Database Models: Beyond RDBMS

UA.DETI.CBD
José Luis Oliveira / Carlos Costa

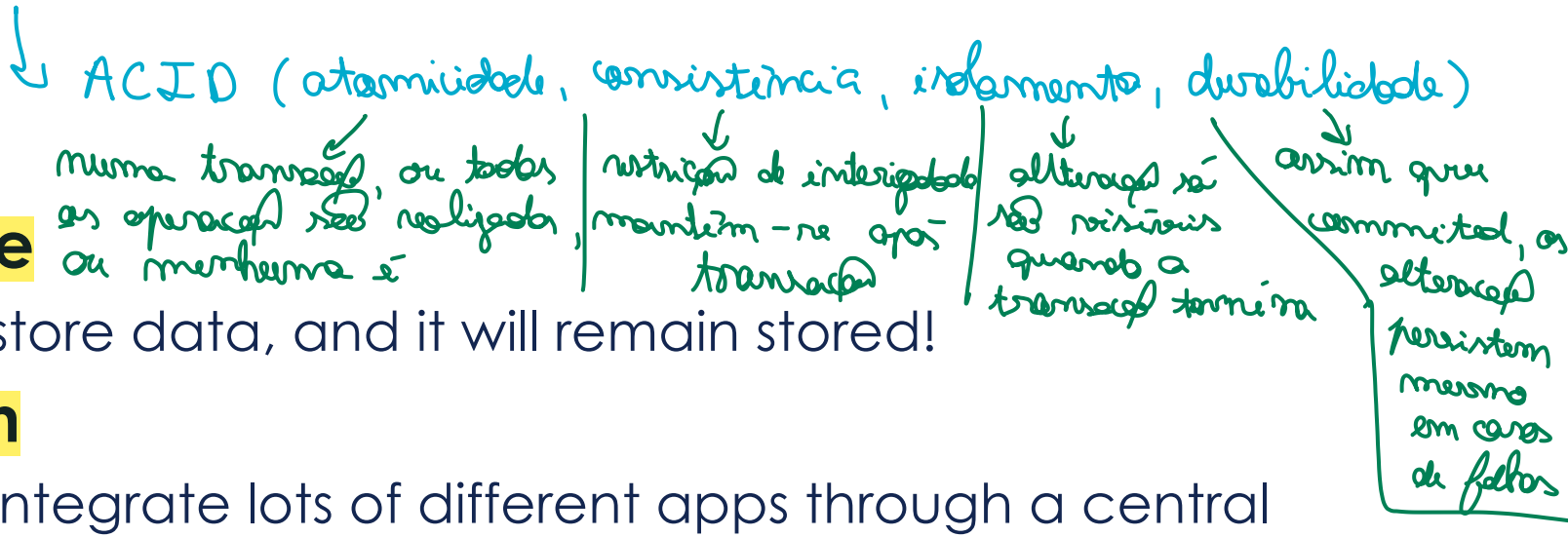
The Battle of the Data Models

- Data models are perhaps the most important part of developing software

(os modelos de dados com os quais os programas vão trabalhar têm um papel fundamental na sua programação)

When we have some data...

- **Relational Databases** solve most data problems



Why?

- **Persistence**
  - We can store data, and it will remain stored!
- **Integration**
  - We can integrate lots of different apps through a central DB
- **SQL**
  - Standard, well understood, very expressive
- **Transactions**
  - ACID transactions, strong consistency

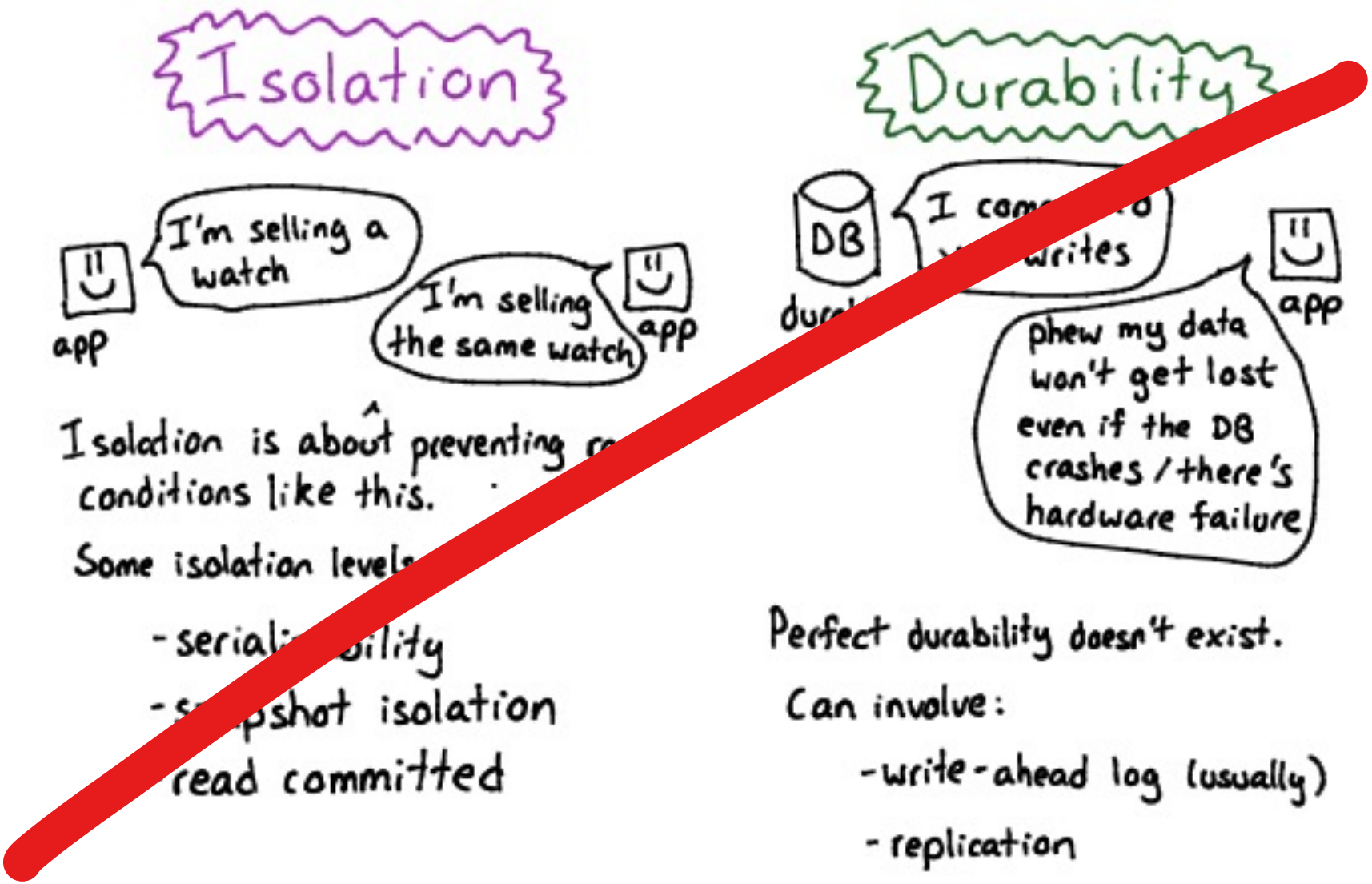
Transactions – ACID Properties

- **Atomic**
  - All of the work in a transaction completes (commit) or none of it completes
- **Consistent**
  - A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
- **Isolated**
  - The results of any changes made during a transaction are not visible until the transaction has committed. Concurrent interactions behave as though they occurred serially
- **Durable**
  - The results of a committed transaction survive failures

ACID

https://jvns.ca/blog/2017/06/11/log-structured-storage/

ACID



https://jvns.ca/blog/2017/06/11/log-structured-storage/

The Relational Model

- The relational model, proposed by Edgar Codd in 1970, is still the best-known data model today.
  - data is organized into relations (in SQL: tables), where each relation is an unordered collection of tuples (rows).

