

# Big Data and Data Mining

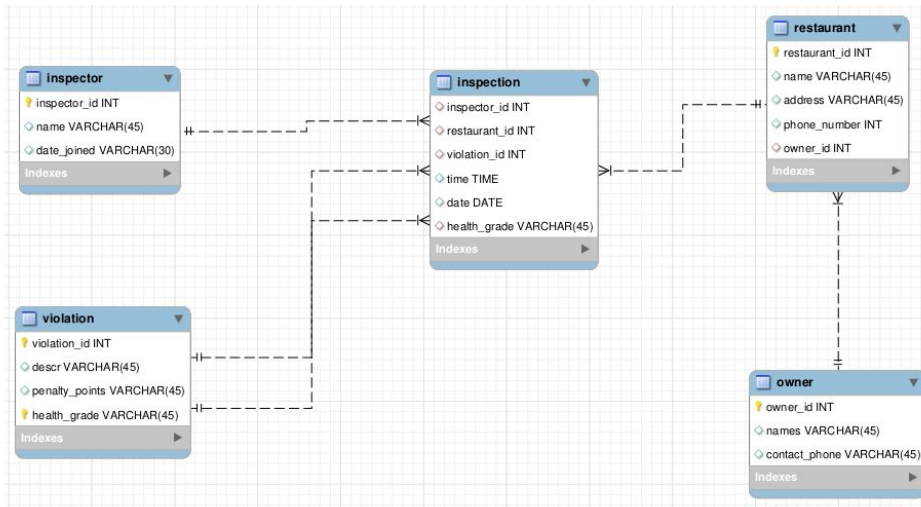
## *NoSQL*

**Flavio Bertini**

[flavio.bertini@unipr.it](mailto:flavio.bertini@unipr.it)

# Relational DBMS: Properties

- **Strong foundation:** Relational Model
- **Highly Structured:** rows, columns, data types
- **Structured Query Language:** standardized
- **ACID properties:** all or nothing
- **Joins:** new views from relationships

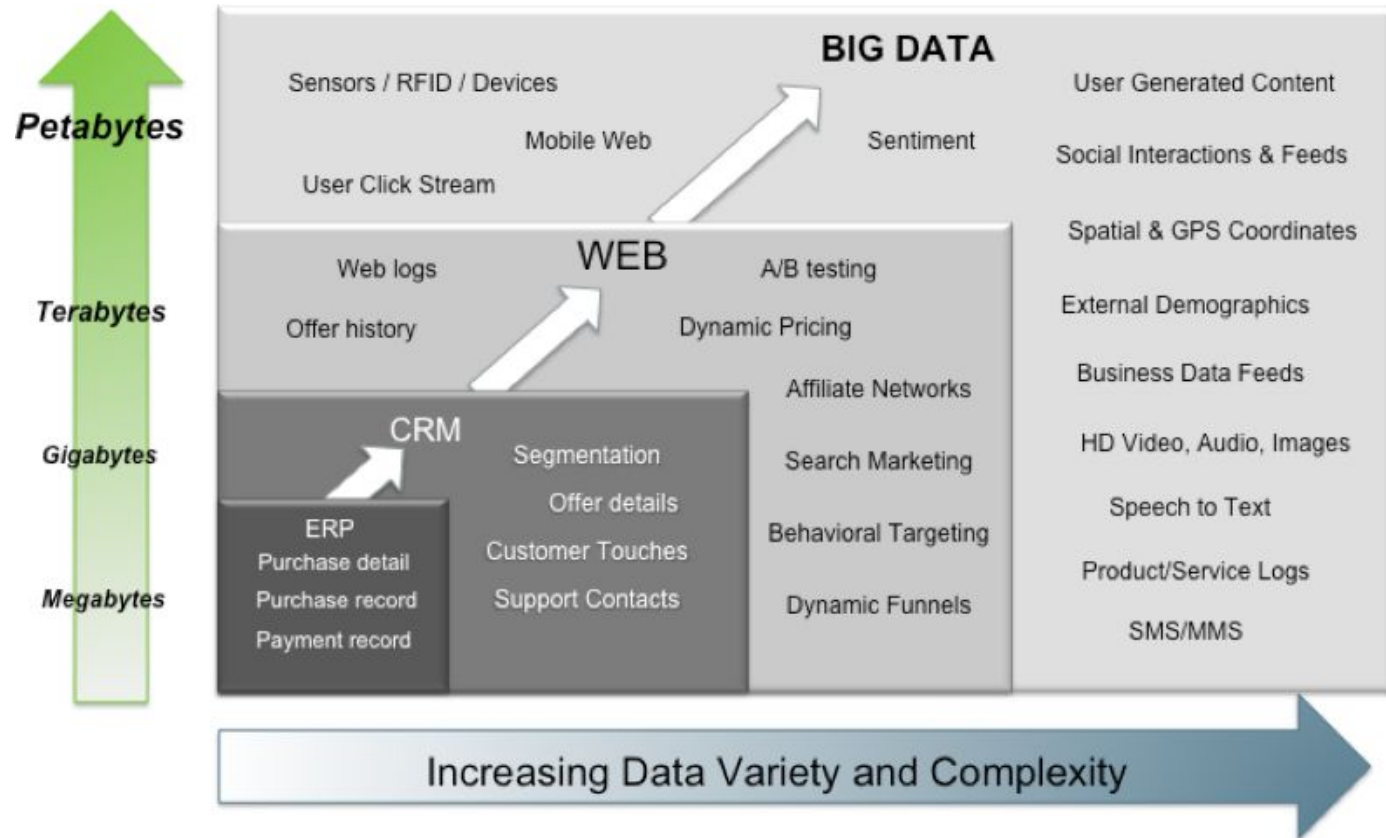


**A** **C** **I** **D**  
ATOMICITY CONSISTENCY ISOLATION DURABILITY

# Changes in scenario: Big Data

## ■ The five (six) V(s) of Big Data:

- **Volume**
- **Velocity**
- **Variety**
- **Variability**
- **Veracity**
- (■ **Value**)



## ■ Needs for:

- Flexible schema for variability and veracity **or no schema at all!**
- Distributed data for volume
- High availability for velocity

# A real example

---

- A vehicle at the **lower end of the autonomous spectrum** will produce about 3 Gbit/s of data, which amounts to about **1.4 terabytes every hour**
- At **higher levels of autonomy**, the total sensor bandwidth will be closer to 40 Gbit/s and approximately **19 terabytes per hour**
- Over a whole year, a vehicle could generate **434 TB for lower level autonomy** or up to **5,894 TB of data for higher-level autonomy**

[Siemens, The Data Deluge: What do we do with the data generated by AVs?](#)

