EISTI
Ecole Internationale des Sciences du Traitement de l'Information

# NoSQL Data Stores

## Data Stores for the Cloud


# Summary

1. Introduction
2. Definition
3. Storage paradigms
    - Column oriented database
    - Key-Value oriented database
    - Document oriented database
    - Graph oriented database
4. API NoSQL
5. Big Table
6. APIs
    - JDO
    - JPA
    - API of low level
    - Other APIs Objectify, twig persist

# Introduction to SQL

- Relational Database Managment System (RDBMS) : System widely used
- Query language SQL
  - Tooled and rich language
  - intuitive
  - mostly integrated
  - Standardized (or nearly!)

# Reminders on SQL JOIN

An SQL JOIN clause combines records from two or more tables in a database.

It creates a set that can be saved as a table or used as it is.

A JOIN is a means for combining fields from two tables by using values common to each.

ANSI-standard SQL specifies four types of JOIN:

INNER, OUTER, LEFT, and RIGHT.

One of the most complex SQL operations.

# SQL Join

- SQL joins allow you to associate multiple tables in a single query. This exploits the power of relational databases to get results that combine data from multiple tables efficiently.

# Types of joins (see your course on database)

- There are several ways to associate 2 tables together. Here is the list of the different techniques that are used

INNER JOIN: internal join to return records when the condition is true in both tables. This is one of the most common joins.

CROSS JOIN: Cross-join to make the Cartesian product of 2 tables. In other words, you can join each row of a table with each row of a second table. Attention, the number of results is usually very high.

# Types of joins (continue..)

LEFT JOIN (or LEFT OUTER JOIN): outer join to return all records from the left table even if the condition is not checked in the other table.

RIGHT JOIN (or RIGHT OUTER JOIN): outer join to return all records in the right table even if the condition is not checked in the other table.

FULL JOIN (or FULL OUTER JOIN): outer join to return results when the condition is true in at least one of the 2 tables.

SELF JOIN: allows you to join a table with itself as if it were another table.

NATURAL JOIN: natural join between 2 tables if there is at least one column that has the same name between 2 SQL tables

# ACID

**is an acronym for a set of properties that you would like to hold when modifying a database via a transaction, i.e. a group of related changes to the database.**

**A RDBMS checks ACID properties**

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**

**A cornerstone of RDBMS**

# ACID

**Atomicity** means that you can guarantee that all of a transaction happens, or none of it does.

**Consistency** means that you can guarantee that your data will be consistent before and after a transaction.

**Isolation** means that one transaction cannot read data from another transaction that is not yet completed.

**Durability** means that once a transaction is complete, it is guaranteed that all of the changes have been recorded to a durable medium (such as a hard disk).

# RDBMS

- Properties :
  - Centralization of data and processing
  - Efficiency through standardization of data and processes
  - Cost optimization: replication, parallelization
- RDBMS responds most industrial problems

# Distributed databases (Scalability)

- Ability to manage more data, calculations without altering the performance of a system
- Ways :
  - add memory
  - mutualization of processors
  - duplication services, load balancing
- Vertical or Horizontal scalability
  - Definition : local/distributed data
  - Execution of transactions
    - local : access to data on site
    - global : access to multiple sites
  - Semantic data controler
  - Transaction manager
  - Recovery manager(fault tolerence)
  - Query processor (query optmization)

# Interests of the distribution of data

- Greater reliability (fault tolerance)
- Better performance
- Facilitate the growth of the system

# Disadvantages of data distribution

- Costs (network traffic, hardware and software)
- Synchronization to access data
- Data security
- The distributed deadlock

# Distribution techniques

- PRINCIPLE

  Distribute a relationship (table) across multiple sites.
  On each site is stored part of the relationship.

- TECHNIQUES

| | | |
|---|---|---|
| **Horizontal Cutting** | ⇔ | **Selection** |
| **Vertical Cutting** | ⇔ | **Projection** |
| **Mixed Cutting** | ⇔ | **Selection et projection** |

- PROBLEM
  - Recompose the initial relationship

# Example

Creating a link on my database

create database link *db_toulouse* connect to *user_distant* identified by *password* using ‘XE’

Remark : using → name of the service

local query and remote query

select * from *table_name*;

select * from *table_name@db_toulouse*;

# CAP Theorem

**You need to understand the CAP theorem when you talk or when designing any distributed system.**

**The CAP theorem states that there are three basic requirements which exist in a special relationship when designing applications for a distributed architecture.**

# CAP Theorem

**Consistency - This means that the data in the database remains consistent after the execution of an operation.**
**For example, after an update operation, all clients see the same data.**

**Availability - This means that the data are available for all users in a constant time.**

**Partition Tolerance - This means that the system continues to function even if the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.**
**Network breakdowns must not alter the response**

