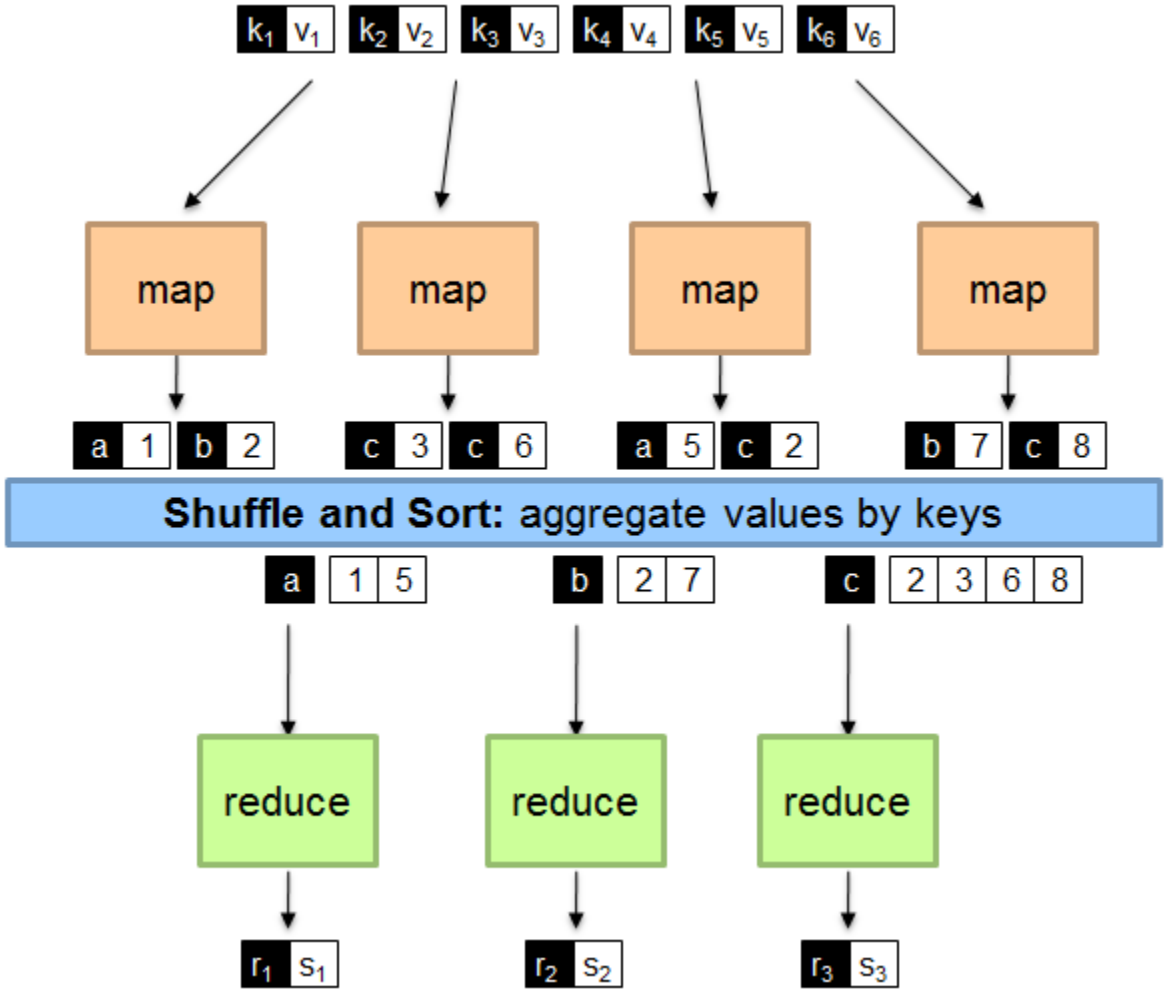
The Idea of MapReduce (1)

- Inspired by the map and reduce functions in functional programming
- We can view map as a transformation over a dataset
  - This transformation is specified by the function *f*
  - Each functional application happens in isolation
  - The application of *f* to each element of a dataset can be parallelized in a straightforward manner
- We can view reduce as an aggregation operation
  - The aggregation is defined by the function *g*
  - Data locality: elements in the list must be "brought together"
  - If we can group elements of the list, also the reduce phase can proceed in parallel
- The framework coordinates the map and reduce phases:
  - Grouping intermediate results happens in parallel