# Relational DBMS: Weakness

- **Joins:** not scalable
- **Transactions:** read & write operations will be slow because of locking resources
- **Fixed definitions (schema):** difficult to work with highly variable data
- **Document integration:** difficult create reports based on structured & unstructured data





# NoSQL: Why, What and When

---

- In 2004 Google and Amazon built their own **non-relational** (do not feature primary / foreign keys, JOINS, or relational calculus of any type) databases designed to scale to **petabyte** of data across **thousands of machines** (Google BigTable and Amazon Dynamo)
- In 2008, Facebook releases its own non-relational database, with a design similar to Google BigTable (Cloud NoSQL database service)
- **Other (very) important reasons:** SQL licenses costs for hundreds of thousands machines!
- **#NoSQL was a twitter hashtag** for a conference in 2009
- The name refers to "non SQL", "non relational" or "not only SQL", but it does not indicate its characteristics
- There is **no strict definition** for NoSQL

# NoSQL DBMSs

- There are currently more than 225 NoSQL databases systems!

[source:

<https://hostingdata.co.uk/nosql-database/>]

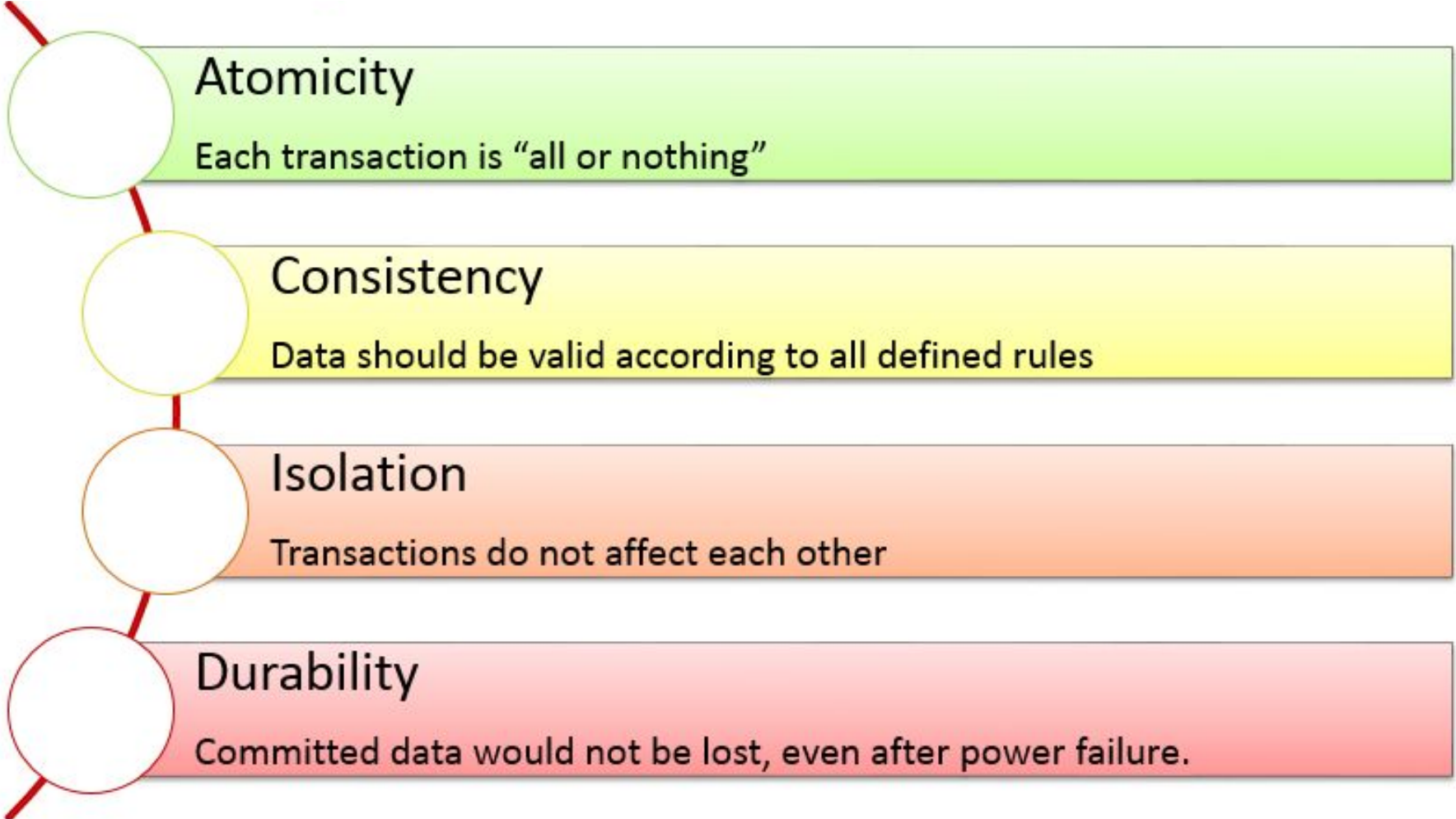


- We will see now the main differences from relational DBMSs (one or both the following):
  - NoSQL systems are **BASE**: they don't follow ACID properties
  - NoSQL systems are **schema-less**: they have a non-relational data model (e.g., key-value, document, graph)

**NoSQL systems are  
BASE**



# Recall ACID



## Atomicity

Each transaction is “all or nothing”

## Consistency

Data should be valid according to all defined rules

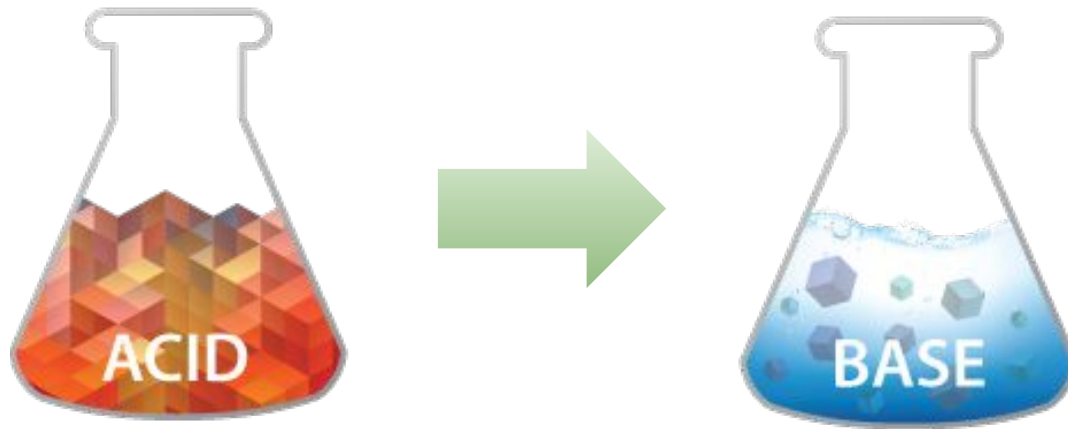
## Isolation

Transactions do not affect each other

## Durability

Committed data would not be lost, even after power failure.