  - The goal of the relational model was to hide that implementation detail behind a cleaner interface.

Rivals of the Relational Model

- Over the years, there have been many competing approaches to data storage and querying.
  - Object databases came and went again in the late 1980s and early 1990s.
  - XML databases appeared in the early 2000s, but have only seen niche adoption.
- Much of what you see on the web today is still powered by relational databases
  - Online publishing, discussion, social networking, e-commerce, games, software-as-a-service productivity applications, or much more.
- Now NoSQL is the most recent attempt to overthrow the relational model's dominance.

Current trends and Issues

- A few key trends and issues have motivated change in relational data storage technologies
  - …In use cases
  - …In technology

- Key trends include:
  - Increasing **volume of data and traffic**
  - More complex data connectedness
- Key Issues include:
  - The **impedance mismatch** problem

bottleneck das bases de dados relacionais é a rapidez de execução das operações

Impedance Mismatch

paradigma entre princípios de engenharia de software e princípios relacionais

- **Object** Orientation
  - based on software engineering principles
- **Relational** Paradigms
  - based on mathematics and set theory

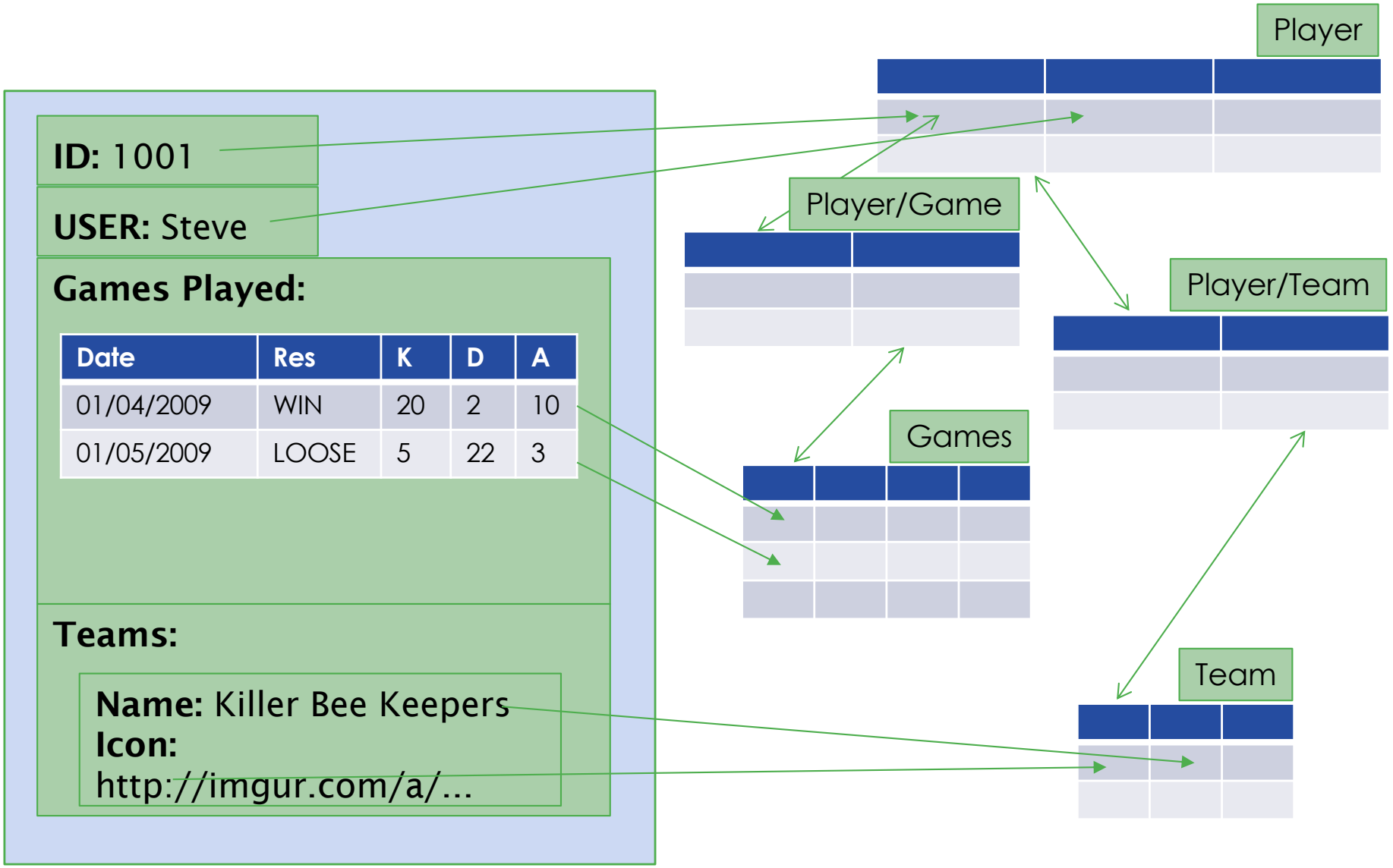
conflito

Ex: • vários objetos que representam funcionários numa empresa
• cada funcionário tem o seu departamento } mas vários funcionários podem trabalhar no mesmo departamento

- To store data persistently in modern programs a single logical structure must be split up
  - The nice word is **normalised**

⇓
iremos ter uma repetição de departamentos nos funcionários e a base de dados não estará normalizada
(redundância de dados)

