Big Data Processing and Applications

Ioana G. CIUCIU
ioana.ciuciu@ubbcluj.ro

Lecture 4-5 - Data systems and the lambda architecture for big data

Babes-Bolyai University, Cluj-Napoca

24,30 Oct. 2023

Agenda

Data Systems Beyond the Relational Model

- Recall on traditional database systems
  - DBMS characteristics
  - ACID database properties
  - Data models
  - The non-relational data model
  - Database systems vs. data warehouse
  - Limitations of traditional data systems

- Big Data Systems
  - What are Big Data Systems?
  - Big Data Systems: Requirements
  - CAP theorem
  - Lambda architecture for Big Data

Course structure

Date	Course/Week	Title
03.10.2025	1	Introduction to Data Science and Big Data (Part 1)
10.10.2025	2	Introduction to Data Science and Big Data (Part 2)
17.10.2025	3	Data systems and the lambda architecture for big data
	4	Industrial standards for data mining projects. Big data case studies from industry – invited lecture from Bosch
	5	Lambda architecture: batch layer
	6	Lambda architecture: serving layer
	7	Lambda architecture: speed layer
	8	NoSQL Solutions for Big Data – invited lecturer from UBB
	9	Data Ingestion
	10	Introduction to SPARK – invited lecture from Bosch
	11	Data visualization
	12	Presentation research essays
	13	Presentation research essays + Project Evaluation during Seminar
	14	Presentation research essays + Project Evaluation during Seminar

Slight modifications in the structure are possible

Semester Project

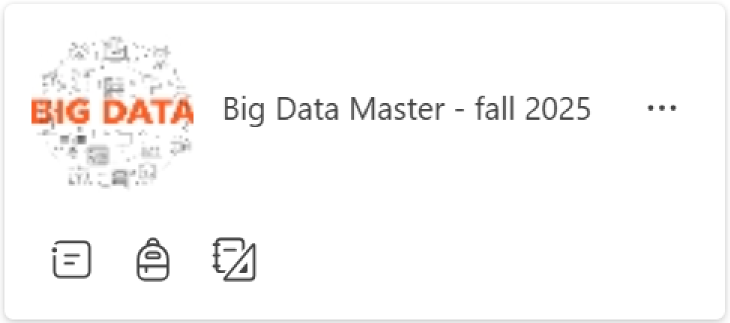
- Team-based (2-5 students with precise roles)
- Multidisciplinary: 3 CS Master Programes (HPC, SI, DS) + Master in Bioinformatics
- Real use cases: collaboration with local IT industry - TBA
- High degree of autonomy in selecting the topic and proposing the solution
- Implementation prototype
- Prototype demo & evaluation – student workshop (last seminar – weeks 13 & 14)
- *Best projects – disseminated in various events (TBD), scientifically disseminated (workshops/conferences/journals) AND/ OR possibility for a dissertation thesis*

Evaluation

- The final grade will be computed as follows:
  - 50% semester project (must be >= 5)
  - 50% research presentation or written exam (must be >= 5)

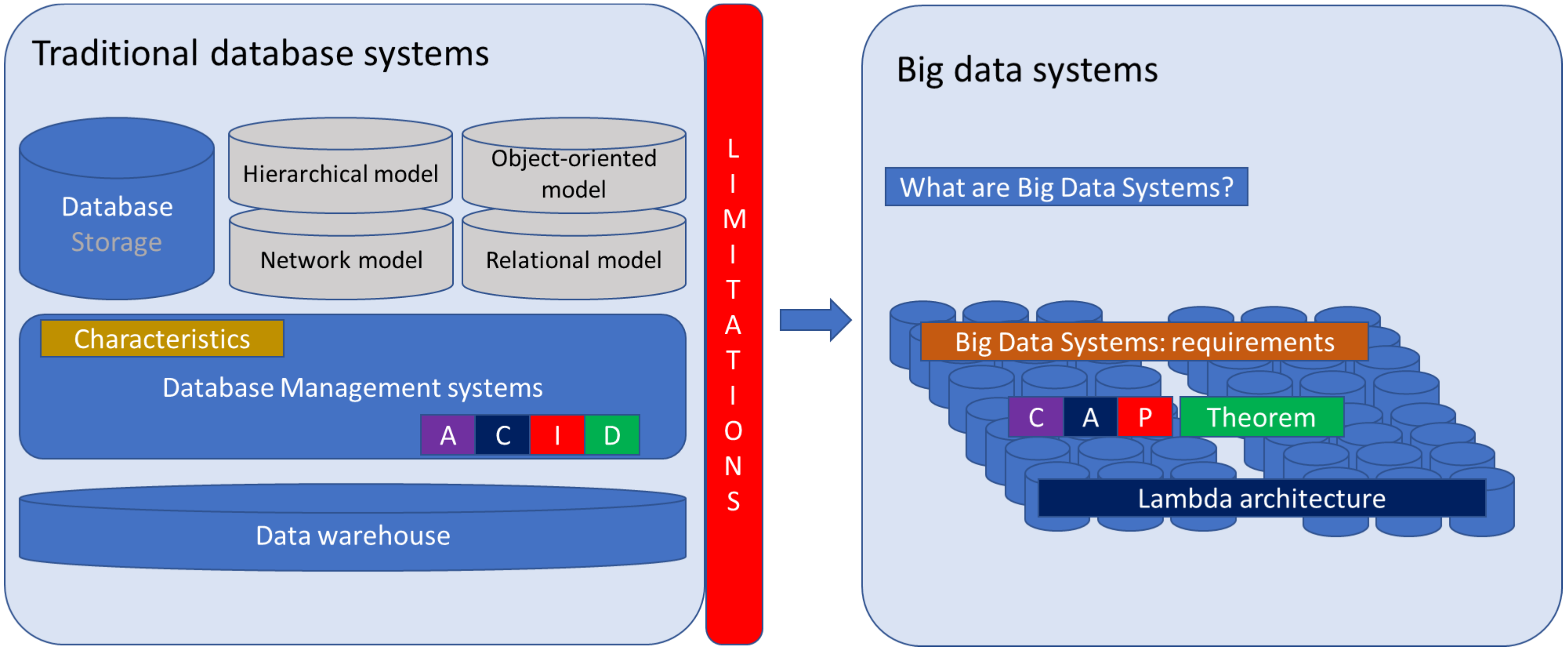
- Semester project

  - Details available on the course team
    - MS Team: **Big Data Master - fall 2025**
    - Access code: **j1exenq**



Big Data Master - fall 2025

Agenda


Traditional database systems
Database
Storage
Hierarchical model
Object-oriented model
Network model
Relational model
Characteristics
Database Management systems
A
C
I
D
Data warehouse
LIMITATIONS
Big data systems
What are Big Data Systems?
Big Data Systems: requirements
C
A
P
Theorem
Lambda architecture

Traditional database systems: recall

Data and databases

Feedback required

QUIZ

- What is data?
  - The practice of collecting and analyzing data
  - The factual information collected from a survey or other source
  - An element or feature that can vary
  - A group chosen from a population

QUIZ

- What is data?

  - The practice of collecting and analyzing data
  - The factual information collected from a survey or other source
  - An element or feature that can vary
  - A group chosen from a population

QUIZ

- What is a database?
  - A file that stores and organizes large amounts of related data
  - An application that allows you to carry out general book-keeping tasks
  - A file that stores your computer's configuration details
  - A device used to store deleted files

QUIZ

- What is a database?

  - A file that stores and organizes large amounts of related data
  - An application that allows you to carry out general book-keeping tasks
  - A file that stores your computer's configuration details
  - A device used to store deleted files

Traditional database systems: recall

- **Data**: *information, especially facts or numbers, collected to be examined and considered and used to help decision-making, or information in an electronic form that can be stored and used by a computer (source https://dictionary.cambridge.org/dictionary/english/data)*

- **Database**: *A collection of data arranged for ease and speed of search and retrieval (source https://www.thefreedictionary.com/database)*

- **The only purpose of a database is to store data**

Traditional database systems: recall

- **Database:** a collection formed of the following components:
  - **A description of the data structures** used (database schema) for data modeling (stored in the database's dictionary)
  - **The data collection** (instances of the schema)
  - **Various components**: views, procedures and functions, users, roles, etc.

There is a **separation** between:

- Data **definition** (kept in a dctionary of the DB),
- Data **management** (add, delete, update) and querying

Database management systems (DBMSs)

Feedback required

QUIZ

- Which of the following are main features of a database management system?
  - Creating databases
  - Processing data
  - Creating and processing forms
  - Administrating the database

QUIZ

- Which of the following are main features of a database management system?
  - Creating databases
  - Processing data
  - Creating and processing forms
  - Administrating the database

Traditional database systems: recall

- **Database management system (DBMS):** *A software system that facilitates the creation, maintenance and use of an electronic database (source https://www.thefreedictionary.com/)*

- **Database management system (DBMS):** collection of programs that are necessary for the management of a database

- **Database system**: database + DBMS

Traditional database systems: recall

- DBMS **main functions**:
  - **Database definition**: data definition language (or applications which generate commands in the definition language)
  - **Data management**: add, update, delete, query
  - **Database administration**: authorizing DB control, monitoring the DB use, monitoring the DB performance, DB performance optimization, etc.
  - **Protection of the information** inside the DB: **confidentiality** (protection against unauthorized access to data), **integrity** (protection against altering the DB contents)

Traditional database systems: recall

Characteristics of (relational) DBMSs

- **Data stored into Tables:**
  - Data is stored into tables, created inside the database
  - DBMS allows to have relationships between tables which makes the data more meaningful and connected
  - The type of data stored specified in the tables definition

- **Reduced Redundancy:** DBMS follow **Normalization** which divides the data in such a way that repetition is minimum

- **Data Consistency:** On live data, i.e. data that is being continuously updated and added, maintaining the consistency of data can become a challenge (Only valid data is saved)

Traditional database systems: recall

Characteristics of (relational) DBMS

- **Support Multiple User and Concurrent Access**: DBMS allows multiple users to work on it (update, insert, delete data) at the same time and still manages to maintain the data consistency
- **Query Language**: DBMS provides users with a simple Query language, allowing data to be easily fetched, inserted, deleted and updated in a database
- **Security**: The DBMS also takes care of the security of data, protecting the data from unauthorized access. In a typical DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access
- DBMS supports **transactions**, which allows us to better handle and manage data integrity in real world applications where multi-threading is extensively used.

Database models

Traditional database systems: recall

Database models

- Until the 60s: classical file-based organization

- During the 60s: 1st generation of DBMS
  - When DBs first appeared, the conceptual level was tightly linked to data representation on physical support
  - The Hierarchical model, the Network model

- During the 70s-80s: 2nd generation of DBMS, more independent from the physical support
  - **The Relational model**

- Beginning of the 90s: 3rd generation DBMS
  - Object-oriented model, …

- The 2000: NoSQL databases
  - Semi-structured/ unstructured data, of higher and higher volumes

Traditional database systems: recall

- The **relational database model –** proposed by Edgar F. Codd in 1970
  - New concepts: data independency, table structure, data query language (Structured Query Language, SQL)
- Any operation or group of operations (transaction) on a relational database must respect the **ACID principle**

ACID principle

Traditional database systems: recall

ACID database principles

- The ACID properties define SQL database key properties to ensure consistent, safe and robust database modification when saved.

a

Atomic – A transaction is treated as a single unit (if a single operation fails to execute, the entire transaction is rolled back). *Transactions are "all or nothing".*

c

Consistent – The database is consistent at the end of a transaction execution (it respects the memorized integrity rules). *Only valid data is saved.*

i

Isolation – While a transaction T executes, other transactions cannot modify the data handled by the transaction T. *Transactions do not affect each other.*

d

Durable – System failures or restarts do not affect committed transactions. *Written data will not be lost.*

Traditional database systems: recall

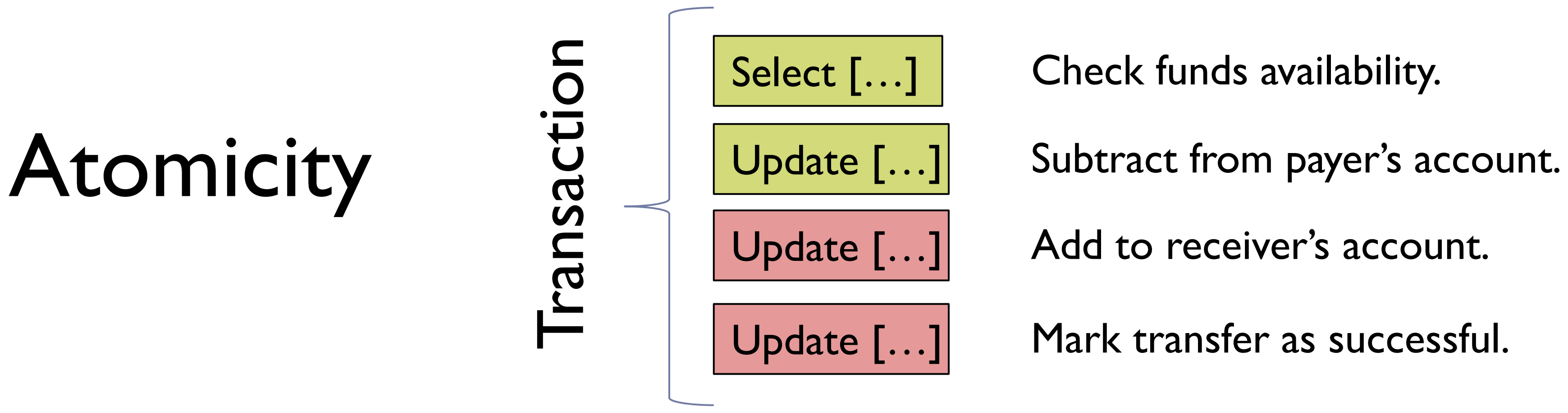
- **ACID database principles explained**
  - Example: Transfer money from one bank account to another

Transaction

Select […]	Check funds availability.
Update […]	Subtract from payer’s account.
Update […]	Add to receiver’s account.
Update […]	Mark transfer as successful.

Traditional database systems: recall

- **ACID database principles explained**
  - Example: Transfer money from one bank account to another


Atomicity
Transaction
Select […]
Update […]
Update […]
Update […]
Check funds availability.
Subtract from payer's account.
Add to receiver's account.
Mark transfer as successful.

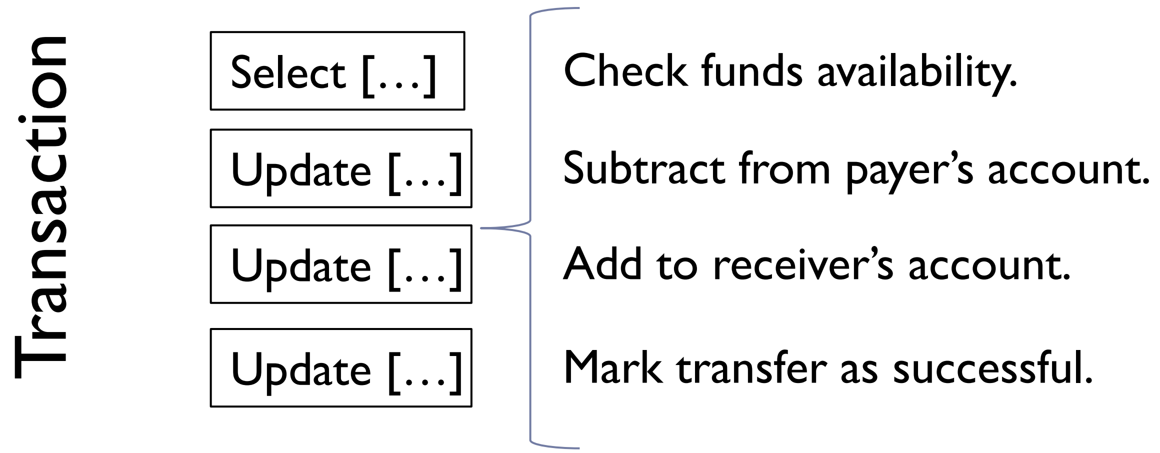
If one part of the transaction fails, none of the results are committed.

Transactions are "all or nothing".

Traditional database systems: recall

- **ACID database principles explained**
  - Example: Transfer money from one bank account to another

Consistency


Transaction
Select […]
Update […]
Update […]
Update […]
Check funds availability.
Subtract from payer's account.
Add to receiver's account.
Mark transfer as successful.