# SQL (ACID properties) vs NoSQL (BASE properties)



- **Basic Availability:** there will be a response to any request (can be failure too)
  - **Soft State:** the state of the system could change over time
  - **Eventual Consistency:** may be inconsistent in short term, consistent in long term
- 
- **It's OK to use stale data**
  - **it's OK to give approximate answers**

# Basically Available

---

- **Basically Available**

- The system does guarantee the availability of the data as regards [CAP Theorem](#) (a.k.a. Brewer's theorem)
- There will be a response to any request but:
  - That response could still be 'failure' to obtain the requested data
  - The data may be in an inconsistent or changing state

- **Soft state**

- The state of the system could change over time
- Even during times without input there may be changes going on due to 'eventual' consistency
- The state of the system is always 'soft'

# Eventual consistency

---

## ■ **Eventual consistency**

- The system will *eventually* become consistent once it stops receiving input
- The data will propagate to everywhere it should sooner or later
- The system will continue to receive input in the meantime
- The system does not check the consistency of every transaction before it moves onto the next one



# ACID vs BASE

---

## ■ ACID

- Strong consistency
- Less availability
- Pessimistic concurrency
- Complex

## ■ BASE

- Availability is the most important thing! Willing to sacrifice other properties for this (like consistency)
- Weaker consistency (Eventual)
- Best effort
- Simple and fast
- Optimistic

## ■ Why can't we have both together?

- A tradeoff exists between consistency, availability, and partition tolerance → **CAP Theorem**

# Another view: CAP trade off

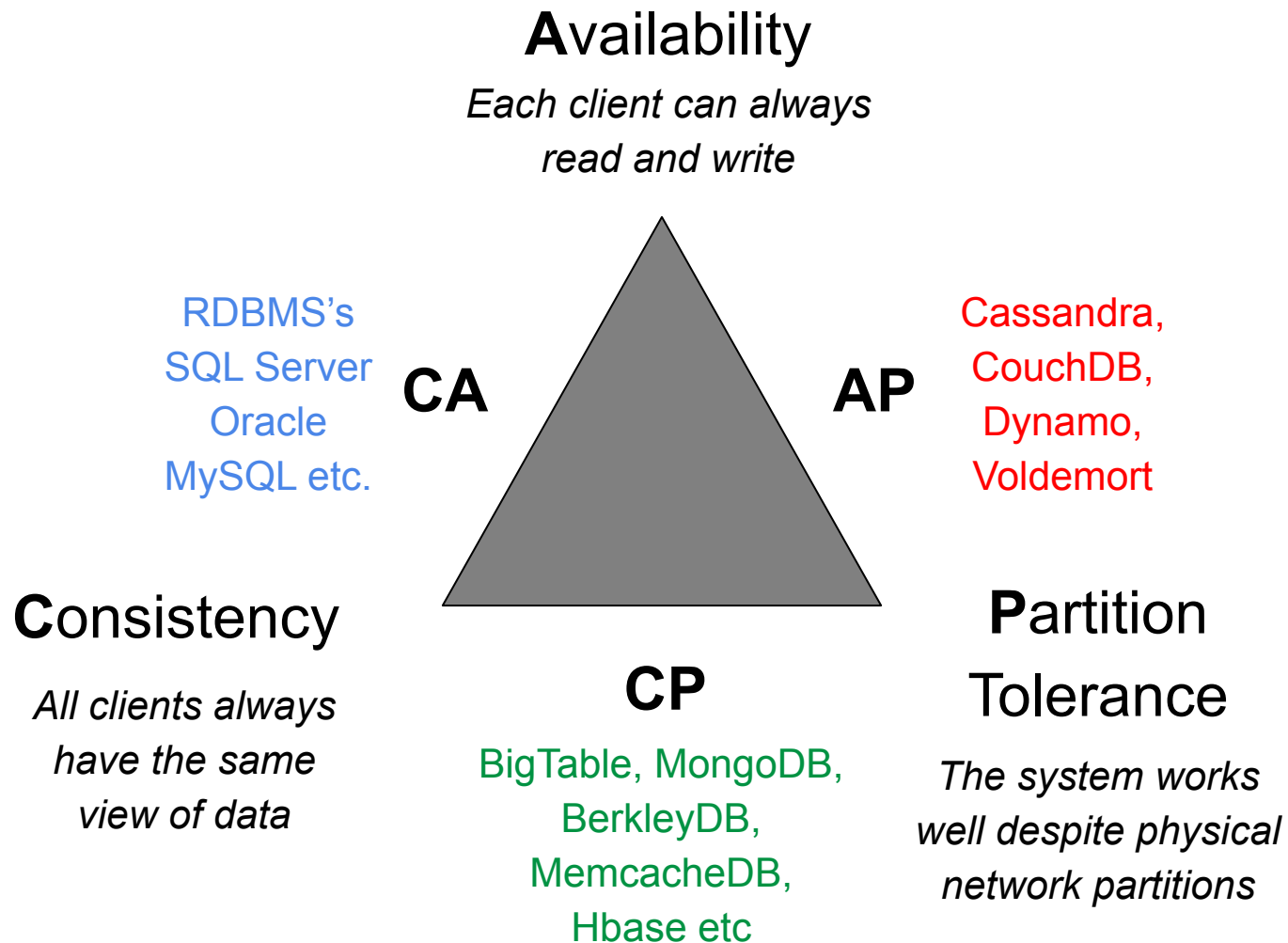
---

- **C**onsistency refers to whether a system operates fully or not. Do all nodes within a cluster see all the data they are supposed to? This is the same idea presented in ACID
- **A**vailability means just as it sounds. Is the given service or system available when requested? Does each request get a response outside of failure or success?
- **P**artition Tolerance represents the fact that a given system continues to operate even under circumstances of data loss or system failure. A single node failure should not cause the entire system to collapse
- In large scale, distributed, non relational systems, they need availability and partition tolerance, so consistency suffers and ACID collapses



# CAP Theorem (or triangle)

Only **two properties out of three** can be satisfied in the same data model! **Pick any two**