# CAP Theorem

**Generally, it is not be possible to fulfill or respect all these three requirements in a distributed system.**

**CAP provides the basic requirements for a distributed system to follow two of the three requirements.**

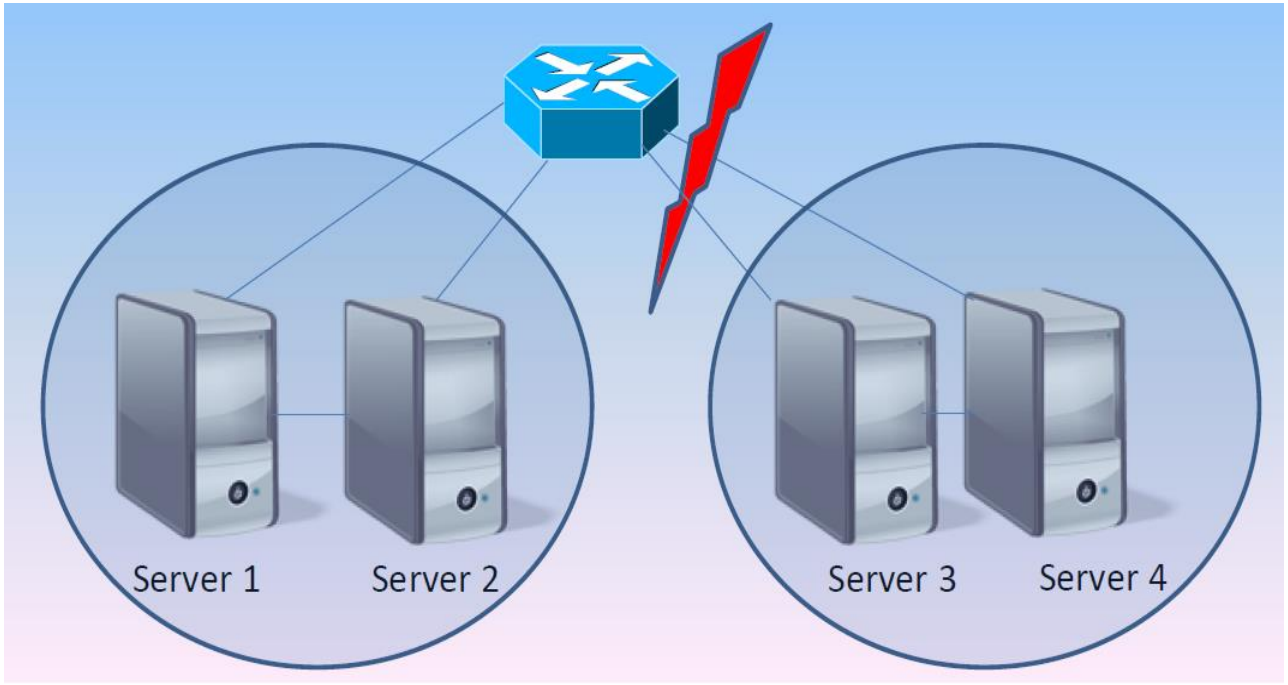
**Distributed systems must be partition tolerant (P), so we have to choose between Consistency and Availability.**

**Current NoSQL databases follow the different combinations of C and A from the CAP theorem.**

# Network Partitions

**A *network partition* can occur when a network failure causes a collection of nodes to be divided into two sub-collections that are isolated from each other.**

**Partitions may occur within or between data centers.**



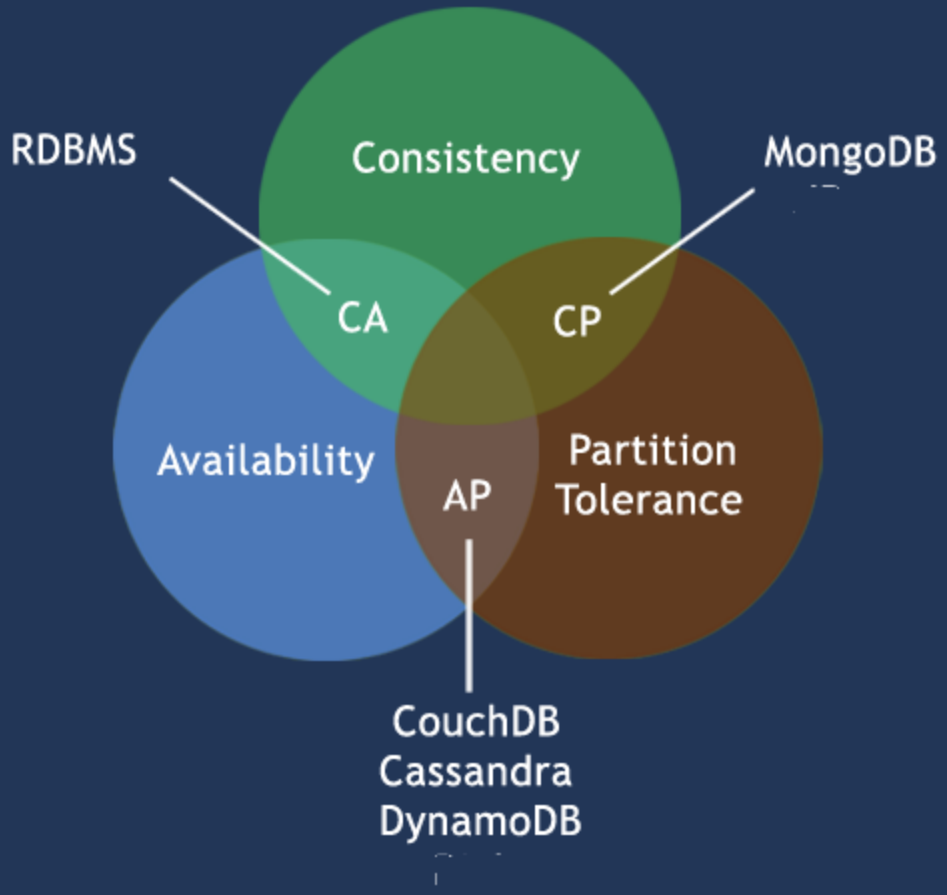
**Here is the brief description of three combinations CA, CP, AP :**

