CMPT 506

Introduction to NoSQL

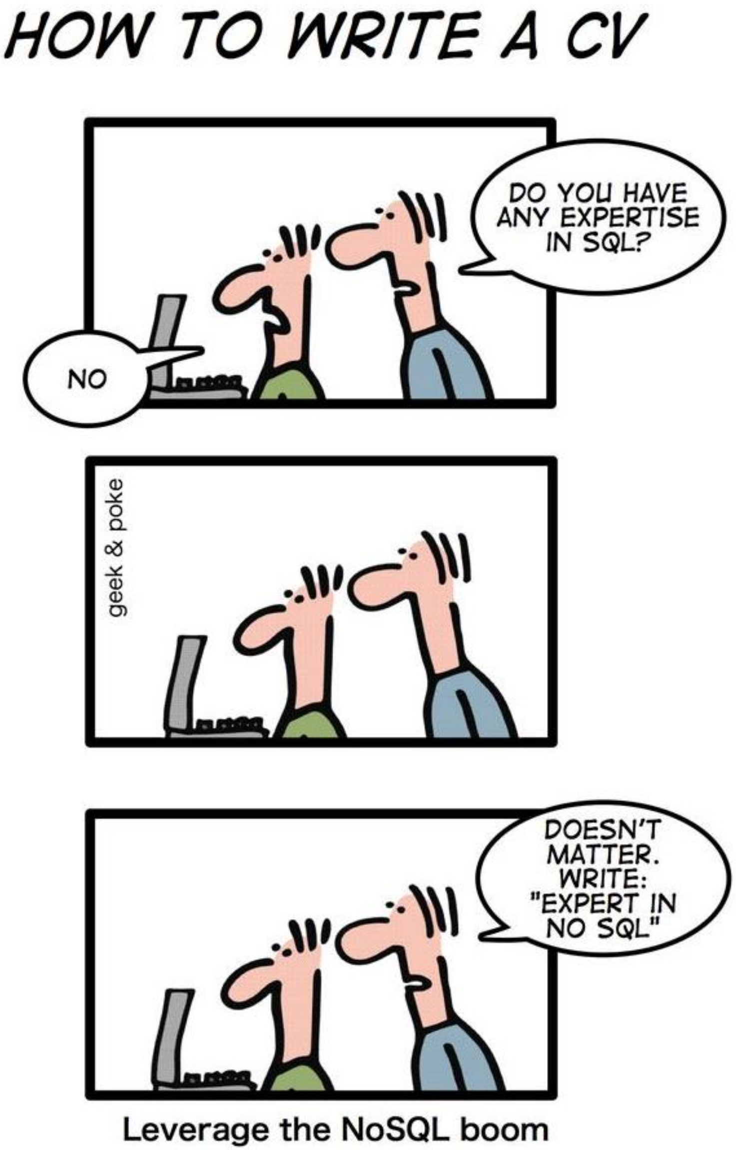
Dr. Abdelkarim Erradi
Computer & Engineering Science Dept.
QU

Outline

1. Why NoSQL?
2. Solutions to address Database Scalability
3. NoSQL Taxonomy
4. CAP Theorem

Acknowledgment

Some slides are based on ‘***NoSQL Distilled***’ book and other online resources



Why NoSQL?

< Back

What is NoSQL?

- NoSQL = alternatives to RDMS to **manage large volumes of data** and achieve **higher availability** using **horizontal scaling** and (often) looser consistency models
- **Horizontal scaling** = scale by adding more commodity servers
- Uses various data models and various query languages
- Some relax ACID (e.g., eventual consistency)

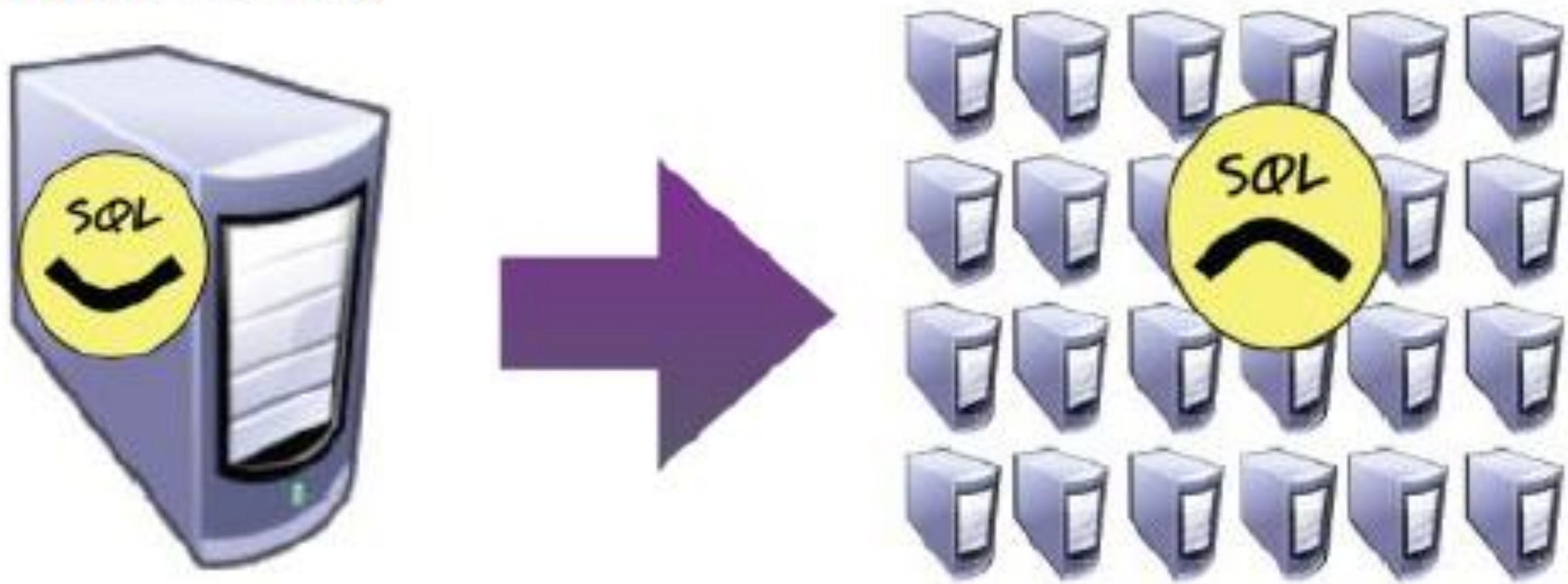
Why NoSQL?

- Two primary reasons for considering NoSQL:
  - Handle ***data access with sizes and performance that demand a cluster***
  - Improve the productivity of application development by using a more convenient data interaction style
    - ***Address the impedance mismatch*** between the relational model and the in-memory data structures

The Scaling Problem of SQL

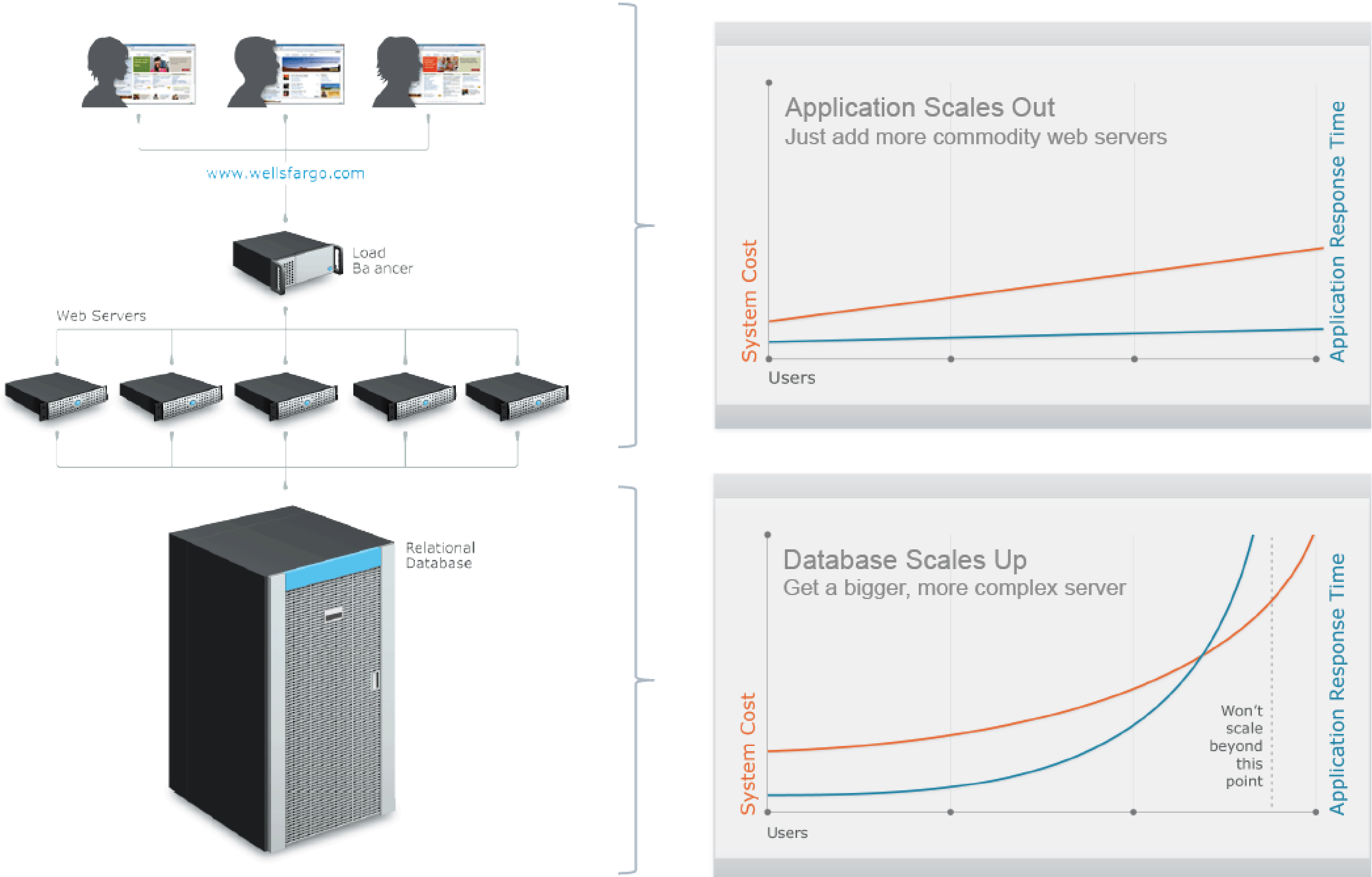
Relational databases are designed to run on a single machine, so to scale, you need buy a bigger Machine (**scale up**)

But it's cheaper and more effective to **scale out** by buying lots of machines.