The Idea of MapReduce (2)

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles "data distribution"
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed file system (HDFS)
- You don't know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

WordCount - Mapper



Map and Reduce Functions

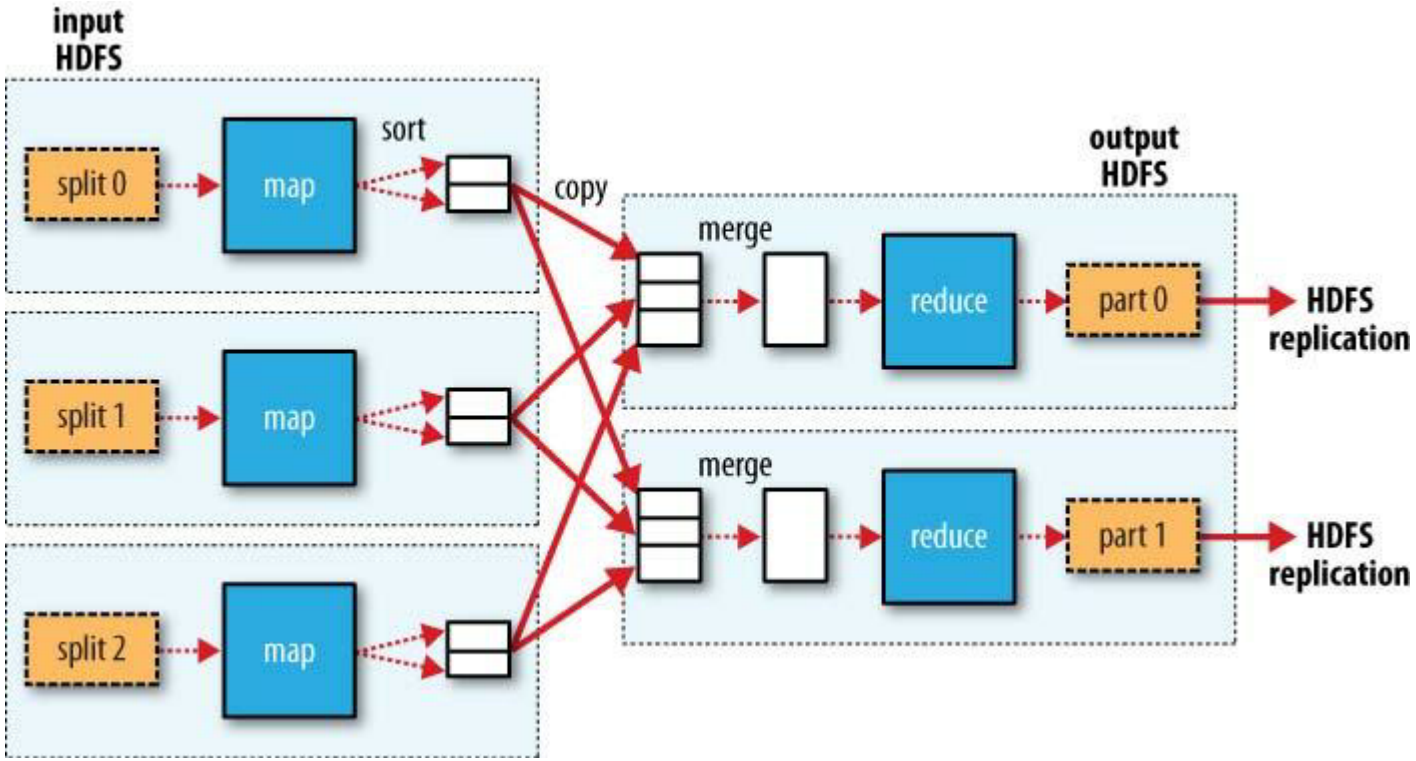
- Programmers specify two functions:
  - **map** (k_1, v_1) → list [<k_2, v_2>]
    - Map transforms the input into key-value pairs to process
  - **reduce** (k_2, list [v_2]) → [<k_3, v_3>]
    - Reduce aggregates the list of values for each key
    - All values with the same key are sent to the same reducer
  - list [<k_2, v_2>] will be grouped according to key k_2 as (k_2, list [v_2])

