University of Udine

Data Management for Big Data

Introduction to Big Data and NoSQL

Andrea Brunello
andrea.brunello@uniud.it

Big Data

Big data are beyond the usual limits of traditional databases, and are characterized by one or more of the properties:

- huge *Volume*
- high *Variety*
- acquired at high *Velocity*

Gartner analyst Doug Laney introduced the 3Vs concept in a 2001 MetaGroup research publication: "*3D data management: Controlling data volume, variety and velocity*"

Volume

Volume basically means the size of stored data, which may derive from human actions or can be machine-generated.

Sometimes the volume of data is so massive that they cannot be stored in their entirety, but have to be compressed/transformed online, as soon as they arrive (e.g., scientific sensor data).

Sometimes data can be stored using traditional RDBMS, while other times this choice may end up being too expensive in terms of cost or time $\rightsquigarrow$ NoSQL solutions, Hadoop.

Volume (Orders Of Magnitude)

IBM 350 disk storage (1956, 3.75 MB)

Walmart's DW (1992, 1 TB)

1 year of CERN's LHC data (15 PB)

Quantities of bytes						
Common prefix				Binary prefix		
Name	Symbol	Decimal SI	Binary JEDEC	Name	Symbol	Binary IEC
kilobyte	KB/kB	10^3	2^{10}	kibibyte	KiB	2^{10}
megabyte	MB	10^6	2^{20}	mebibyte	MiB	2^{20}
gigabyte	GB	10^9	2^{30}	gibibyte	GiB	2^{30}
terabyte	TB	10^{12}	2^{40}	tebibyte	TiB	2^{40}
petabyte	PB	10^{15}	2^{50}	pebibyte	PiB	2^{50}
exabyte	EB	10^{18}	2^{60}	exbibyte	EiB	2^{60}
zettabyte	ZB	10^{21}	2^{70}	zebibyte	ZiB	2^{70}
yottabyte	YB	10^{24}	2^{80}	yobibyte	YiB	2^{80}

In 2025, it is estimated that 175 ZB of data will be generated on the Internet: `https://en.wikipedia.org/wiki/Zettabyte_Era`

Variety

Differences between formats and the absence of a common structure are a typical characteristic of big data.

Structured, semi-structured, and unstructured data.

Data may come from different sources. Considering the web it may come from humans, like *user-generated content*, or it can be machine-generated, such as *logs, packet traces, etc.*

Heterogeneity of formats, structures and sources make it difficult to process and store such data using traditional tools.

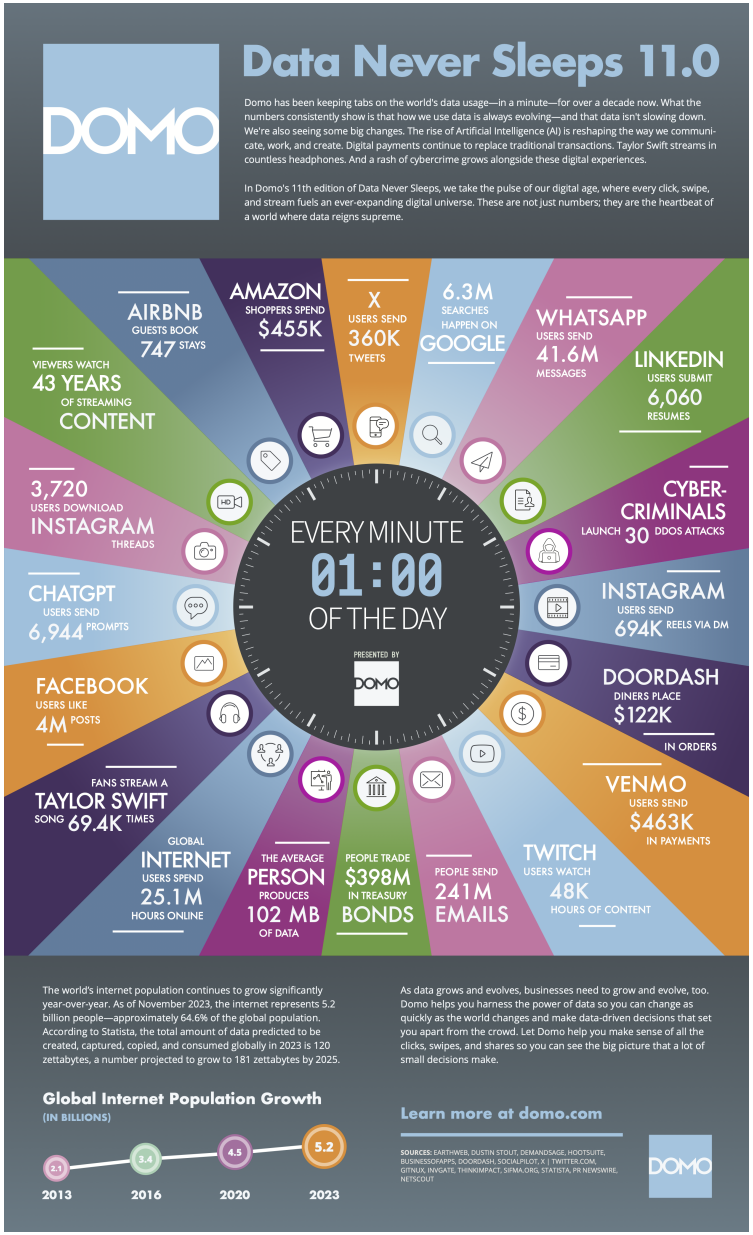
Velocity

Data acquired via sensors, or scientific instruments, may come at a high speed.

Some data have to be stored or analyzed as soon as they arrive, since they are transient (e.g., logs, data streams).

For companies that rely upon fast-generated data it is also important to exploit/analyze such data as fast as possible.