**CA - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.**

**CP - Some data may not be accessible, but the rest is still consistent/accurate.**

**AP - System is still available under partitioning, but some of the data returned may be inaccurate.**

# CAP Theorem


RDBMS
Consistency
MongoDB
CA
CP
Availability
Partition
Tolerance
AP
CouchDB
Cassandra
DynamoDB

# Business Perspective

**In the business world …**

**Availability is more valuable than Consistency**

- **When in doubt, take the customer's order.**
- **Apologize, fix, and compensate later.**

**Do not say**

**"Sorry, we cannot take your money. Our computers are down!"**

# Limitations

Problem of managing large quantities of data (One size fits all).

Solutions envisaged then assimilated:
- Object databases
- XML storage
- Outsourcing

Problems

- Solution few chosen
- Management and material cost  higher

# NoSQL

## What is NoSQL?

**"NoSQL is the term used to designate database management systems (data stores) that differ from traditional relational database management systems.**

**These data stores may not require fixed-table schemas, and usually avoid join operations and typically scale horizontally."**

**- Wikipedia**

**"Non-relational" may be more accurate term than "NoSQL", as some NoSQL DBs do support a subset of SQL.**

**NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways.**

**It is designed for distributed data stores where very large scale of data storing needs to be available (For example, Google or Facebook which collects terabits of data every day for their users).**

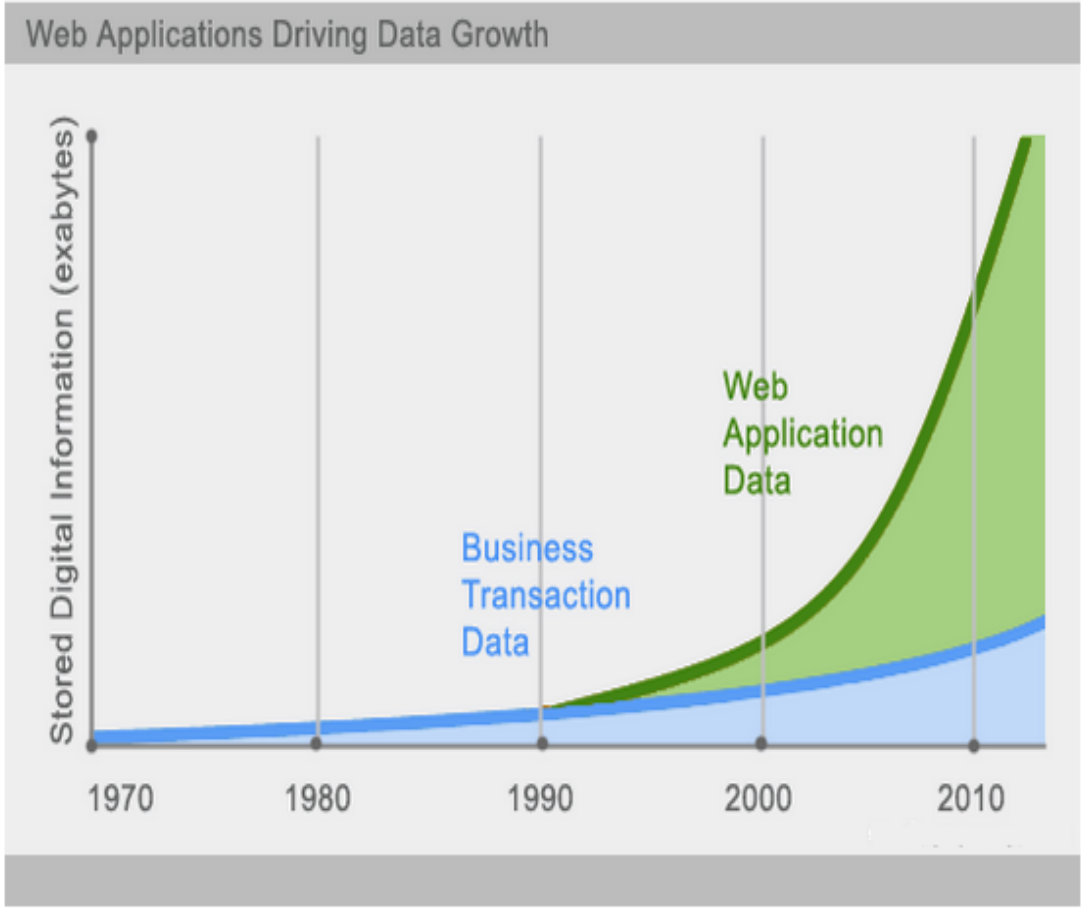
**These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.**