- The MapReduce environment takes in charge of everything else…

- A complex program can be decomposed as a succession of Map and Reduce tasks

More Detailed MapReduce Dataflow

- When there are multiple reducers, the map tasks partition their output:
  - One partition for each reduce task
  - The records for every key are all in a single partition
  - Partitioning can be controlled by a user-defined partitioning function

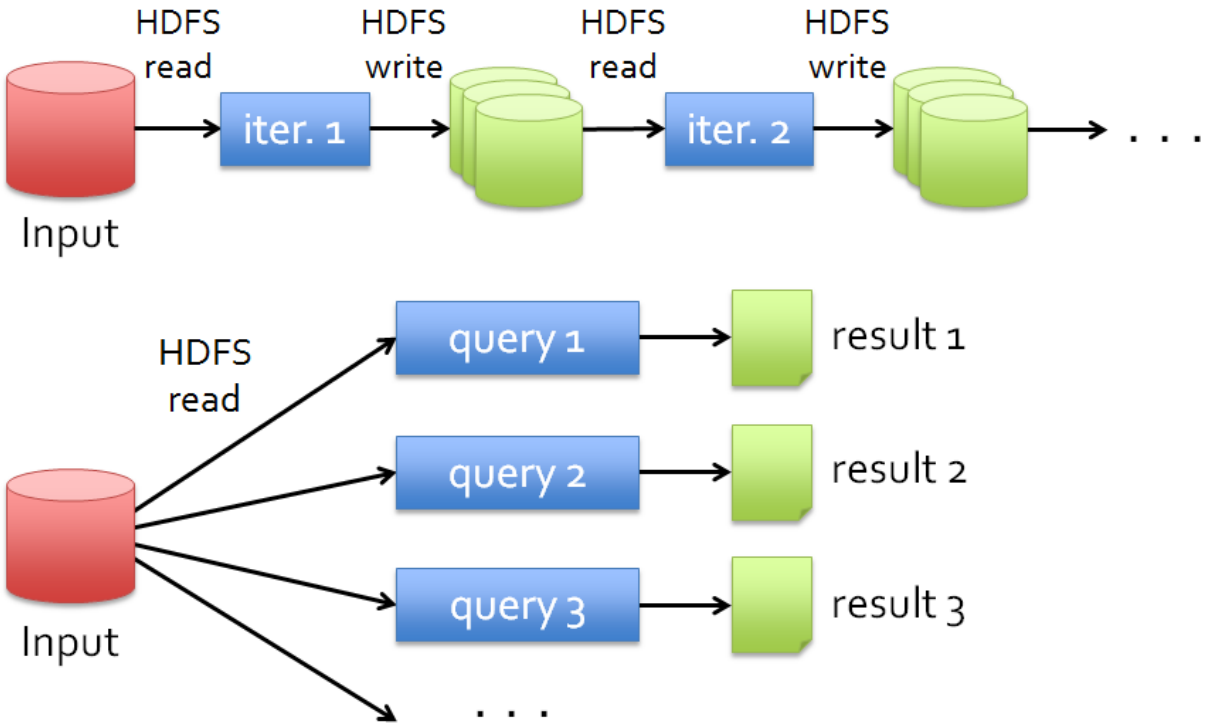


Apache Spark

Motivation of Spark

- MapReduce greatly simplified big data analysis on large, unreliable clusters. It is great at one-pass computation
- But as soon as it got popular, users wanted more:
  - More **complex**, multi-pass analytics (e.g. ML, graphs)
  - More **interactive** ad-hoc queries
  - More **real-time** stream processing
- All 3 need faster **data sharing** across parallel jobs
  - One reaction: specialized models for some of these apps, e.g.,
    - Pregel (graph processing)
    - Storm (stream processing)