"Just in its 1st phase, the SKA telescope will produce some 160 TB of raw data per second that the supercomputers will need to handle."
`https://www.skatelescope.org/frequently-asked-questions/`


DOMO
Data Never Sleeps 11.0
Domo has been keeping tabs on the world's data usage—in a minute—for over a decade now. What the numbers consistently show is that how we use data is always evolving—and that data isn't slowing down. We're also seeing some big changes. The rise of Artificial Intelligence (AI) is reshaping the way we communicate, work, and create. Digital payments continue to replace traditional transactions. Taylor Swift streams in countless headphones. And a rash of cybercrime grows alongside these digital experiences.
In Domo's 11th edition of Data Never Sleeps, we take the pulse of our digital age, where every click, swipe, and stream fuels an ever-expanding digital universe. These are not just numbers; they are the heartbeat of a world where data reigns supreme.
AIRBNB GUESTS BOOK 747 STAYS
AMAZON SHOPPERS SPEND $455K
X USERS SEND 360K TWEETS
6.3M SEARCHES HAPPEN ON GOOGLE
WHATSAPP USERS SEND 41.6M MESSAGES
LINKEDIN USERS SUBMIT 6,060 RESUMES
VIEWERS WATCH 43 YEARS OF STREAMING CONTENT
3,720 USERS DOWNLOAD INSTAGRAM THREADS
CYBER-CRIMINALS LAUNCH 30 DDOS ATTACKS
EVERY MINUTE 01:00 OF THE DAY
PRESENTED BY DOMO
CHATGPT USERS SEND 6,944 PROMPTS
INSTAGRAM USERS SEND 694K REELS VIA DM
FACEBOOK USERS LIKE 4M POSTS
DOORDASH DINERS PLACE $122K IN ORDERS
FANS STREAM A TAYLOR SWIFT SONG 69.4K TIMES
VENMO USERS SEND $463K IN PAYMENTS
GLOBAL INTERNET USERS SPEND 25.1M HOURS ONLINE
THE AVERAGE PERSON PRODUCES 102 MB OF DATA
PEOPLE TRADE $398M IN TREASURY BONDS
PEOPLE SEND 241M EMAILS
TWITCH USERS WATCH 48K HOURS OF CONTENT
The world's internet population continues to grow significantly year-over-year. As of November 2023, the internet represents 5.2 billion people—approximately 64.6% of the global population. According to Statista, the total amount of data predicted to be created, captured, copied, and consumed globally in 2023 is 120 zettabytes, a number projected to grow to 181 zettabytes by 2025.
As data grows and evolves, businesses need to grow and evolve, too. Domo helps you harness the power of data so you can change as quickly as the world changes and make data-driven decisions that set you apart from the crowd. Let Domo help you make sense of all the clicks, swipes, and shares so you can see the big picture that a lot of small decisions make.
Global Internet Population Growth
(IN BILLIONS)
2.1
3.4
4.5
5.2
2013
2016
2020
2023
Learn more at domo.com
SOURCES: EARTHWEB, DUSTIN STOUT, DEMANDSAGE, HOOTSUITE, BUSINESSOFAPPS, DOORDASH, SOCIALPILOT, X | TWITTER.COM, GITNUX, INVGATE, THINKIMPACT, SIFMA.ORG, STATISTA, PR NEWSWIRE, NETSCOUT
DOMO

Velocity

Data *velocity* is rarely a static metric.

Internal and external changes to a system and to the context in which it operates can have a considerable impact on the rate at which data have to be managed.

Variable velocity, coupled with large volume, requires data stores to handle **sustained levels** of high read and write loads, and also **peaks**.