# Web 2.0

- Emerging markets
  - Web application
  - Indexing
  - Statistics
- New problems
  - Large data volumes
  - Large demand for access to data
  - Immediate response time, or at least constant

**Today, data is becoming easier to access and capture through third parties such as Facebook, Google+ and others.**

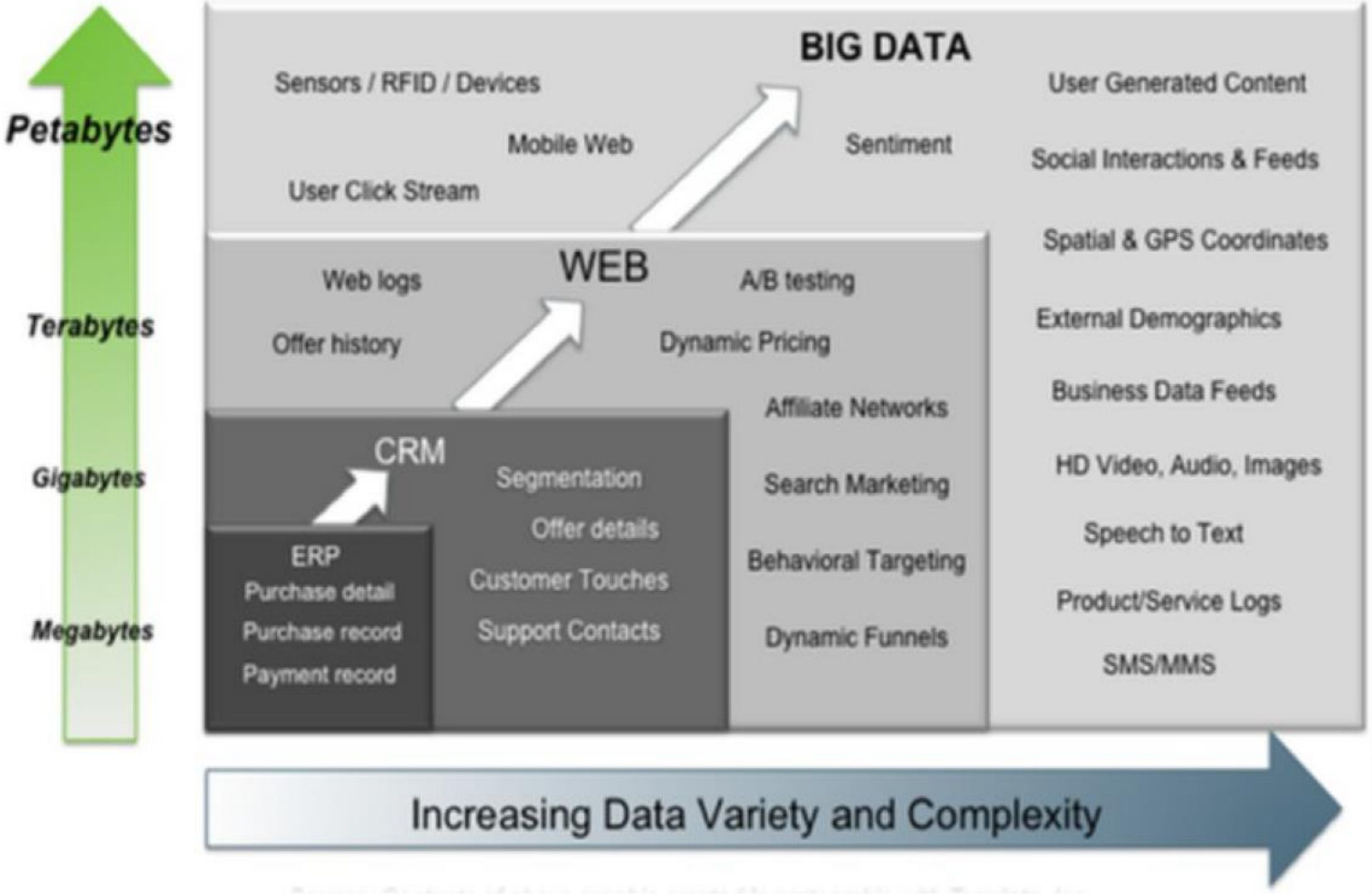
**Personal user information, social graphs, geo-location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially.**