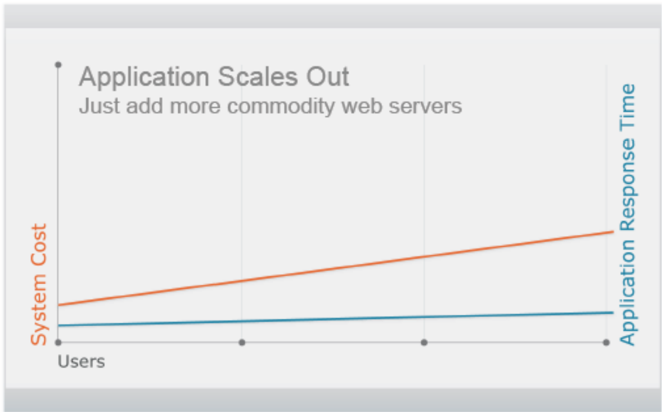
http://martinfowler.com/articles/nosql-intro.pdf

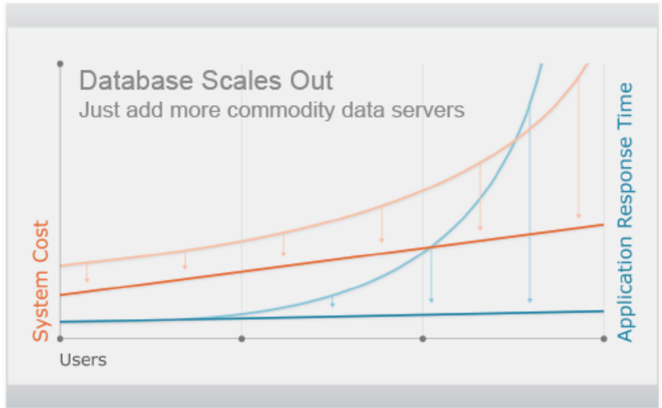
RDMS Key Issue = Database layer does not scale with large numbers of users or large amounts of data



Source: http://www.slideshare.net/Couchbase/nosql-database-technology-why-nosql-why-now

NoSQL database allows the Data layer to scale with linear cost and constant performance





Need to **support large volumes of data by running on clusters**. Relational databases are not designed to run efficiently on clusters.

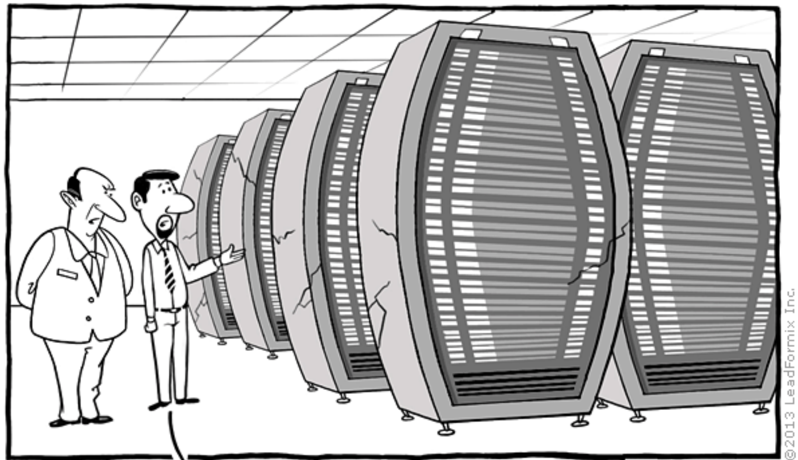
Source: http://www.slideshare.net/Couchbase/nosql-database-technology-why-nosql-why-now

Problem 1: Big Data – the Data Deluge

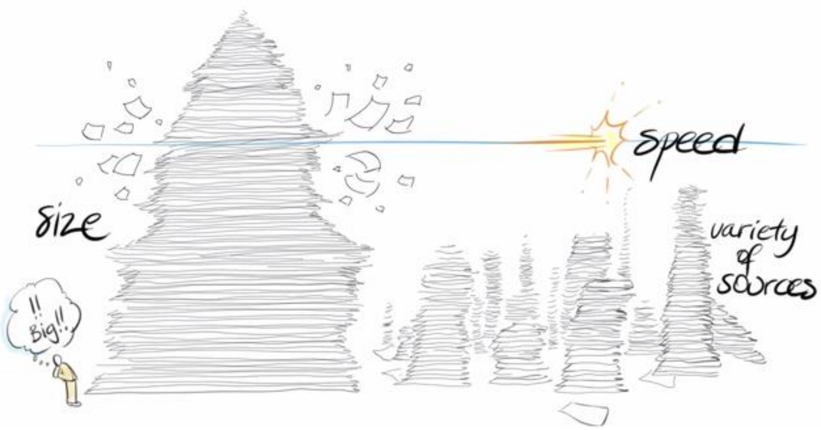
- The world is creating ever more data
- Gartner reported that every day 2.5 exabytes of data are created

(Exabyte = 1 million terabytes = 1billiongigabytes)

- Examples:
  - Facebook has 50 billion photos from its user base, and Facebook users share 30 billion pieces of contents every month
  - Facebook collects over 500 Terabytes of Data in a day
  - Twitter processes over half a billion tweets a day



Our infrastructure can't keep up.
Marketers are flooding us with big data.



World of Big Data

- Big Data are high-**volume**, high-**velocity**, and/or high-**variety** information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization.
  - [Gartner 2012 report]
- Some big data numbers
  - Facebook data in 2010[http://www.facebook.com/note.php?note_id=409881258919]
    - 500 million active users
    - 100 billion hits per day
    - 50 billion photos
    - 2 trillion objects cached, with hundreds of millions of requests per second
    - 130TB of logs every day
  - Facebook garnered more than 1 trillion pageviews per month in June and July (2011), according to data from DoubleClick. [http://mashable.com/2011/08/24/facebook-1-trillion-pageviews/]
    - 23148 views per minute
- Planet scale computing

A Useful Career Advice

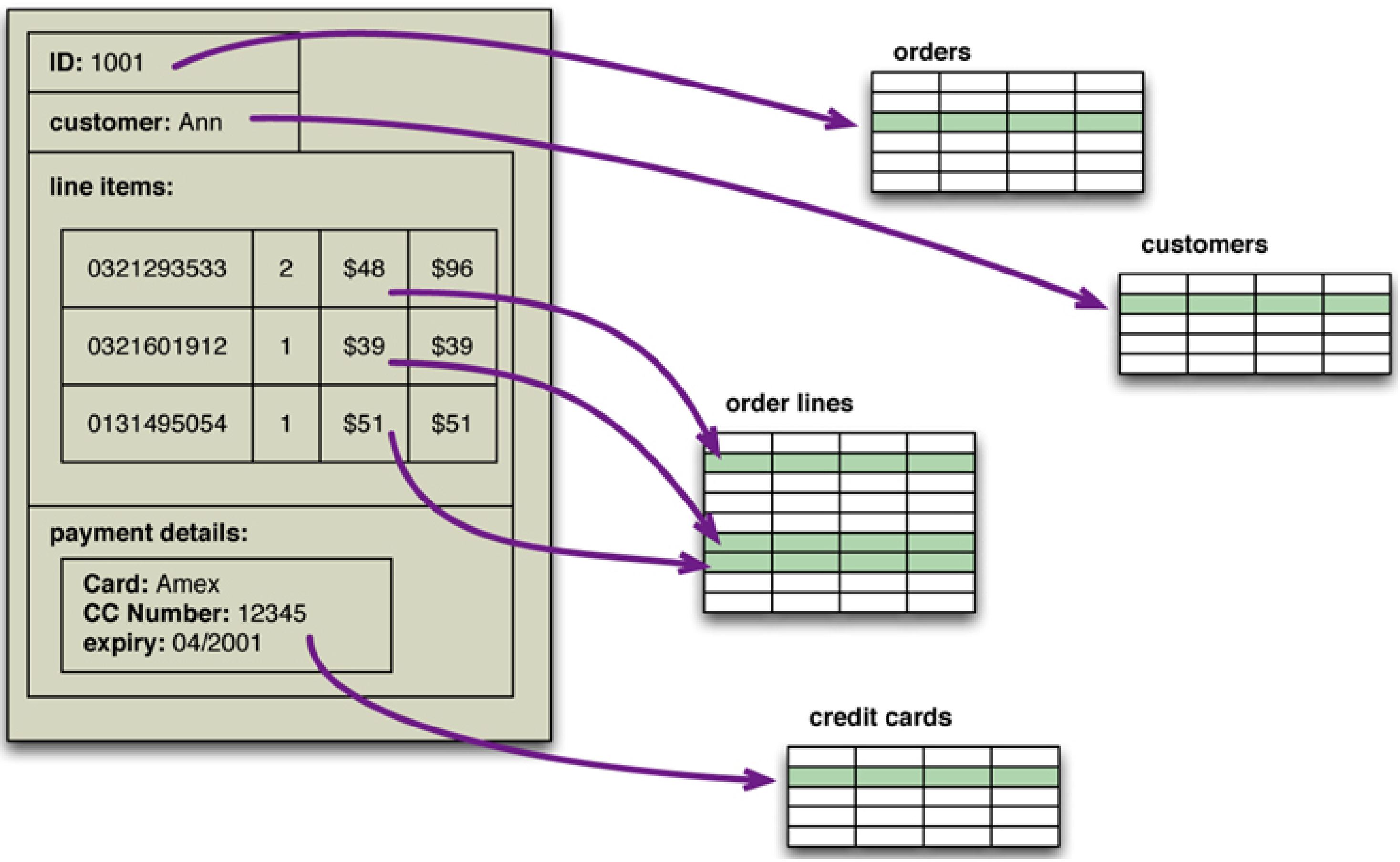
"If you are looking for a career where your service will be in high demand, you should find something where you provide a **scarce**, **complementary service** to something that is getting ubiquitous and cheap.