**NoSQL systems are  
schema-less**

# Relational vs Schema-less 1/2

- In relational Databases:
  - You can't add a record which does not fit the schema
  - You need to add NULLs to unused items in a row
  - We should consider the data types, i.e. you can't add a string to an integer field
  - You can't add multiple items in a field (you have to create another table: primary-key, foreign key, joins, normalization, ... !!!)

```
create table customers (id int, firstname text, lastname text)  
insert into customers (firstname, middlename, lastname) values (...
```




# Relational vs Schema-less 2/2

- In NoSQL Databases:
  - There is no schema to consider
  - There is no unused cell
  - There is no datatype (implicit)
  - Most of considerations are done at the *application layer*
  - We gather all items in an aggregate (document)

## Relational Model



## Document Model

Collection ("Things")

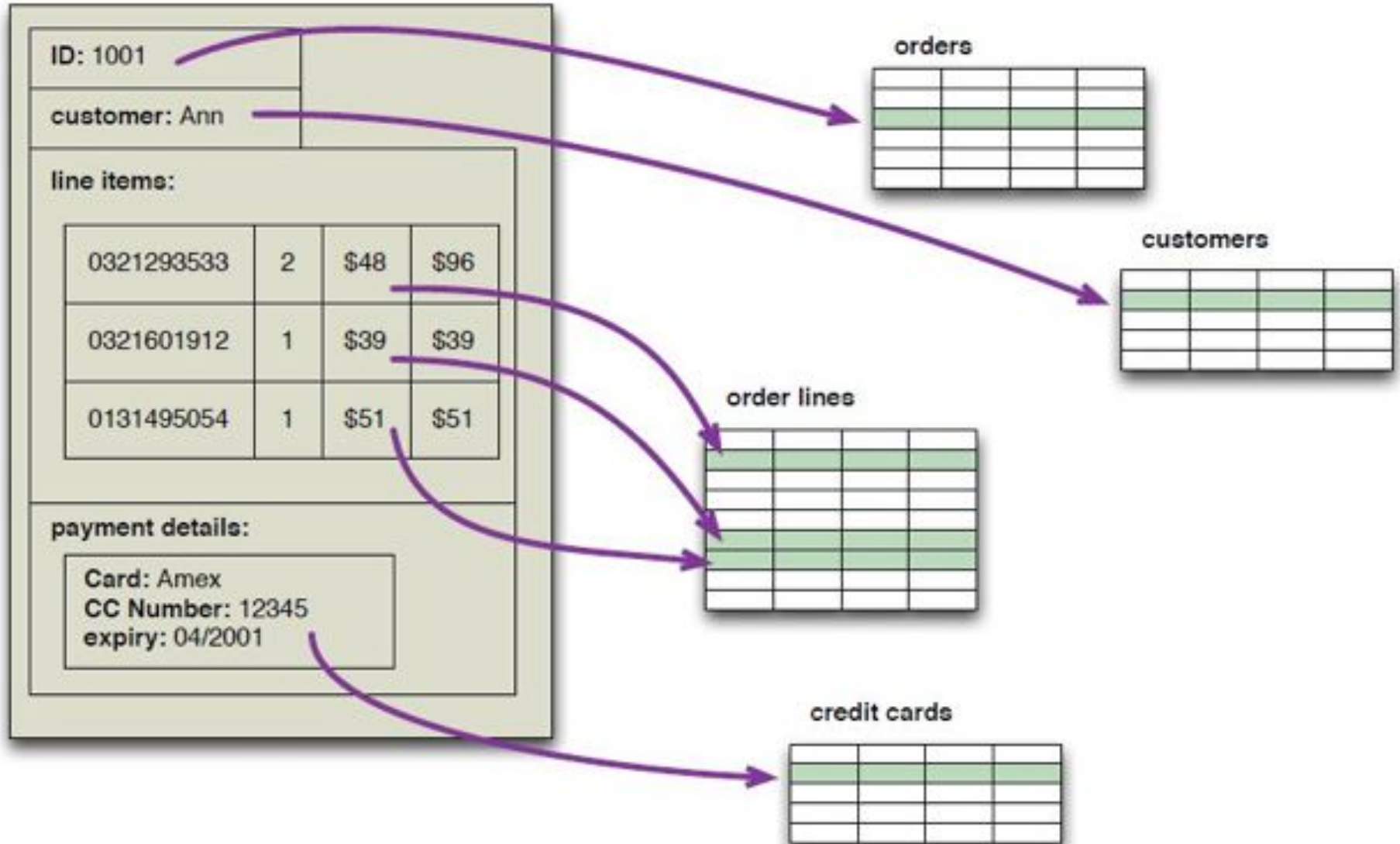


# Aggregation

---

- The term comes from Domain-Driven Design
- An aggregate is a cluster of domain objects that can be treated as a single unit
  - For example, a web document is an aggregate with a title, a body, an image inside the body with its caption etc.
- Aggregates are the basic element of transfer of data storage: you request to load or save whole aggregates
- Transactions should not cross aggregate boundaries
- This mechanism reduces the join operations to a minimal level
  - Related things are already together