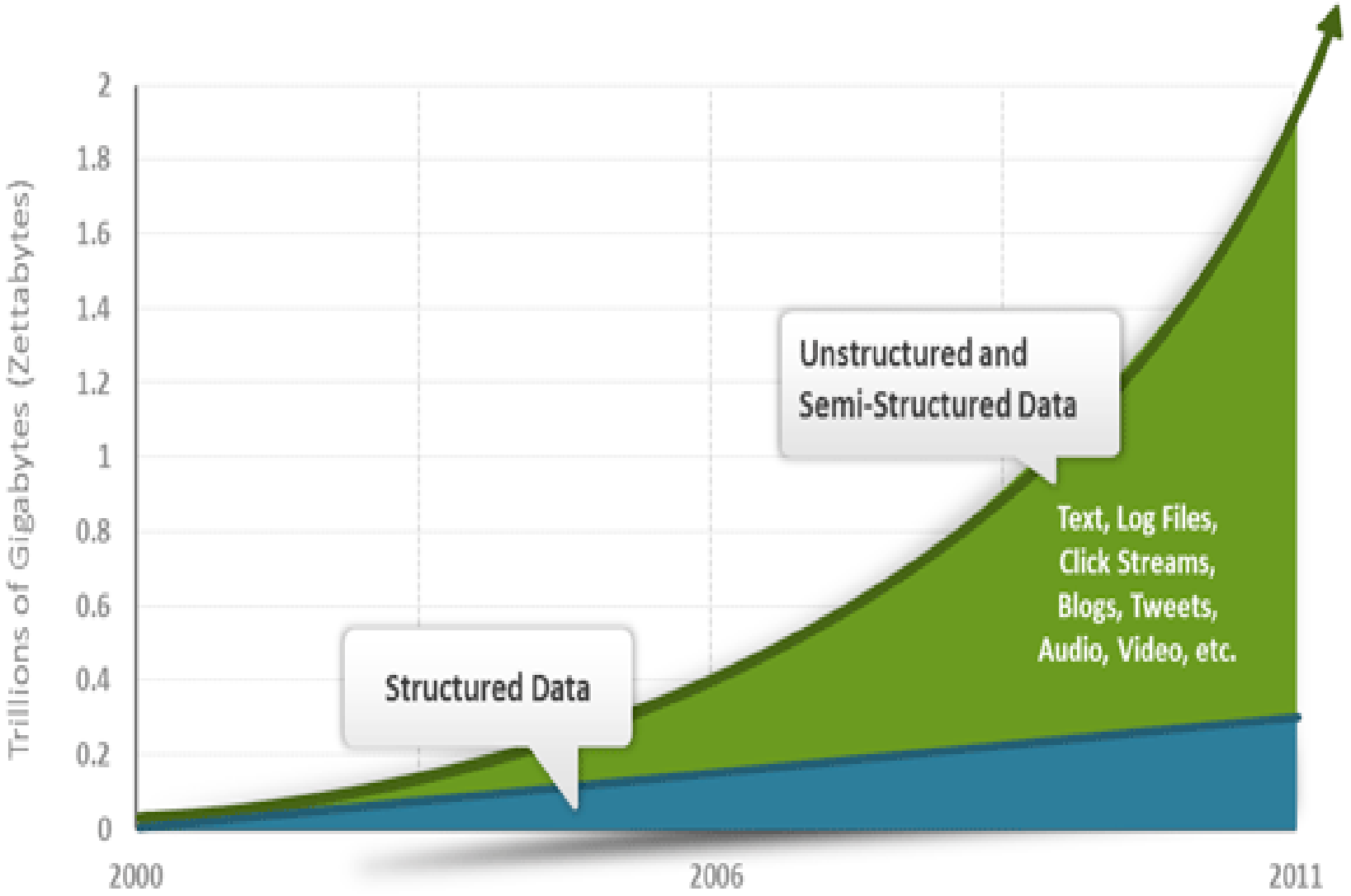



**To use the above services properly requires the processing of huge amounts of data. Which SQL databases are no good for, and were never designed for.**

**NoSQL databases have evolved to handle this huge data properly.**

# Big Data = Transactions + Interactions + Observations






**RDBMS has a problem with Unstructured or Semi-Structured Data**

# Example

## Social-network graph :

```
Each record: UserID1, UserID2
Separate records: UserID, first_name,last_name, age, gender,...
Task: Find all friends of friends of friends of ... friends of a given user.
```

## Wikipedia pages :

```
Large collection of documents
Combination of structured and unstructured data
Task: Retrieve all pages regarding athletics of Summer Olympic before 1950.
```

**A typical traditional, structured, table-based relational database**

## Example of standard structured (relational) database table for user records

| id | lastname | firstname |
|---|---|---|
| 1 | Devin | Florent |
| 2 | Le Nir | Yannick |
| 3 | Loubière | Peio |

**Fixed number of fields for each record – highly structured**

# NoSQL

- **In early 2000 major actors: Google, Amazon**
- **Storage server and machine time available**
- **NoSQL (Not only SQL)**
- **Web technology linked to cloud**
- **objectives:**
  - **size adaptability**
  - **data distribution**
  - **flexibility of structure**
  - **scalability**
- **How: by releasing one of the ACID properties**

# Different Types of NoSQL Systems (4 paradigms to storage data)

- **Distributed Key-Value Systems** - Lookup a single value for a key
  - Amazon's Dynamo

- **Document-based Systems** - Access data by key or by search of "document" data.
  - CouchDB
  - MongoDB