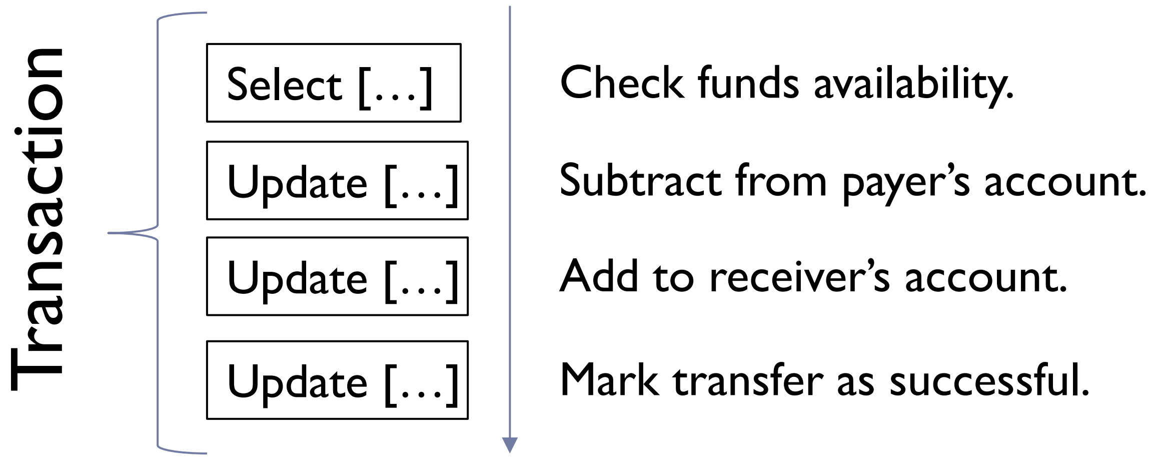
After transaction is committed, data is in a valid state (submitted to constraints):
e.g. the bank account must not be negative.

Only valid data is saved.

Traditional database systems: recall

- **ACID database principles explained**
  - Example: Transfer money from one bank account to another

Isolation



Transaction
Select […]
Update […]
Update […]
Update […]
Check funds availability.
Subtract from payer's account.
Add to receiver's account.
Mark transfer as successful.

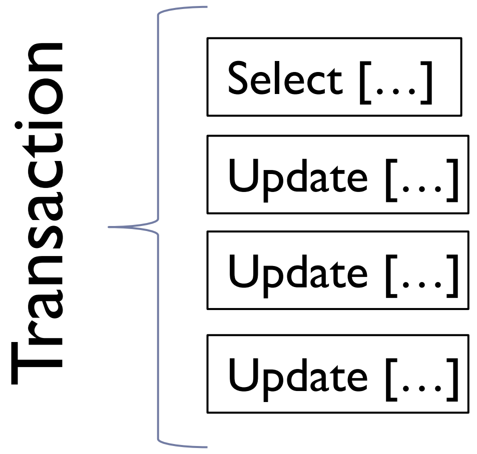
Transactions execute sequentially and they do not affect each other.

Transactions do not affect each other.

Traditional database systems: recall

- **ACID database principles explained**
  - Example: Transfer money from one bank account to another

Durability



Check funds availability.

Subtract from payer's account.

Add to receiver's account.

Mark transfer as successful.

Committed changes are permanently stored.

Written data will not be lost.

Relational DBMS advantages

Traditional database systems: recall

Advantages of RDBMS

- Minimal data redundancy
- Easy retrieval of data using the SQL
- Reduced development time and maintenance need
- Seamless integration with APIs (which makes it very easy to add a database to almost any application or website)

Popular DBMS

- MySql
- Oracle
- SQL Server
- IBM DB2
- PostgreSQL
- Amazon SimpleDB (cloud based) etc.

Traditional database systems: recall

The relational model

- Relational DBs are a traditional and well-developed tool for the backend of internet shops and other online apps
- They are used in many solutions for shopping charts, product catalogues, and related data products
- **Then *why would these not fit the need* of a giant like Amazon?**

Traditional database systems: recall

The relational model: technical limitations

- The DB is made up of relations with a **fixed schema**. Modifying the schema (i.e., the virtual structure of the DB) is a heavy task, which can involve high costs
- It is difficult to query data which depend on other data (from other DBs), therefore a **data interconnectivity** would be necessary
- The **size of the data** can increase drastically (for example, in DBs which store huge quantities of text and binary information: images, movies)
- To ensure **scalability**, the DBs would need to be distributed across multiple servers, which would make the table management very difficult/ impossible

The need of a non-traditional (non-relational) alternative solution to data representation

The relational model: technical limitations

- Main reasons why traditional relational database management systems (RDBMS) do not fit well into the big data scenario:
  - RDBMS are not good in data distribution and partitioning
  - RDBMS produce an overhead that can be avoided when
    - Many read or write operations occur as key-value pairs
    - Only large data blobs have to be stored
- Problems like these have led to new database systems generations / trends
  - **The NoSQL family**
  - NewSQL
  - Data grid/cache products (in-memory DBs)

The need of a non-traditional (non-relational) alternative solution to data representation

The NoSQL Family

- NoSQL stands for "not only SQL", a collection of paradigms which lead to the following approaches:
  - Flexible schemas
  - Queries without JOINs
  - Horizontally scalable
- Offer more possibilities of developing new solutions or solving problems that with classical RDBMS
- However, it is also a *step away from multiple purpose solutions* towards a plethora of multiple solutions for different data types and processing needs

The need of a non-traditional (non-relational) alternative solution to data representation

The NoSQL Family

- With **NoSQL**, the ACID principle is difficult to be respected (especially due to distribution and replication), and therefore it is replaced with the **BASE model**:
  - **Basic Availability**: all clients receive an answer to their query (instead of using a single data source, the data collection is replicated and distributed, therefore somewhere in the network, the data must be found)
  - **Soft State**: the DB consistency is not verified by the DBMS, it needs to be ensured instead by the client (program) who has modification rights on the DB
  - **Eventual Consistency**: the DB can be in an inconsistent state (e.g., different values for the same data), but it is assumed that the data will become consistent. Eventually. The propagation of the updates to all the replicas of that data will be done in the future.

The non-relational model

- Classification of the NoSQL models:

Data Model	Implementations
Key-value stores (collections of key-value pairs)	Amazon Dynamo, Redis, Membase, MemcacheDB, Scalaris, Tokyo Cabinet, Voldemort, Riak
Wide-column store/ Column family	Google Bigtable, Cassandra (Facebook), Hadoop/HBase, HyperTable, Amazon SimpleDB
Graph data model	Neo4j, InfiniteGraph, InfoGrid, GrafBase, HypergraphDB
Document data model (collections of documents)	MongoDB, Couchbase, CouchDB, Terrastore, RavenDB

The non-relational model

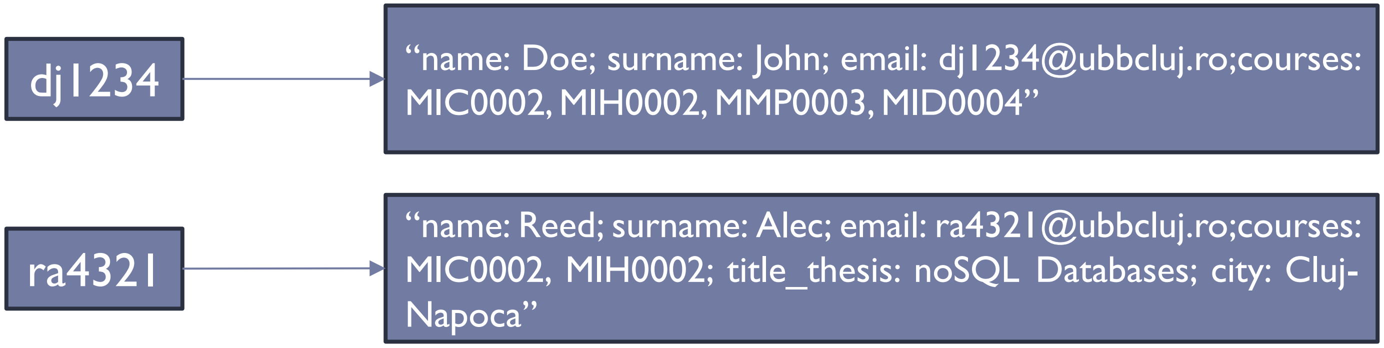
1. **Key-value data model**

- Most basic NoSQL data model
- The DB is formed of a **collection of pairs (key-value)**
- The **key** part of the key-value pair is also a unique identifier and it is used for data access
- The **value** part of the key-value pair is usually opaque at the database level
  - The value part can contain complex structures, but these structures are not visible at the database level
- Operations allowed:
  - Inserting a new value for a key
  - Returning the value of a key
  - Deleting a key and its value

The non-relational model

1. **Key-value data model**

- Example:


dj1234
“name: Doe; surname: John; email: dj1234@ubbcluj.ro;courses: MIC0002, MIH0002, MMP0003, MID0004”
ra4321
“name: Reed; surname: Alec; email: ra4321@ubbcluj.ro;courses: MIC0002, MIH0002; title_thesis: noSQL Databases; city: Cluj-Napoca”

The non-relational model

2. Wide-column store data model (Column family)

- It is based on the **multi-dimensional map** data structure
- In this context, a **column** is in fact a **key-value pair** in which the key is the name of the column and the value is the actual value for the column
- The value part can store complex structures as well (like arrays or objects)
- A **column-family** is a group of columns that contain related data (similar to a relational table)
- A column family resembles a relational table, but it has a flexible schema and does not store NULL values if there is no value assigned to a specific column

The non-relational model

2. Wide-column store data model (Column family)

- Example:

Assume we have the following non1NF table:

ID	name	surname	email	study_contract
1	Doe	John	dj1234@ubbcluj .ro	"MIC0002", "MIH0002", "MMP0003", "MID0004"
2	Popa	Radu		
3	White	Ana	wa4321@ubbcluj .ro	"MLR0020", "MLR0002", "MLR5004"

- The associated column-oriented model is the following (the ID column (key) was used for every column):

ID	name
1	Doe
2	Popa
3	White

ID	surname
1	John
2	Radu
3	Ana

ID	email
1	dj1234@ubbcluj.ro
3	wa4321@ubbcluj.ro

ID	study_contract
1	MIC0002
1	MIH0002
1	MMP0003
1	MID0004
3	MLR0020
3	MLR0002
3	MLR5004

The non-relational model

2. Wide-column store data model (Column family)

- Example:
- In a table cell can be stored multiple versions of the value. For example, tables with columns 'name' and 'email' could have the following contents after a while (an extra column was addes, **ts-timestamp**, with meaning: time of the update):

ID	ts	name
1	t1	Doe
2	t1	Popa
3	t1	White
3	t5	Richards

ID	ts	email
1	t1	dj1234@ubbcluj.ro
1	t2	djjr1234@ubbcluj.ro
1	t3	doe_john@yahoo.com
3	t1	wa4321@ubbcluj.ro
3	t4	white.ana@gmail.com

- Examples of possible column families: {name, surname}, {email}, {study_contract}

The non-relational model

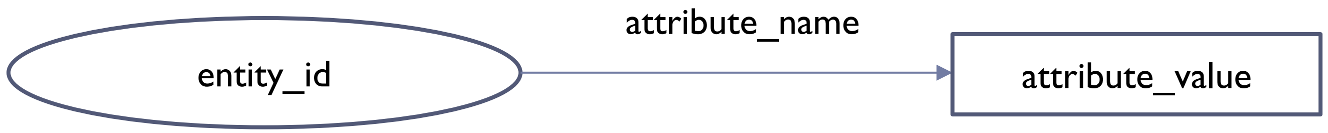
3. The graph data model

- A **graph database is** a database that uses **graph structures** to represent and store data (a set of edges and vertices)
- Suitable for highly associative data (e.g., social networks)

- Assume we need to store a set of semi-structured data, or a table from a relational DB
  - Every entity (record) identifies with a unique key
  - Such an entity has values for some attributes (properties, predicates, columns)
  - We will build a set of **triples** having the following structure:

(entity_id, attribute_name, attribute_value)

- An oriented graph can be built from these triples

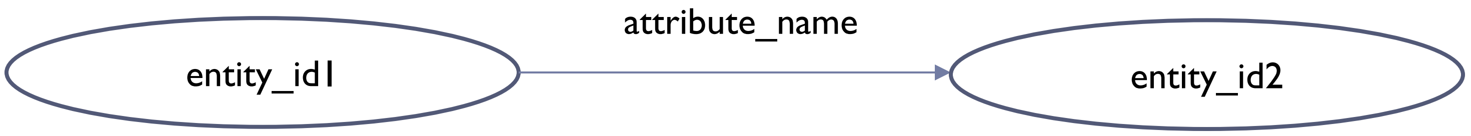


The non-relational model

3. The graph data model

- The vertices of the graph can be of two types:
  - **entity id** (drawn as an ellipse), or
  - **constant** (drawn as a rectangle)

- The value of an attribute can be an entity id, in which case it is of the first type



The non-relational model

3. The graph data model

- Example: Assume we have a relational DB with the following tables:

students

IDstud	name	surname
1	Doe	John
2	Popa	Radu
3	White	Ana

discipline

code	title
MIC0002	Distributed Operating Systems
MIH0002	Databases
MMP0003	Probabilities
MLR0002	Mathematical Analysis
MLR5004	Big Data

contracts

IDstud	ccode
1	MIC0002
1	MIH0002
1	MMP0003
2	MIH0002
2	MLR5004
3	MLR0002
3	MLR5004

The non-relational model

3. The graph data model

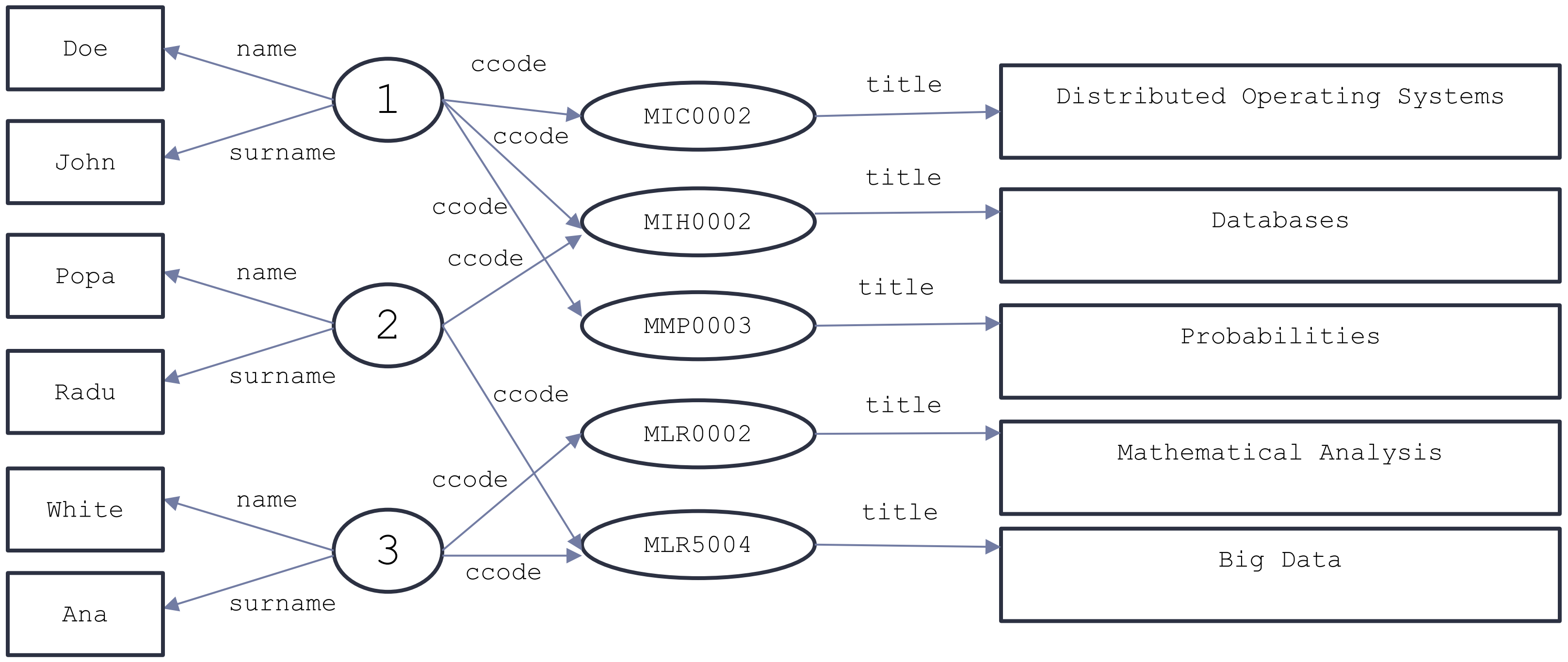
- Example
- The corresponding triples are:
- On column 'attribute_value', the values which represent entity ids are marked in blue

entity_id	attribute_name	attribute_value
1	name	Doe
1	surname	John
2	name	Popa
2	surname	Radu
3	name	White
3	surname	Ana
MIC0002	title	Distributed Operating Systems
MIH0002	title	Databases
MMP0003	title	Probabilities
MLR0002	title	Mathematical Analysis
MLR5004	title	Big Data
1	ccode	MIC0002
1	ccode	MIH0002
1	ccode	MMP0003
2	ccode	MIH0002
2	Ccode	MLR5004
3	ccode	MLR0002
3	ccode	MLR5004