So what is getting ubiquitous and cheap? **Data**. And what is complementary to data? **Analysis**."

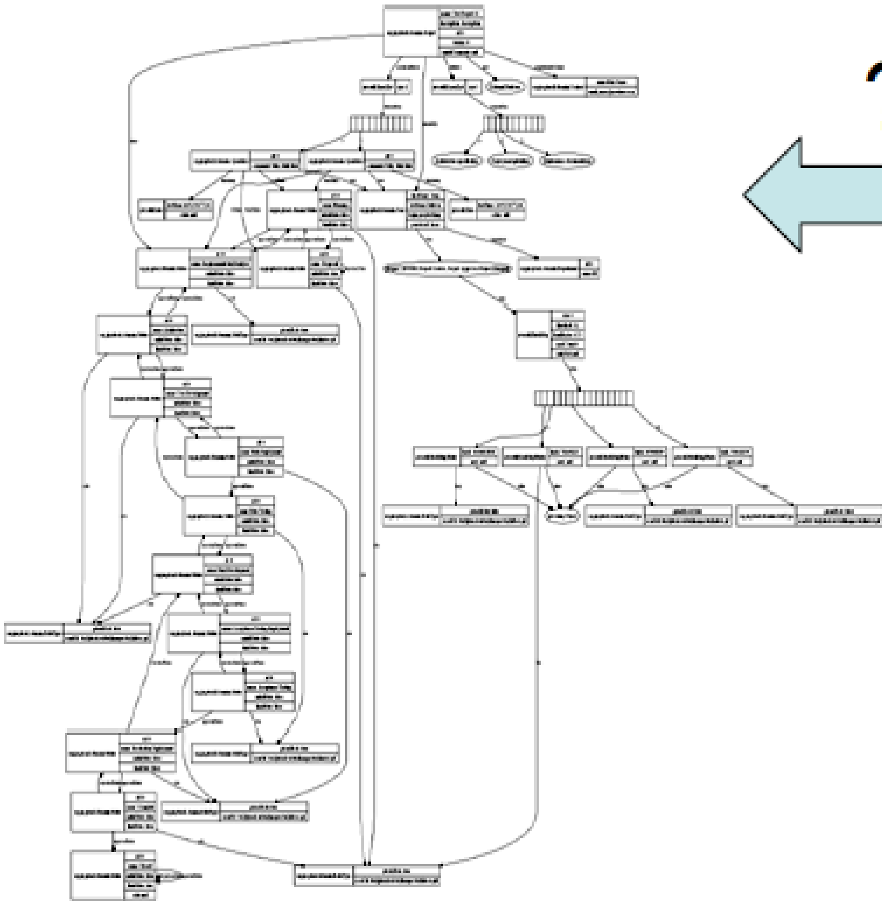
--- Prof. Hal Varian, UC Berkeley

Chief Economist at Google

Problem 2: Impedance mismatch between the relational model and the in-memory data structures



Complex object graphs need to be Mapped to the Relational Model



?!

ID	COL1	COL2	COL...

Poor performance

- Many inserts
- Many joins

Problem 3: Semi-structured data

Customer table

Id	Name	Street	...	Other_Attributes
1	Acme Inc	180 Main		XML/JSON/Blob
2	Failed Bank	1 Wall Street		
...	...	...		

Can't be queried easily

The Fixed Schema Problem of SQL

- In a relational database
  - Table structure are predefined
  - Tables are related with relationships are predefined as well
- Schema evolution in RDBMS is hard and has large impact on queries and applications
- Example
  - MediaWiki had been through 171 schema versions between April 2003 and November 2007.
    - MySQL backend
    - ~ 34 tables, ~242 columns, ~700GB in wikipedia (note: 2008 data)
  - Schema change has big impact on queries
    - Large number of queries could fail due to schema change.

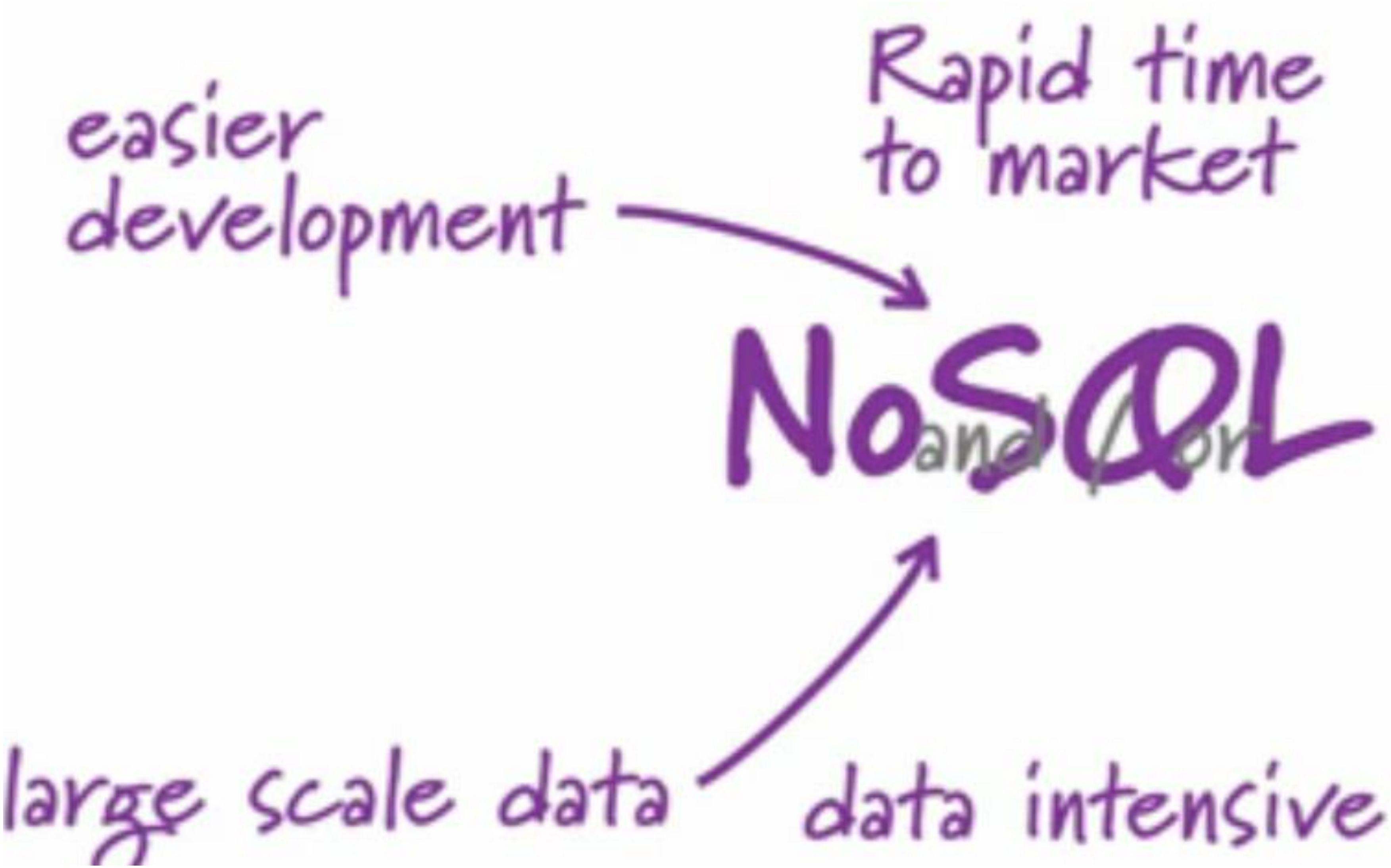
http://yellowstone.cs.ucla.edu/schema-evolution/documents/curino-schema-evolution.pdf

The common characteristics of NoSQL databases

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web-scale data
- Schemaless / flexible data model

=> The most important result of the rise of NoSQL is Polyglot Persistence.

Summary - Why NoSQL



Solutions to Address Database Scalability

< Back

What is scalability?

A service is said to be scalable if when we increase the resources in a system, it results in increased performance in a manner proportional to resources added.