Impedance Mismatch – example



Impedance Mismatch – example

http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table

id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

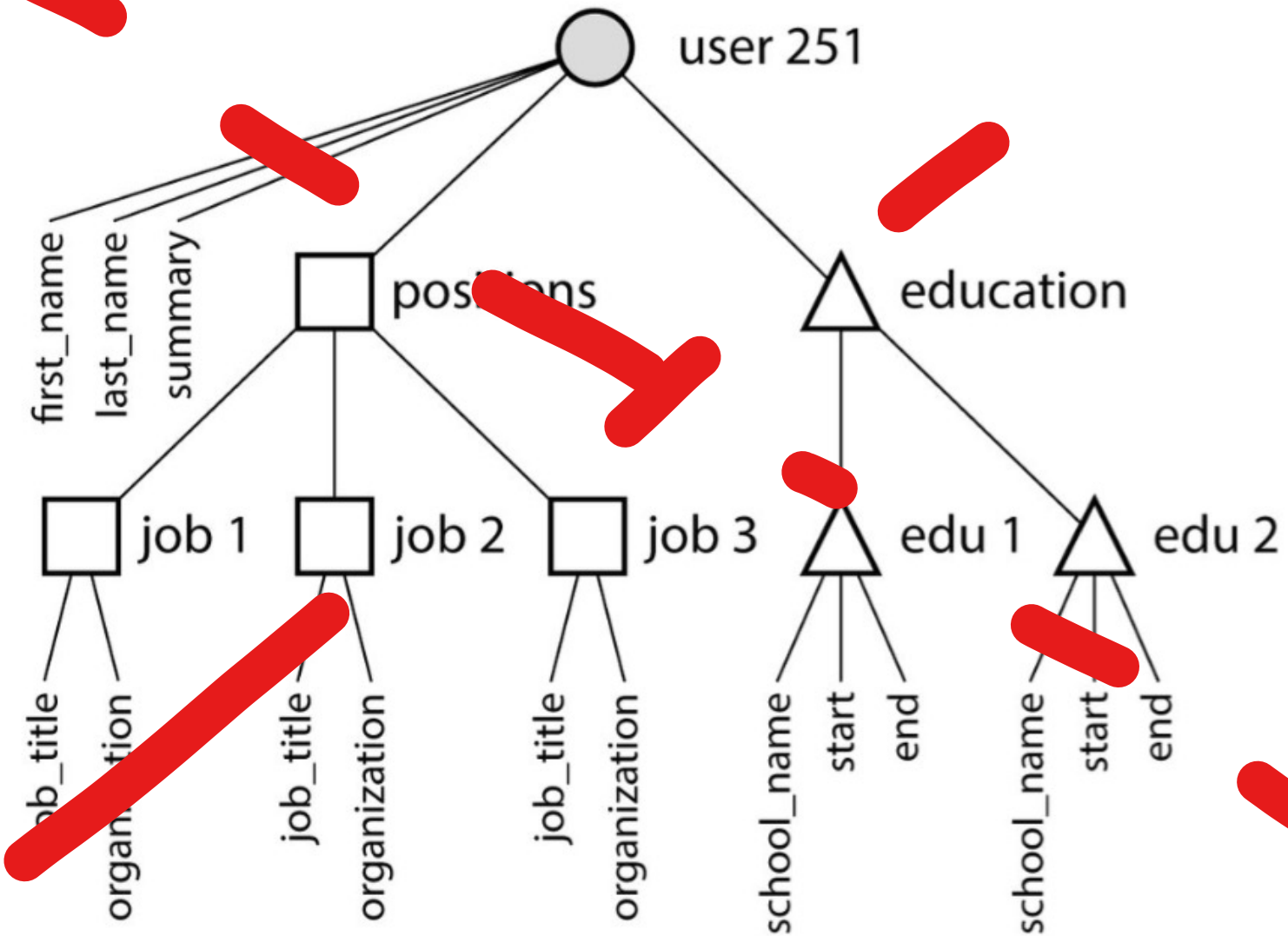
Normalization

→ visa reduzir a redundância de dados

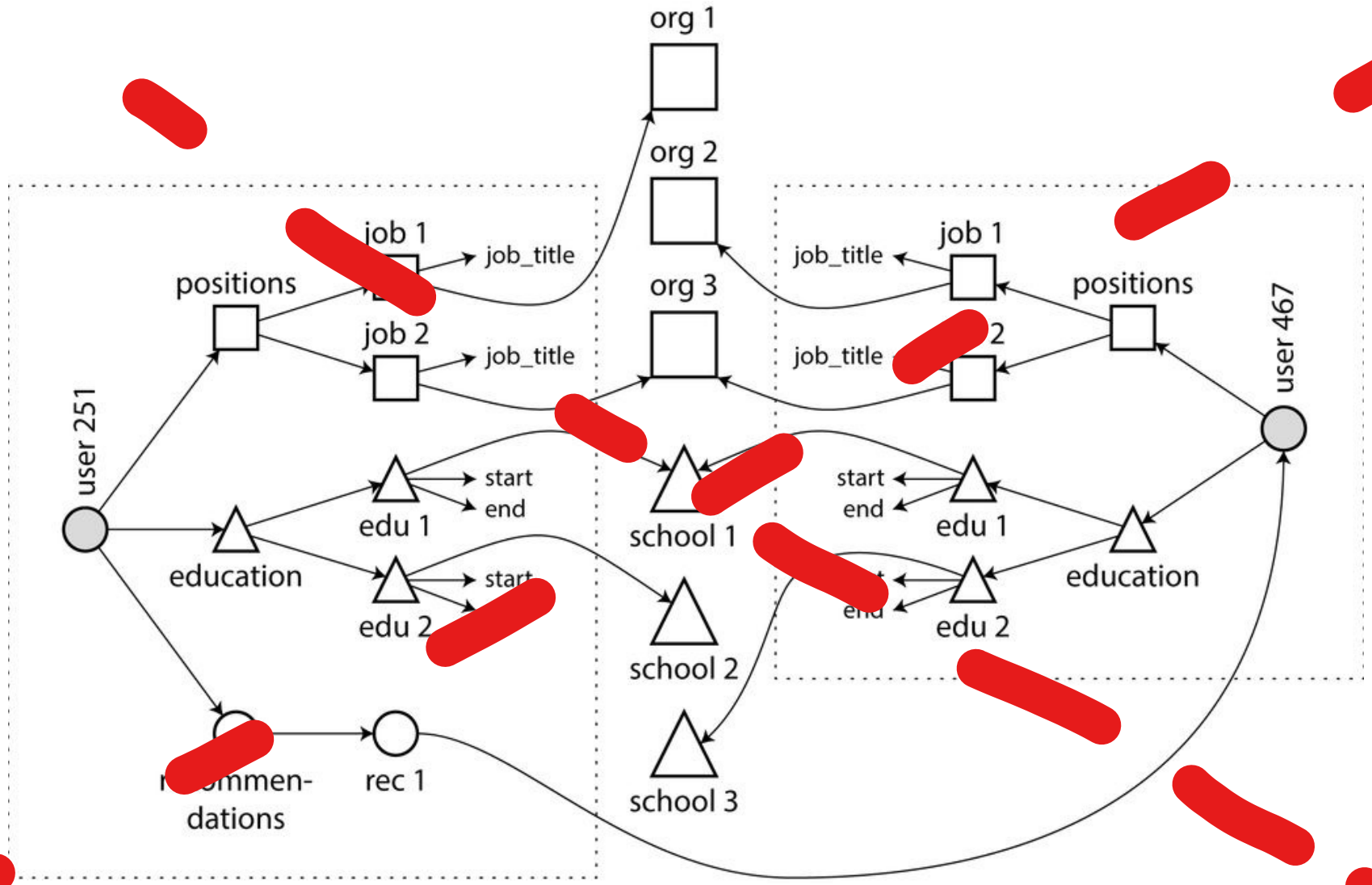
UTILIZA IDs (para identificar entidades)

- Why IDs (region_id, industry_id, ..) and not plain-text?
  - Consistent style and spelling across profiles,
  - Avoiding ambiguity, e.g. if there are several cities with the same name, (se houverem entidades semelhantes)
  - The name is stored only in one place, so it is easy to update, (facilita alterações - informação está organizada em só 1 tabela
  - Simplify translation into other languages,
- A database in which entities like region and industry are referred to by ID is called **normalized**.
- A database that duplicates the names and properties of entities on each document is **denormalized**.