Data Sharing in MapReduce



Slow due to replication, serialization, and disk IO

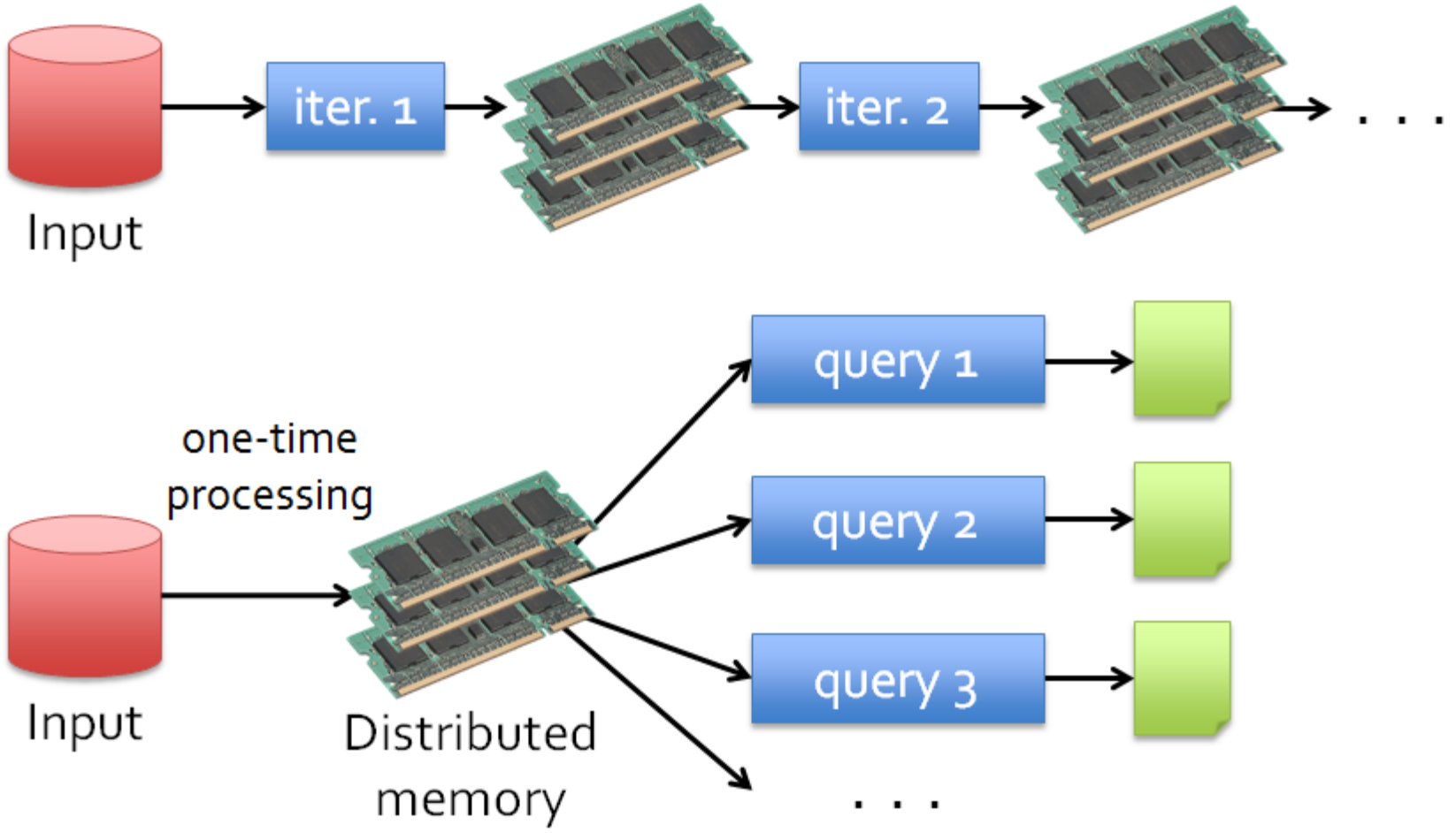
- Complex apps, streaming, and interactive queries all need one thing that MapReduce lacks:

  Efficient primitives for **data sharing**

Goals of Spark

- Keep more data in-memory to improve the performance!
- Extend the MapReduce model to better support two common classes of analytics apps:
  - Iterative algorithms (machine learning, graphs)
  - Interactive data mining
- Enhance programmability:
  - Integrate into Scala programming language
  - Allow interactive use from Scala interpreter

Data Sharing in Spark with RDDs (Resilient Distributed Dataset)