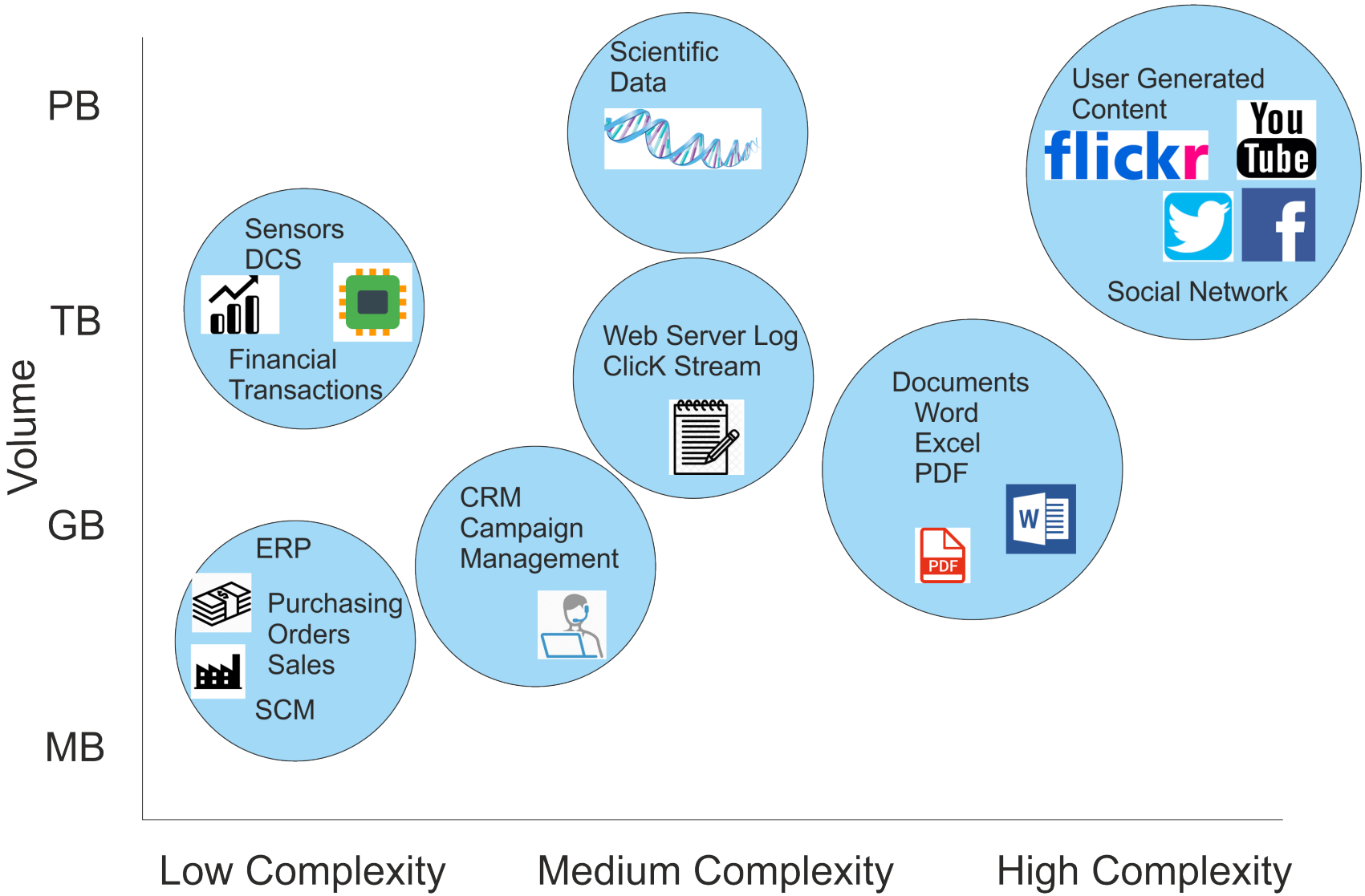
Big Data
Additional Vs

- **Volume**: scale of the data
- **Variety**: different forms of data
- **Velocity**: e.g., analysis of streaming data
- **Variability**: changes in the characteristics of the data
- **Value**: revenues, hypotheses that may arise from the data
- **Veracity**: trustworthiness, origin and reputation

Big Data
And Even More Vs...

- **7 Vs:** *https://impact.com/marketing-intelligence/7-vs-big-data/*
- **10 Vs:** *tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx*
- **17 Vs**: *https://www.irjet.net/archives/V4/i9/IRJET-V4I957.pdf*
- **42 Vs:** *https://www.kdnuggets.com/2017/04/42-vs-big-data-data-science.html*

Big Data
Classification by Volume and Complexity



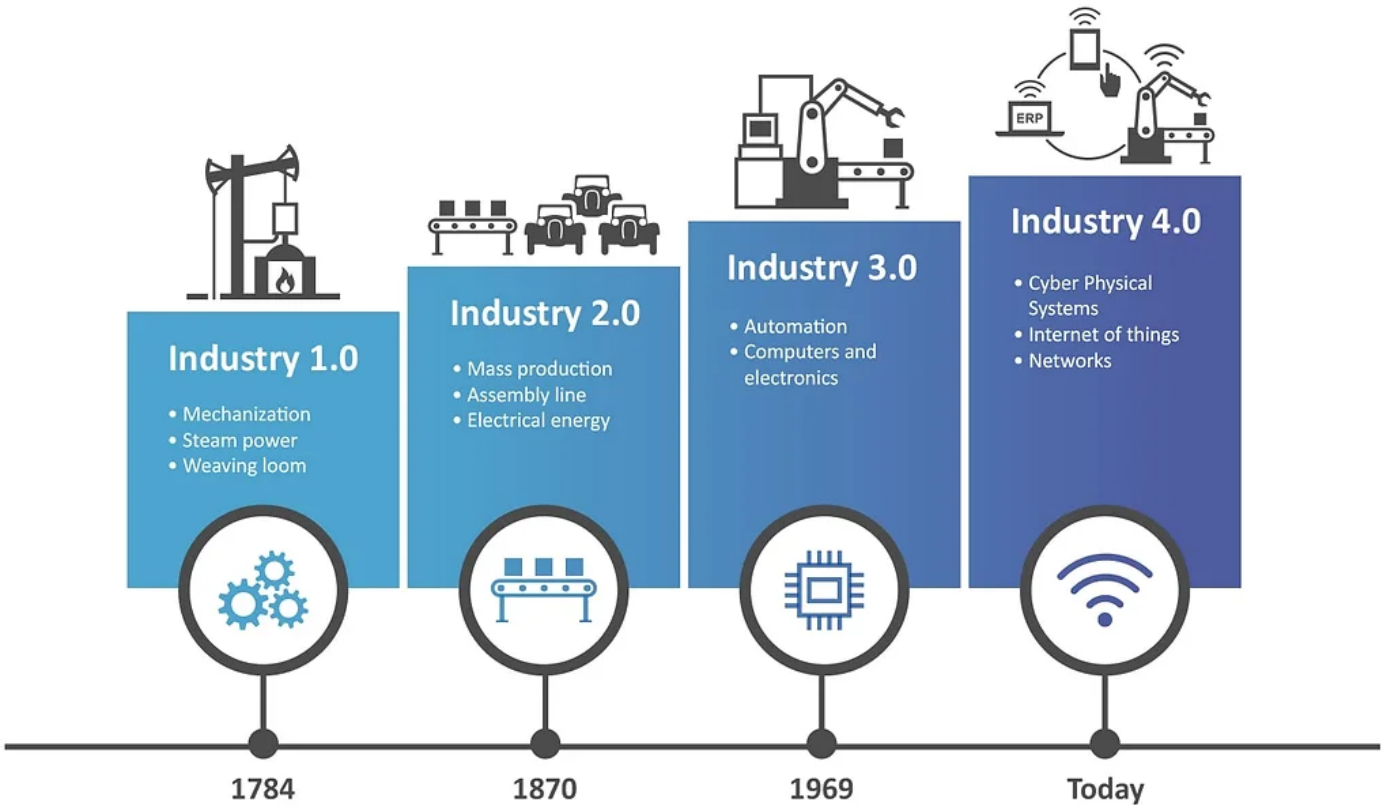
Big Data Use Cases
Data from the Web

Data from the Web plays a big role in big data realm, and are characterized by volume, variety and velocity. Examples:

- HTML pages (in any language)
- Tweets
- Social network content (Facebook, LinkedIn, etc.)
- Forum comments and blog posts
- Documents in several formats: XML, PDF, Word, Excel, etc.

The Internet Of Things (IoT) is about (daily life) objects that are equipped with sensors and connectivity, acting as sources of data

- the concept can involve both industry and consumer products (like connected automobiles, machinery, etc.)
- data can be used for tasks such as surveillance, predictive maintenance, or performance enhancement in general
- if objects are attached to people even human behaviour and well-being can be analyzed


Industry 1.0
• Mechanization
• Steam power
• Weaving loom
Industry 2.0
• Mass production
• Assembly line
• Electrical energy
Industry 3.0
• Automation
• Computers and electronics
Industry 4.0
• Cyber Physical Systems
• Internet of things
• Networks
ERP
1784
1870
1969
Today

Big data come with challenges and opportunities:

- *business:* big data give companies the opportunity to develop new business models so to get advantages with respect to competitors
- *technology:* size and complexity of big data require adequate solutions
- *financial aspects:* several use cases show that exploiting big data may lead to economic benefits. To this end, it is also important evaluate the costs involved with their management (e.g., cloud solutions)
- *privacy:* who owns the data?