# Aggregated vs Relational



# Different types of NoSQL (data models)

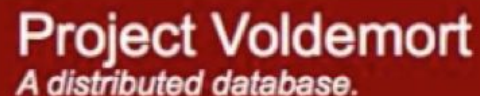
## DOCUMENT



## COLUMN



## KEY-VALUE



## GRAPH

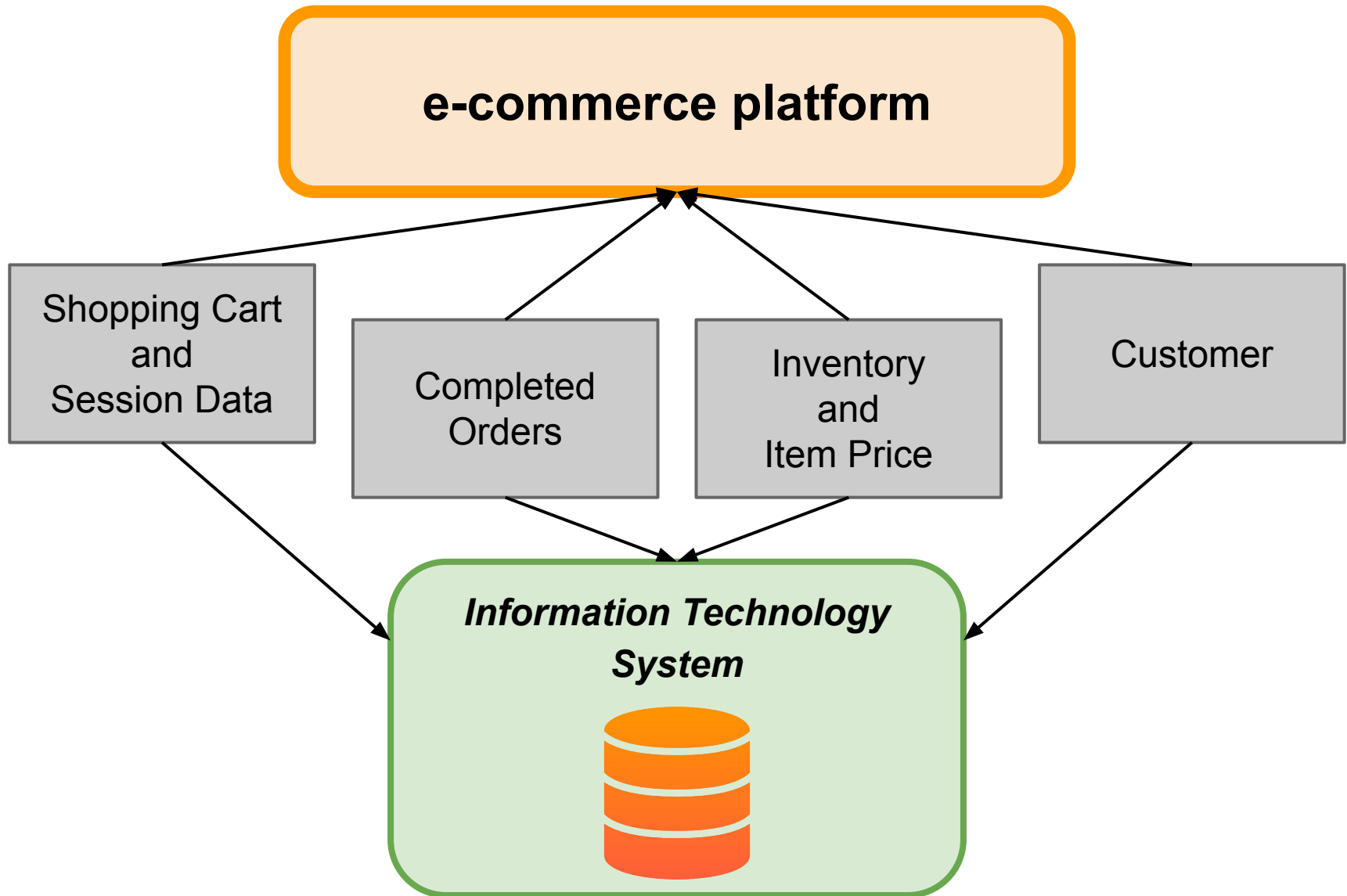


# Why many models?

---

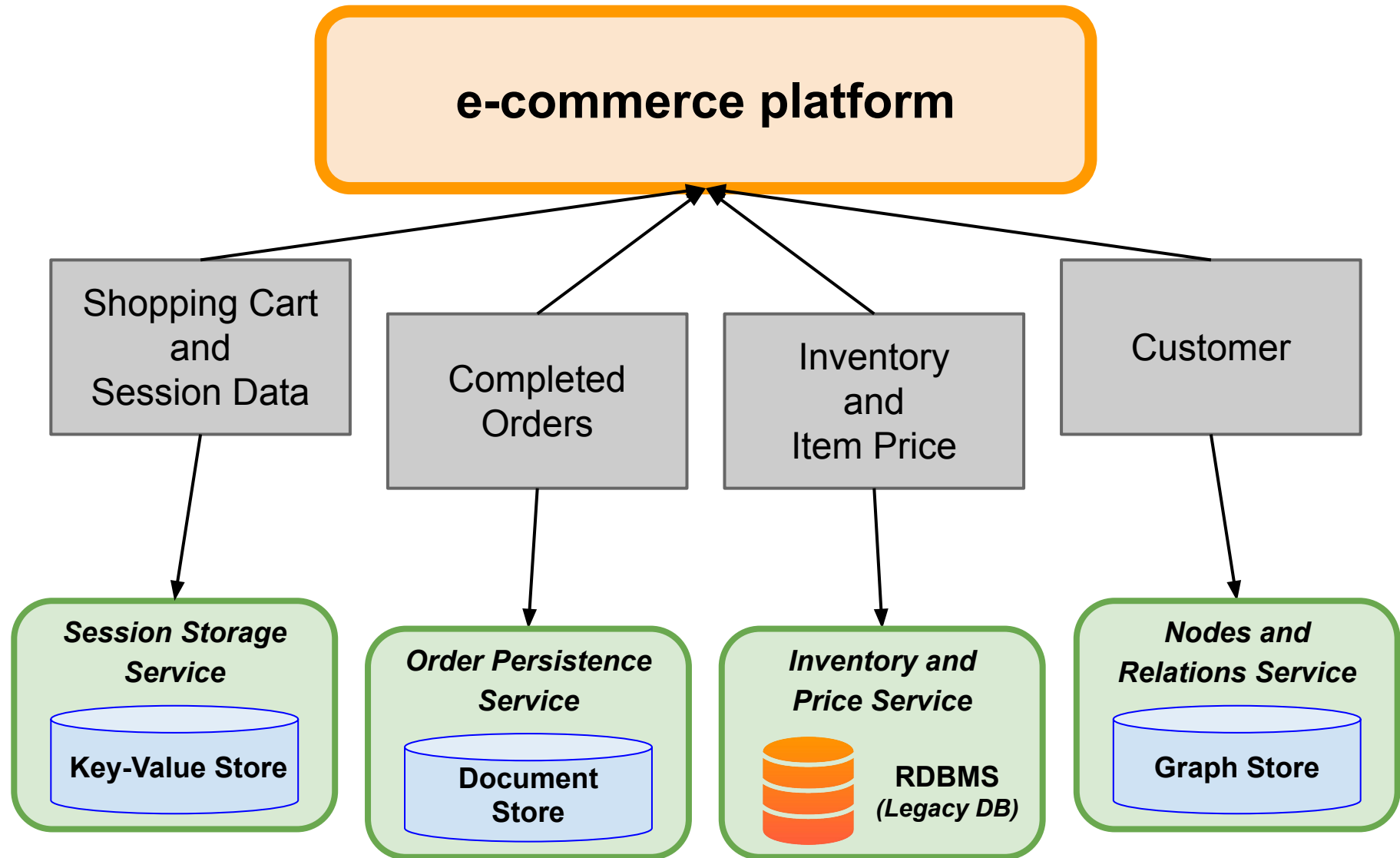
- Different data models are designed to solve different problems
  - Different kind of data
  - Different needs of access (temporary, very fast, historical)
- Using single database engine for all the requirements leads to non-performant solutions
- The solution is **polyglot persistence**: a hybrid approach to data persistence