10-100× faster than network and disk

Solution: Resilient Distributed Datasets

- *Resilient Distributed Datasets (RDDs)*
  - Distributed collections of objects that can be cached in memory across cluster
  - Manipulated through parallel operators
  - Automatically recomputed on failure based on lineage
- RDDs can express many parallel algorithms, and capture many current programming models
  - Data flow models: MapReduce, SQL, …
  - Specialized models for iterative apps: Pregel, …

RDD Traits

- **In-Memory**, i.e. data inside RDD is stored in memory as much (size) and long (time) as possible
- **Immutable** or **Read-Only**, i.e. it does not change once created and can only be transformed using transformations to new RDDs
- **Lazy evaluated**, i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution
- **Cacheable**, i.e. you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed)
- **Parallel**, i.e. process data in parallel
- **Typed**, i.e. values in a RDD have types, e.g. RDD[Long] or RDD[(Int, String)]
- **Partitioned**, i.e. the data inside an RDD is partitioned (split into partitions) and then distributed across nodes in a cluster (one partition per JVM that may or may not correspond to a single node)

RDD Operations

- **Transformation:** returns a new RDD
  - Nothing gets evaluated when you call a Transformation function, it just takes an RDD and return a new RDD
  - Transformation functions include *map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, filter, join, etc.*
- **Action:** evaluates and returns a new value
  - When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned
  - Action operations include *reduce, collect, count, first, take, countByKey, foreach, saveAsTextFile, etc.*

How Spark Works?

- User application create RDDs, transform them, and run actions
- This results in a DAG (Directed Acyclic Graph) of operators
- DAG is compiled into stages
- Each stage is executed as a series of Task (one Task for each Partition)

```
val file = sc.textFile("hdfs://...")

val counts = file.flatMap(line => line.split(" "))
    .map(word => (word,1))
    .reduceByKey(_ + _)

counts.saveAsTextFile("hdfs://...")
```

Data Curation

Garbage in...

source

My “super cool” MapReduce or Spark solution

source

“around 20% of records in any data source are garbage.” *M. Stonebraker*

Data Curation is a must in any Big Data project!

What is data curation?

"Data curation is the process of identifying which data sources are needed, putting that data in the context of the business so that business users can interact with it, understand it, and use it to create their analysis."

https://www.tableau.com/reports/business-intelligence-trends/data-curation

Data Quality Dimensions

- Accessibility
- Appropriate amount of data
- Believability
- Completeness
- Concise representation
- Ease of manipulation
- Free-of-error
- Interpretability
- Objectivity
- Relevancy
- Reputation
- Security
- Timeliness
- Understandability
- Value-added

https://dl.acm.org/citation.cfm?id=506010

Data Curation

- Ingestion
- Validation
- Transformation
- Correction
- Consolidation
- Visualization

Data Tamer: A Scalable Data Curation System. Michael Stonebraker, Database Group, MIT CS and AI Lab, 2013

Generations of Data Curation Approaches

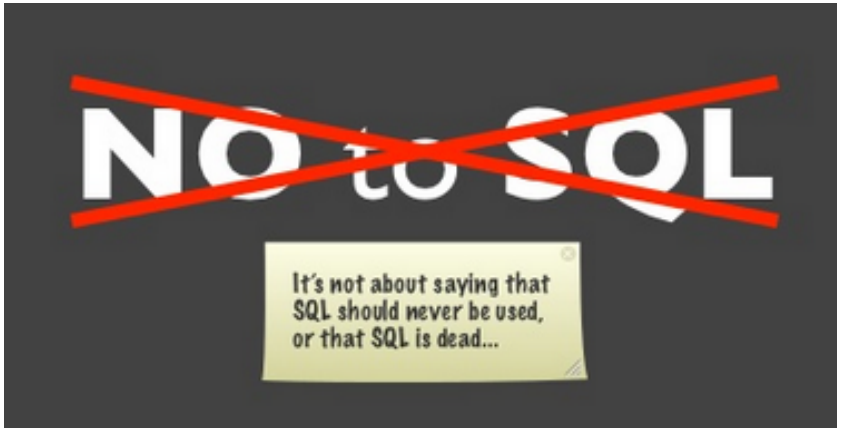
- 1st Generation (90s) -> Traditional Extraction-Transform-Load (ETL)
- 2nd Generation (2000s) -> ETL enhanced
- 3rd Generation (now) -> Scalable Data Curation

Scalable Data Curation
Michael Stonebraker, Strata + Hadoop Word, 2015, San Jose (CA)

NoSQL Technologies

What is NoSQL?