NoSQL

What is NoSQL?

The term NoSQL (Not only SQL) refers to data stores that are **not relational databases**, rather than explicitly describing what they are.

A possible (rather general) definition: *"Next Generation DBs mostly addressing some of the points: being non-relational, distributed, open source and horizontally scalable"*.

NoSQL storage technologies have very **heterogeneous** operational, functional, and architectural characteristics (*one size does not fit all*).

NoSQL proposals have been developed starting from 2009 trying to address new challenges that emerged in that decade.

The Rise of NoSQL

Most enterprise level applications used to run on top of relational databases (MySQL, PostgreSQL, etc).

Over the years data got bigger in volume, started to change more rapidly, and to be in general more structurally varied than those commonly handled with traditional RDBMS.

As the *volume* of data increases dramatically, so does query execution time, as a consequence of the size of tables involved and of an increased number of JOIN operations (*join pain*).

Benefits of NoSQL

High volume and high heterogeneity data:

- RDBMS: capacity and constraints at their limits
- NoSQL are specifically designed for this scenario

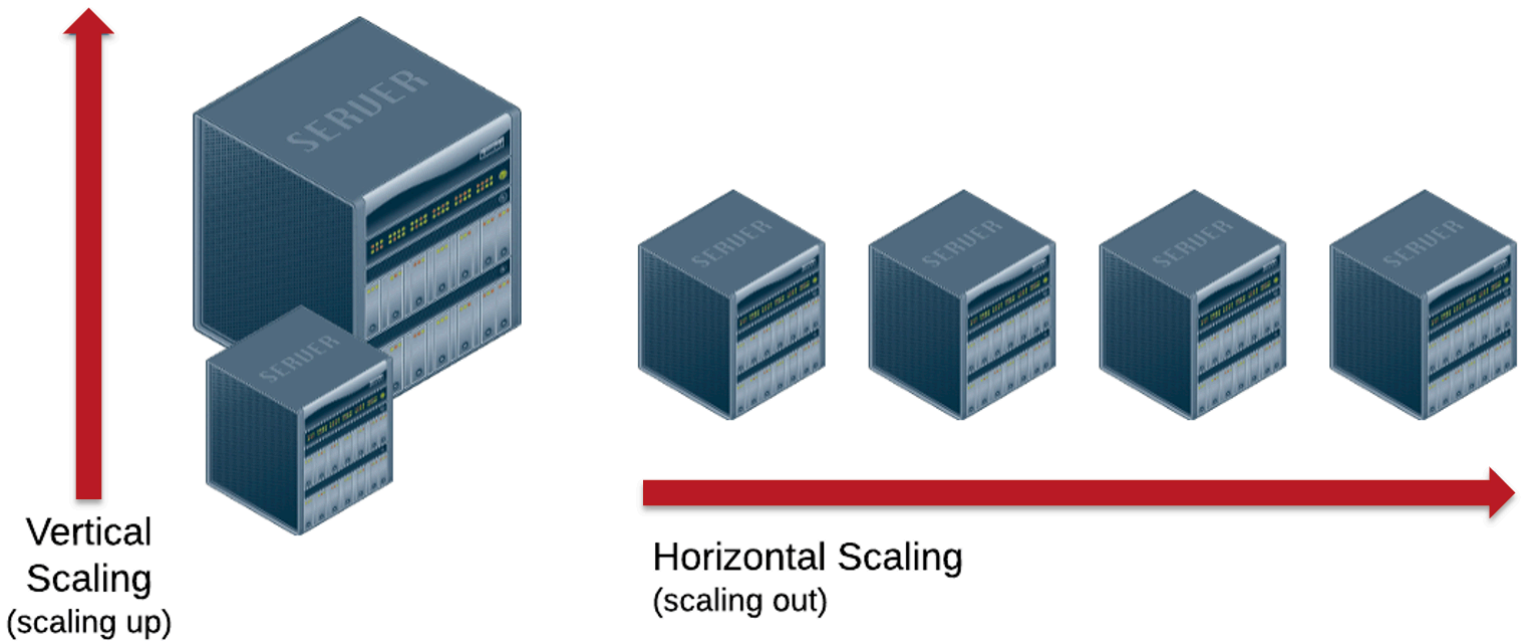
Elastic Scaling:

- RDBMS scale up: bigger load $\Rightarrow$ bigger server
- NoSQL scale out: distribute data across multiple nodes, adding/removing them seamlessly

DBA Specialists:

- RDBMS require highly trained experts to monitor DB
- NoSQL require less management: automatic repair and simpler data models

Benefits of NoSQL
Vertical vs horizontal scaling



Drawbacks of NoSQL

Several default constraints of RDBMS are not supported at the database level (e.g., foreign keys).

If not properly managed, the absence of a fixed structure may become an issue.

The design process is not as straightforward and consolidated as in the relational model.

Lack of SQL (though there are some SQL-inspired languages).

Often, lack of one or more ACID properties.