One-to-Many relations



Many-to-Many relationships



Increased Data Volume

- We are creating, storing, processing more data than ever before!
  - *"From 2005 to 2020, the digital universe will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, or 40 trillion gigabytes (more than 5,200 gigabytes for every man, woman, and child in 2020)".*
    - THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East, Idc 2012, John Gantz and David Reinsel
  - *"IDC predicts that the collective sum of the world's data will grow from 33 zettabytes this year to a 175ZB by 2025."*
    - The Digitization of the World, From Edge to Core, Nov 2018
    - https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

EB = 10^{18} ZB = 10^{21}

Increased Data Connectivity

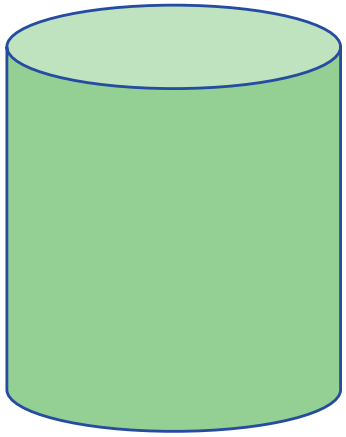
- The data we're producing has fundamentally changed
  - from isolated Text Documents (early 1990s)
  - … to html pages with links (early web)
  - … to blogs with pingback, RSS feeds (web 2.0)
  - … to social networks (… add links between people)
  - … to massive linked open data sets (web 3.0… one of them anyway)

Dealing with data size Trends

- Two options when dealing with these trends:

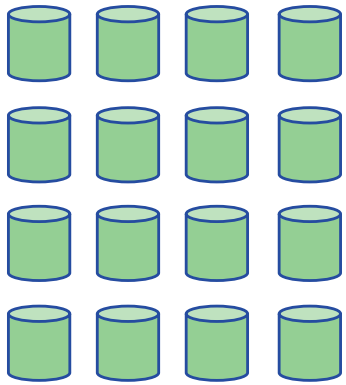
(construir bases de dados maiores)

- Build Bigger Database machines
  - This can be expensive
  - Fundamental limits to machine size (limitações físicas)



(criar um sistema distribuído de BD)

- Build Clusters of smaller machines
  - Lots of small machines (commodity machines)
  - Each machine is cheap, potentially unreliable
  - Needs a DBMS which understands clusters



mais barata, mas menos fiável

RDBMS have fundamental issues

(Relational database management system)

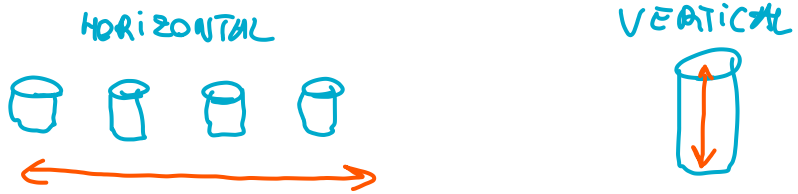
- In dealing with (horizontal) scale
  - Designed to work on single, large machines
  - Difficult to distribute effectively

(bom escalamento vertical, mau escalamento horizontal)

More subtle: An Impedance Mismatch

⚠ Capacidade ≠ Desempenho

Escalar verticalmente (scale up) significa adicionar recursos em um único nó do sistema (mais memória ou um disco rígido mais rápido). Escalar horizontalmente (scale out) significa adicionar mais nós ao sistema, tais como um novo computador



The NoSQL Movement

(Not only SQL)

NoSQL

- The term NoSQL is unfortunate, since it doesn't refer to any technology
  - "Not only SQL"
- Nevertheless, the term struck a nerve, and quickly spread through the web startup community and beyond.
- Several interesting database systems are now associated with the #NoSQL hashtag.

The NoSQL movement (base de dados não relacionais)

- Key attributes include:
  - **Non-Relational**
    - They can be, but aren't good at it
  - **Simple API**
    - No Join
  - **BASE** & **CAP** Theorem
    - No ACID requirements
  - **Schema-free**
    - Implicit schema, application side (sem configuração da BD)
  - **Inherently Distributed**
  - **Open Source**
    - mostly

BASE Transactions (oposição ao princípio ACID)

em resposta às limitações de consistência que um certo SD suporta

- Acronym contrived to be the opposite of ACID
  - Basic Availability
    - The database appears to work most of the time.
  - Soft-state
    - Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time. (escritas num nó da base de dados não têm de ser escritas garantidamente em simultâneo nos outros nós).
  - Eventual consistency
    - Stores exhibit consistency at some later point (e.g., lazily at read time).
- Characteristics
  - Optimistic
  - Simpler and faster
  - Availability first
  - Best effort
  - Approximate answers OK

(consistência atingida em algum momento posterior)
(quando nós estão desatualizados em relação a outros a determinado momento)

torna a BD + rápida

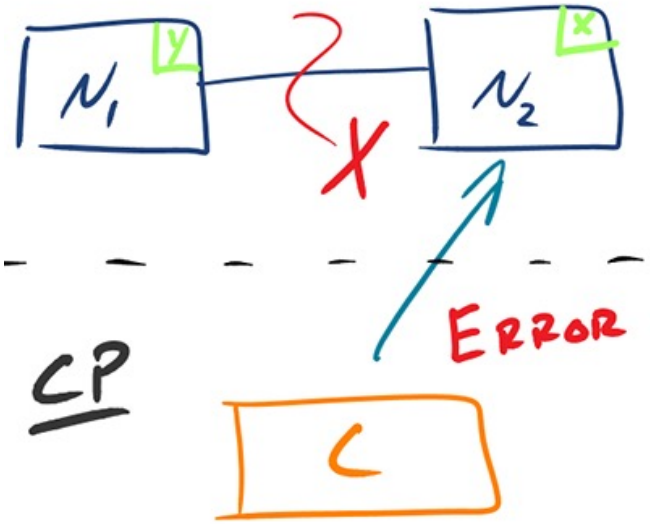
Brewer's CAP Theorem

- A distributed system can support only two of the following characteristics:
  - **Consistent**
    - writes are atomic, all subsequent requests retrieve the new value
  - **Available**
    - The database will always return a value so long as the server is running
  - **Partition Tolerant** (tolerante a falhas – um sistema consegue funcionar mesmo que 1 nó deixe de responder)
    - The system will still function even if the cluster network is partitioned (i.e. the cluster loses contact with parts of itself)
- The overly stated well cited issue is:
  - We can only ever build an algorithm which satisfies 2 of 3.
    - But .. horizontal scaling strategy is based on data partitioning;
    - Therefore, designers are forced to decide between consistency and availability.