- The name stands for Not Only SQL
- Does not use SQL as querying language
- Class of non-relational data storage systems
- The term NOSQL was introduced by Eric Evans when an event was organized to discuss open source distributed databases
- It's not a replacement for a RDBMS but compliments it
- All NoSQL offerings relax one or more of the ACID properties (will talk about the CAP theorem)



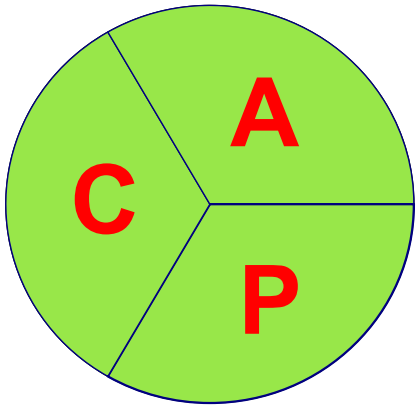
Why NoSQL?

- Web apps have different needs (than the apps that RDBMS were designed for)
  - Low and predictable response time (latency)
  - Scalability & elasticity (at low cost!)
  - High availability
  - Flexible schemas / semi-structured data
  - Geographic distribution (multiple datacenters)

- Web apps can (usually) do without
  - Transactions / strong consistency / integrity
  - Complex queries

CAP Theorem

- Consider these three properties of a distributed system (sharing data)
  - Consistency:
    - all copies have same value
  - Availability:
    - reads and writes always succeed
  - Partition-tolerance:
    - system properties (consistency and/or availability) hold even when network failures prevent some machines from communicating with others

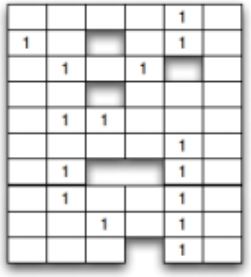


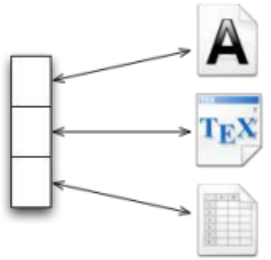
CAP Theorem

- Brewer's CAP Theorem:
  - *For any system sharing data, it is "impossible" to guarantee simultaneously all of these three properties*
  - You can have at most two of these three properties for any shared-data system
- Very large systems will "partition" at some point:
  - That leaves either **C** or **A** to choose from (traditional DBMS prefers **C** over **A** and **P**)
  - In almost all cases, you would choose **A** over **C** (except in specific applications such as order processing)

No SQL Taxonomy

- Key-Value stores
  - Simple K/V lookups (Distributed Hash Table (DHT))
- Column stores
  - Each key is associated with many attributes (columns)
  - NoSQL column stores are actually hybrid row/column stores
    - Different from “pure” relational column stores!
- Document stores
  - Store semi-structured documents (JSON)
- Graph databases
  - Neo4j, etc.
  - Not exactly NoSQL
    - can’t satisfy the requirements for High Availability and Scalability/Elasticity very well





What is HBase?

- HBase is an **open-source**, **distributed**, **column-oriented** database built on top of HDFS and based on BigTable
  - Distributed – uses HDFS for storage
  - Row/column store
  - Column-oriented and sparse - nulls do not occupy any storage space
  - Multi-Dimensional (versions)
  - Untyped - stores byte[]
- HBase is part of Apache Hadoop
- HBase is the Hadoop application to use when you require real-time, fast read/write random access to very large datasets
  - Aims to support low-latency random access