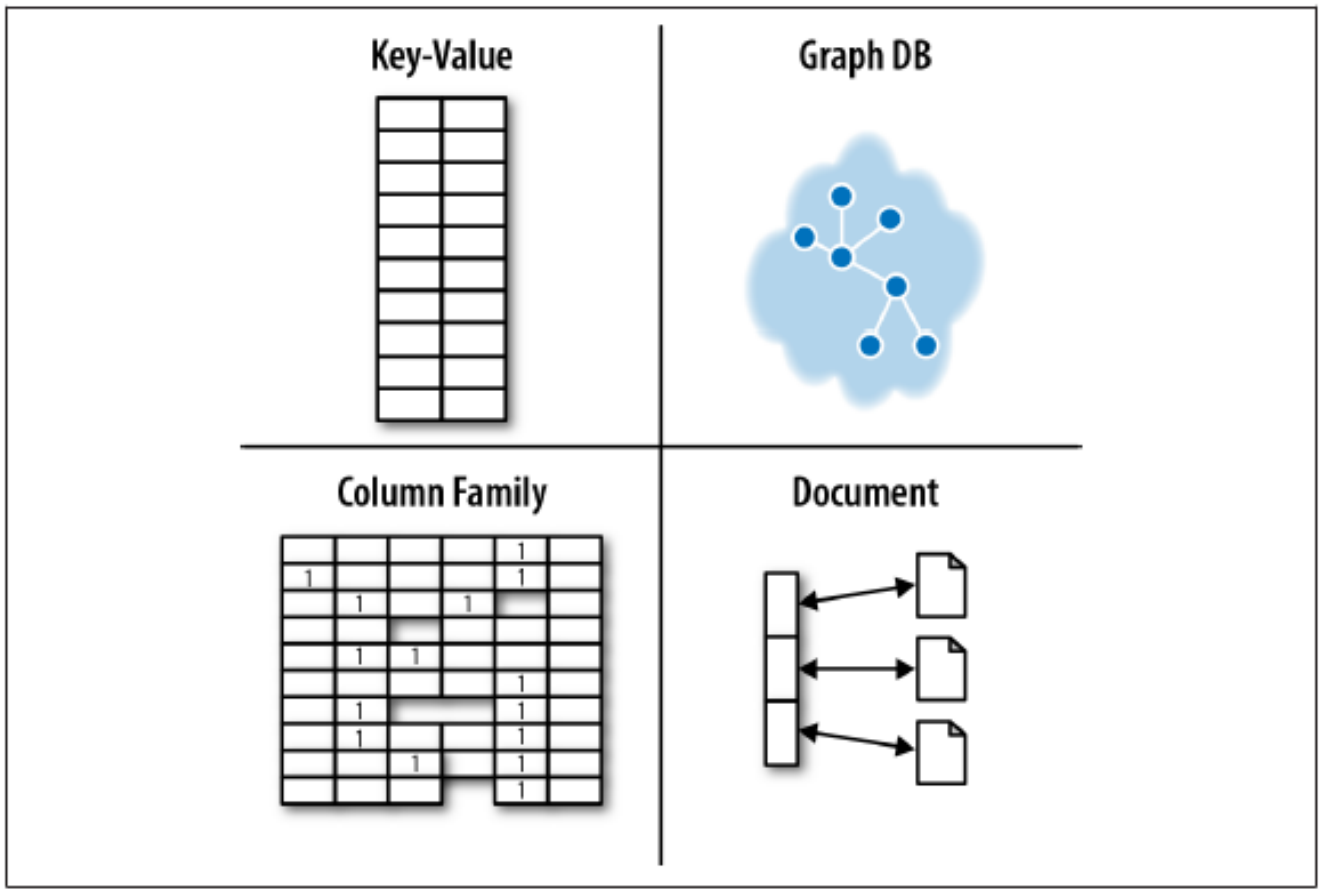
BASE Properties

In order to better support the characteristics of the novel kinds of data, NoSQL systems rely on the BASE properties instead:

- *Basically available* The store appears to be accessible most of the time (for instance, tolerance to node failures)
- *Soft state* Stores do not need to be write-consistent, nor do different replicas have to be mutually consistent all the time
- *Eventual consistency* The system eventually exhibits consistency at some later point

Of course, this kind of relaxed consistency would be a problem, for instance, in a banking database. In other cases, it is perfectly acceptable, e.g., for social networks.

The NoSQL Quadrant



Key/Value Stores

Key/value stores are based on the concept of *associative array* (i.e., dictionary, or map), a data structure that contains couples of <key, values>; the key is used to access the values.

E.g., in a database storing personal data, one array could have as keys the SSN of customers associated to their phone numbers as values, while in another array the same keys could be associated with their profile pictures.

Operations allowed over an associative array are:

- *Add*: add a new pair to the array
- *Remove*: remove a pair from the array, based on the key
- *Modify*: change the value associated to a key
- *Find*: search for a value by the key

Key/Value Store - Focus

Key/values stores offer just the add/remove/modify/find operations on data, which nevertheless are executed at a fixed (fast) computational cost, thanks to the associative array.

A value can be anything and of any size, but typically is totally opaque for the user (schemaless approach).

Some typical relational operations, such as complex filters and JOINS are not possible (unless a custom script is written).

Also, important RDBMS functionalities like foreign keys are not supported.

Key/value stores are very simple pertaining to to their data model, but they can be extremely sophisticated regarding horizontal scalability.

Key/Value Store - Focus

Array "emp_name":
[<XA13, John Doe>, <XZ42, Mary Jane>, <XG56, Jhonny Donny>, . . .]

Array "emp_department":
[<XA13, Physics>, <XZ42, Physics>, <XG56, Computer Science>, . . .]

Array "emp_salary":
[<XA13, 45000>, <XZ42, 50000>, <XG56, 52000>, . . .]

Array "dept_budget":
[<Physics, 950000>, <Computer Science, 780000>, . . .]

By convention, even if opaque and schemaless, all the values within a same array should be of a same kind.

Key/Value Store - Examples

Use cases:

- Storage of user profiles, or session data on the Web
- Storage of multimedia objects
- Storage of machinery or sensors data

Exemplary DBMSs:

- **DynamoDB** developed by Amazon and available on the AWS (Amazon Web Services) platform
- **Project Voldemort** is a (LinkedIn-born) distributed key/value store based on Berkeley DB, designed for performance and fault protection
- **Redis** (in-memory database): `https://redis.io/`
- **Aerospike**: `https://aerospike.com/`

Document-oriented Databases

Document databases store, retrieve and manage document oriented information, also known as semi-structured data.

Documents do not have a fixed structure, nonetheless they contain **tags** or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, they can be considered as a kind of **self-describing structure**.