The non-relational model

3. The graph data model

Example: The corresponding graph is



Example of using this storage model for Google search:
Introducing the Knowledge Graph, http://www.youtube.com/watch?v=mmQl6VGvX-c

The non-relational model

4. The document data model

- A document DB contains **collections of documents**, similar to tables in a relational DB
  - A collection groups together documents that are useful for a query
- **No join operations** between different collections
- The **DB and** the **collections** inside the DB are identified with a **name**
- A document has **no predefined structure**
- A **document** is formed of a set of **name/value** pairs

The non-relational model

4. The document data model

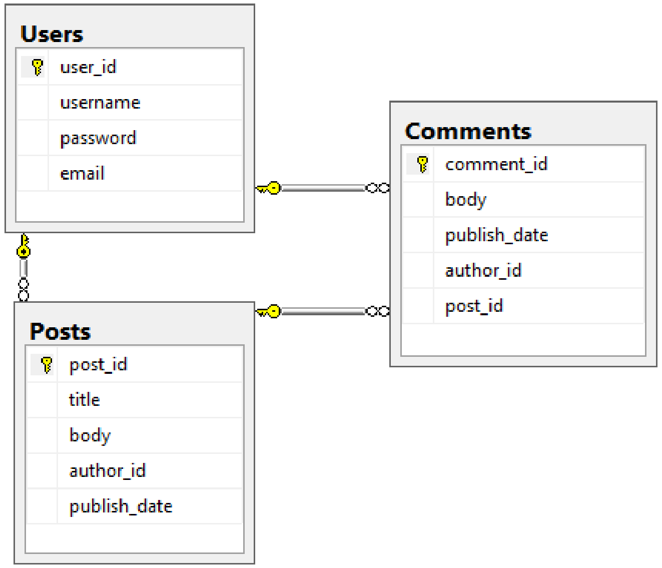
- Each document has a **unique identifier** (which corresponds to the key part of a key-value pair) and a **value** (which corresponds to the value part of a key-value pair)
- The most popular format is JSON (JavaScript Object Notation)
  - A JSON document is a collection of key-value pairs, and the value part can be a complex structure (like a collection of values or a collection of key-value pairs)
  - Other document formats are XML and YAML
- It allows **very expressive query languages**
- The structure of the value part is *visible* at the database level

The non-relational model

4. The document data model

- Example:
- The relational version of the database contains three tables: **Users**, **Posts** and **Comments**

  - There is a **one to many relationship** between **Users** and **Posts** (one user can have many posts, but one post belongs to a single user)
  - There is a **one to many relationship** between **Users** and **Comments** (one user can have many comments, but one comment belongs to a single user)
  - There is a **one to many relationship** between **Posts** and **Comments** (one post can have many comments, but one comment belongs to a single post)
  - There is also a **unique constraint** defined on the column "username", because generally, the username is unique



The non-relational model

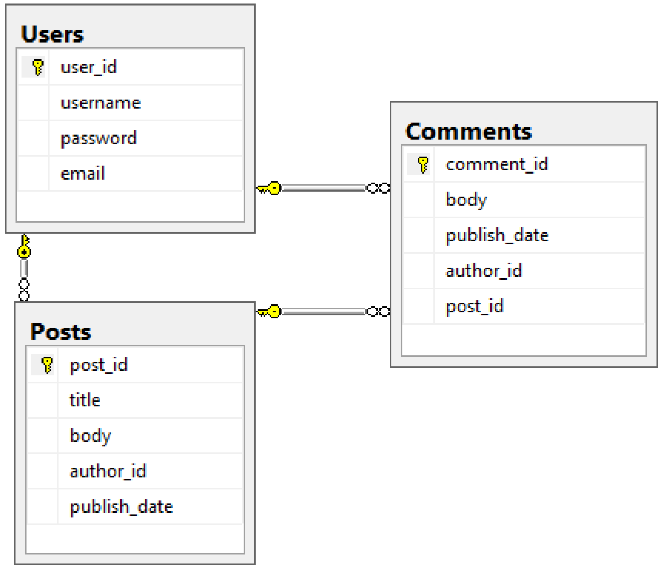
4. The document data model

- Example: Solution – MongoDB
- In MongoDB, a collection has a **flexible schema** (you don't have to define the schema when you create a collection)
  - Documents that belong to the same collection can have different schemas (collections do not enforce document structure)
- **No** need for **normalization**
- The things you have to think about are the **data access pattern** and the **most frequent queries**
- In MongoDB, a **one to many** relationship can be modeled in more than one way:
  - **A single collection** is created and the documents that belong to the **many** side of the relationship are embedded in the document that belongs to the **one** side of the relationship
  - **Two collections** are created and each document stored in the collection that belongs to the **many** side of the relationship contains a reference to a document stored in the collection that belongs to the **one** side of the relationship (and that field contains the value of the "_id" field)

The non-relational model

4. The document data model

- Example: Solution – MongoDB
  - In our application's case, the portions of data that are accessed together are:
    - A post and all its comments
    - All posts written by a user
  - For each post and each comment, only the username is required
  - In this case, the database will contain two collections:
    - **users** collection
    - **posts** collection



The non-relational model

4. The document data model

- Example: Solution – MongoDB

- A document that belongs to the **users** collection:

```
{
"_id" : "Bob",
"password" : "233567",
"email" : "bob@gmail.com"
}
```

The non-relational model

4. The document data model

- Example: Solution – MongoDB

- A document that belongs to the **posts** collection:

```
{"_id" : ObjectId("5a53b2c51d2f5a15e12d3109"),
 "title" : "A sunny day",
 "author" : "Bob",
 "body" : "In a sunny day, I saw a nice flower…….",
 "publish_date" :  ISODate("2018-01-08T18:04:00Z"),
 "comments" : [
                   {"author" : "Tom",
                    "body" : "Nice post ",
                    "publish_date" :  ISODate("2018-01-08T19:04:00Z")
                   } ]
}
```

Relational vs. non-relational data models. A comparative analysis

Characteristics	NoSql	SQL
Data storage	- Document (JSON), columns, .. - Flexible - No need for each record to store same properties/attributes - New properties/attributes can be added dynamically - Suitable for semi-structured and unstructured data	- DB tables - Less flexible - All records must have same attributes - Adding a new attribute will imply a schema modification - Suitable for structured data
Schema	- Allows for dynamic, flexible schemas - The application is the one dictating the schema (schema on read)	- Only allow strict schemas - Schema must synchronize between the application and the DB (schema on write)
Transactions	- The support for ACID transactions is different for different solutions	- ACID transactions
Consistency and Availability	- Depend on the solution - Consistency, availability and performance can be decided based on the application needs	- Strong consistency - Consistency takes precedence over data availability and performance

Databases and data warehouses

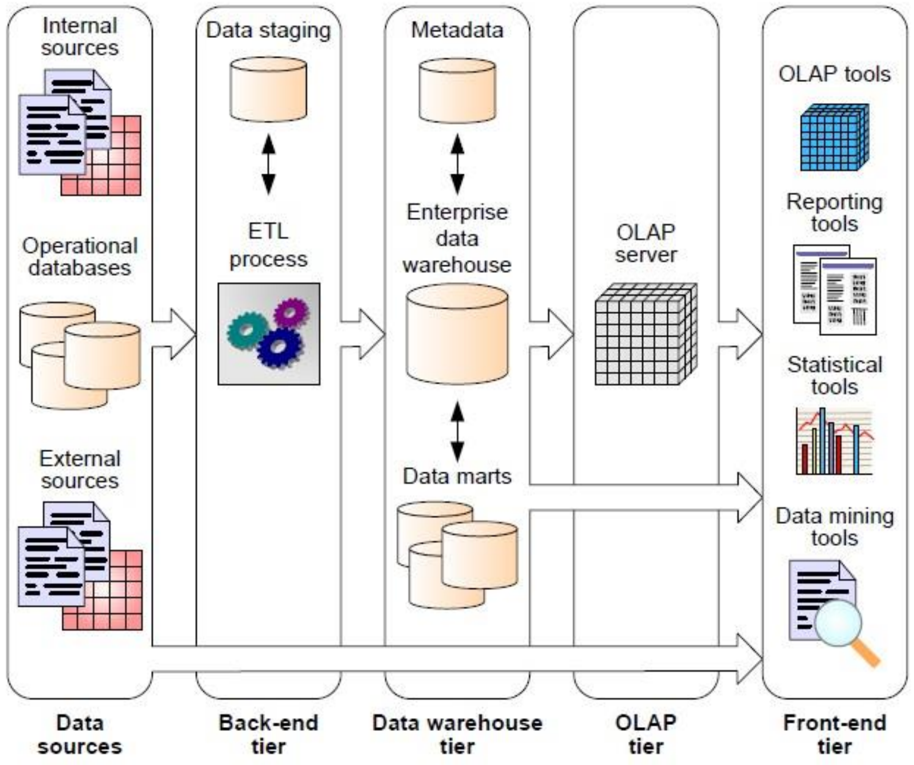
Traditional database systems: recall

- **Database system**: database + DBMS
- **Data warehouse**: a repository of integrated data obtained from several sources for multidimensional data analysis

Traditional database systems: recall

Differences between Databases and Data Warehouses

Database	Data Warehouse
An organized collection of data	A central repository of integrated data from one or more sources
Usually tied to a single application such as a ticketing system	Usually store data from any number of applications
Primarily insert/write data	Primarily read/retrieve data
Data is normalized to allow quick response times	Data is denormalized for analytical an reporting efficiencies
Current / Point-in-time data	Historical data
Online Transactional Processing	Online Analytical Processing
Provides a detailed relational view	Provides a summarized multi-dimensional view
For many concurrent transactions	Not for a large amount of concurrent transactions



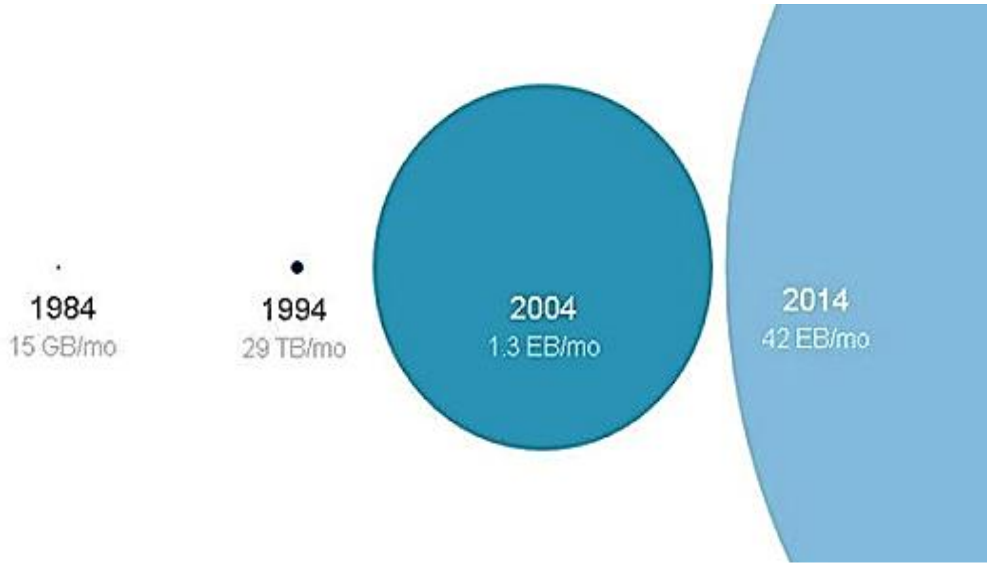
[Vaisman, Zimányi 2014]

Limitations of traditional database systems

Limitations of traditional database systems

Facts:

- In the 2000s, the big majority of business sectors were facing the "explosion of data"
- Internet companies like Google, Yahoo!, Facebook, eBay were ingesting massive volumes of data



The data problem: data were increasing in size, variety and velocity day by day

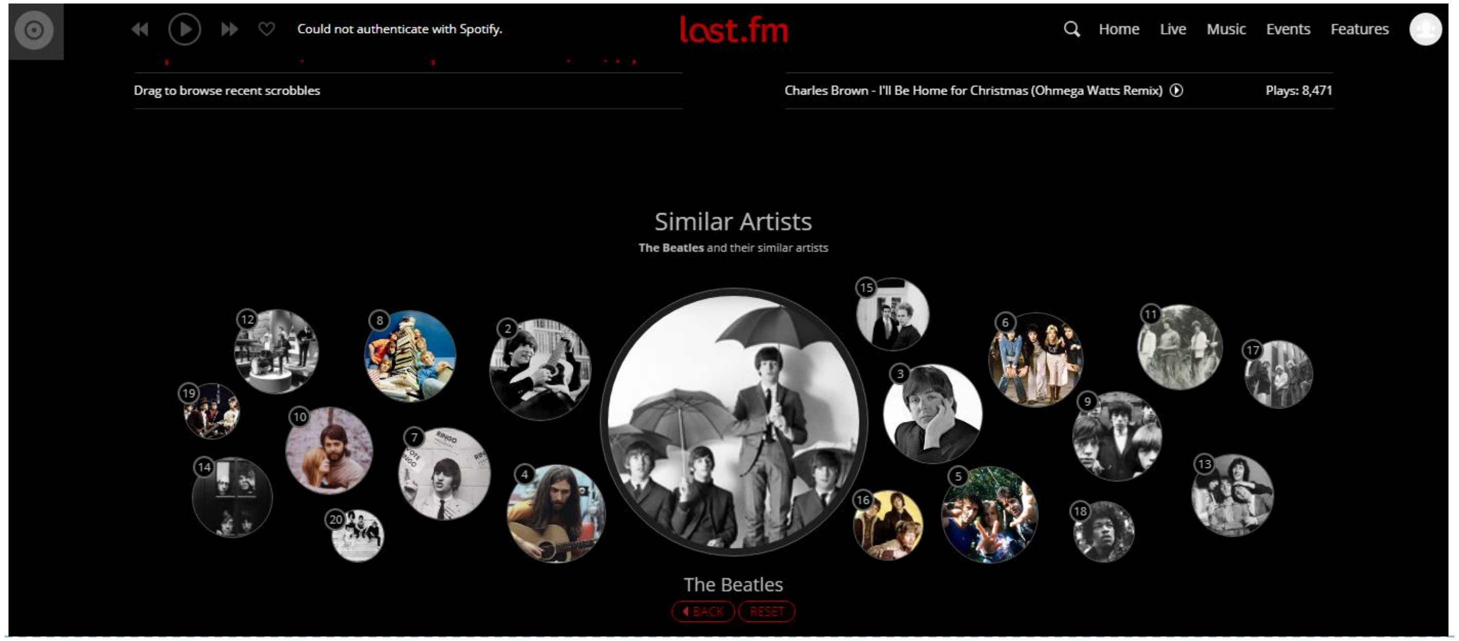
The problem that had to be solved in order to stay in business.