- **Column-based Systems**
  - Google's BigTable
  - Facebook's Cassandra

- **Graph-based Systems** - Use a graph structure
  - Google's Pregel
  - Neo4j

# NOSQL Database Types

## Key-Value


## Column Family


## Document


## Graph

# Relational database

Relational database contains a set of attributes. A data is an element of this set, it is defined by the set of attributes constituting the relation

Exemple:

| id | lastname | firstname | age |
|---|---|---|---|
| 1 | Devin | Florent | 22 |
| 2 | Le Nir | Yannick | 18 |
| 3 | Loubière | Peio | |

# Key-Value Pair (KVP) Storage

- Presentation
  - Classical storage map type
  - No constraint on the key or value
  - A key is associated with complete data

Used by:

- Amazon dynamo,
- Project Voldemort (Linkedin)

# Key-Value Pair (KVP) Storage

- Objectives
  - A different structure for each data
  - A distributed storage
  - Fast  to retrieve data
  - Joints to make statistics on the columns

# Key-Value Pair (KVP) Stores

- Example

  Data1 : (“prof1” : “lasttname” : ”Devin”, “firstname”: “Florent”, “age”:22))
  Data2 : (“prof2” : “lasttname” : ”LeNir”, “firstname”: “Yannickt”, “age”:18))
  Data3 : (“prof3” : “lasttname” : ”Loubiere”, “firstname”: “Peio”))

## Key-Value Pair (KVP) Storage

- **Access data (values) by strings called keys.**
- **Data has no required format – data may have any format**

**Extremely simple interface**

- **Data model: (key, value) pairs**
- **Basic Operations: `Insert(key,value)`, `Fetch(key)`, `Update(key)`, `Delete(key)`**

**Implementation: efficiency, scalability, fault-tolerance**

- **Records distributed to nodes based on key**
- **Replication**
- **Single-record transactions, "eventual consistency"**

# Column-based Stores

- Presentation
  - Data organized in columns Family
  - Each family contains one attribute of all data
  - The union of the columns family rebuilt the data
  - A common identifier allows the unification of data

Used by:

- Google(Big Table),
- Facebook (Cassandra),
- Amazon(SimpleDB)

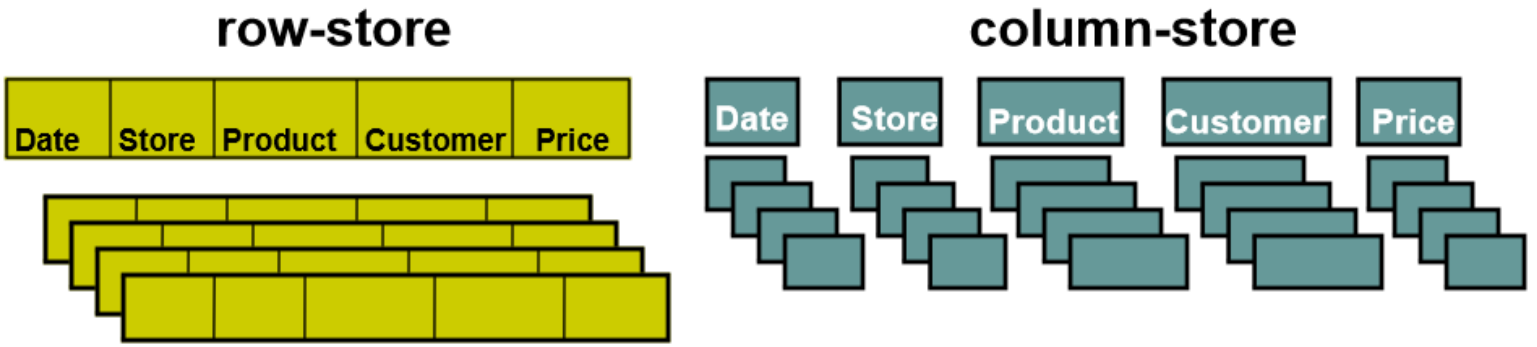
# Column oriented storage

- Objectives
  - A different structure for each data
  - A distributed storage
  - Extremely fast for operation on one attribute (statistics)
  - Joints to rebuild the data are expensive in time

# Column oriented storage

- Example (three columns)
  Firstname family (”Devin”, “Le Nir”, “Loubière”)
  Lastname family (”Florent”, “Yannick”, “Peio”)
  Age family (22, 18)

## Data tables are stored as sections of columns of data, rather than as rows of data.




+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

=> *suitable for read-mostly, read-intensive, large data repositories*

# Column-oriented Storage

**This type of data store is good for**

- Distributed data storage, especially versioned data because of the time-stamps.
- Large-scale, batch-oriented data processing: sorting, parsing, conversion, algorithmic crunching, etc.
- Exploratory and predictive analytics – Business Intelligence.