Elasticsearch

source

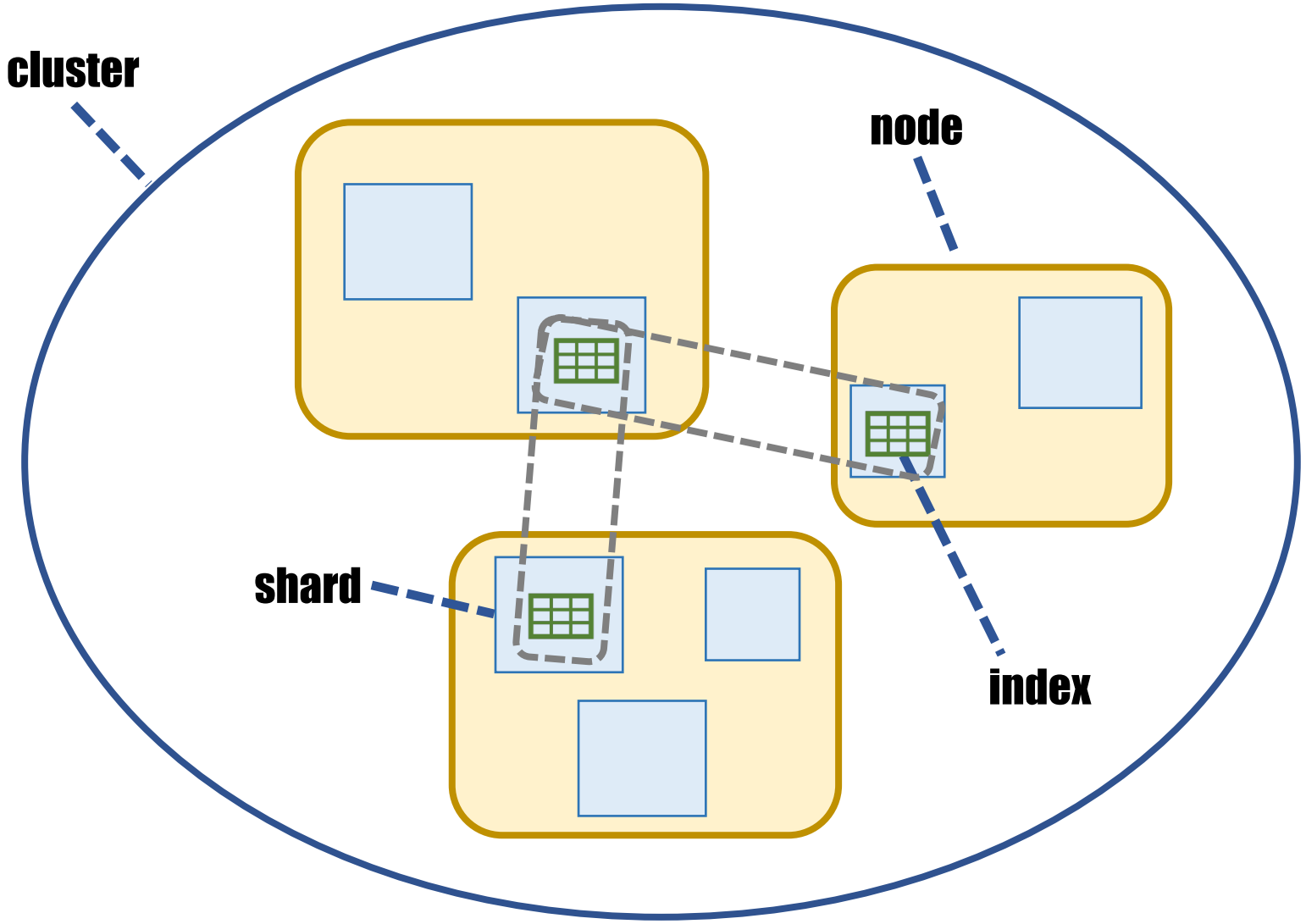
- Open source search engine based on Apache Lucene
- Initial release in 2010
- Provides a distributed, full-text search engine with a REST APIs
  - E.g. GET http://localhost:9200/person/student/8871