The need of a non-traditional (non-relational) alternative solution to data representation

- Two examples of players who have adopted big data architectures early on
  - last.fm
  - Amazon

The need of a non-traditional (non-relational) alternative solution to data representation

- **Last.fm** - internet radio and a music community website, provides
  - Music streams & downloads
  - **User-specific services**, such as music and event recommendation
  - Personalized chart lists
  - Presentation according to personal listening habits, taste in music, similar music and stylistic similarities
  - In particular, provides these services for **many concurrent users**
- In 2012, it had more than 25 million users / month


Could not authenticate with Spotify.
last.fm
Home
Live
Music
Events
Features
Drag to browse recent scrobbles
Charles Brown - I'll Be Home for Christmas (Ohmega Watts Remix)
Plays: 8,471
Similar Artists
The Beatles and their similar artists
The Beatles
BACK
RESET

The need of a non-traditional (non-relational) alternative solution to data representation

- **Last.fm** - internet radio and a music community website, provides
  - Music streams & downloads
  - **User-specific services**, such as music and event recommendation
  - Personalized chart lists
  - Presentation according to personal listening habits, taste in music, similar music and stylistic similarities
  - In particular, provides these services for **many concurrent users**
- In 2012, it had more than 25 million users / month

!The aspect of redundant distributed storage of large amounts of data!

The need of a non-traditional (non-relational) alternative solution to data representation

- **Amazon** - e-commerce company in the US
  - Handles many separate applications that

use big data sets, such as:
    - Seller lists
    - Shopping charts
    - Customer preferences
    - Session management
    - Sales rank
    - Product catalogue
  - Websites are rendered by using data, aggregations and analysis results of the different applications

The need of a non-traditional (non-relational) alternative solution to data representation

- This leads to the following **strong requirements** for their infrastructure (Amazon)
  - **High availability** - for read-write operations
  - **High performance** - short response times
  - **Service-level agreement** between applications - short, guaranteed response times even under high load

The need of a non-traditional (non-relational) alternative solution to data representation

- This leads to the following **strong requirements** for their infrastructure (Amazon)
  - **High availability** - for read-write operations
  - **High performance** - short response times
  - **Service-level agreement** between applications - short, guaranteed response times even under high load

!The question of databases whose size exceeds the capabilities of traditional relational database systems!

The need of a non-traditional (non-relational) alternative solution to data representation

- Historically, last.fm and Amazon provide examples of two different aspects:
  - (1)redundant distributed storage of large amounts of data and
  - (2) databases whose size exceeds the capabilities of traditional databases - of big data systems, which lead to two models:
  - Model 1: Hadoop
  - Model 2: Amazon Dynamo

Model 1: Hadoop

- Last.fm uses Apache Hadoop since 2006 as an infrastructure to stem the heavy computational load created by its broad user base

- What makes Hadoop a good solution in this case?
  - Hadoop provides a distributed filesystem with redundant backups
  - It is scalable with "commodity hardware"
  - It is available for free
  - It is an open source system which is extensible with your own features
  - It is flexible and easy to learn

- In 2010 last.fm was managing about 100 TB of data using 50 nodes with a total of 300 computer cores

- More examples of famous Hadoop users on Hadoop's Powered By website: http://wiki.apache.org/hadoop/PoweredBy

Model 2: Amazon Dynamo

- Amazon deals with extremely large amounts of data
- Even though Amazon data are structured in a way to suggest the use of traditional relational database systems, the sheer amount of data makes their use prohibitive
- For this (and other) reason, Amazon has developed their own big data database system, Dynamo
- Dynamo comes with the following design principles:
  - **Incremental scalability**: Dynamo should be able to scale out one storage host ("node") at a time
  - **Symmetry**: Every node in Dynamo should have the same set of responsibilities as its peers
  - **Decentralization**: Decentralized peer-to-peer techniques over centralized control
  - **Heterogeneity**: The system needs to be able to exploit heterogeneity in the infrastructure it runs on. This is essential in adding new nodes with higher capacity without having to upgrade all hosts at once.

- This, together with Google's BigTable[1] marked the beginning of a new generation of database systems specifically designed to handle big data problems

- [1] https://cloud.google.com/bigtable/

Limitations of traditional database systems

- Traditional systems' main problem with big data is that they were not designed for it

Limitations in traditional systems	Solution proposed by big data systems
Vertical scalability	Horizontal scalability
Schema on write	Schema on read
Cost of storage	Local storage
Cost of proprietary hardware	Commodity hardware
Complexity	Simplicity
Causation	Large scale analysis
Bring data to the programs	Bring programs to the data

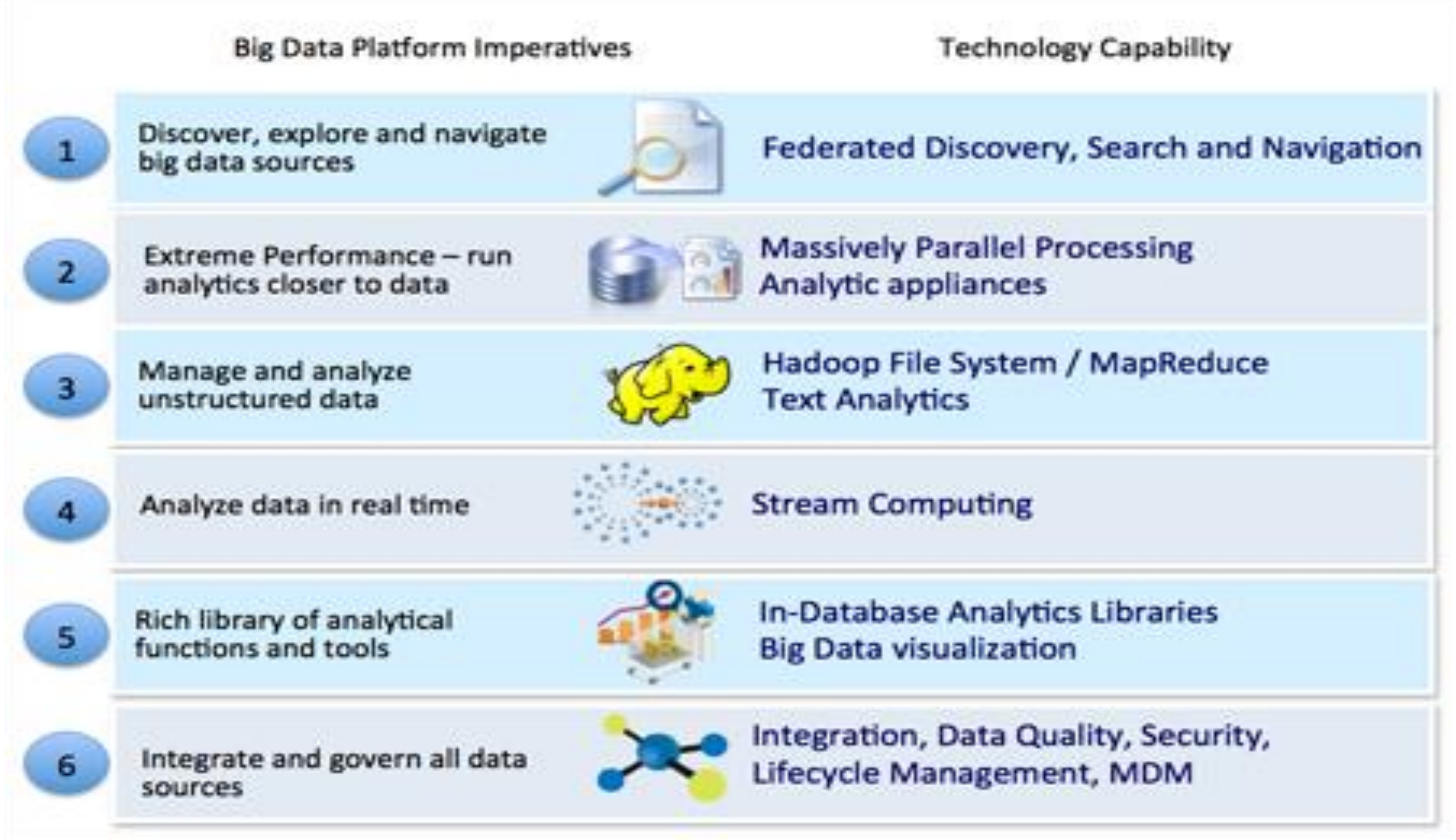
Big Data Systems

Big Data Systems

- What is a big data system?
- When is a system capable of handling big data?
- CAP theorem
- Lambda architecture

Big Data Ecosystem: imperatives

- Big Data Imperatives explains 'what big data platforms should do'

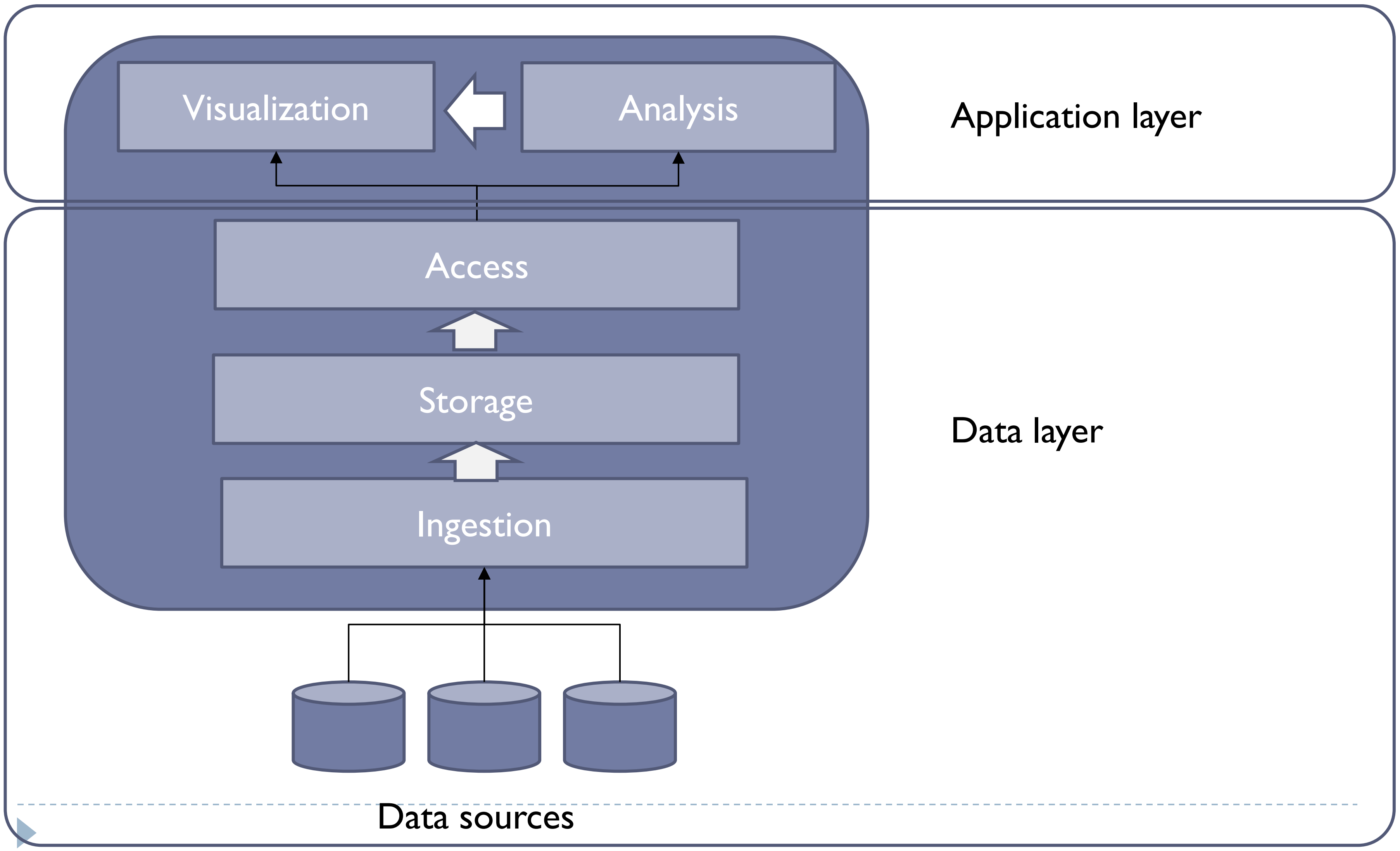


	Big Data Platform Imperatives	Technology Capability
1	Discover, explore and navigate big data sources	Federated Discovery, Search and Navigation
2	Extreme Performance – run analytics closer to data	Massively Parallel Processing Analytic appliances
3	Manage and analyze unstructured data	Hadoop File System / MapReduce Text Analytics
4	Analyze data in real time	Stream Computing
5	Rich library of analytical functions and tools	In-Database Analytics Libraries Big Data visualization
6	Integrate and govern all data sources	Integration, Data Quality, Security, Lifecycle Management, MDM

Big Data Platform Manifesto: imperatives and underlying technologies

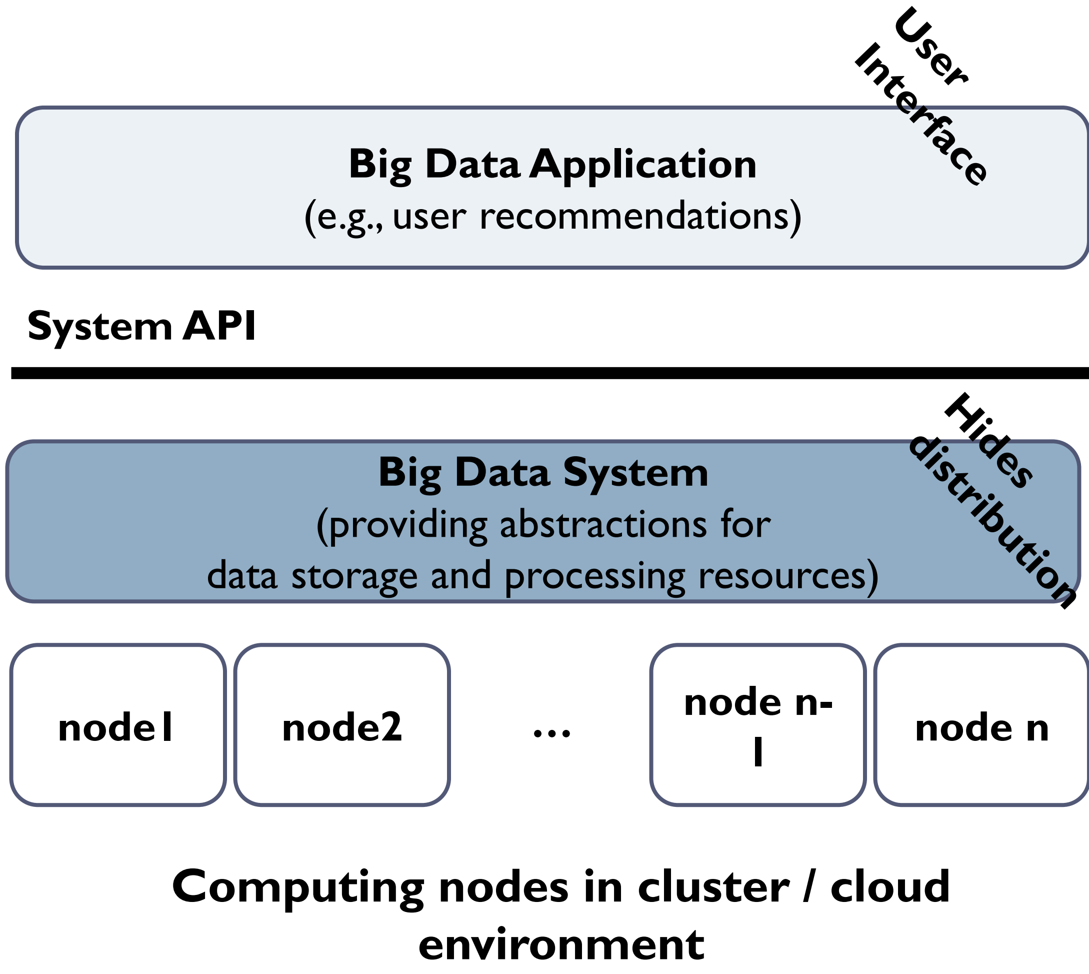
Source: https://www.slideserve.com/cargan/big-data-powerpoint-ppt-presentation

Big Data Architecture: main components



Big Data Systems: preamble

- Are inherently **distributed**
- Big data algorithms and applications must be **parallelizable**
- As in the case of operating systems, there should be a **separation** between the actual application and a big data system in the background
- The big data system **hides** data distribution and its handling from the application programmer
- It provides a **restricted set of operations** on data which are automatically parallelizable and hence lead to applications making transparent use of data and algorithm parallelization


User Interface
Big Data Application
(e.g., user recommendations)
System API
Hides distribution
Big Data System
(providing abstractions for
data storage and processing resources)
node1
node2
…
node n-1
node n
Computing nodes in cluster / cloud environment

Big Data Systems: requirements and properties

- What is a big data system? When is a system capable of handling big data?