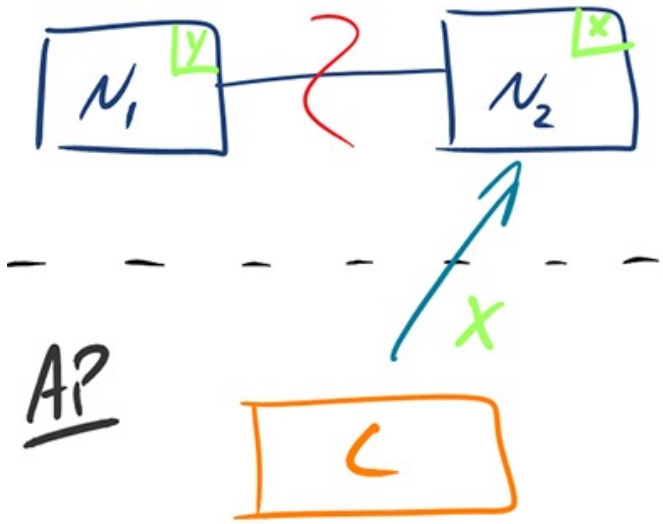
↳ há sempre data partitioning (P)
- CP: all clients will have the same view of the data
- AP: each client can always read and write

Brewer's CAP Theorem

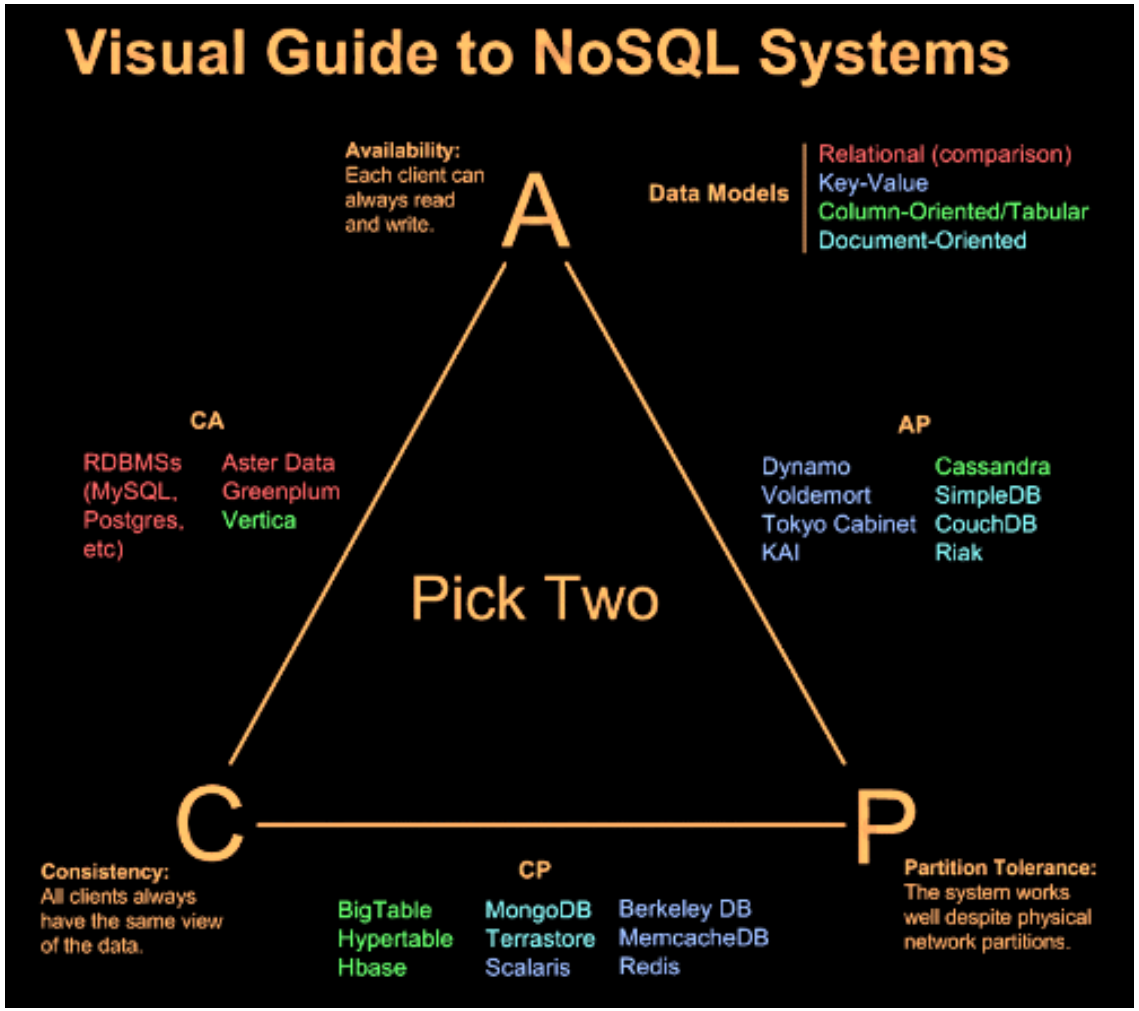
- **CP** - Consistency/Partition Tolerance



- **AP** - Availability/Partition Tolerance

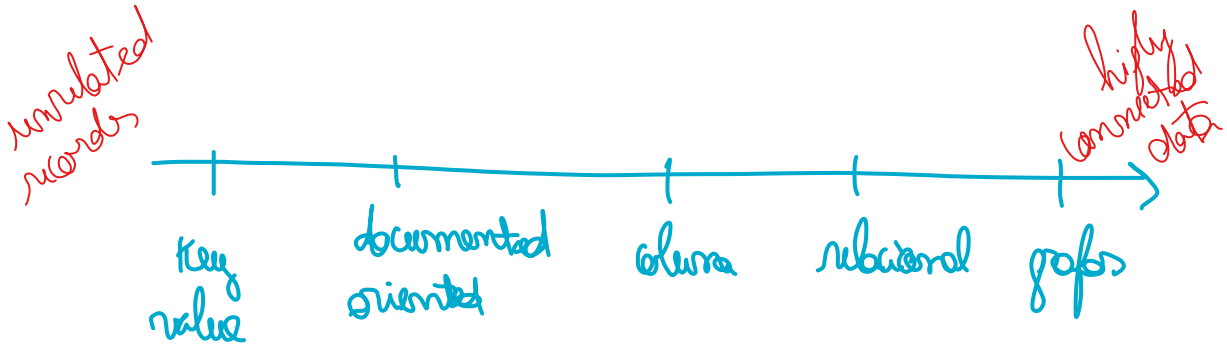


CAP Theorem


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

Types of NoSQL Databases

- Core types
  - **Key-value** stores
  - **Document** stores
  - **Column** stores
  - **Graph** databases



- Non-core types
  - **Object** databases
  - Native **XML** databases
  - **RDF** stores
  - …