# Hybrid Approach: an Example (1/2)



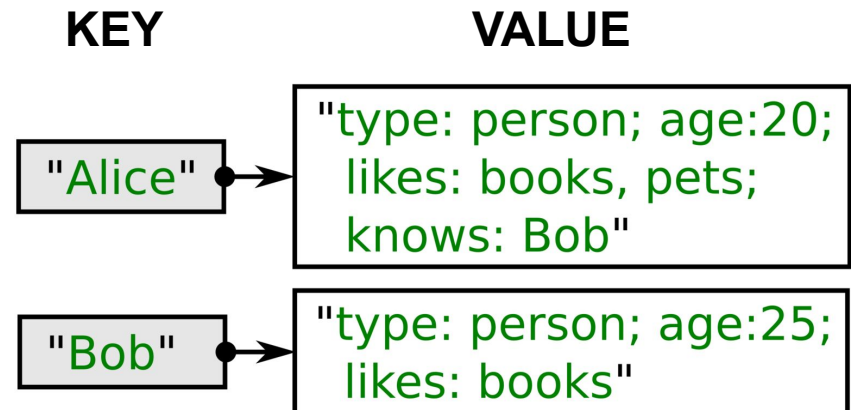


# Hybrid Approach: an Example (2/2)



# Key-value data model

- **Key-value:** A very simple structure. Sets of named keys and their value(s), typically an uninterpreted chunk of data
  - Sometimes that simple value may in fact be a JSON or binary document
- Can be in memory only, or be backed by disk persistence
- Designed to handle massive data loads
- Supports versioning
- Examples:
  - Voldemort (LinkedIn)
  - Amazon SimpleDB
  - Redis
  - Memcache
  - BerkleyDB
  - Oracle NoSQL



# Key-value main idea

- Data model: (key, value) pairs
- Access data (values) by strings called keys
- The main idea is the use of a hash table
- Data has no required format, data may have any format
- Basic Operations:
  - *Insert(key,value)*
  - *Fetch(key)*
  - *Update(key,value)*
  - *Delete(key)*

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year: 2005 Transmission: Auto

# Key-value store in practice

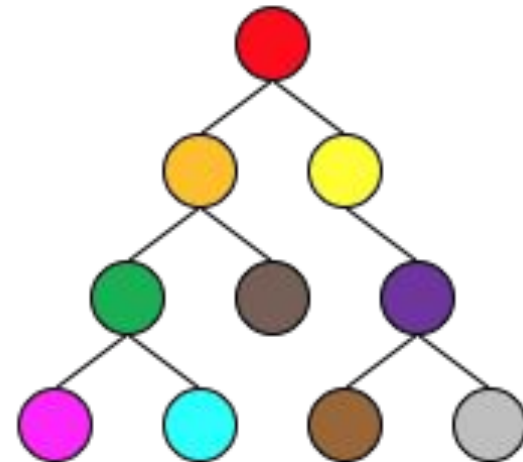
---

- “Value” is stored as a “blob”
  - Without caring or knowing what is inside, or how long it is
  - **Application is responsible for understanding the data**
- Main observation from Amazon (using Dynamo)
  - “There are many services on Amazon’s platform that only need primary-key access to a data store.”
  - e.g., Best seller lists, shopping carts, customer preferences, session management, sales rank, product catalog



# Document data model

- **Document:** XML, JSON, text, or binary blob
- Any treelike structure can be represented as an XML or JSON document, including things such as an order that includes a delivery address, billing details, and a list of products and quantities
- Similar to Key-value, except value is a document!
- Examples:
  - Couchbase
  - MongoDB
  - RavenDB
  - ArangoDB
  - MarkLogic
  - OrientDB
  - Redis
  - RethinkDB