Werner Vogels *CTO - Amazon.com*

Scale up

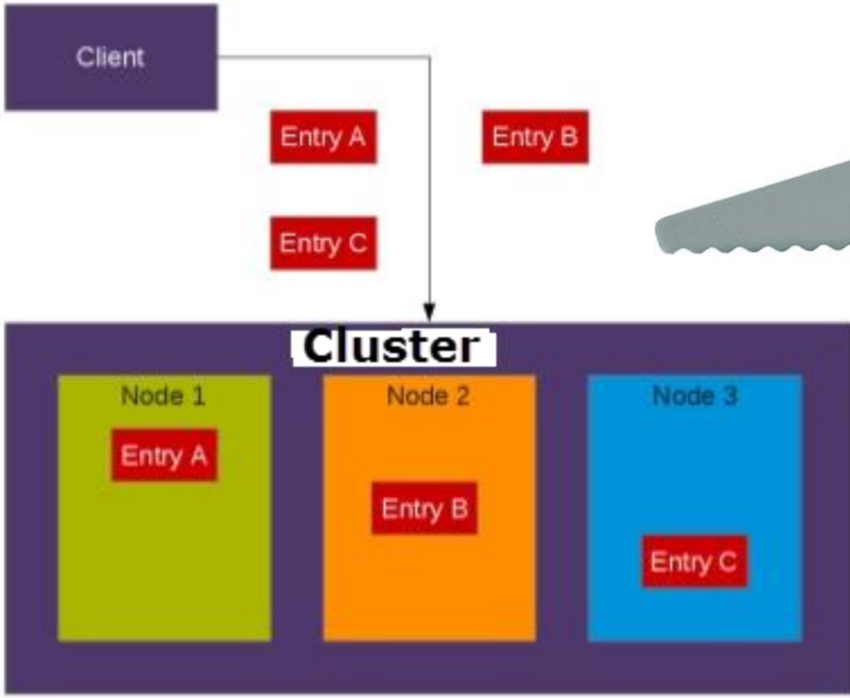
Scale out

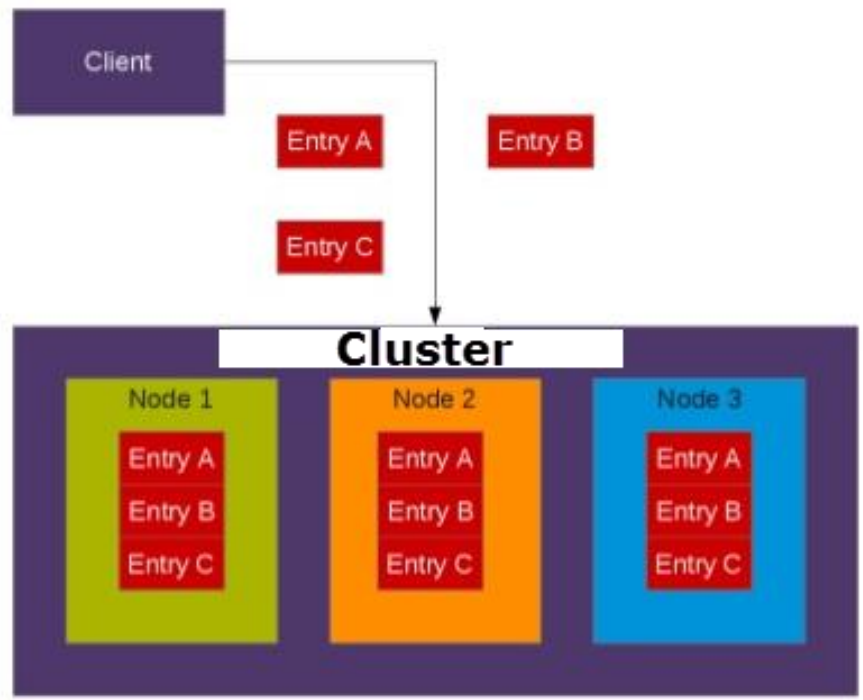
Solutions to address scalability

Two styles of **distributing data** to enhance scalability:

- **Sharding** distributes different data across multiple servers
  - each server acts as the single source for a subset of data.
- **Replication** copies data across multiple servers
  - each bit of data can be found in multiple places
  - Replication → good for performance/reliability
    - Key challenge → keeping replicas up-to-date

Sharding vs. Replication

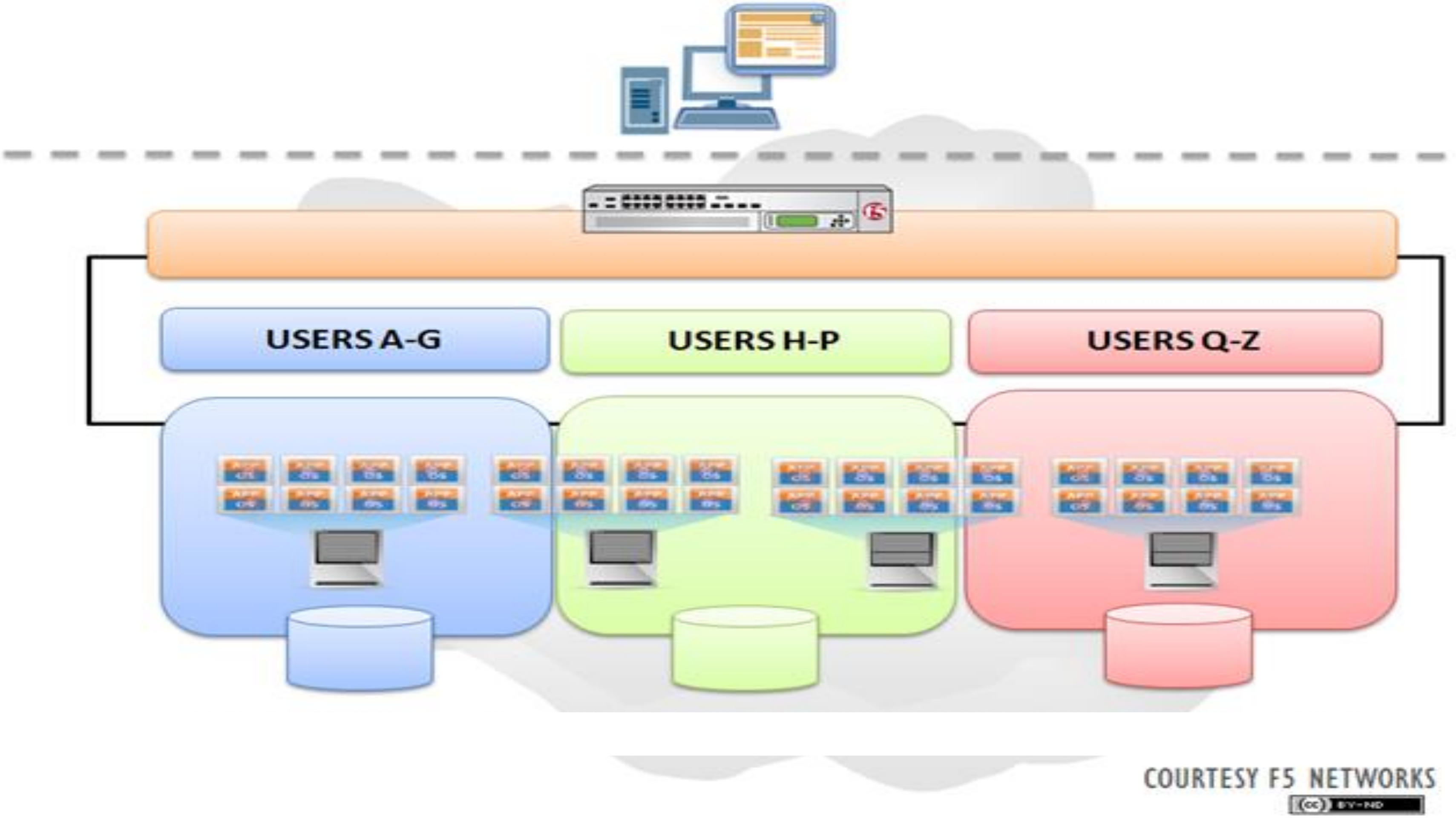




What is Sharding?

- **Problem:** one database can't handle all the data
  - Too big, poor performance, needs geo distribution, …
- **Solution:** split data across multiple databases
  - One **Logical Database**, multiple **Physical Databases**
  - *Scales well for both reads and writes*
- Each Physical Database Node is a **Shard**