- Two central requirements are defining such a system:
  - 1) Horizontal scalability
  - 2) Availability and fault tolerance

- Current big data systems are based on computing nodes each of which has its own main memory and secondary storage; the nodes are connected by network only (shared nothing architecture)

Big Data Systems: requirements and properties

- **Database scalability** is the ability to **scale out (horizontal scalability)** or **scale up (vertical scalability)** a **database** to allow it to hold increasing amounts of data without sacrificing performance.

- Many relational **databases** (RDBMS) such as Oracle, MySQL, Postgres or Microsoft SQL Server were designed to run on a single server.

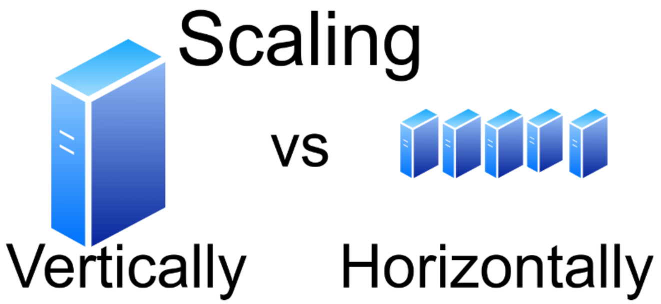
How do they scale?

Big Data Systems: requirements and properties

1) Horizontal scalability

- In the sense that adding **more computing nodes** provides **more storage** and **processing capacity** to the application, without the need to modify its implementation
- This is ensured by the system

- Contrasted by ***vertical* scalability**: this means increasing computer resources and /or storage resources in a single node
  - This approach to scalability reaches natural limits and is therefore *not suited* for big data approaches



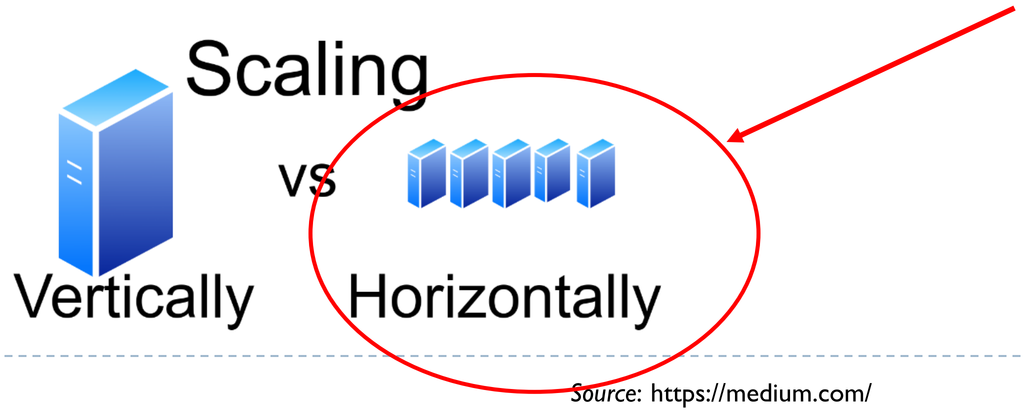
Source: https://medium.com/

Big Data Systems: requirements and properties

1) Horizontal scalability

- In the sense that adding **more computing nodes** provides **more storage** and **processing capacity** to the application, without the need to modify its implementation
- This is ensured by the system

- Contrasted by ***vertical* scalability**: this means increasing computer resources and /or storage resources in a single node
  - This approach to scalability reaches natural limits and is therefore *not suited* for big data approaches

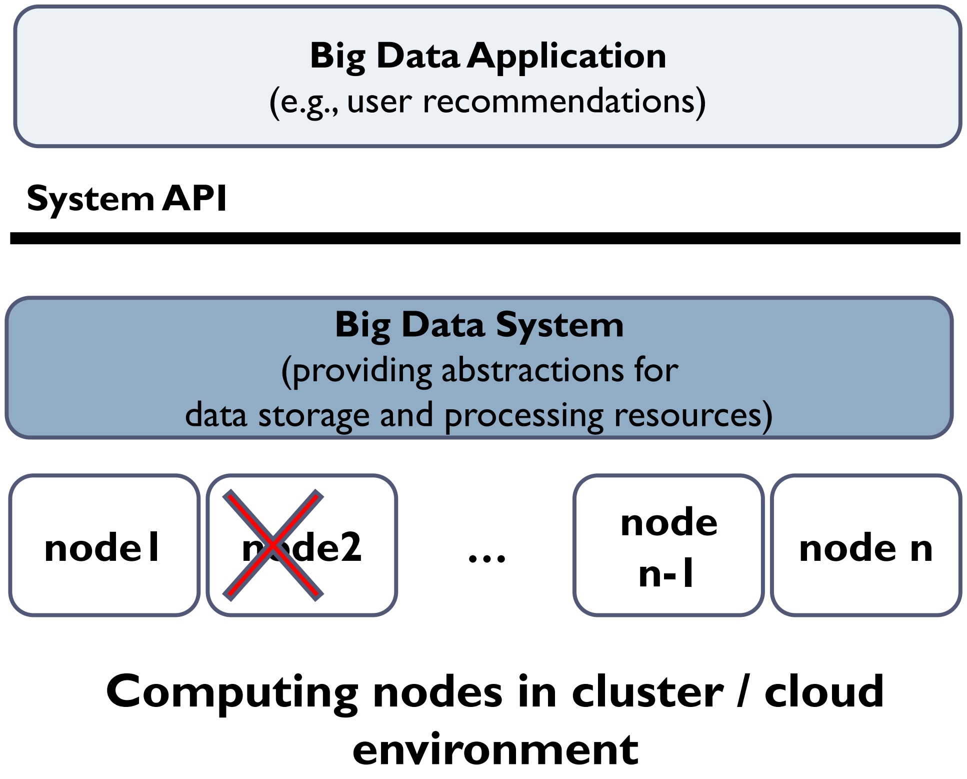


Source: https://medium.com/

Big Data Systems: requirements and properties

2) Availability and fault tolerance

- When using large numbers of nodes, any computing node may fail at any time (stop working - fail stop model) and needs to be repaired after some time
- Data availability is used to describe systems which ensure that data continues to be available at a required level of performance in situations ranging from normal to "disastrous"


Big Data Application
(e.g., user recommendations)
System API
Big Data System
(providing abstractions for data storage and processing resources)
node1
node2
…
node n-1
node n
Computing nodes in cluster / cloud environment

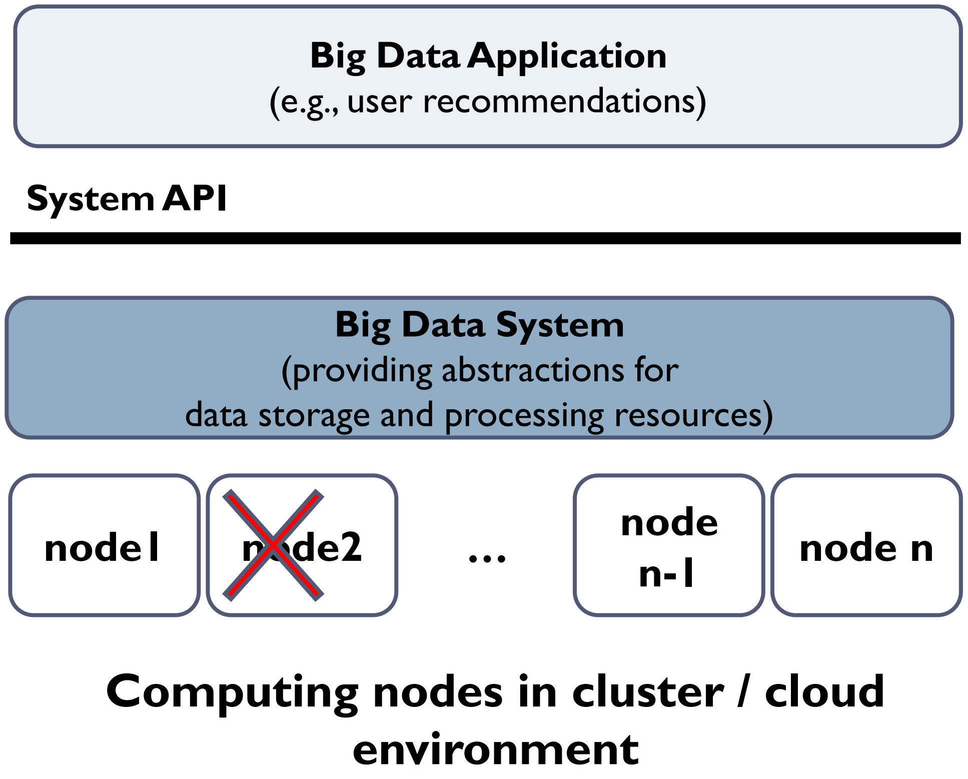
Big Data Systems: requirements and properties

2) Availability and fault tolerance

- The availability of a computing node is based on two measures:
  - MMTF - Mean Time to Failure
  - MMTR - Mean Time to Repair
- Based on these, the availability of a system is given by:

Availability = MMTF / (MMTF + MMTR)

- A **big data system must be fault-tolerant:** It must continue working even when individual nodes fail


Big Data Application
(e.g., user recommendations)
System API
Big Data System
(providing abstractions for
data storage and processing resources)
node1
node2
…
node n-1
node n
Computing nodes in cluster / cloud environment

Big Data Systems: additional properties

- Additional desired properties of a Big Data System:

1. Low latency reads and updates
2. Generalization (should support a wide range of applications)
3. Extensibility (new functionality to be added with minimum effort)
4. Minimal maintenance
5. Debuggability (should provide the information necessary to debug the system when things go wrong)

Big Data Systems: CAP Theorem

The CAP Theorem

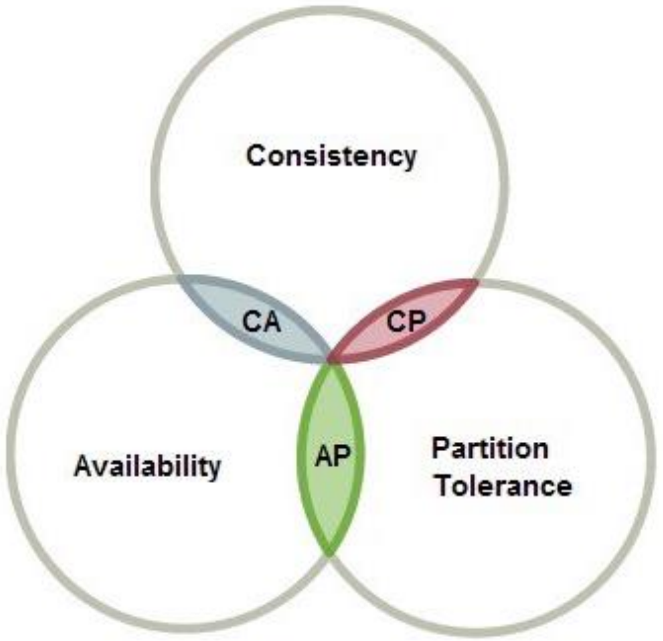
- Applies to all distributed systems (it does not apply to traditional DB systems)
- It states a fundamental limitation of distributed systems with respect to the following properties of the system: Consistency (C), Availability (A) and Partitioning Tolerance (P)
- **The theorem is important especially in the design of a Big Data System** when it needed to make a trade off between the three

Big Data Systems: CAP Theorem

The CAP Theorem

Theorem: A distributed system can fulfill at most two of the following qualities at the same time:

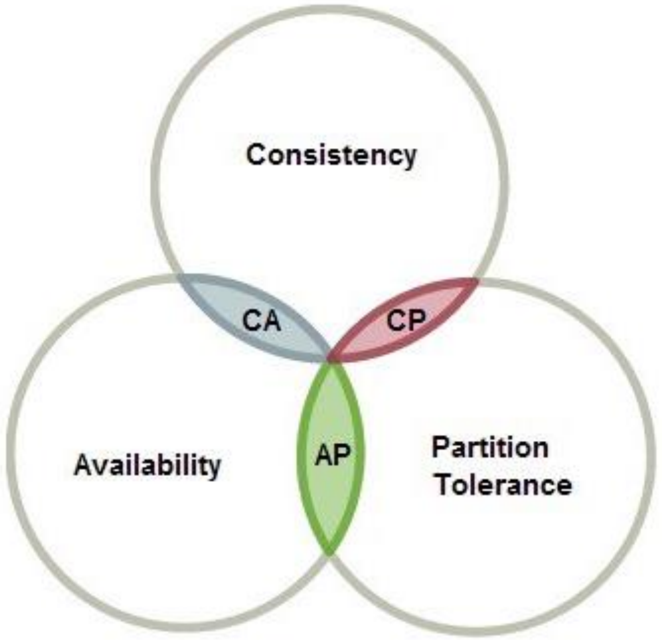
- ***Consistency (C)**: all replicates of data are equal and all nodes see the same state at the same time*
- ***Availability (A)**: every request is handled by the system*
- ***Partition (P)**: The system continues to work even at loss of messages, network nodes or a network partition*



Big Data Systems: CAP Theorem

CAP explained

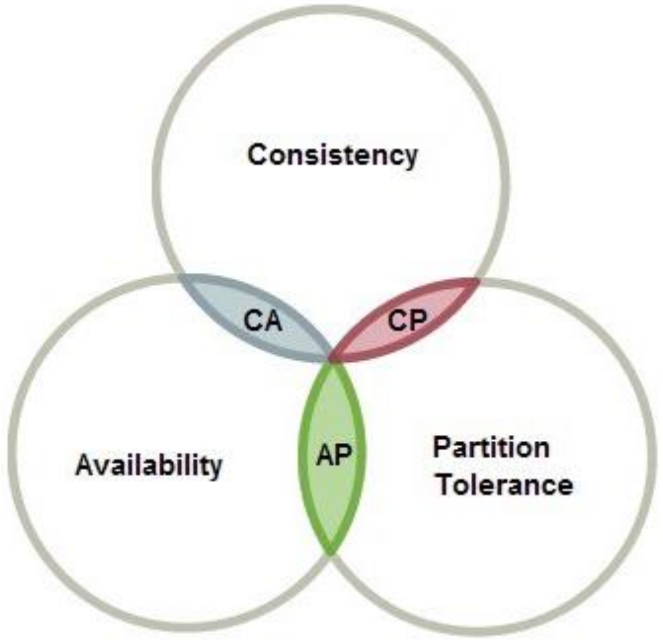
Consistency: *This condition states that all nodes see the same data at the same time.* Simply put, performing a *read* operation will return the value of the most recent *write* operation causing all nodes to return the same data. A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state.



Big Data Systems: CAP Theorem

CAP explained

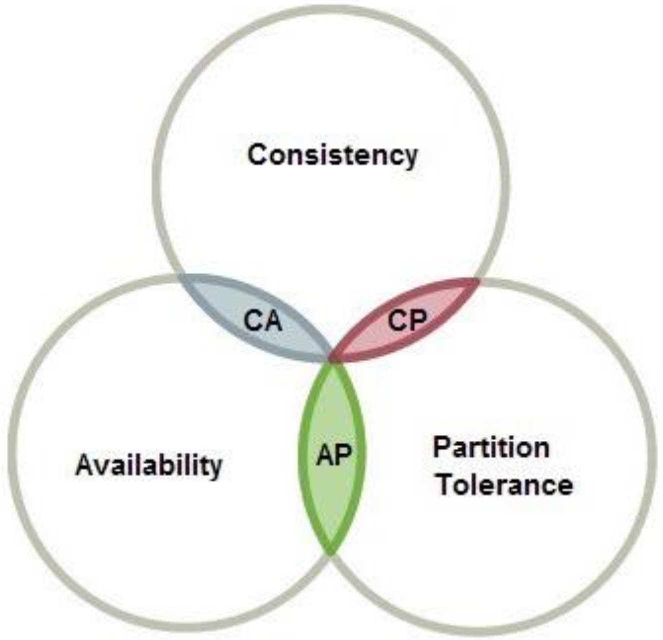
Availability: This means that every request gets a response on success/failure. It requires that the system remains operational 100% of the time. Every client gets a response, regardless of the state of any individual node in the system. This metric is trivial to measure: either you can submit read/write commands, or you cannot.