Key-Value Databases – Basics

- Data model
  - The most simple NoSQL database type
  - Works as a simple hash table (mapping)
- Key-value pairs
  - **Key** (id, identifier, primary key) – usually a string.
  - **Value**: can be anything (text, structure, image, etc.) – a black box for the database system.
- Query patterns
  - Create, update or remove value for a given key
  - Get value for a given key
- Characteristics
  - great performance, easily scaled, …
  - not for complex queries nor complex data

Key-Value Databases – Basics

- Suitable use cases
  - Session data, user profiles, user preferences, shopping carts, …
    - I.e. when values are only accessed via keys
- When not to use
  - Relationships among entities
  - Queries requiring access to the content of the value part

Armazenam dados não persistentes e não permitem relações entre entidades

SAI NO TESTE

Document Databases – Basics

- Data model - Documents
  - Self-describing complex data structure
  - **Hierarchical tree structures** (JSON, XML, …)
    - Scalar values, maps, lists, sets, nested documents, …
  - Identified by a unique identifier (key, …)
- Document data stores understand their documents
  - Queries can run against values of document fields
  - Indexes can be constructed for document fields
- Query patterns
  - Create, update or remove a document
  - Retrieve documents according to complex queries
- Difference from Key-Value stores
  - Extended key-value stores. The value part is examinable!

Document Databases – Basics

```
{
"_id": "1",
  "name": "steve",
  "games_owned": [
    {"name":"Super Meat Boy"},
    {"name":"FTL"},
  ],
}
```

```
{
  "_id": "2",
  "name": "darren",
  "handle":"zerocool",
  "games_owned": [
    {"name":“FTL"},
    {"name":“Assassin’s Creed 3“, “dev”: “ubisoft”},
  ],
}
```

JSON é um formato de troca de informação feito através de base de dados de tipos document oriented, com diversas quantidades de chaves estrangeiras

⇓

apenas se colocam os dados que interessam

Document Databases – Basics

- Suitable use cases
  - Event logging, content management systems, blogs, web analytics, e-commerce applications, …
  - I.e. for **structured documents with similar schema**
- When not to use
  - Set operations involving multiple documents
  - Design of document structure is constantly changing
    - I.e. when the required level of granularity would outbalance the advantages of aggregates

⊕ simples e código fácil, flexibilidade de esquema e melhor performance para as operações SIMPLES, sem JOINS (nem existe)

⊖ não gostam de alterar valores frequentemente

Column Databases – Basics

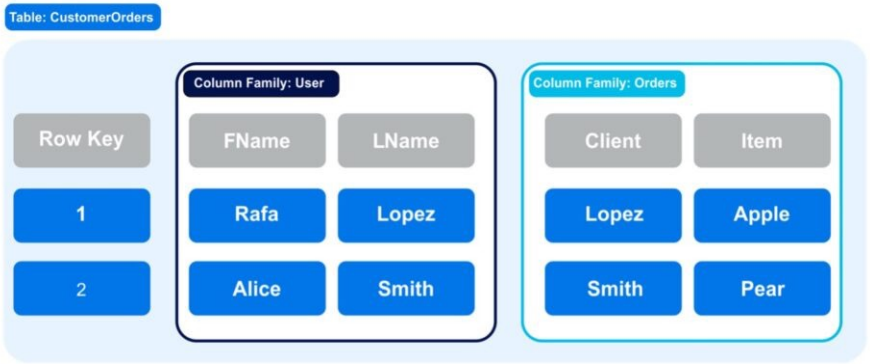
- Data model
  - Column family (table)
    - Table is a collection of similar rows (not necessarily identical)
  - Row
    - Row is a collection of columns - should encompass a group of data that is accessed together
    - Associated with a unique row key
  - Column
    - Column consists of a column name and column value (and possibly other metadata records)
    - Scalar values, but also flat sets, lists or maps may be allowed
- Query patterns
  - Create, update or remove a row within a given column family
  - Select rows according to a row key or simple conditions

Column Databases – Basics

- Suitable use cases
  - Event logging, content management systems, blogs, …
    - I.e. for structured flat data with similar schema
  - Batch processing via mapreduce
- When not to use
  - ACID transactions are required
  - Complex queries: aggregation (SUM, AVG, …), joining, …
  - Early prototypes: i.e. when database design may change
- Examples
  - Apache Cassandra, Apache HBase, Apache Accumulo, Hypertable, Google Bigtable

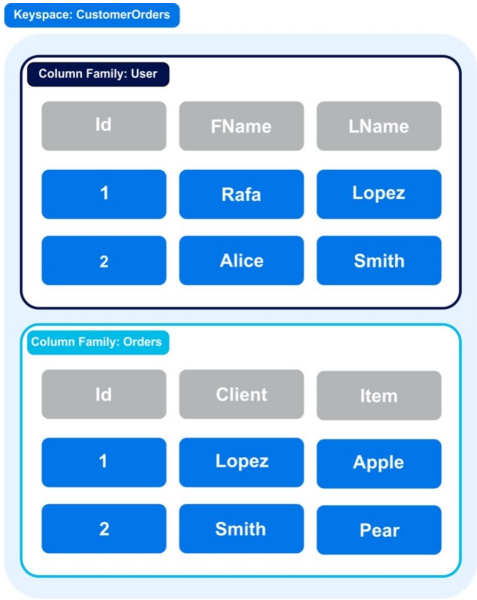
Column Databases – Approaches

- Apache HBase versus Apache Cassandra
  - **HBase**
    - data model is the column-oriented table
    - rows are divided into related columns of data called column families




```
> get 'CustomerOrders ', '1'
COLUMN                          CELL
User: FName timestamp = 1675123184293, value = Rafa
User: LName timestamp = 1675123184293, value = Lopez
Orders: Client timestamp = 1675123388213, value = Lopez
Orders: Client timestamp = 1675123388214, value = Apple
4 row(s) in 0.0260 seconds
```

  - **Cassandra**:
    - data model is best described as a partitioned row store
    - at top-level data model, keyspaces are column-families