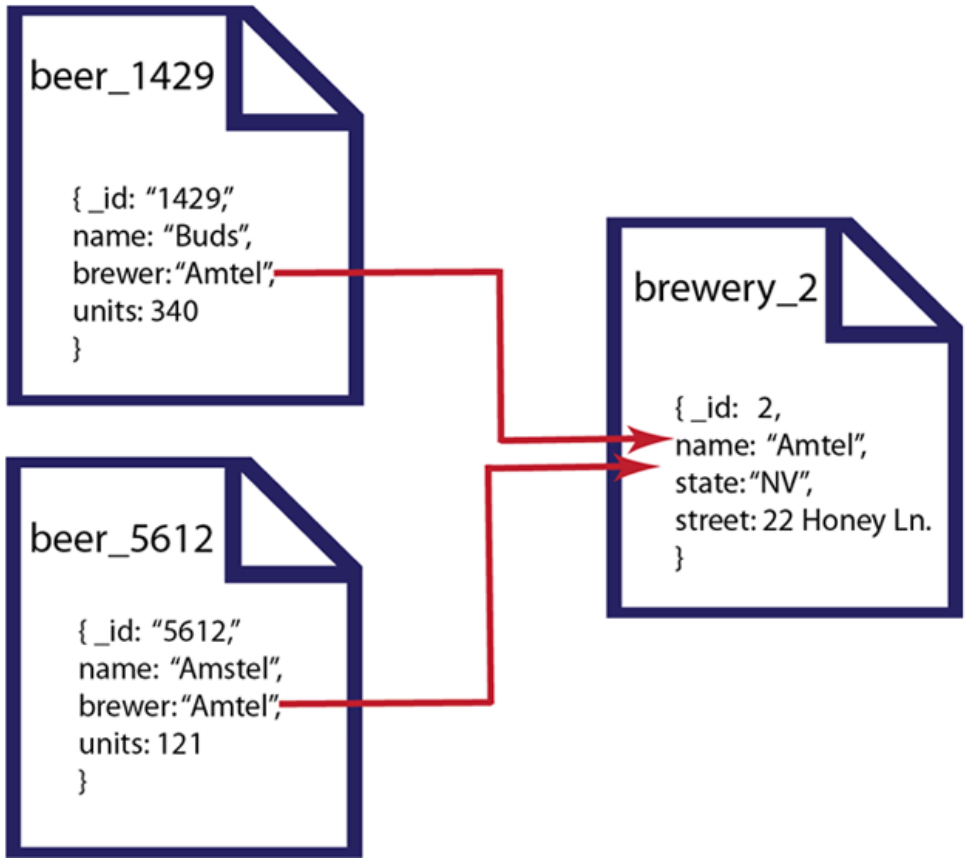
Documents can be encoded into formats like XML and JSON.

Document stores represent a step up with respect to key-value stores, defining a structure over keys and values, thus allowing the users to operate on the internal document structure.

Document example

```
<Phone campaign>
    <Survey ID = 1>
        <Name> John Easter </Name>
        <Question ID = 1> yes </Question>
        <Question ID = 2> yes </Question>
        <Question ID = 2.1> 5 </Question>
        <Question ID = 2.2> 7 </Question>
        <Question ID = 3> no </Question>
    </Survey>
    <Survey ID = 2>
        <Name> Mary Christmas </Name>
        <Question ID = 1> yes </Question>
        <Question ID = 2> no </Question>
        <Question ID = 3> yes </Question>
        <Question ID = 3.1> yes </Question>
        <Note> The person seems to be rather interested in the offered product </Note>
    </Survey>
</Phone campaign>
```

Document example



Custom scripts are still often required to perform JOIN operations or to enforce data consistency between documents

Documents and relationships

Relationships are represented by nesting documents: *"Data that is accessed together should be stored together"*

USERS

ID	first_name	last_name	cell	city	latitude	longitude
1	Leslie	Yepp	8125552344	Pawnee	39.170344	-86.536632

HOBBIES

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working

JOB HISTORY

ID	user_id	job_title	year_started
20	1	Deputy Directory	2004
21	1	City Councilor	2012
22	1	Director, National Parks Service, Midwest Branch	2014

USERS

```
{
   "_id": 1,
   "first_name": "Leslie",
   "last_name": "Yepp",
   "cell": "8125552344",
   "city": "Pawnee",
   "location": [ -86.536632,
                 39.170344 ],
   "hobbies": [
     "scrapbooking",
     "eating waffles",
     "working"
   ],
   "jobHistory": [
     {
       "title": "Deputy Director",
       "yearStarted": 2004
     },
     {
       "title": "City Councillor",
       "yearStarted": 2012
     },
     {
       "title": "Director, National Parks
                 Service, Midwest Branch",
       "yearStarted": 2014
     }
   ]
}
```

Document-oriented Databases – Operations

The core operations that a document-oriented database supports for documents are similar to other databases, and are named as CRUD:

- *Creation*: of a new document
- *Retrieval*: based on key, content, or metadata
- *Update*: the content or metadata of the document
- *Deletion*: of a document

Typical use cases: similar to key-value stores, but more complex data can be handled (product data management, inventory).