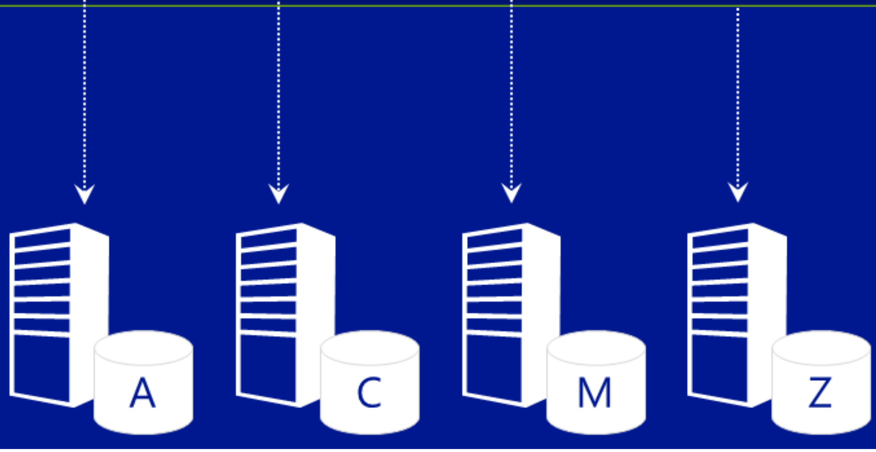
Database Sharding



All shards have same schema

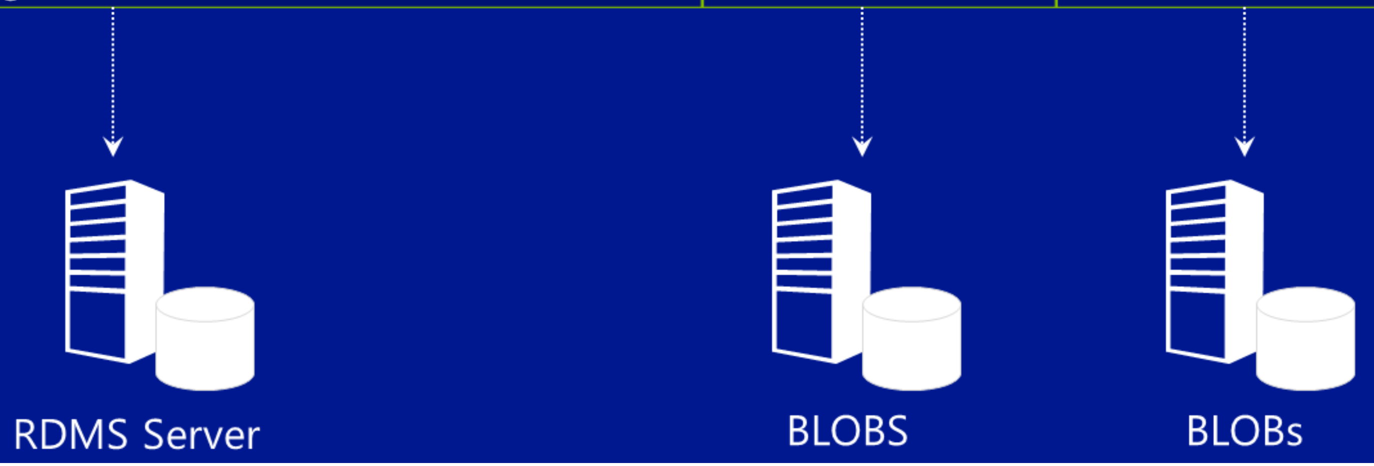
Horizontal Partitioning

First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB



Vertical Partitioning

First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB



Hybrid Portioning

First Name	Last Name	Email	Thumbnail	Photo
David	Alexander	davida@contoso.com	3kb	3MB
Jarred	Carlson	jaredc@contosco.com	3kb	3MB
Sue	Charles	suec@contosco.com	3kb	3MB
Simon	Mitchel	simonm@contoso.com	3kb	3MB
Richard	Zeng	richard@contosco.com	3kb	3MB

Sharding Challenges

- Not transparent, application needs to be partition-aware
  - Sharding is largely managed outside RDBMS
  - Recent version of RDBMS may provide limited support for sharding

Joins become too expensive

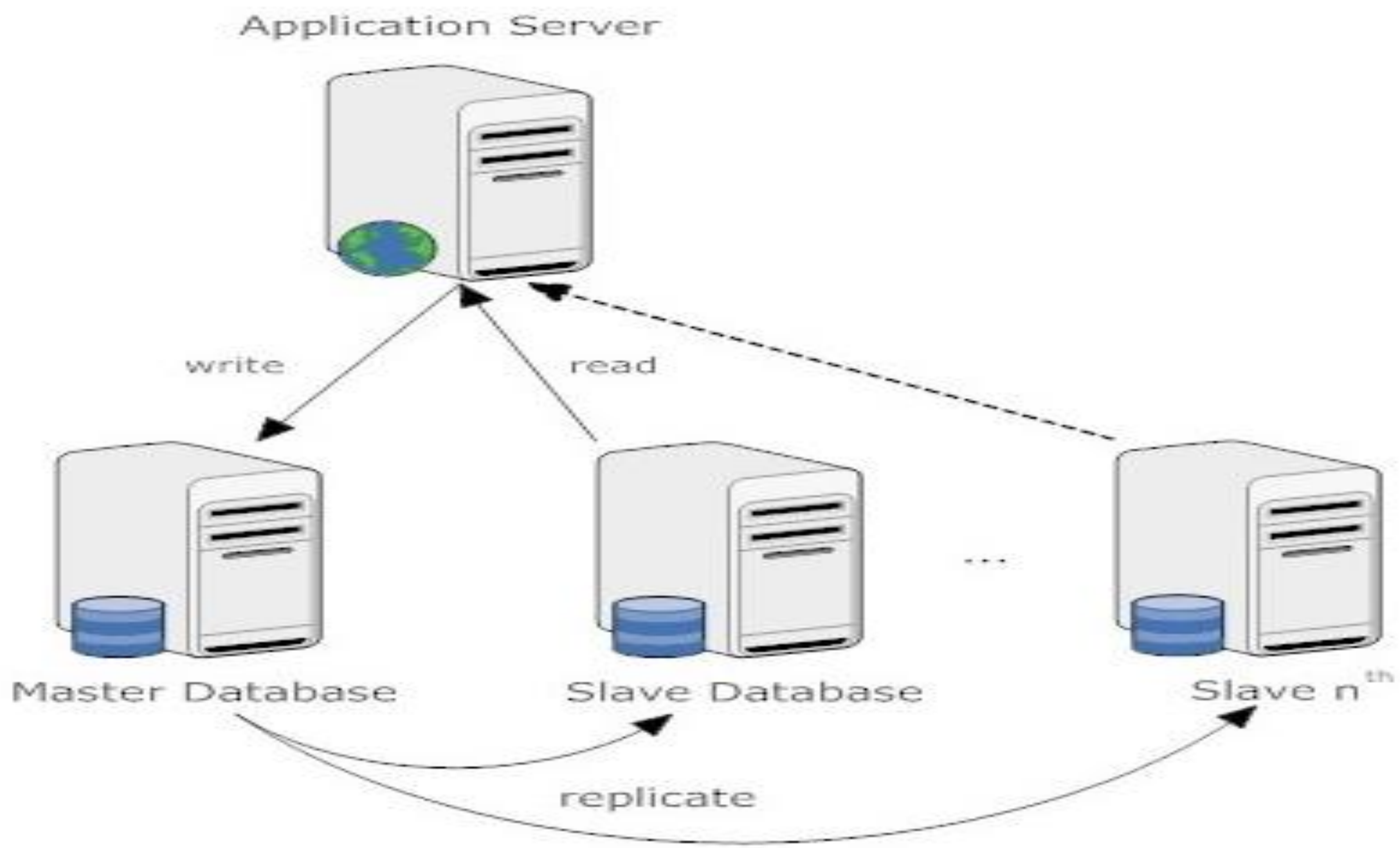
- Loss of referential integrity across shards. So the constraints are moved away from datastore and are part of application

- Re-balancing or Re-Sharding is hard
  - What to do when data do not fit in one shard
- Deciding on a partition factor/plan is hard
  - May generate hotspots

Sharding is Difficult

- What defines a shard? (Where to put stuff?)
  - Example - use country of origin: customer_us, customer_fr, customer_cn, customer_ie, …
  - Use same approach to find records
  - May generate hotspots
- What happens if a shard gets too big?
  - Rebalancing shards can get complex
- Query / join / transact **across** shards

Master-Slave Replication



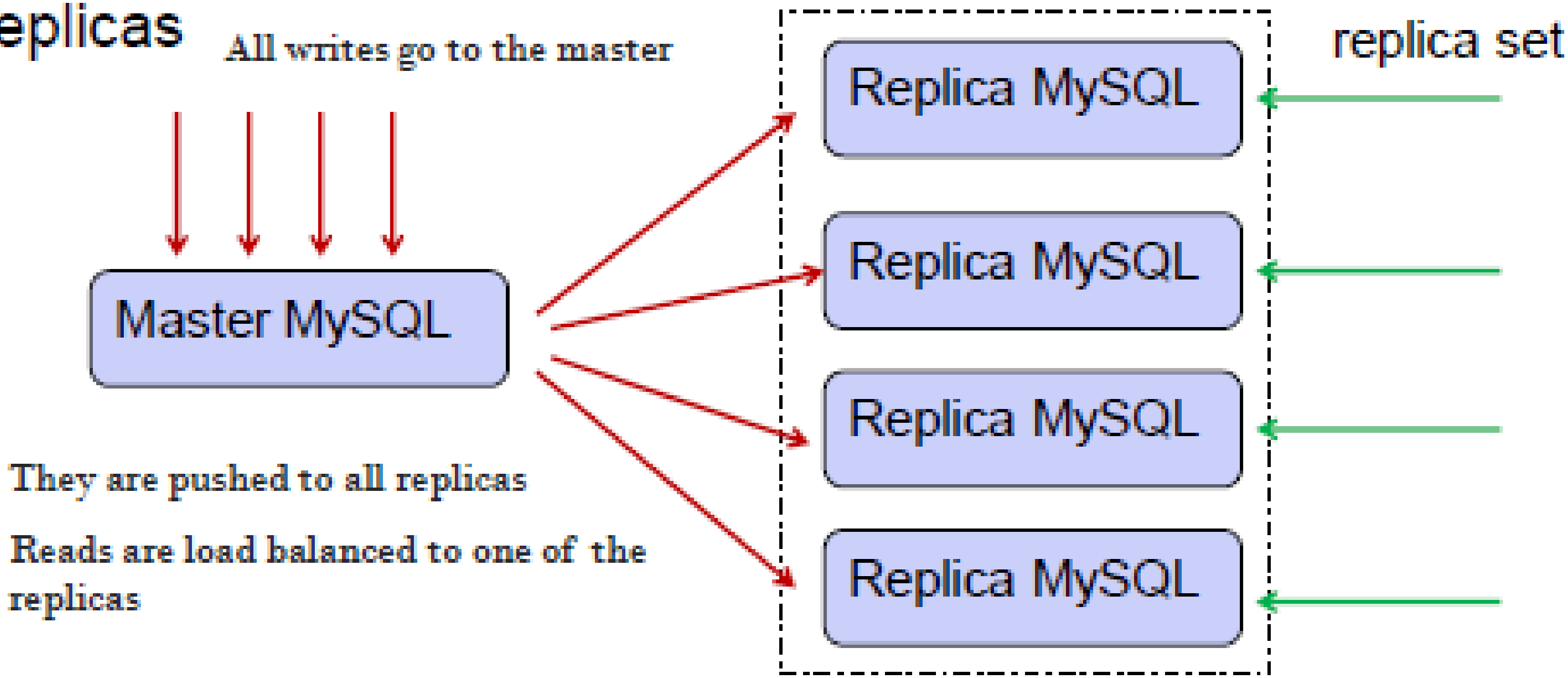
- Replication increases resilience and availability but introduces new class of consistency problems