Document is a tree-like hierarchical structure

# Data model: Relational to Document

## Relational

Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

no relation



## MongoDB Document

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, ... },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, ... }
  ]
}
```



# Example: SQL vs MongoDB

RDBMS		MongoDB
Database	↔	Database
Table	↔	Collection
Row	↔	Document
Index	↔	Index
JOIN	↔	Embedded / Reference

# Document Model Benefits

---

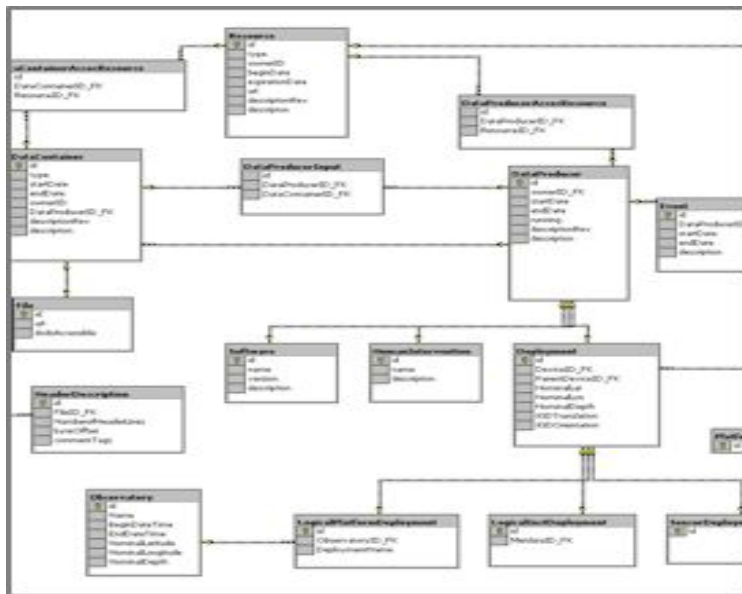
- Rich data model, natural data representation
  - Embed related data in sub-documents & arrays
  - Support indexes and rich queries against any element
- Data aggregated to a single structure (pre-JOINed)
  - Programming becomes simple
  - Performance can be delivered at scale
- Dynamic schema
  - Data models can evolve easily
  - Adapt to changes quickly: agile methodology



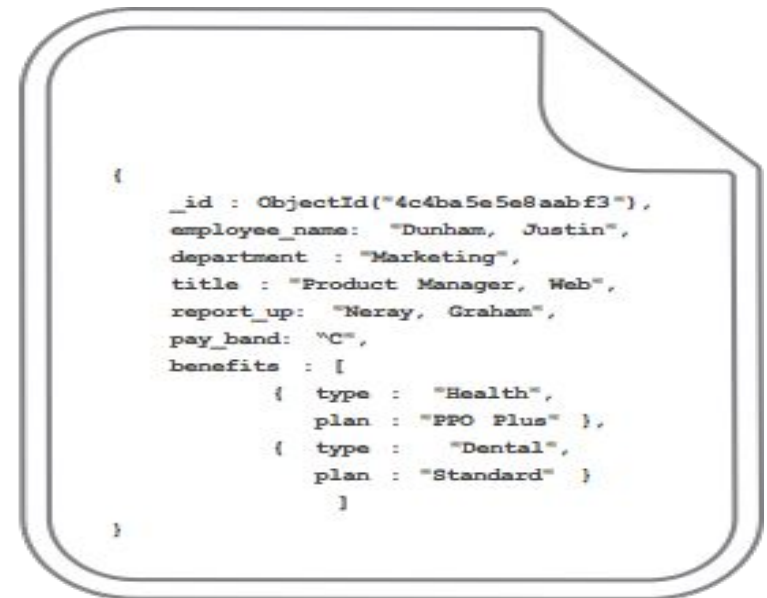
# Join vs Aggregate

- Complex objects are maintained in a single place, not across tables

# Relational



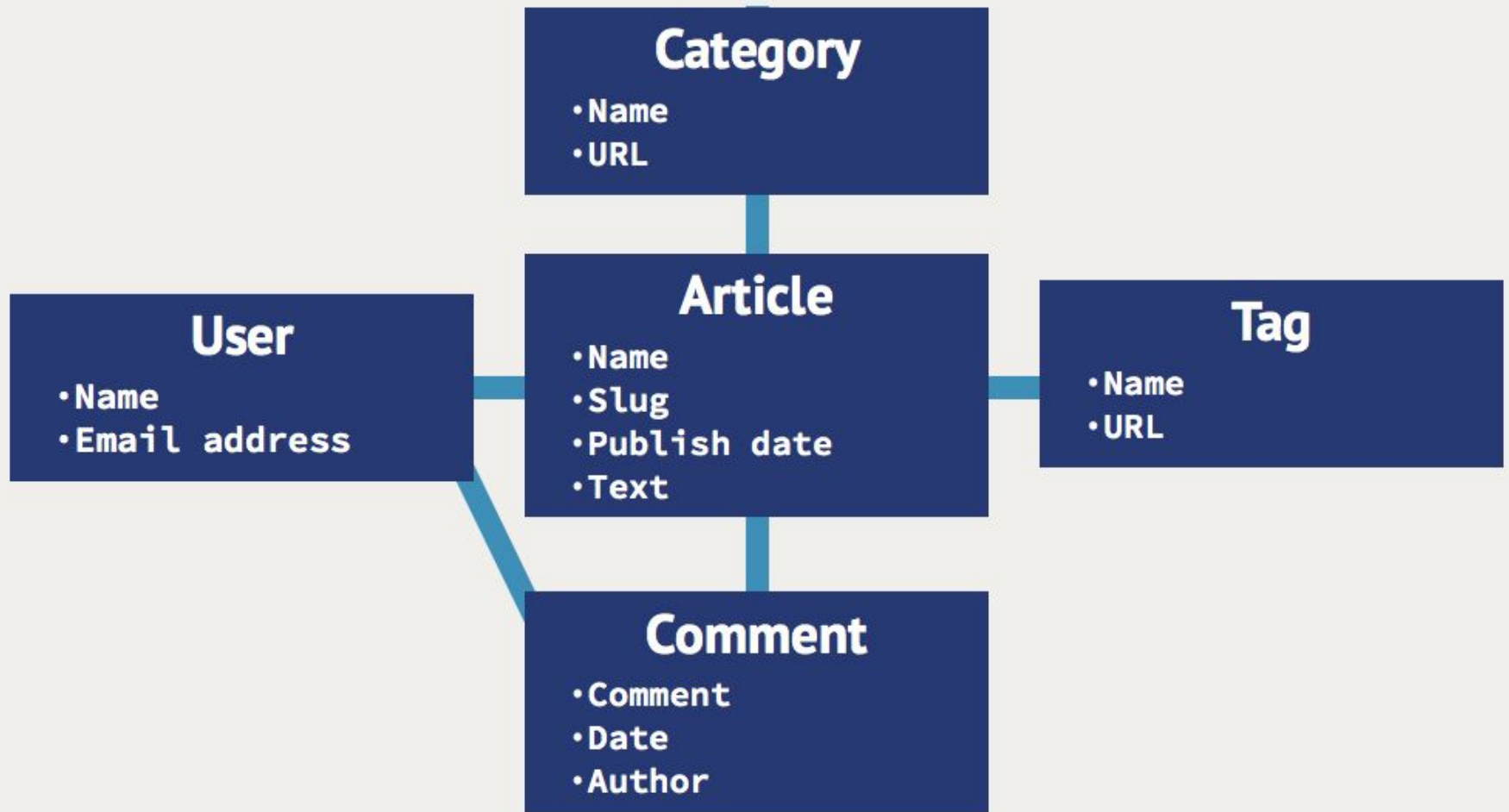
# Document Model



- Schema-less models do not need beforehand design!

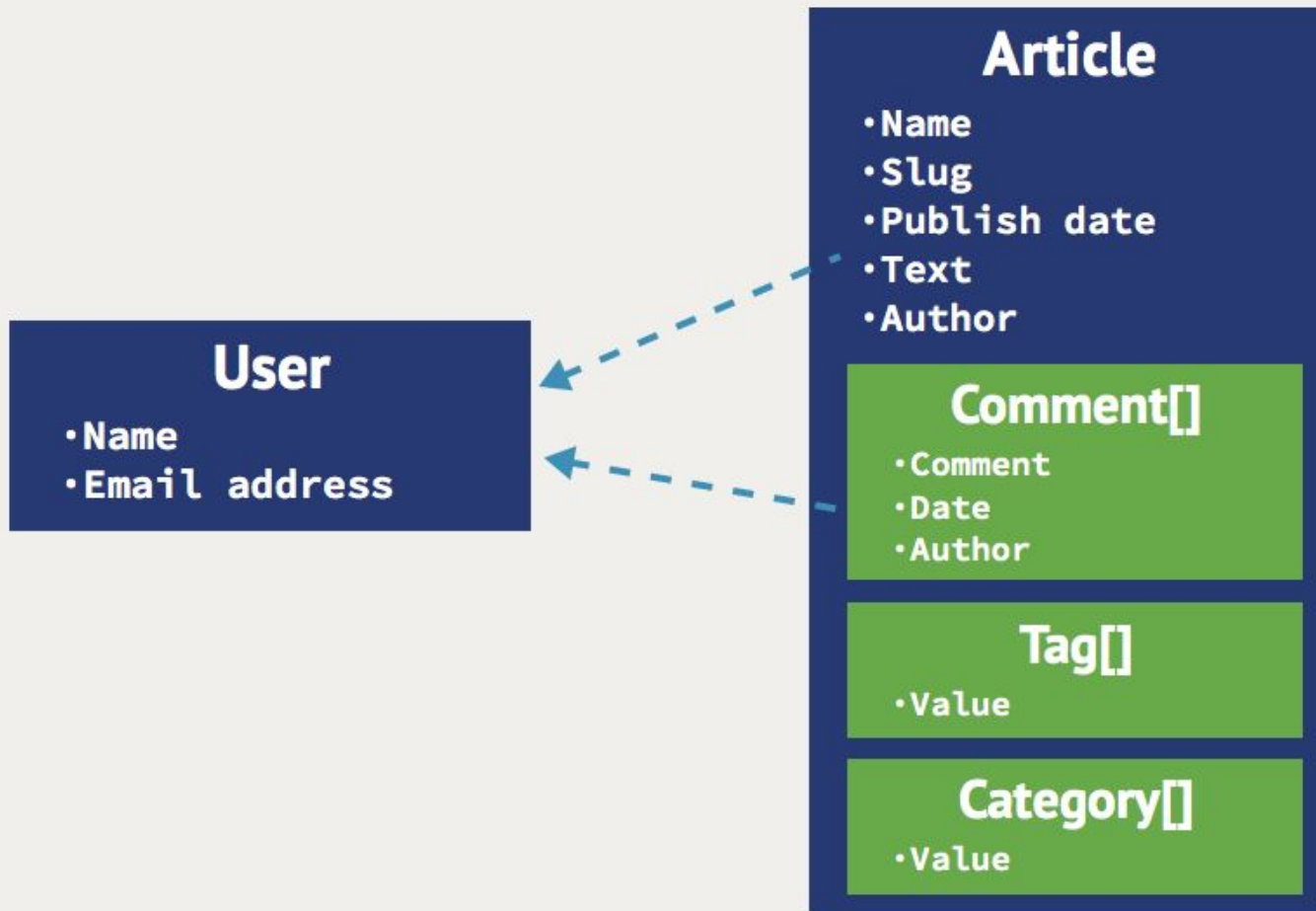


# Example: Blogging Platform (Relational)



- Join between 5 table is needed!

# Example: Blogging Platform (Document model)



- Higher Performance: Data Locality



# Relational vs Document model: Operations

Application	Relational model Action	Document model Action
Create Product Record	<b>INSERT</b> to (n) tables (product description, price, manufacturer, etc.)	<b>insert()</b> to 1 document with sub-documents, arrays
Display Product Record	<b>SELECT</b> and <b>JOIN</b> (n) product tables	<b>find()</b> aggregated document
Add Product Review	<b>INSERT</b> to “review” table, foreign key to product record	<b>insert()</b> to “review” collection, reference to product document

# Columnar data model 1/3

- Columnar data models store tables by columns of data instead of by rows of data

ID	Last	First	Bonus
1	Doe	John	8000
2	Smith	Jane	4000
3	Beck	Sam	1000

## ROW-BASED