Big Data Systems: CAP Theorem

CAP explained

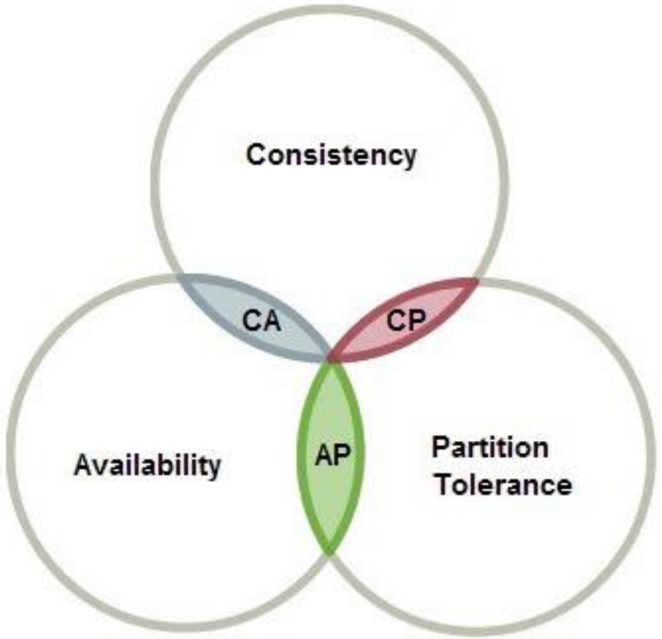
Partition Tolerance: This condition states that the system continues to run, regardless the state of individual nodes. A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network. Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages. **In modern distributed systems, Partition Tolerance is not an option. It's a necessity. Hence, we have to trade between Consistency and Availability.**



Big Data Systems: categories

The CAP theorem categorizes systems into three categories: CP, CA and AP

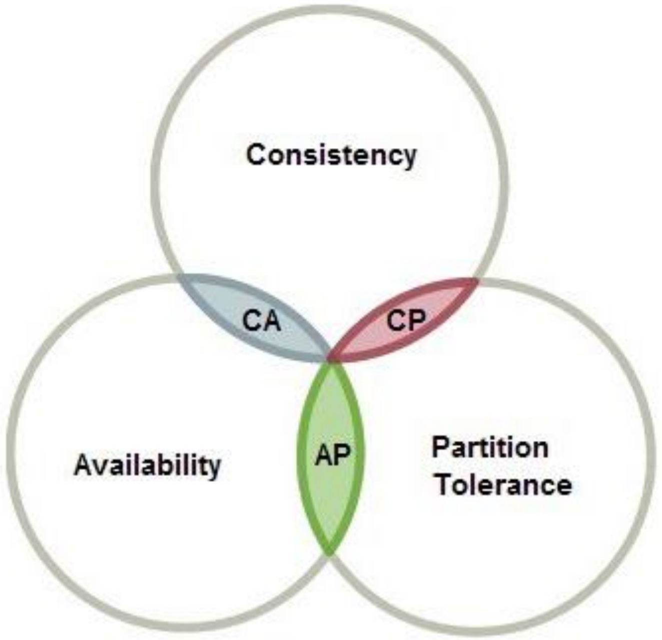
- **CP (Consistent and Partition Tolerant)** — At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available. CP is referring to a category of systems where availability is sacrificed only in the case of a network partition (network split due to a failure).



Big Data Systems: categories

The CAP theorem categorizes systems into three categories: CP, CA and AP

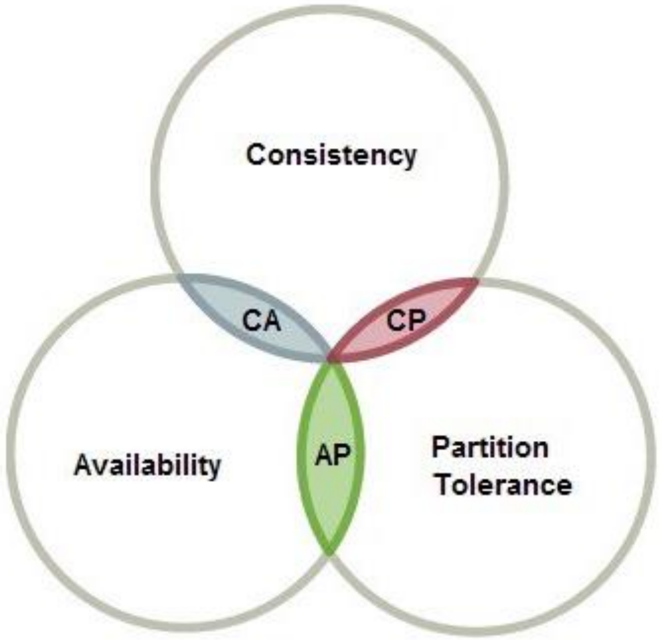
- **CA (Consistent and Available)** — CA systems are consistent and available systems in the absence of any network partition.



Big Data Systems: categories

The CAP theorem categorizes systems into three categories: CP, CA and AP

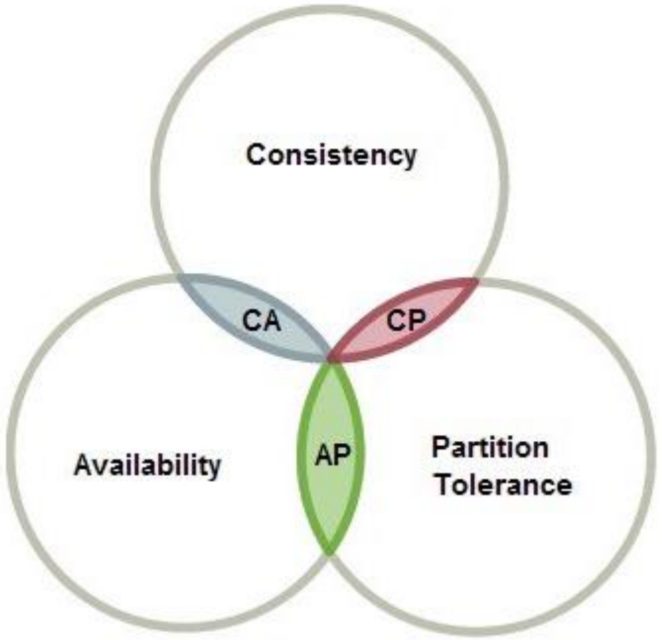
- **AP (Available and Partition Tolerant)** — These are systems that are available and partition tolerant but cannot guarantee consistency.



Big Data Systems: categories

In which category would you place RDBMS:

- CA
- CP
- AP

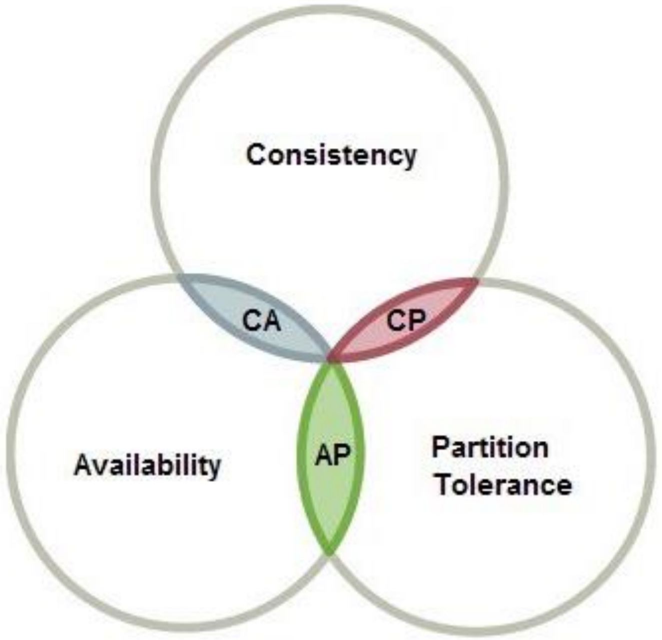


Big Data Systems: categories

In which category would you place RDBMS:

- CA
- CP
- AP

Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems. The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the preview of CAP.



Big Data Systems: Lambda Architecture

- Is a **constructive template** for the conception and design of big data applications
  - Published by Nathan Marz and James Warren[1]

- Is an architecture whose modularization reflects **typical requirements to big data applications** and combines them in a systematic picture

- The name *lambda architecture* is derived from a functional point of view regarding data processing: all data processing is understood as the application of a function to all data

1 Nathan Marz, James Warren;:Big Data - Principles and Best Practices of Scalable Real-Time Data Systems; Manning Publications; 2015 for further information

Big Data Systems: Lambda Architecture

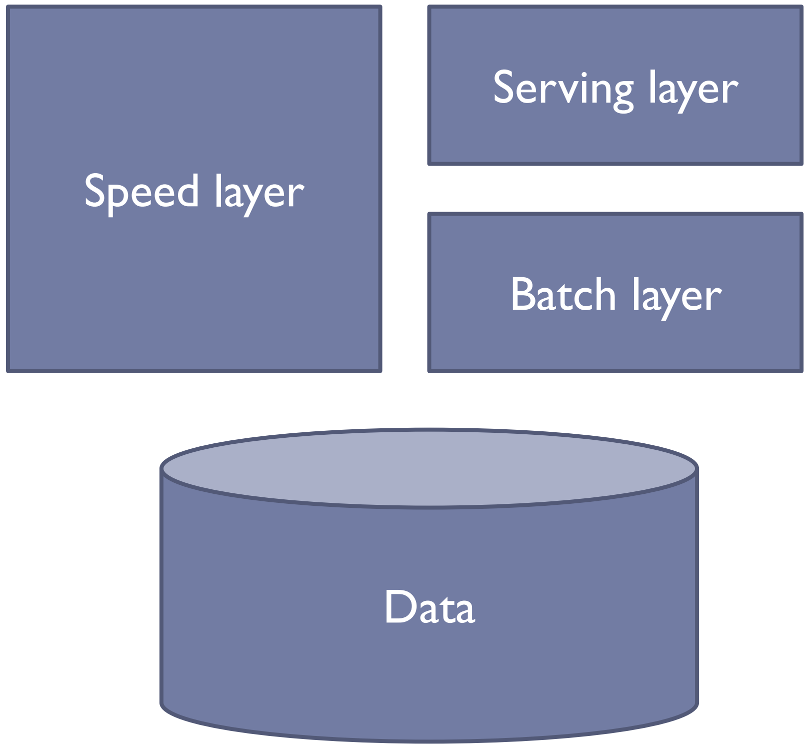
- The template makes it possible to identify the necessary services and allows justifying the selection of necessary components

- In the lambda architecture, original **data is tagged by a time-stamp and recorded** without loss of information

- In the data storage, the complete and growing set of **original data remains available**; In this way, all recorded information can be used in new, corrected, or extended functionality

Big Data Systems: Lambda Architecture

Main idea: build Big Data systems as a series of layers. Each layer satisfies a subset of the properties and builds upon the functionalities of the layers beneath them.



Why a layered system for Big Data?

Big Data Systems: Lambda Architecture

Why a layered system for Big Data?

Running a simple query on a large data set involves to read the whole dataset with billion of rows. Imagine you have to run that query several times a day.

All data
query = function (all data)

Even though running the query would be feasible, it would require an unreasonable amount of resources. **So what could be an alternative?**

Big Data Systems: Lambda Architecture

Why a layered system for Big Data?

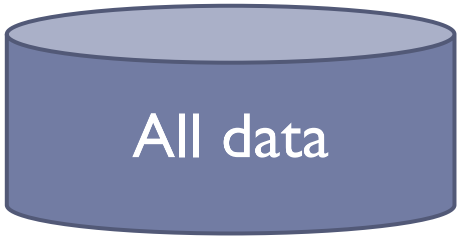
Scan only a subset of the whole data

query = function (all data)

vs

batch view = function (all data)
query = function (batch view)

Scan all dataset



Results are faster and there are less resources required

Big Data Systems: Lambda Architecture

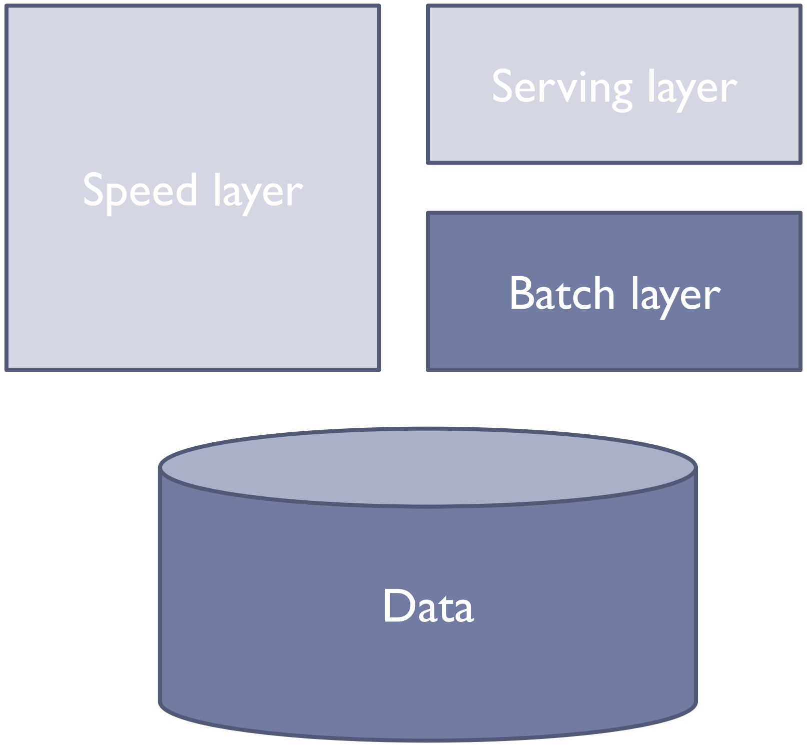
Batch Layer

Batch Layer: implements the equation below

batch view = function (all data)

- **Batch layer functionalities**
  - stores a master copy of the dataset
  - Precomputes batch views

- **Batch layer should be able to**
  - Store constantly growing master dataset
  - Compute arbitrary functions on that dataset



Big Data Systems: Lambda Architecture

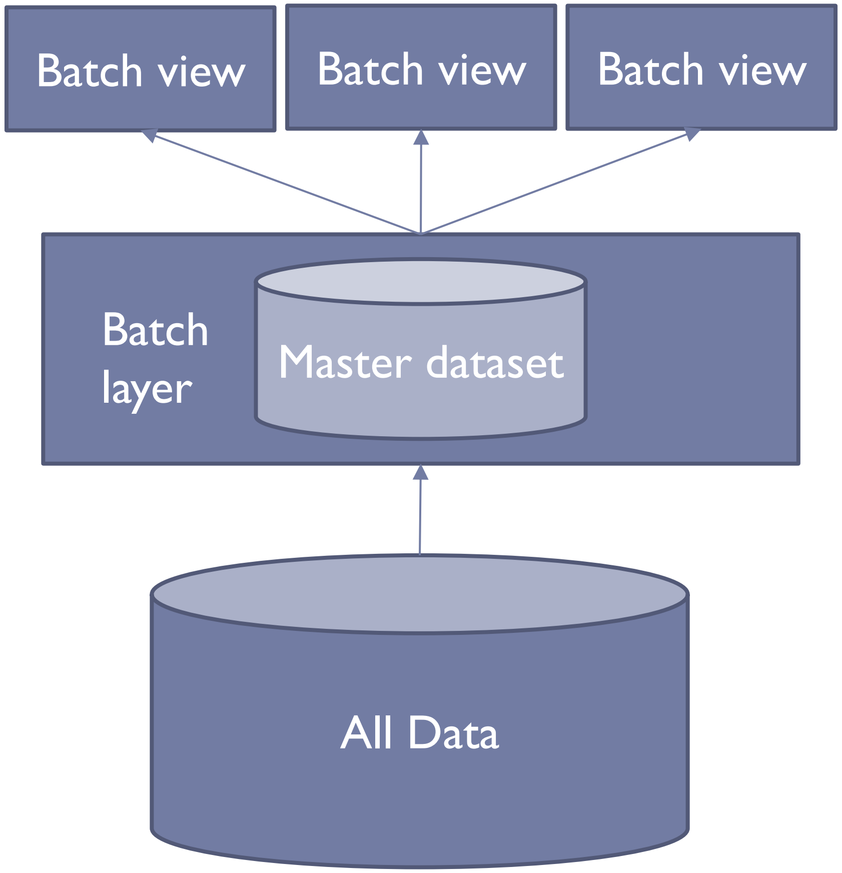
Batch Layer

Batch Layer: implements the equation below

batch view = function (all data)