Master-Slave Replication

- Makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads
  - *Reads may be inconsistent as writes may not have been propagated down*
  - *Large data sets can pose problems as master needs to duplicate data to slaves*

Master-Slave Replication Example

- Example: Wikipedia has one Master database and many replicas



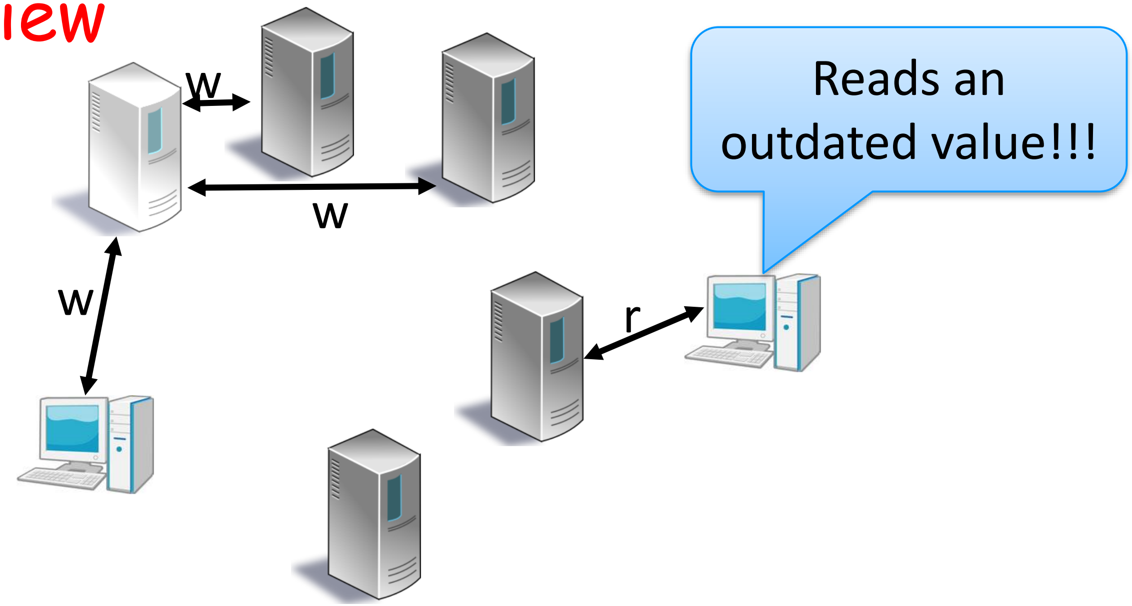
http://www.nedworks.org/~mark/presentations/san/Wikimedia%20architecture.pdf

Master-Slave Replication Implications

- When the master dies
  - One of the replica can be elected as the new master
- Some read may return old data if the latest value has not been pushed from the master
- It is possible to let Master handle read request for data requiring strong consistency
- Relatively easy to setup in most RDBMS

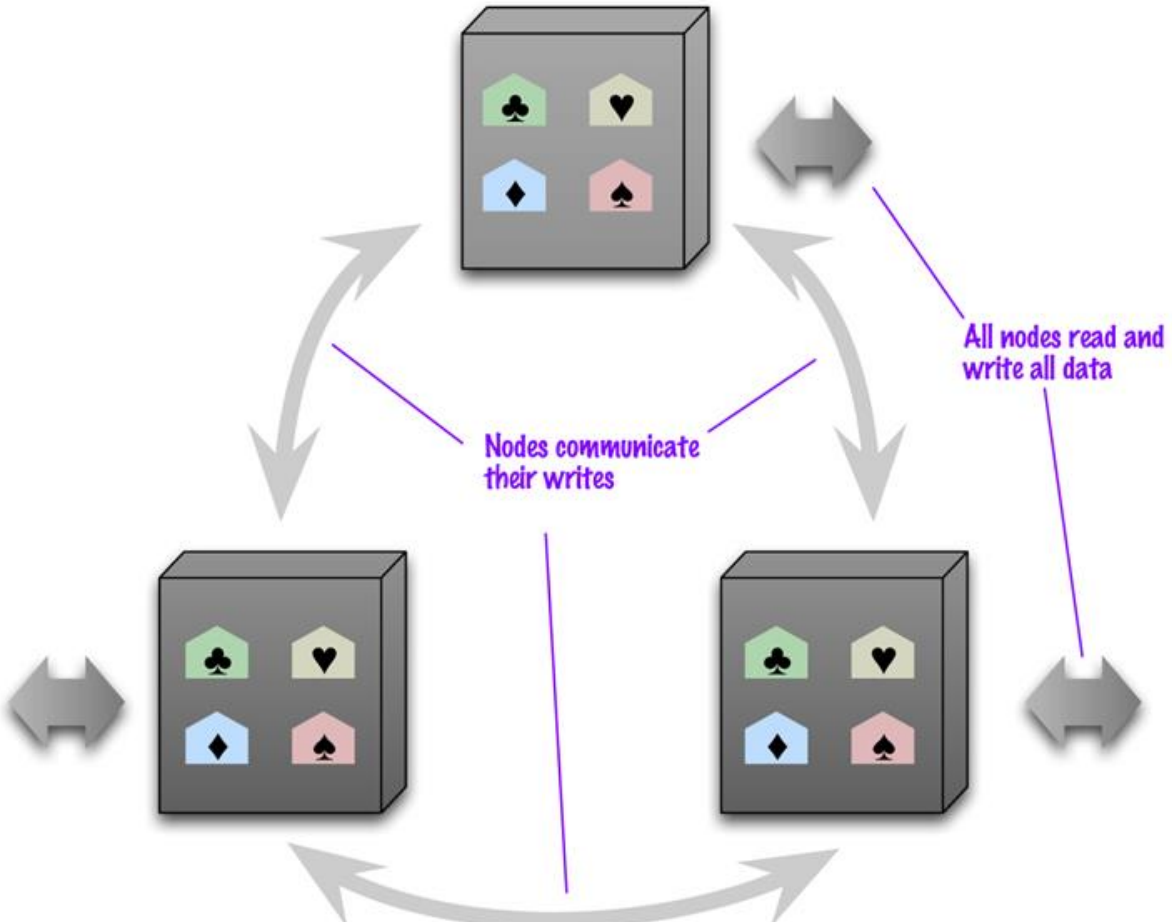
Problems with Master-Slave Replication

- If the clients can only send read requests to the Master, the system stalls if the Master fails
- However, if the clients can also send read requests to the other servers, the clients may not have a consistent view



- Master-slave replication helps with read scalability but doesn't help with scalability of writes

Peer-to-Peer Replication



- ***Peer-to-peer replication*** allows writes to any node; the nodes coordinate to synchronize their copies of the data.

Summary

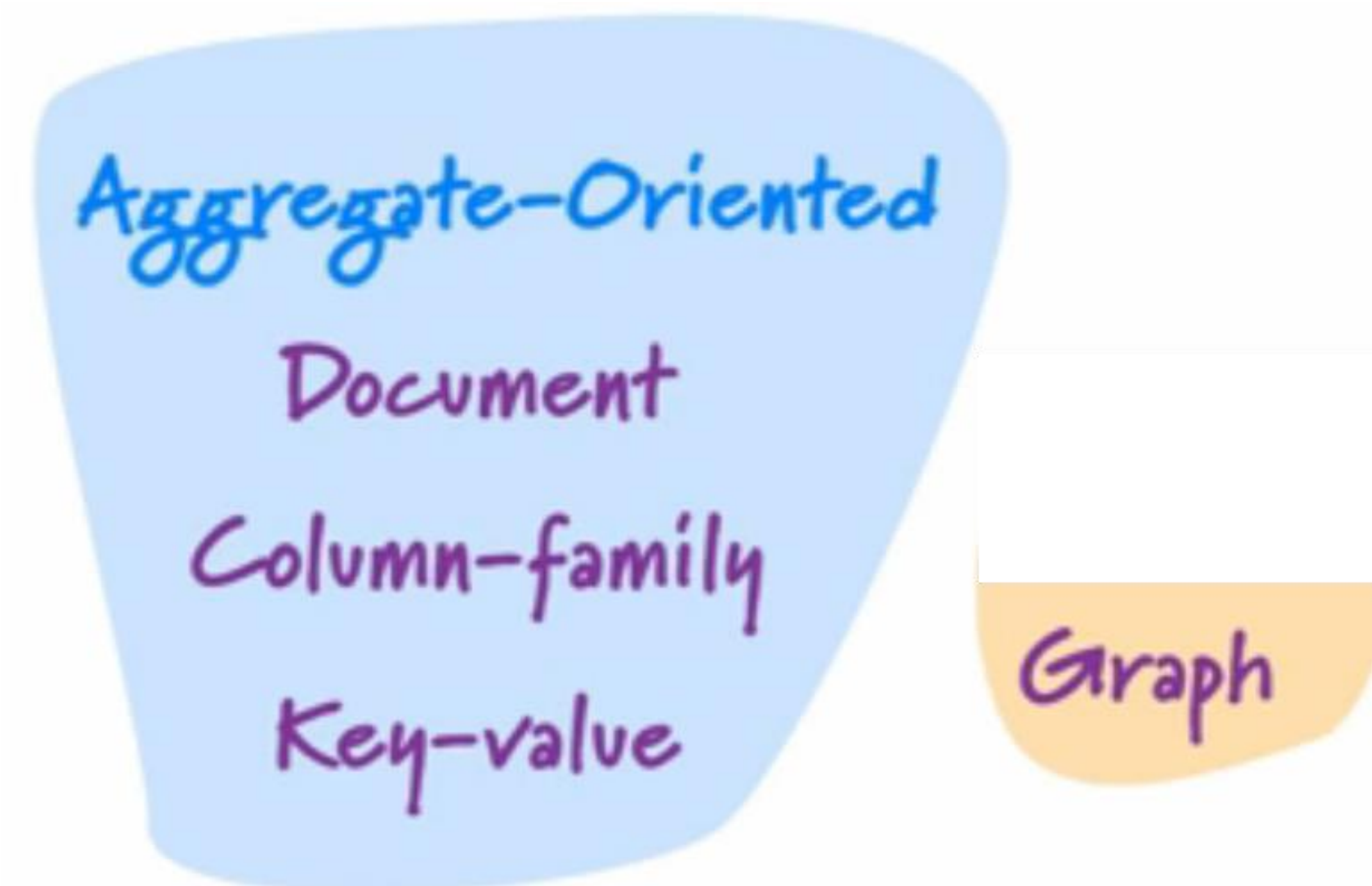
- Master-slave replication reduces the chance of update conflicts but peer-to-peer replication avoids loading all writes onto a single point of failure.
- Replication and sharding are strategies that are often combined.