```
1, Doe, John, 8000;  
2, Smith, Jane, 4000;  
3, Beck, Sam, 1000;
```

## COLUMN-BASED

```
1, 2, 3;  
Doe, Smith, Beck;  
John, Jane, Sam;  
8000, 4000, 1000;
```

# Columnar data model 2/3

---

- **Columnar:** extension to traditional table structures
- Supports variable sets of columns (column families) and is optimized for column-wide operations (such as count, sum, and mean average)
- Compression: column data is of uniform type
- Multiple values (columns) per key
- Examples:
  - Cassandra
  - Hbase
  - Amazon Redshift
  - HP Vertica
  - Teradata

# Columnar data model 3/3

- The column is lowest/smallest instance of data
- It is a tuple that contains a key, a value and a timestamp

ColumnFamily: Authors		
Key	Value	
"Eric Long"	Columns	
	Name	Value
	"email"	"eric (at) long.com"
	"country"	"United Kingdom"
	"registeredSince"	"01/01/2002"
"John Steward"	Columns	
	Name	Value
	"email"	"john.steward (at) somedomain.com"
	"country"	"Australia"
	"registeredSince"	"01/01/2009"
"Ronald Mathies"	Columns	
	Name	Value
	"email"	"ronald (at) sodeso.nl"
	"country"	"Netherlands, The"
	"registeredSince"	"01/01/2010"



# Columnar data model: Performance

---

- Some statistics about Facebook Search (using Cassandra)

## **MySQL > 50 GB Data**

Writes average: ~300 ms

Reads average: ~350 ms



## **Rewritten with Cassandra > 50 GB Data**

Writes average: 0.12 ms

Reads average: 15 ms

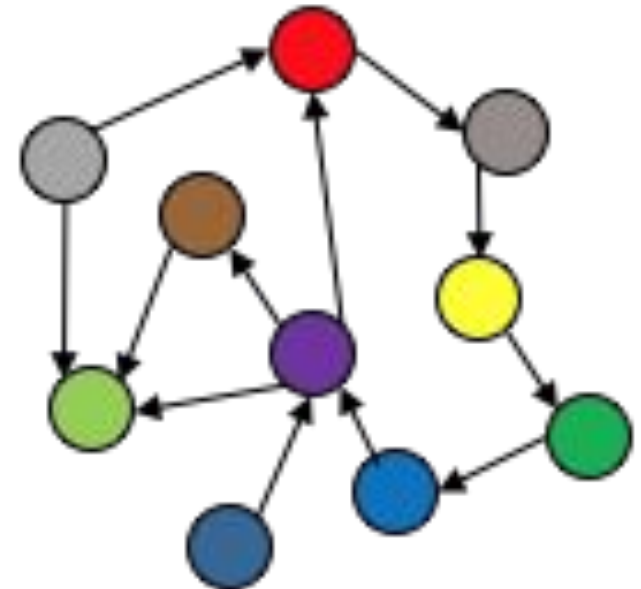


**Cassandra is 2500x faster!**



# Graph data model

- **Graph:** structure of data is graph based
- Uses Property Graph data model (Nodes, Relationships, properties)
- Based on Graph Theory
- Scale vertically, no clustering
- You can use graph algorithms easily
- ACID-compliant transactions
- Examples:
  - Neo4j
  - InfiniteGraph
  - OrientDB
  - Titan GraphDB



# References

---

- **Making Sense of NoSQL (2013) - Dan McCreary and Ann Kelly**
- **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence (2012) - Pramod J. Sadalage and Martin Fowler**
- **Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence (2013) - John Sharp, Douglas McMurtry, Andrew Oakley, Mani Subramanian, Hanzhong Zhang**
- **Distributed Systems: Concepts and Design (5th Edition) (2011) - Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair. Boston: Addison-Wesley. ISBN 0-132-14301-1**