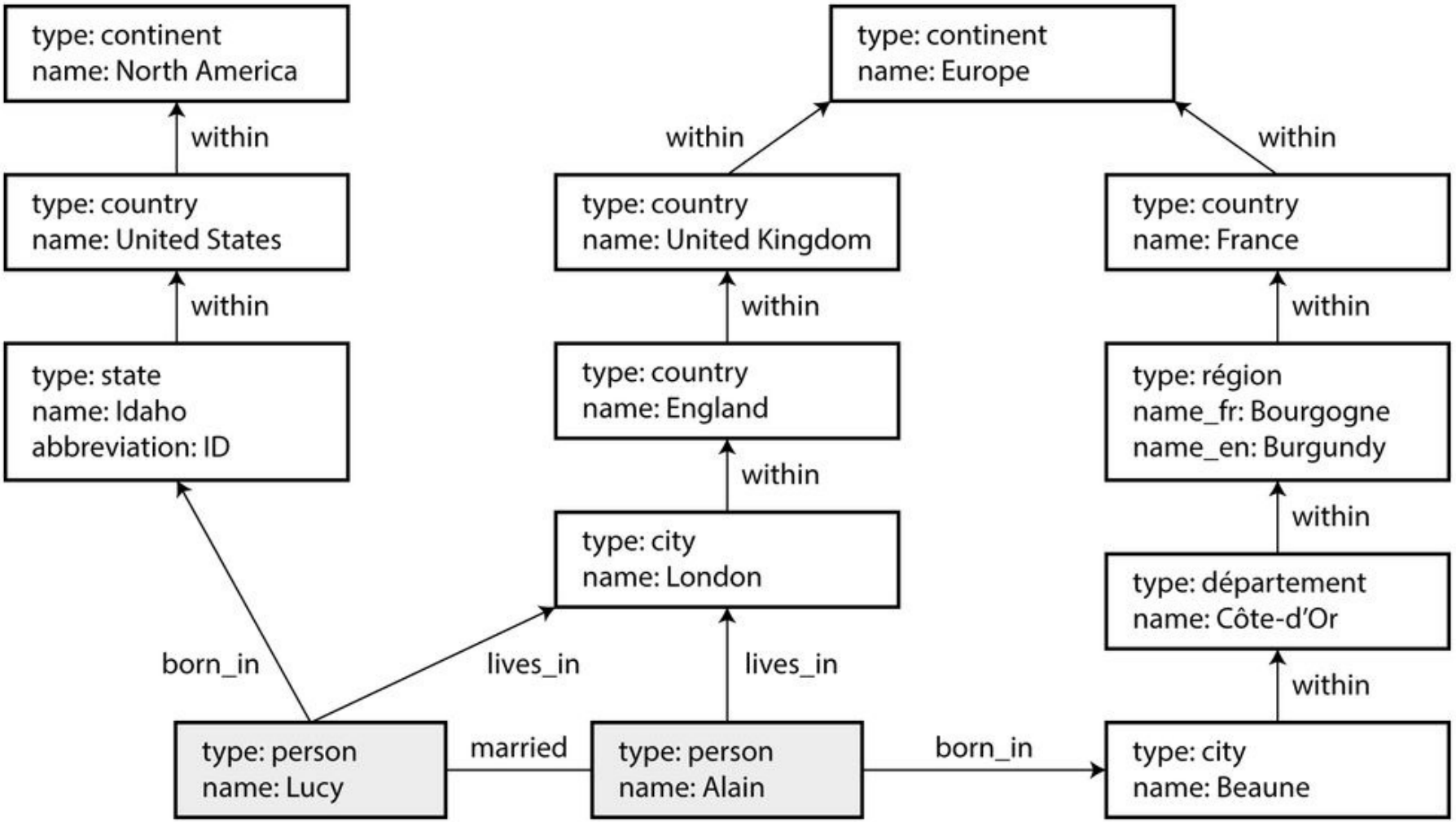



```
cqlsh> select * from User;

id |  fname | lname
----+--------+-------
  1 | Rafa   | Lopez
  2 | Alice  | Smith

(2 rows)
```

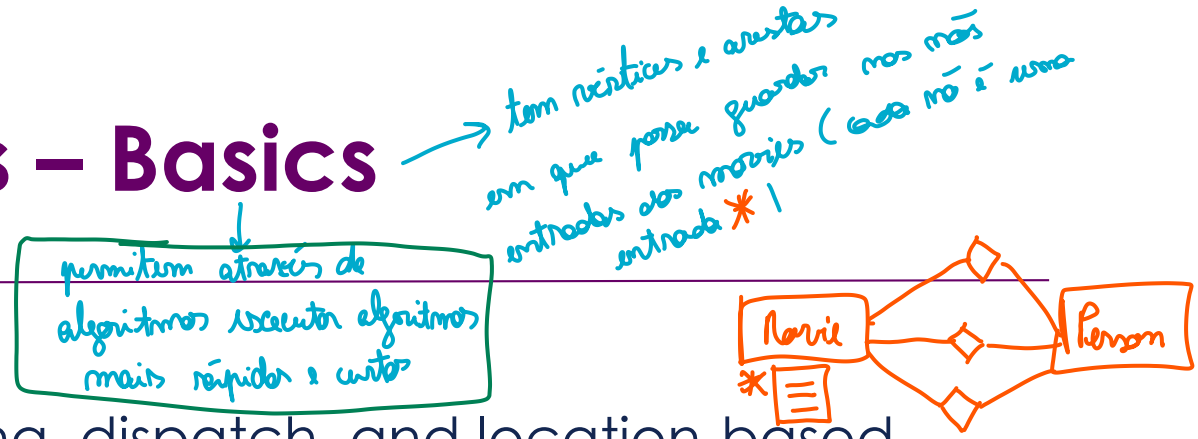
Graph Databases – Basics

- Data Model
  - Focus on modelling graphs' structure and properties
  - Directed / undirected graphs, i.e. collections of …
    - nodes (vertices) for real-world entities, and
    - relationships (edges) between these nodes
  - Both the nodes and relationships can have properties
- Query patterns
  - Create, update or remove a node / relationship in a graph
    - Graph algorithms (shortest paths, spanning trees, …)
    - General graph traversals
    - Sub-graph queries or super-graph queries
    - Similarity based queries (approximate matching)

Graph Databases – Basics



Graph Databases – Basics



- Suitable use cases
  - Social networks, routing, dispatch, and location-based services, recommendation engines, chemical compounds, biological pathways, linguistic trees, …
- When not to use
  - Extensive batch operations are required
    - Multiple nodes / relationships are to be affected
  - Too large graphs to be stored
    - Graph distribution is difficult or impossible at all
- Examples
  - Neo4j, Titan, Apache Giraph, InfiniteGraph, FlockDB
  - *Multi-model*: OrientDB, OpenLink Virtuoso, ArangoDB

Native XML Databases – Basics (JSON-like)

- Data model
  - XML documents
  - **Tree structure** with nested elements, attributes, and text values (beside other less important constructs)
  - Documents are organized into collections
- Query languages
  - **XPath**: XML Path Language (navigation)
  - **XQuery**: XML Query Language (querying)
  - **XSLT**: XSL Transformations (transformation)
- Examples
  - Sedna, Tamino, BaseX, eXist-db
  - *Multi-model*: MarkLogic, OpenLink Virtuoso

Resource Description Framework (linguagem própria)

RDF Databases – Basics

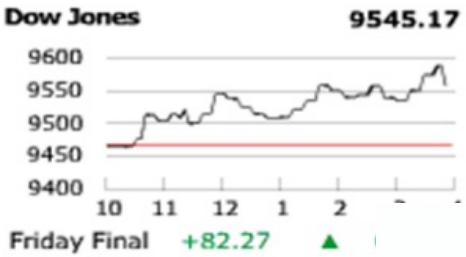
- Data model
  - RDF **triples**
    - Components: **subject**, **predicate**, and **object**
    - Each triple represents a statement about a real-world entity
  - Triples can be viewed as graphs
    - Vertices for subjects and objects
    - Edges directly correspond to individual statements
- Query language
  - **SPARQL**: SPARQL Protocol and RDF Query Language
- Examples
  - Apache Jena, rdf4j (Sesame), Algebraix
  - *Multi-model*: MarkLogic, OpenLink Virtuoso