→ **MongoDB** is an example of a document-oriented database

- It actually offers a very crude join operation

Column-oriented Stores

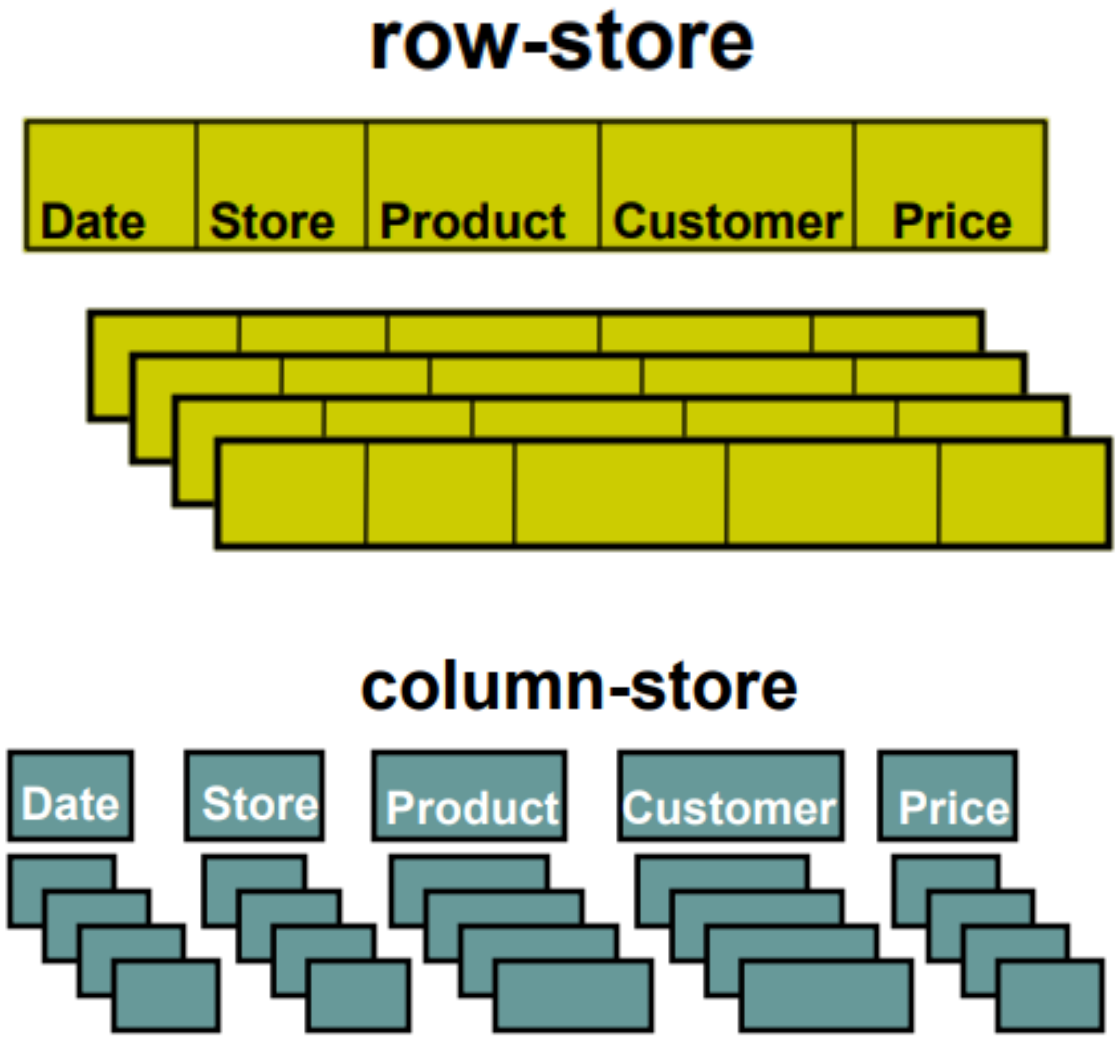
The roots of colum-store DBMSs can be traced back in the 1970s.

However, due to market needs and non favorable-technology trends it was not until the 2000s that they took off.

A column-oriented DBMS stores each database table column separately, in different disk locations, or even different machines.

Values belonging to the same column are packed together, as opposed to traditional DBMSs that store entire rows one after the other.

Rows vs Columns



Rows vs Columns

In a classical relational database each field is stored adjacent to the next in the same block on the hard drive:

```
512,Seabiscuit,Book,10.95,201712241200,goodreads.com
513,Bowler,Apparel,59.95,201712241200,google.com
514,Cuphead,Game,20.00,201712241201,gamerassaultweekly.com
```

In a columnar database, blocks on the disk for the above data might look like this:

```
512,513,514
Seabiscuit,Bowler,Cuphead
Book,Apparel,Game
10.95,59.95,20.00
201712241200,201712241200,201712241201
goodreads.com,google.com,gamerassaultweekly.com
```

Column Store Pros

Beneficial when the typical application is reading a subset of columns, or performing **aggregate functions** over them (AVG, MIN, MAX, ...).

Efficient storing capabilities due to an easier **compression of data**, which is performed by column (low information entropy).

Very scalable, and well suited for *massively parallel processing* (MPP), which involves having data partitioned and spread across a large cluster of machines.

Column Store Cons

The main drawbacks of column-based stores are related to write operations and tuple (re)construction.

Newly inserted tuples have to be **broken down** to their component attributes, and each attribute must be written separately.

Also tuple construction is considered problematic, since information about a logical entity is stored in multiple locations, which brings an **overhead** for queries that access many attributes from an entity.

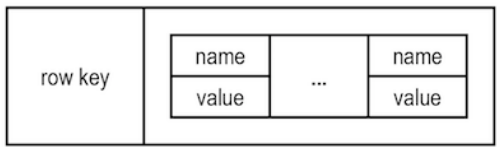
Column vs column-family stores

A column-oriented store *is not the same* as a column-family one.

A column-oriented database simply stores data tables by column rather than by row, as we have seen.

→ **Apache Druid**, **MonetDB** and **Vertica** are examples of column-oriented stores.

A column-family database sees a row as a varying collection of columns, intuitively like a two-level key-value store (thus, it is simpler, but more efficient than a document store).



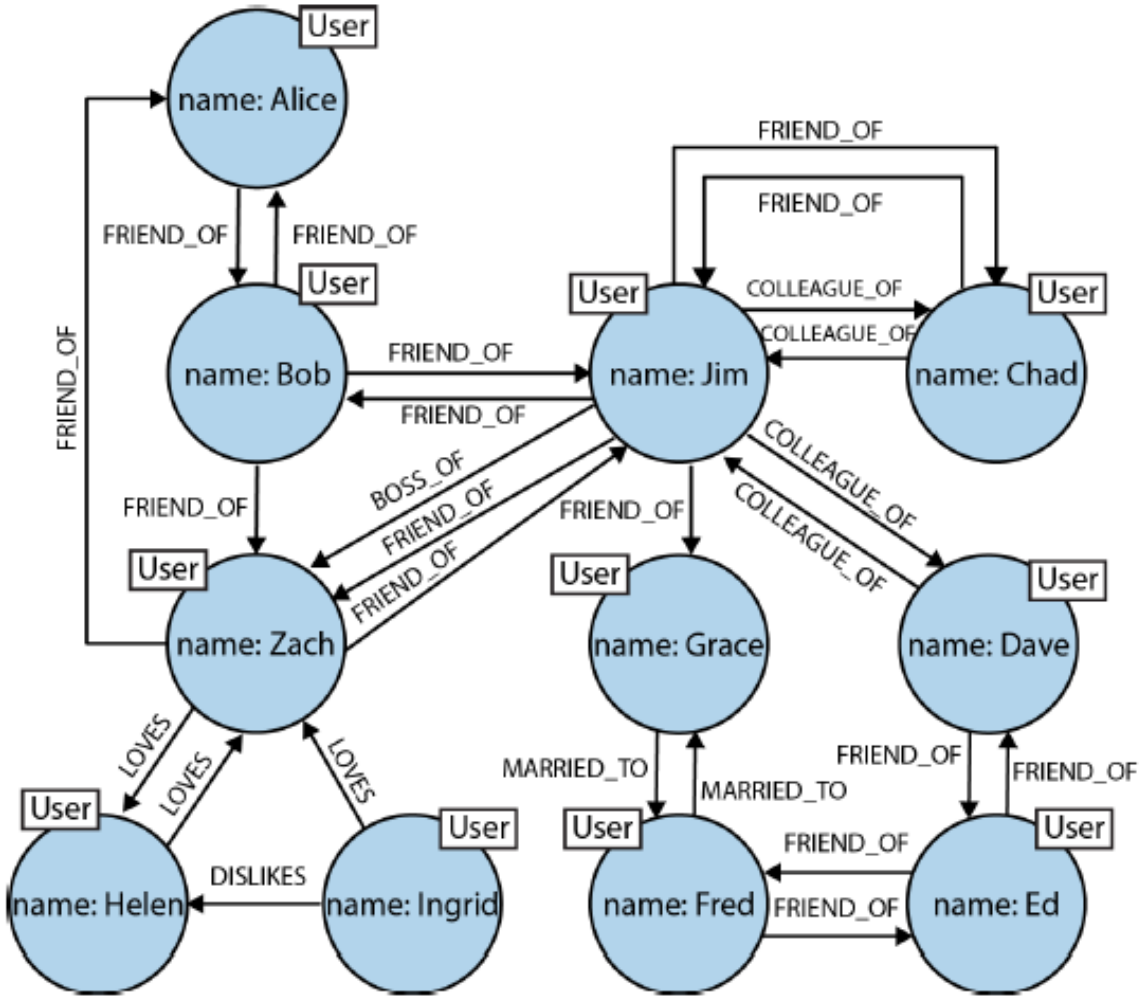
→ **Cassandra**, **Apache Hbase** and **Google BigTable** are examples of column-family stores (aka wide-column stores).