NoSQL Taxonomy

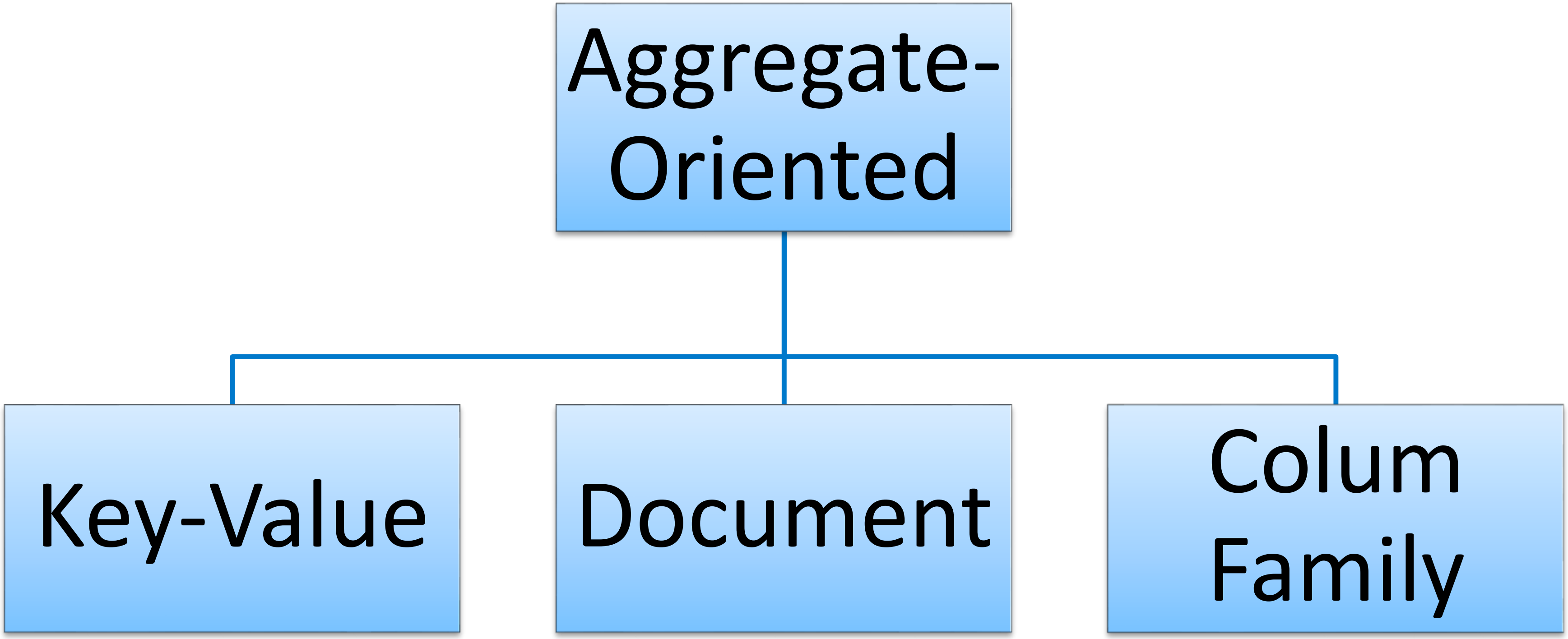
< Back


Document
mongoDB
RAVEN DB
CouchDB
relax
Colum
n-
family
Cassandra
APACHE
HBASE
Neo4j
NOSQL for the Enterprise
Project Voldemort
A distributed database.
riak
Graph
Key-value
redis