- **Batch layer functionalities**
  - stores a master copy of the dataset
  - Precomputes batch views

- **Batch layer should be able to**
  - Store constantly growing master dataset
  - Compute arbitrary functions on that dataset

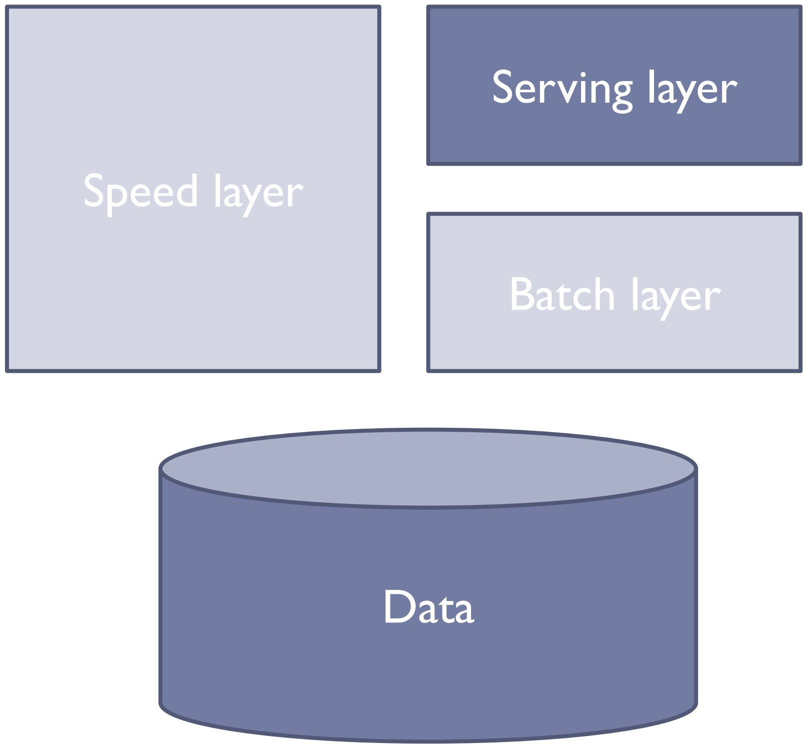


Big Data Systems: Lambda Architecture

Serving Layer

Serving Layer: is a specialized distributed database that loads in a batch view and allows random reads on it.

- **Serving layer**
  - Should be able to support batch updates and random reads
  - It does not need to support random writes -> these databases are extremely simple

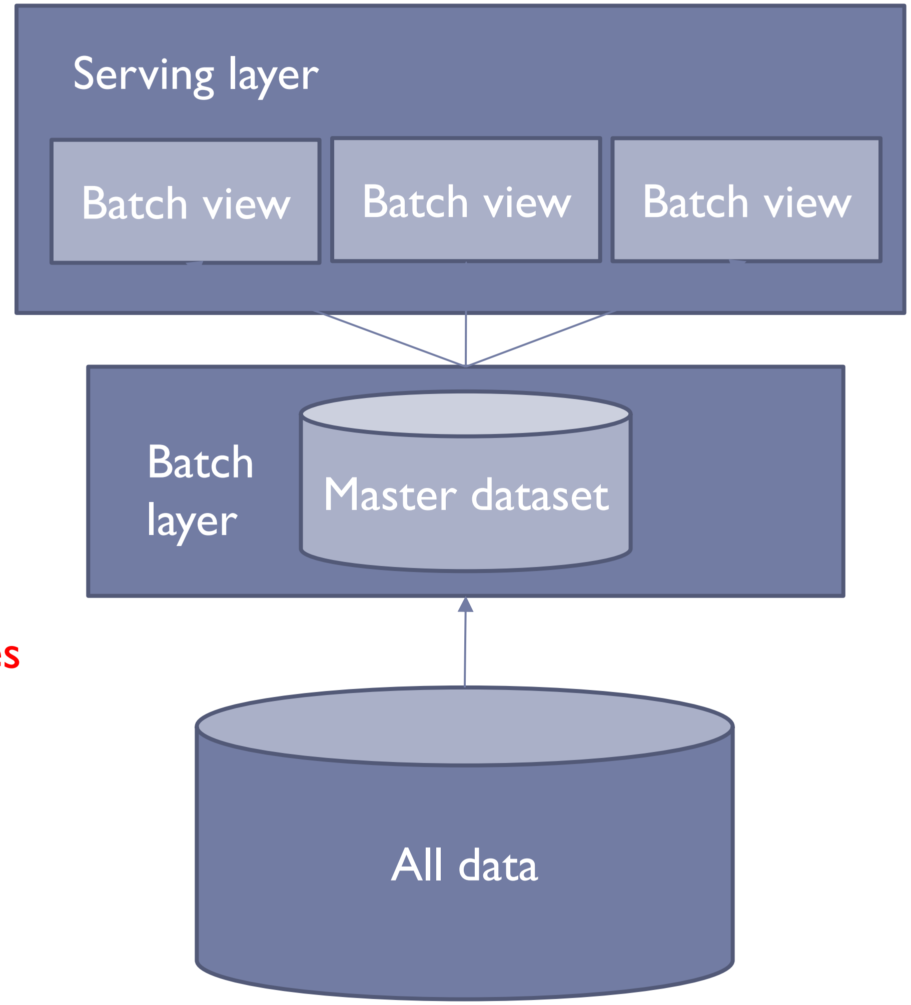


Big Data Systems: Lambda Architecture

Serving Layer

Serving Layer: is a specialized distributed database that loads in a batch view and allows random reads on it.

- **Serving layer**
  - Should be able to support batch updates and random reads
  - It does not need to support random writes -> the databases are extremely simple

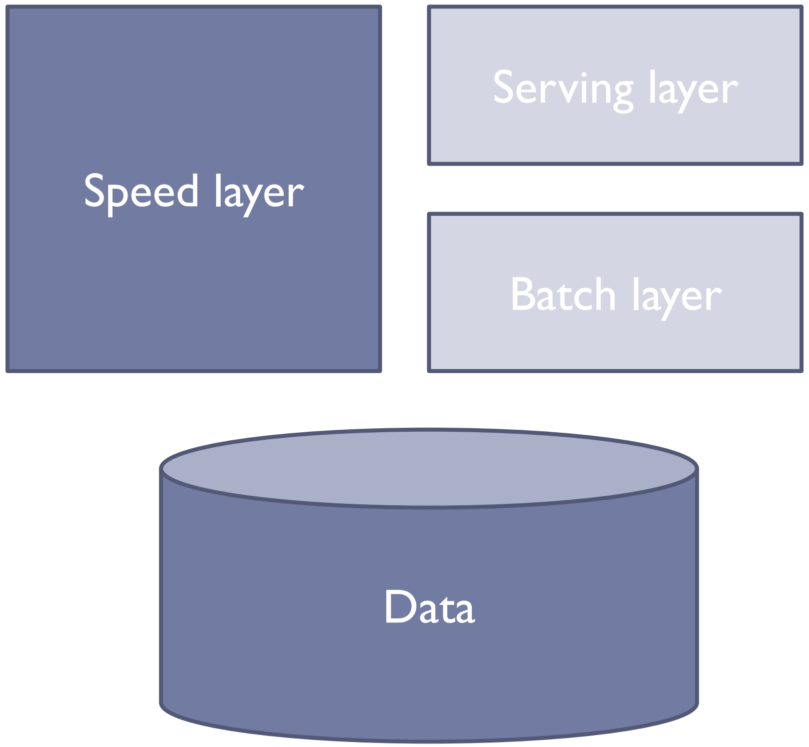


Big Data Systems: Lambda Architecture

Speed Layer

Speed layer: its goal is to ensure new data is represented in the query functions as quickly as required by the application requirements.

- **Speed layer:** can be seen as the batch layer in that it produces views based on data it receives.
- **Speed layer looks only at recent data** while **batch layer looks at the all data**.

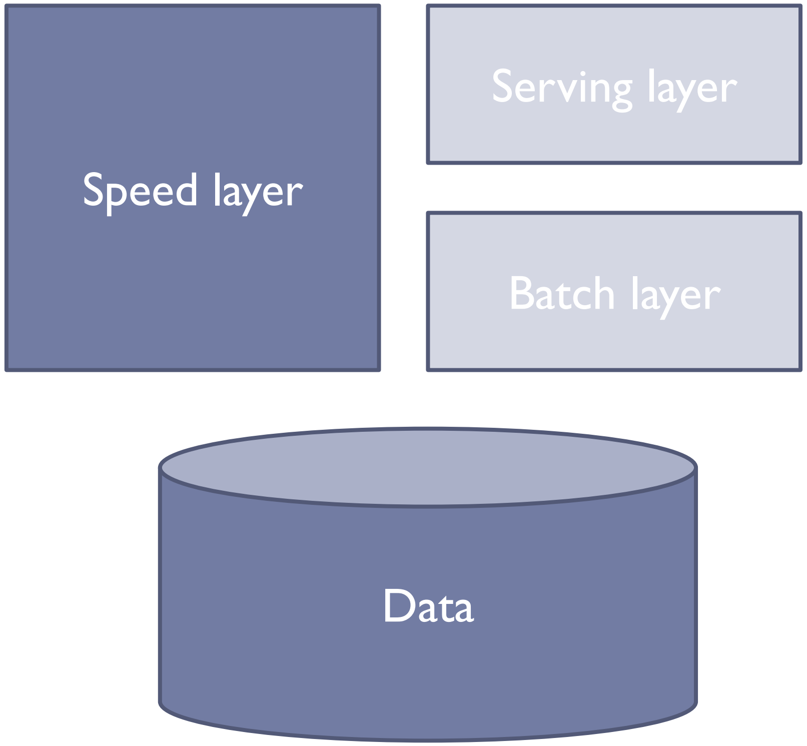


Big Data Systems: Lambda Architecture

Speed Layer

Speed layer: implements the following equation:

realtime view = function (realtime view, new data)



- **Speed layer** updates the realtime views as it receives new data (incremental update) instead of recomputing them from scratch.
- **Speed layer** uses databases that support random reads and random writes, much more complex than the ones used in the serving layer.

Big Data Systems: Lambda Architecture

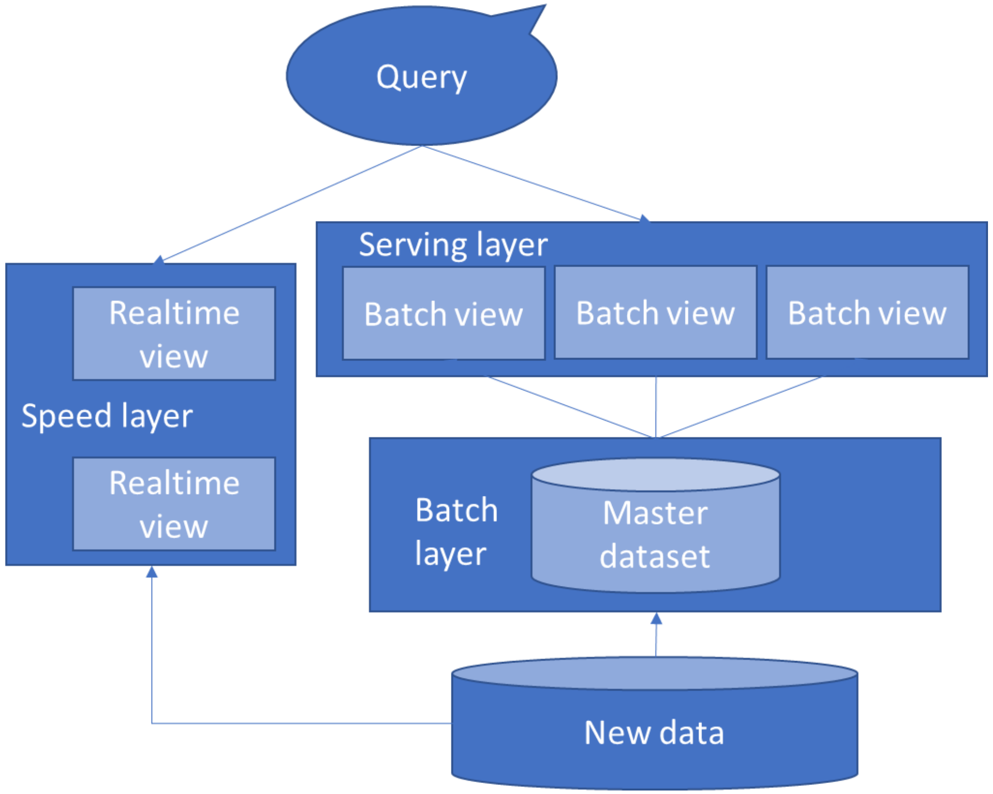
Lambda Architecture Summary

Three equations to summarize the Lambda Architecture.

batch view = function (all data)

realtime view = function (realtime view, new data)

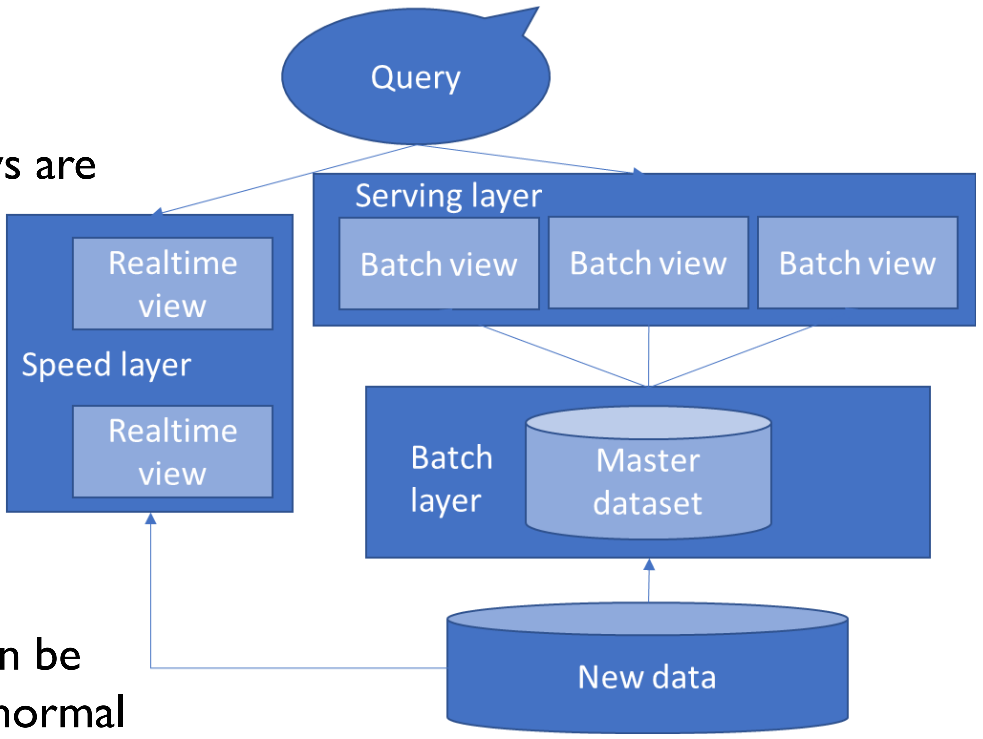
query = function (batch view, realtime view)