Graph Databases

A graph database is a database that uses graph structures for semantic queries with *nodes* (possibly of different kinds), *edges*, and *properties* to represent and store data.

Graph databases can naturally represent certain kinds of semi-structured, **highly interconnected data**, such as those present in social networks, or in geospatial and biotech applications.

Graph Example



Graph Databases – Why

Of course, the previous graph could be modeled using other kinds of NoSQL approaches, as well as RDBMSs.

Nevertheless, the resulting databases would be very difficult to query, update, and populate.

On the contrary, graph databases can easily answer to queries such as:

- what is the shortest path connecting node *X* with node *Y*?
- what are the friends of *Billy*?
- what are the friends of friends of *Billy*?

In native implementations, nodes have **direct pointers** connecting them, so the navigation time is independent from the graph size.

Examples of Graph Databases

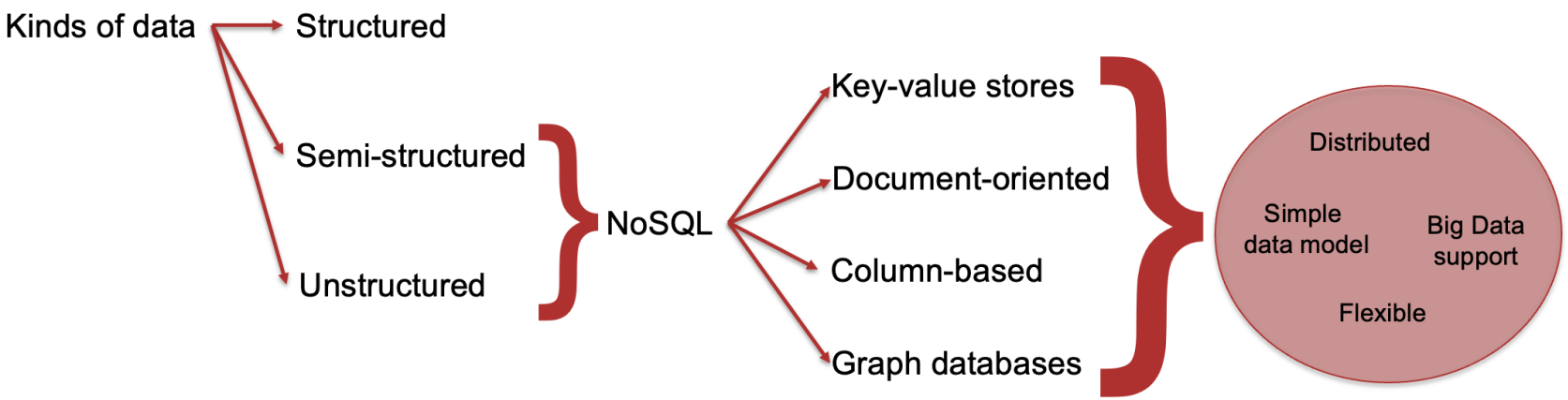
Neo4J is a very performing, native, open source property graph database that guarantees ACID properties offering scalability up to billions of nodes.

- Online sandbox: `https://neo4j.com/sandbox/`

AllegroGraph is a closed source triplestore, designed to store RDF triples; it supports ACID properties.

ArangoDB is a multi-model, open source, database system that supports three data models (key/value, documents, graphs).

Recap


Kinds of data
Structured
Semi-structured
Unstructured
NoSQL
Key-value stores
Document-oriented
Column-based
Graph databases
Distributed
Simple data model
Big Data support
Flexible

The twilight zone: NewSQL

In recent years there has been a progressive **(re)approach between relational solutions and NoSQL**, with the birth of the so-called NewSQL

NewSQL DBMSs try to combine the performances of NoSQL with the data guarantees offered by relational DBs

For example, there are NoSQL systems that support transactions (Neo4j); others offer languages inspired by SQL, but generally less powerful than the latter (Cassandra's CQL)

Conversely, relational DBMSs allow, today, to store semi-structured data inside tables, thanks to the JSON and JSONB formats, while others allow to store time series data generated quickly and in large quantities (TimescaleDB)

Bibliography

Stavros Harizopoulos, Daniel Abadi, Peter Boncz (2009), Column-Oriented Database Systems, VLDB 2009 Tutorial

Ian Robinson, Jim Webber & Emil Eifrem, Graph Databases Second Edition, O'Reilly Media, Inc.

Big Data Management and NoSQL Databases Lecture 7. Column-family stores, Doc. RNDr. Irena Holubova, Ph.D.

Andrew Pavlo, Matthew Aslett (2016), What's really new with NewSQL?, ACM Sigmod Record, 45(2), 45-55