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//archive/bigtable-osdi06.pdf

# Document oriented Storage

- Presentation
  - Extension of the key-value pair model
  - Stores as a structured document
  - Implementation in different languages
  - Documents” are encoded in a standard data exchange format such as XML, JSON (JavaScript Object Notation) or BSON (Binary JSON).
- Used by:
  - MongoDB,
  - CoucheDB(Ubuntu one)

# Document oriented Storage

- Objectives
  - A different structure for each data
  - A distributed storage
  - Fast to retrieve data or column
  - Adapted to web applications

# Document oriented Storage

- Example (XML)

Data1 : (“prof1” :
<lasttname>Devin</lasttname><firstname>Florent</firstname><age>:22</age>))
Data2 : (“prof2” :
<lasttname>LeNir</lasttname><firstname>Yannick</firstname><age>:18</age>))
Data3 : (“prof3” :
<lasttname>Loubiere</lasttname><firstname>Peio</firstname>))

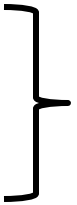
# Document oriented Storage

**Records within a single table can have different structures.**

**An example record from Mongo, using JSON format, might look like**

```
{
"_id" : ObjectId("4fccbf281168a6aa3c215443"),

"first_name" : "Thomas",
"last_name" : "Jefferson",
"address" : {
        "street" : "1600 Pennsylvania Ave NW",
        "city" : "Washington",
        "state" : "DC"
  }
}
```



**Embedded object**

**You can modify the structure of any document on the fly by adding and removing members from the document, either by reading the document into your program, modifying it and re-saving it, or by using various update commands.**

# Document oriented Storage

**Document databases are good for storing and managing**

- **Big Data-size collections of literal documents, like text documents, email messages, and XML documents,**
- **Conceptual "documents" like denormalized (aggregate) representations of a database entity such as a product or customer**

**They are also good for storing "sparse" data in general, that is to say irregular (semi-structured) data that would require an extensive use of "nulls" in an RDBMS.**

# Graph oriented Storage

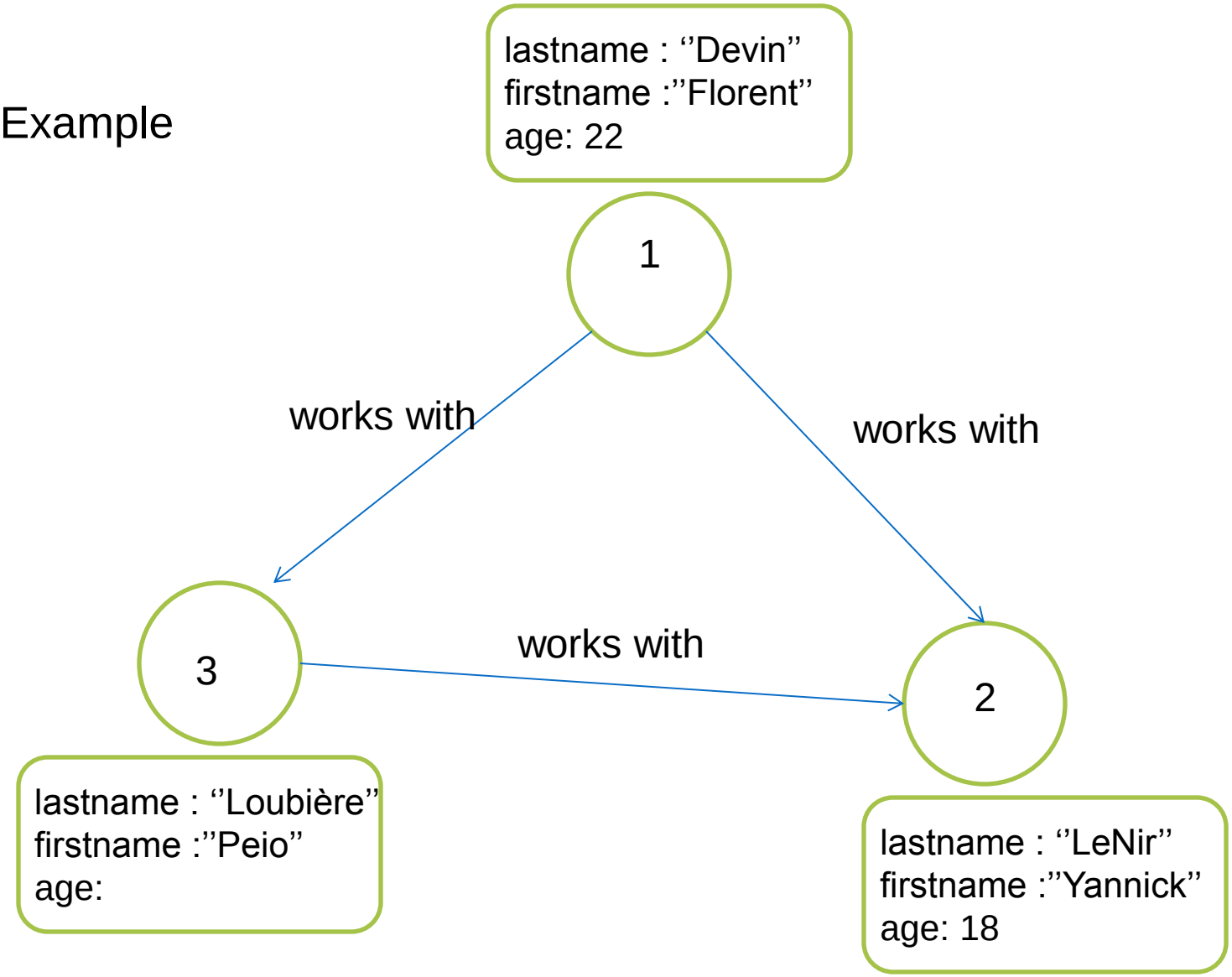
- Presentation
  - Graph where data are vertices
  - Each vertex can contain a weight to guide a search
  - The relationship between these data are represented by oriented edges
  - Model still little used