Big Data Systems: Lambda Architecture

Lambda Architecture Summary: complexity isolation property

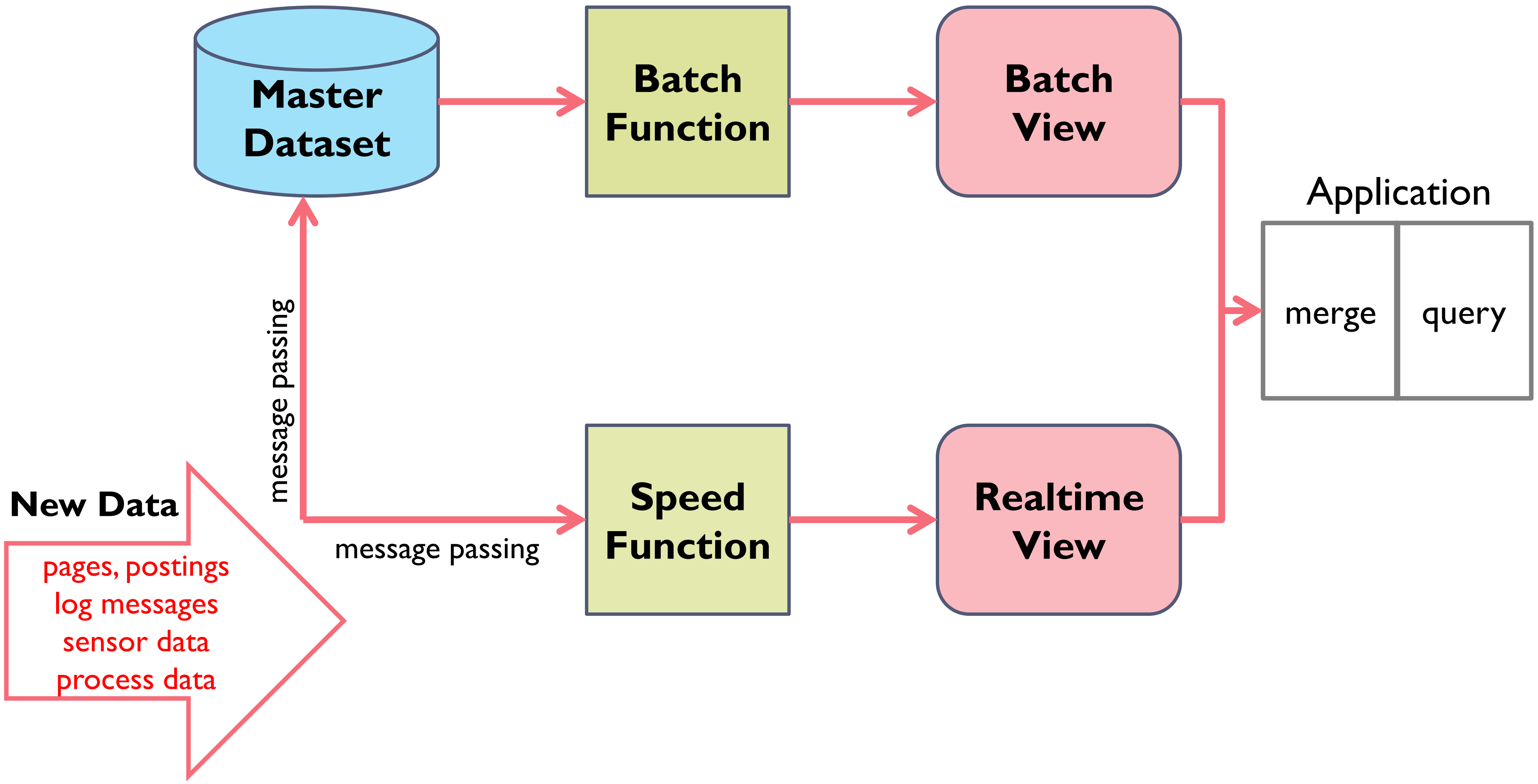
Once the data is in the serving layer, the corresponding results in the real time views are no longer needed.



If anything goes wrong, the speed layer can be discarded and everything will be back to normal within few hours.

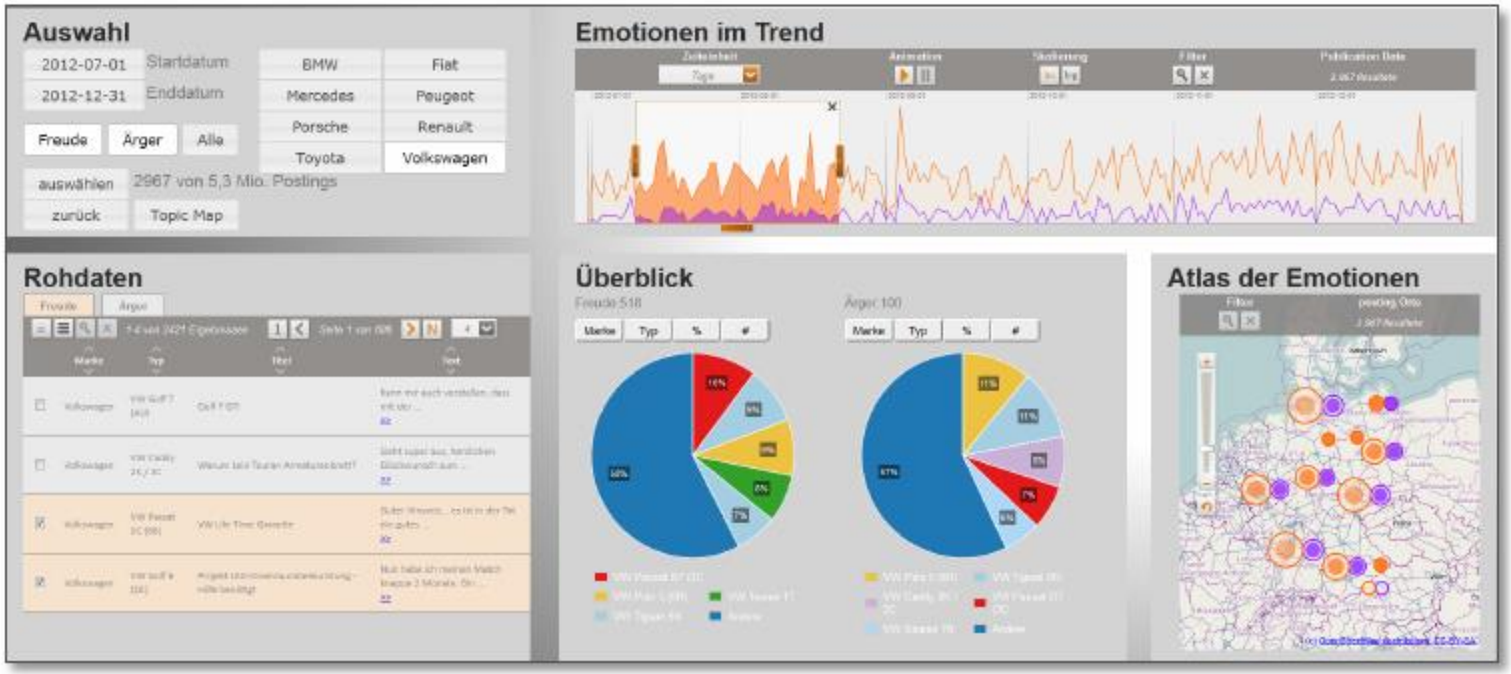
Lambda Architecture - A Running Example

- Lambda Architecture – Components (template)



Lambda Architecture - A Running Example

- The analysis of posts to the automobile enthusiast's forum MOTOR-TALK (https://www.motor-talk.de/)



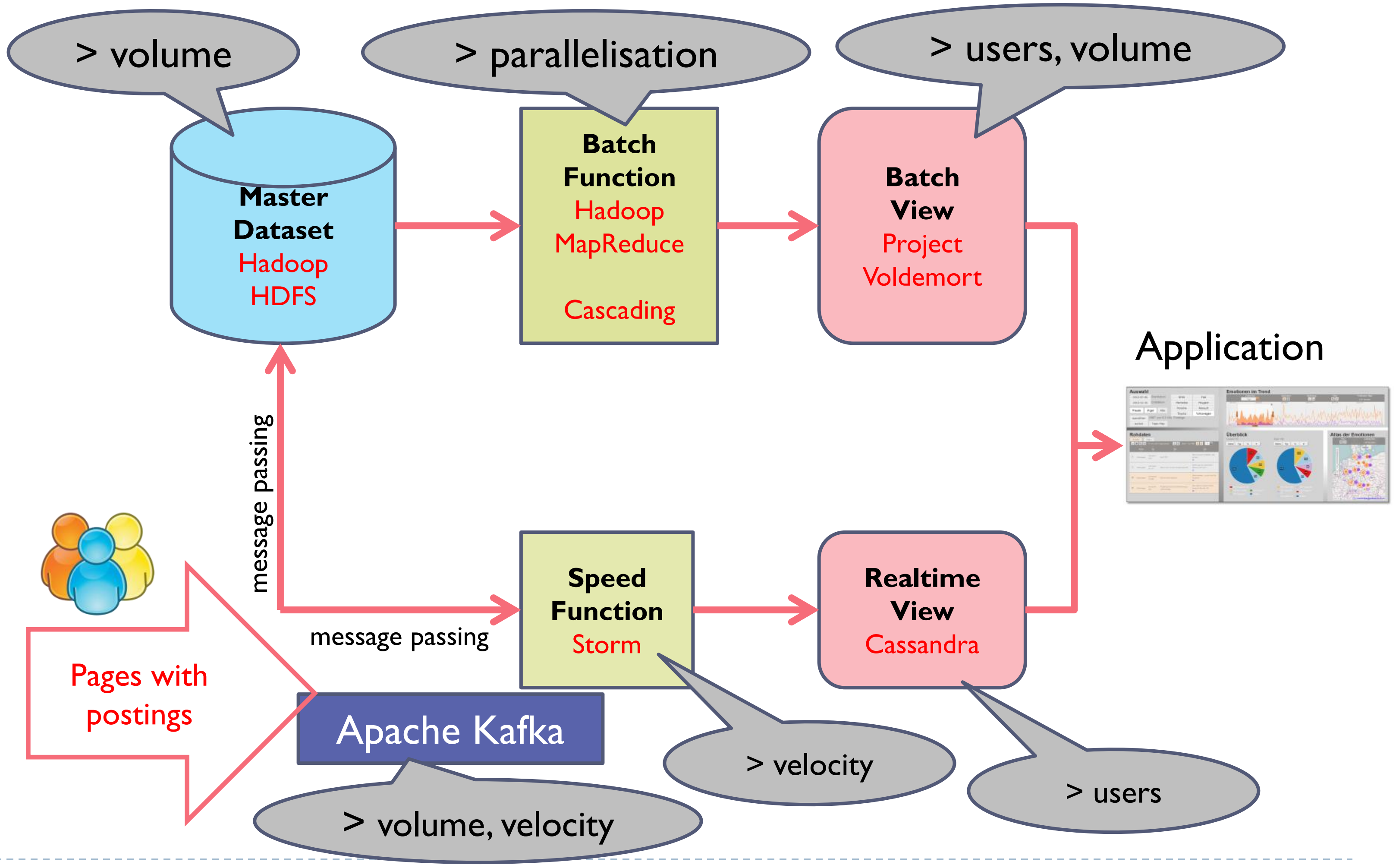
User interface of the example system

Lambda Architecture - A Running Example

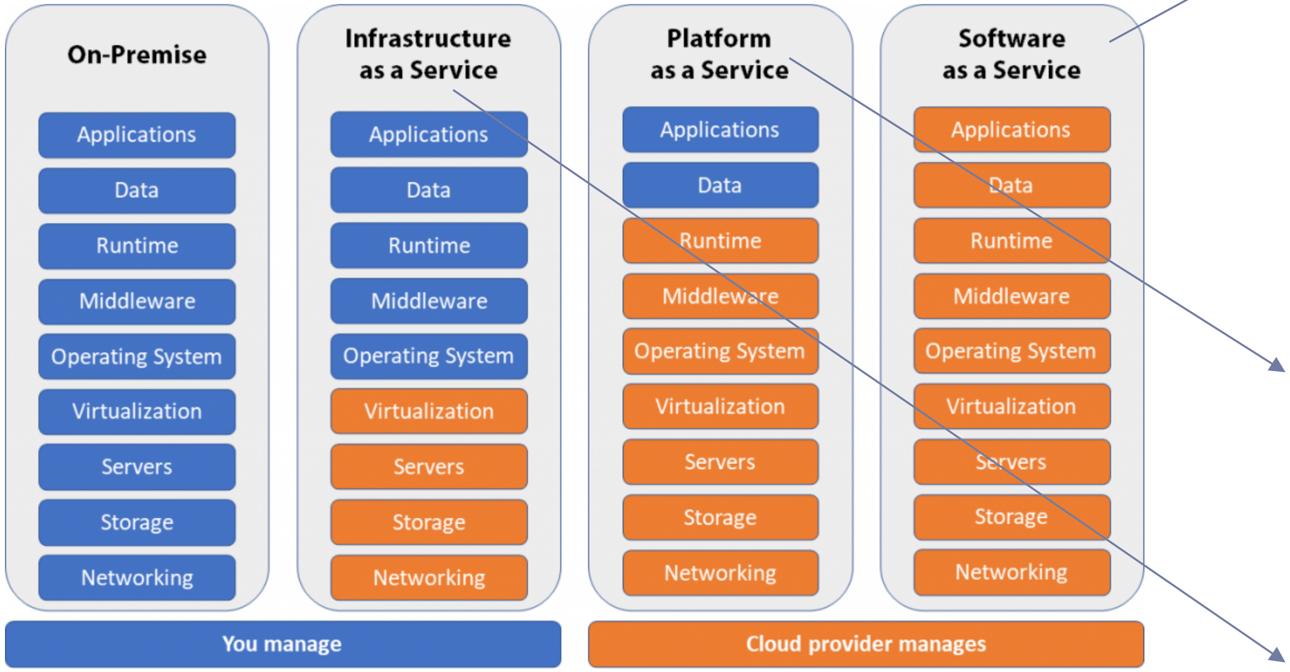
- UI is based on forum posts about cars and their parts
- The application analyses each post to prepare information about which part of which car the post is about and what the *predominant emotion* is in this context
  - i.e., whether the author of the post seems happy, angered, or neutral about the part of the car he is commenting on
- The final application allows to analyze these data in different contexts like brand, location, and time span

Lambda Architecture - A Running Example

- Lambda architecture completed with module choices for the running example



Infrastructure Models for Big Data Applications



SaaS: no implementation required, the provided functionality is used (usually via UI or API).
Ex: Office 365, Dropbox, Google Suite

PaaS: services that provide solutions without being concerned about the operation of the system
Ex: Cloudfoundry, Google App Engine, Azure App Service, managed databases (RDS, CloudSQL)

IaaS: usually virtual machines; the rest of the models are built on top of them.
Ex: Amazon EC2, Google GCE

Database systems and the lambda architecture for big data – take home

- Limitations of the traditional database systems
- Requirements and properties of Big Data systems
- CAP theorem -> important for the design of Big Data systems
- Lambda architecture

Bibliography

- Codd, E., A Relational Model of Data for Large Shared Data Banks, IBM Research Laboratory, Communications of the ACM, vol. 13 (6) (1970)
- Seth Gilbert, Nancy Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT News, Volume 33 Issue 2, June 2002, Pages 51-59, sau: http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf (2002)
- Nathan Hurst, Visual Guide to NoSQL Systems, http://blog.nahurst.com/visual-guide-to-nosql-systems
- Fowler, M., Sadalage, P. NoSQL Distilled: A brief guide to the emerging world of polyglot persistence, 1st ed. Addison-Wesley Professional (2012)
- Introducing JSON, http://www.json.org/
- Guide to the Non - Relational Universe, http://nosql-database.org/
- Perdue, Tim, NoSQL: An Overview of NoSQL Databases, http://newtech.about.com/od/databasemanagement/a/Nosql.htm
- Data Warehouse Systems: Design and Implementation, Vaisman, A.A., Zimányi, E., Data-Centric Systems and Applications, Springer, ISBN: 978-3-642-54654-9 (2014)
- The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Kimball, R., Ross, M. , Wiley Computer Publishing (2013)
- Nathan Marz, James Warren: Big Data - Principles and Best Practices of Scalable Real-Time Data Systems.Manning Publications 2015
- European Data Science Academy (EDSA), EU Project: https://edsa-project.eu/resources/courses
- Leon Tambulea, Databases, UBB Cluj
- Cosmin Lazar, Big Data course, UBB, 2021Marz
- Camelia Andor, NoSQL Databases course, UBB 2023
- Horea Grebla, Cloud Computing – Intro, TDIH Project, UBB 2024