Elasticsearch

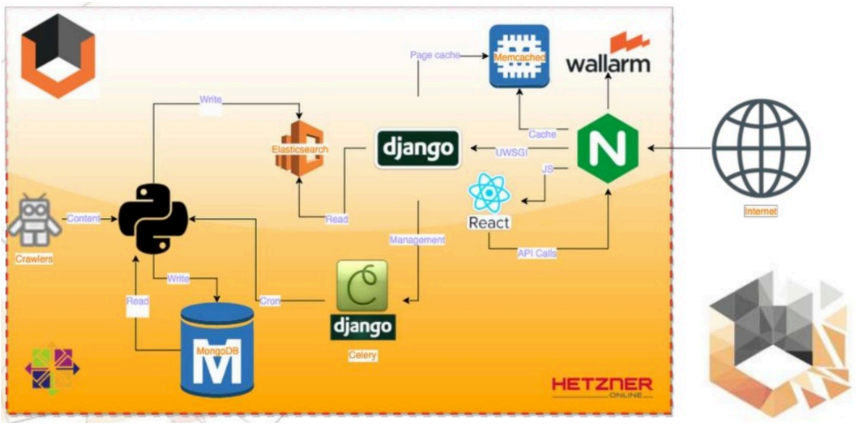
source

- Document oriented (JSON as serialization format for documents)
- Developed in Java (cross platform)
- Focused on scalability – distributed by design
- Highly efficient search

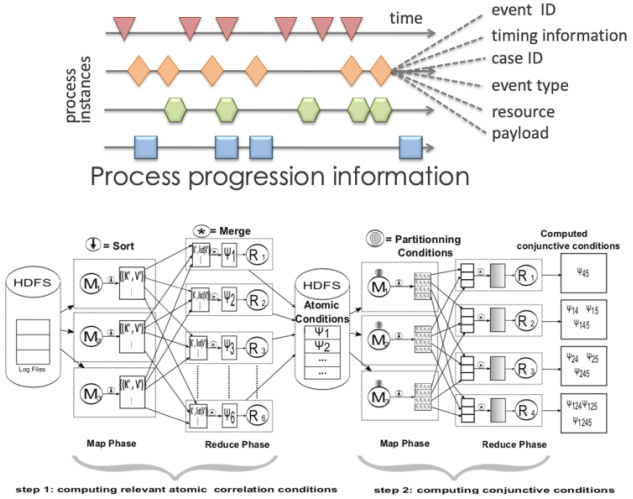
Elasticsearch Ecosystem



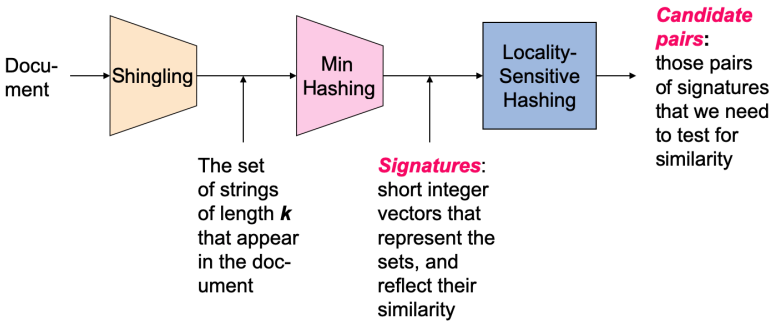
Application Scenarios and Use Cases



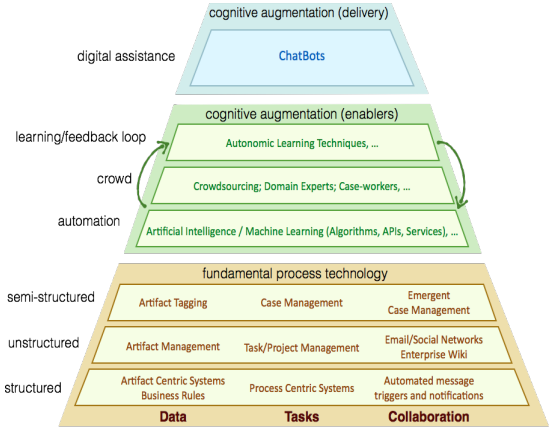
Cybersecurity



Process Mining



Finding Similar Items



Case Management and Augmented Intelligence

https://www.slideshare.net/AlexanderLeonov2/vulnerability-intelligence-and-assessment-with-vulnerscom

(*) Reguieg et al., 2012, September. Using mapreduce to scale events correlation discovery for business processes mining. BPM'12, (pp. 279-284). Springer, Berlin, Heidelberg

Thanks