# Graph oriented Storage

- Objectives
  - A different structure for each data
  - Modelizes social network
  - Allows to use the algorithm to browse a graph

# Graph oriented Storage

- Example



# Graph oriented Storage

**Apply graph theory in the storage of information about the relationship between entries**

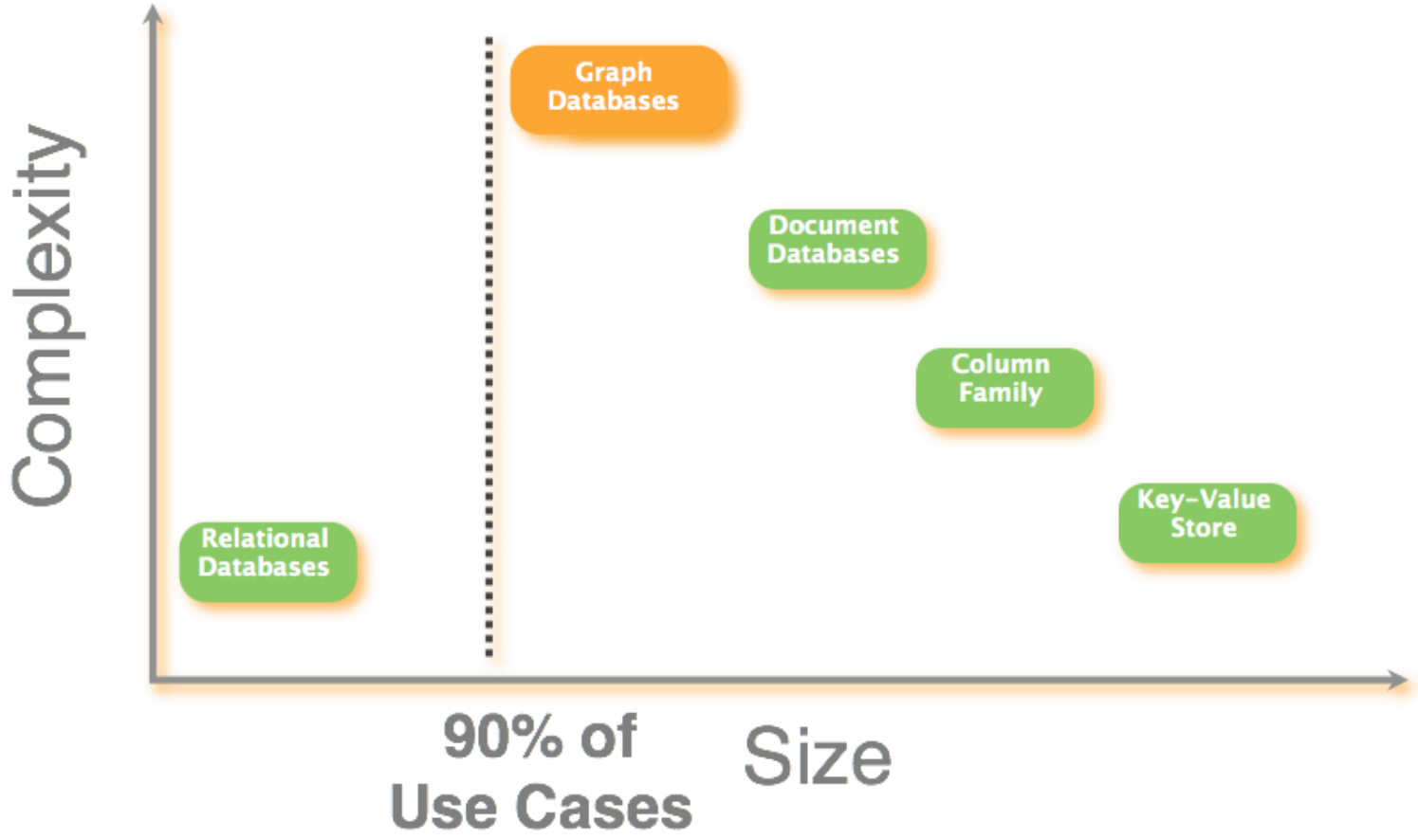
**A graph database is a database that uses graph structures with nodes, edges, and properties to represent and store data.**

**By definition, a graph database is any storage system that provides index-free adjacency.**

**- This means that every element contains a direct pointer to its adjacent element and no index lookups are necessary.**

**In general, graph databases are useful when you are more interested in relationships between data than in the data itself: for example, in representing and traversing social networks, generating recommendations, or conducting forensic (medico-legal ) investigations (e.g. pattern detection).**

# Complexity vs Size



# RDBMS versus NoSQL

## Work to do!