Time Series Databases – basics

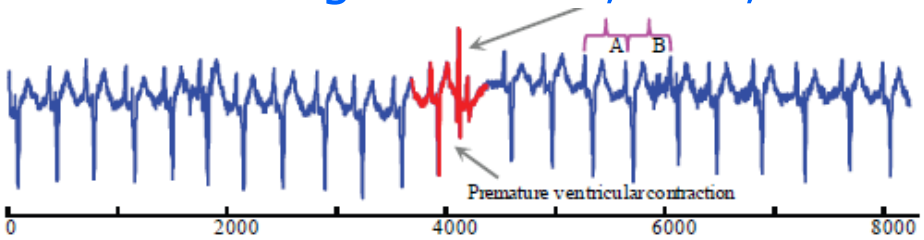
- Data model
  - Stores pairs "Time:Value"
- Query language
  - **Proprietary**: InfluxQL, …
  - **SQL:** some multi-model engines
- Usage
  - store profiles, curves, traces or trends
  - fewer relationships between data entries
  - long sets of data
  - data patterns are "appreciated"
    - compression algorithms to manage the data efficiently
- Examples
  - InfluxDB, Prometheus, Graphite
  - *Multi-model*: Kdb, TimescaleDB

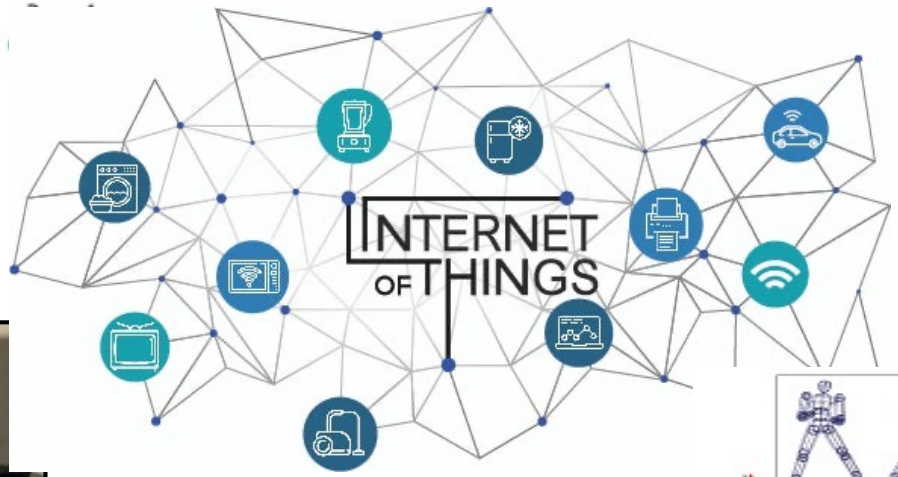
Time Series Examples

Stocks Data

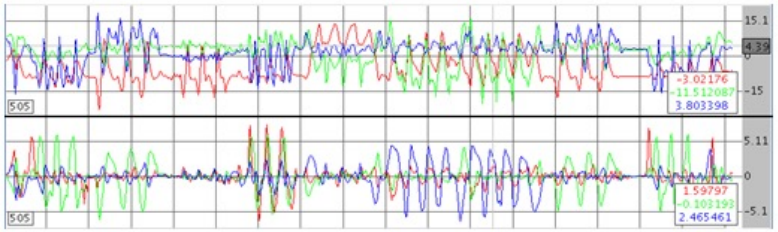


Vital Signals: ECG, EEG,...

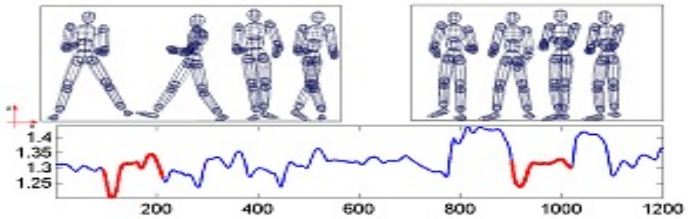


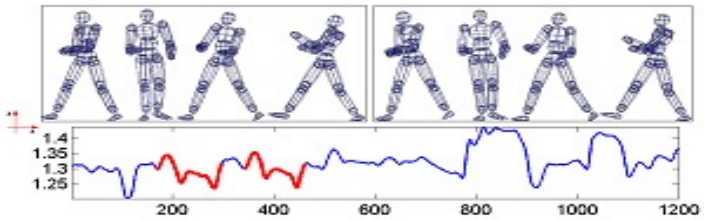


Sensors



Motion Data





NoSQL Databases

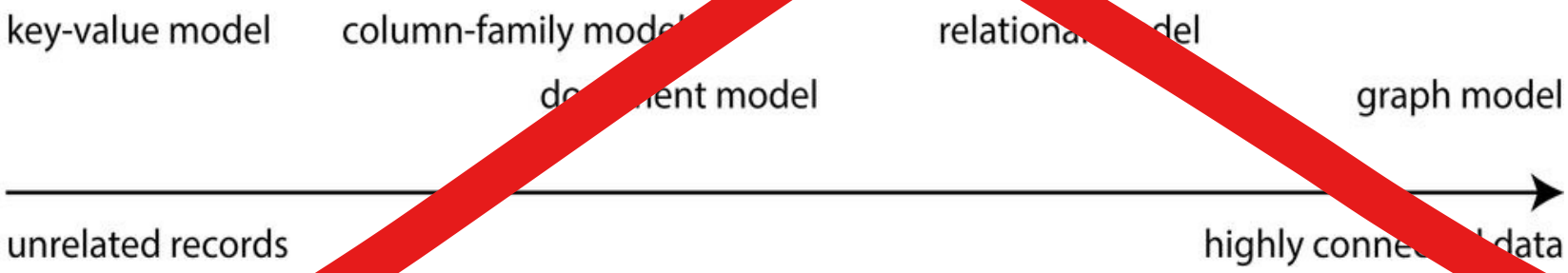
- The end of relational databases?

- **Certainly no!**
  - They are still suitable for most projects (90%)
  - Familiarity, stability, feature set, available support, …

- However, we should also consider different database models and systems
  - **Polyglot persistence** = usage of different data stores in different circumstances

Databases and data connectivity

- Relational model
- NoSQL models
  - Key-value stores
  - Document stores
  - Column stores
  - Graph databases



What next?

- Basic principles
  - Data formats: JSON, YAML, XML, RDF, …
  - Distribution, scaling, sharding, replication, consistency
  - Parallelism, transactions, visualization, processing of graphs
- NoSQL technologies: principles, models, interfaces, languages, …
  - Core databases: Redis, MongoDB, Cassandra, Neo4j
  - MapReduce: Apache Hadoop

Resources

- Martin Kleppmann, ***Designing Data-Intensive Applications***, O'Reilly Media, Inc., 2017.
- Pramod J Sadalage and Martin Fowler, ***NoSQL Distilled*** Addison-Wesley, 2012.
- Eric Redmond, Jim R. Wilson, ***Seven databases in seven weeks***, Pragmatic Bookshelf, 2012.
- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, ***Database systems: the complete book*** *(2nd Ed.)*, Pearson Education, 2009.