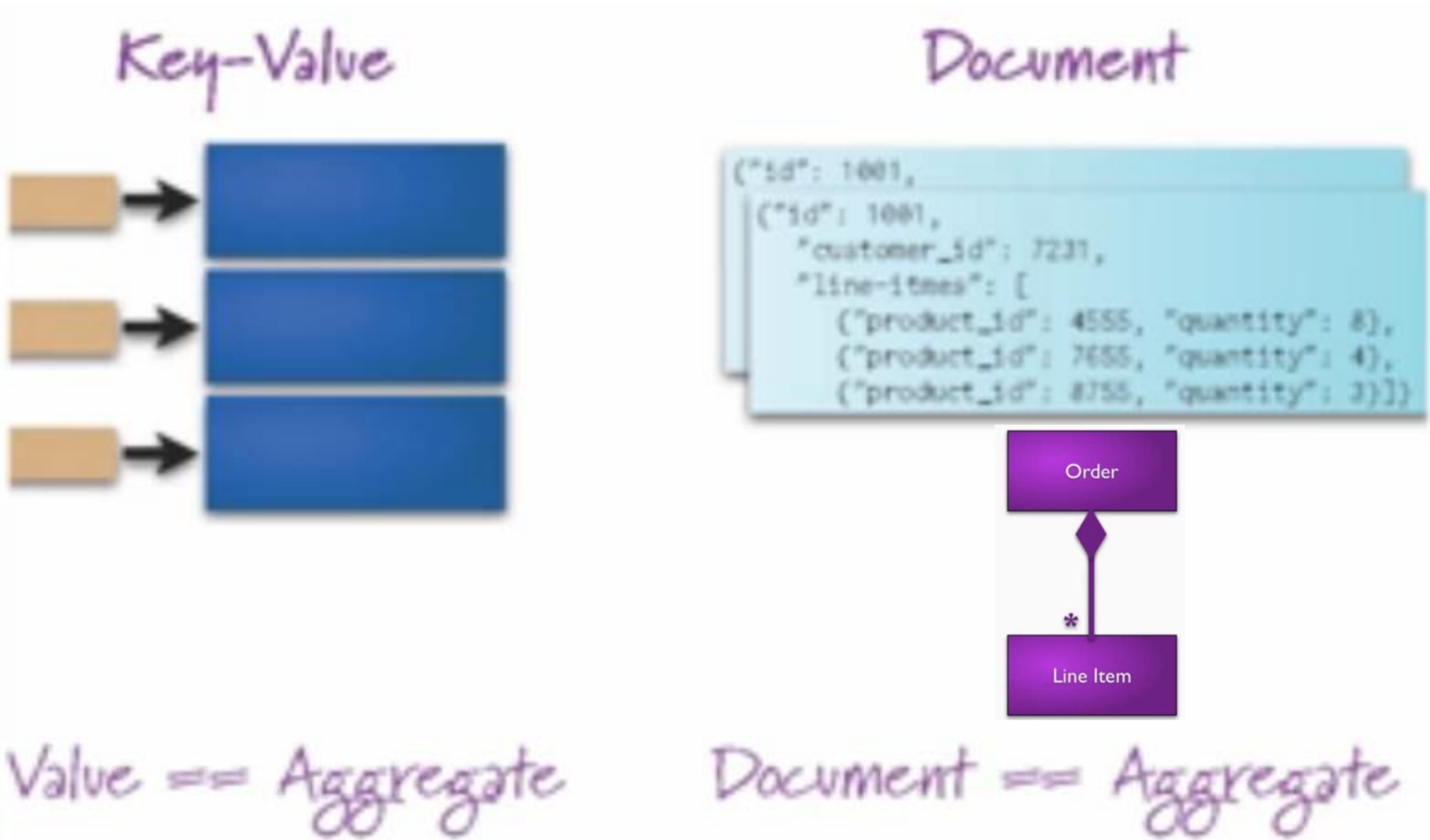
Two Major Categories



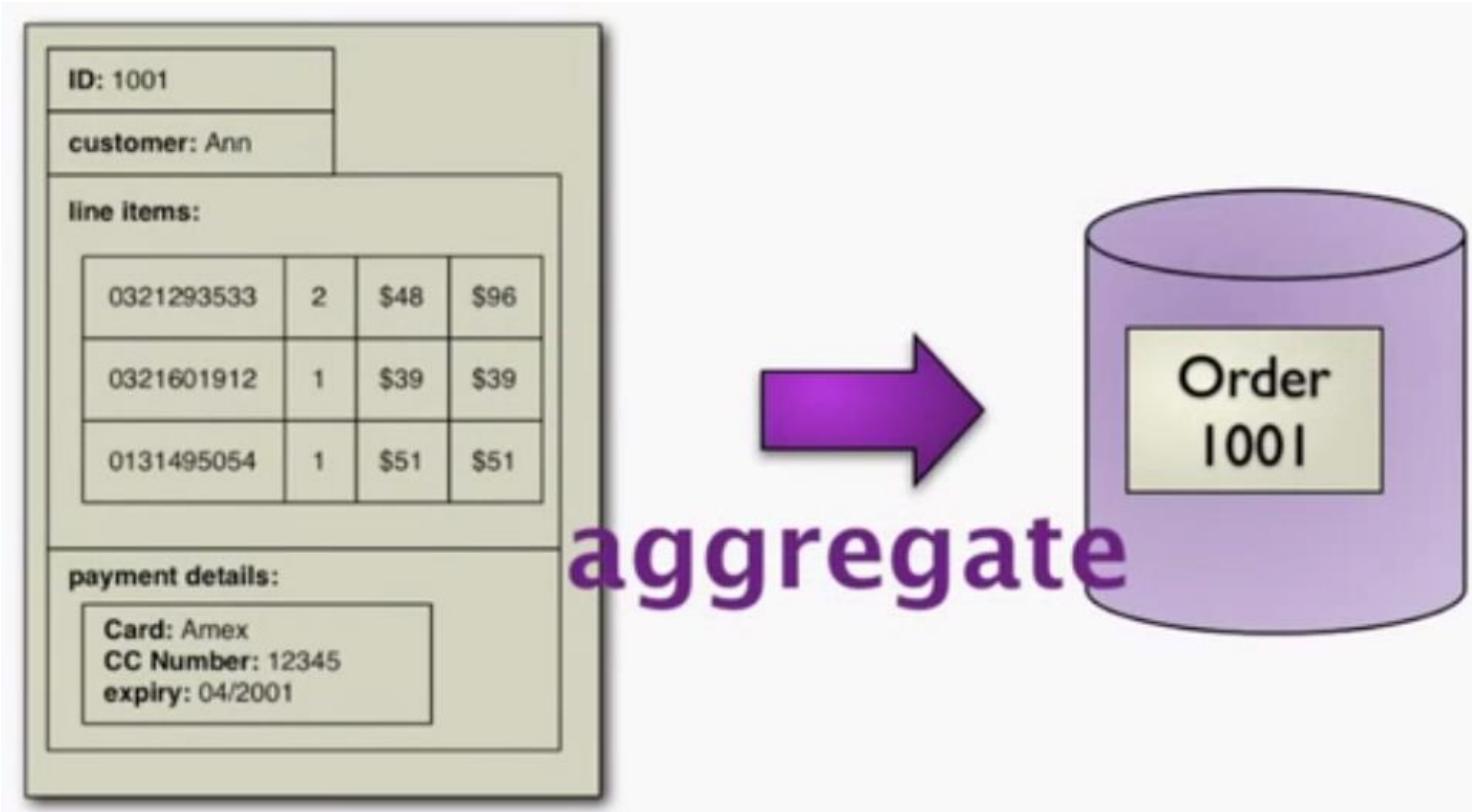
Aggregate-Oriented



Aggregate-Oriented

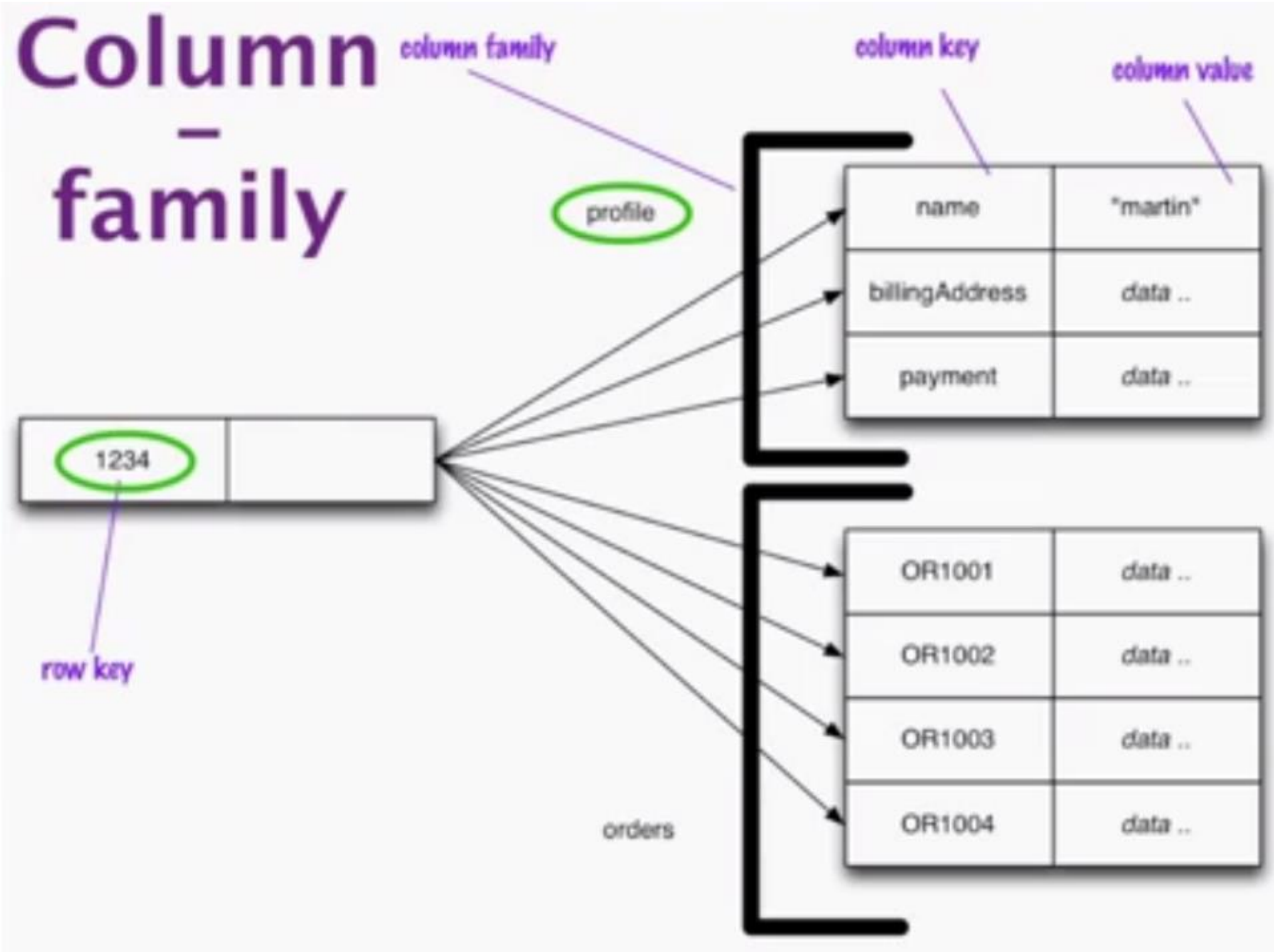


Aggregate Example



- Aggregate brings **cluster friendliness** as a whole aggregate can be stored in one node of the cluster

Column Family



A column-family database with different fields for the columns

Row Key	Column Families			
CustomerID	CustomerInfo		AddressInfo	
1	CustomerInfo:Title CustomerInfo:FirstName CustomerInfo:LastName	Mr Mark Hanson	AddressInfo:StreetAddress AddressInfo:City AddressInfo:County AddressInfo:PostCode	999 Thames St Reading Berkshire RG99 922
2	CustomerInfo:Title CustomerInfo:FirstName CustomerInfo:LastName	Ms Lisa Andrews	AddressInfo:StreetAddress AddressInfo:City AddressInfo:State AddressInfo:ZipCode	888 W. Front St Boise ID 54321
3	CustomerInfo:Title CustomerInfo:FirstName CustomerInfo:LastName	Mr Walter Harp	AddressInfo:StreetAddress AddressInfo:City AddressInfo:State AddressInfo:ZipCode	999 500th Ave Bellevue WA 12345

NoSQL Taxonomy

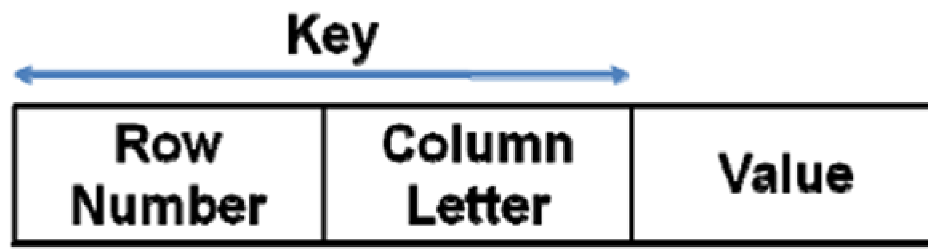
Conceptual Structures:

Key Value Stores

Schema-less system

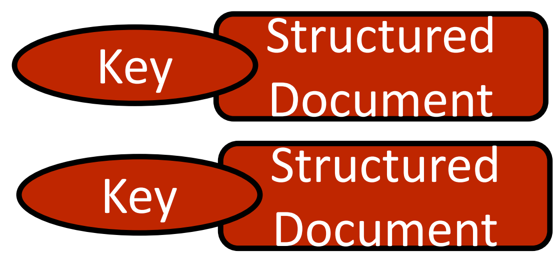
Column Family databases

key is mapped to a value that is a set of columns



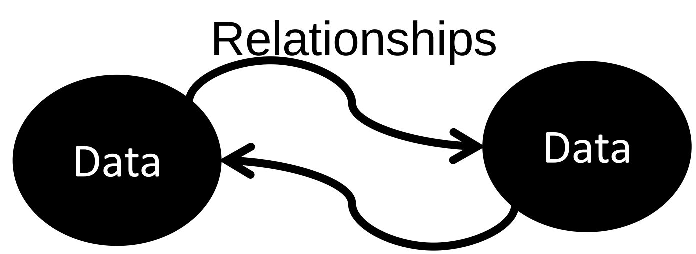
Document Oriented Database

Stores documents that are semi-structured



Graph Databases

Uses nodes and edges to represent data



NoSQL Taxonomy - Examples

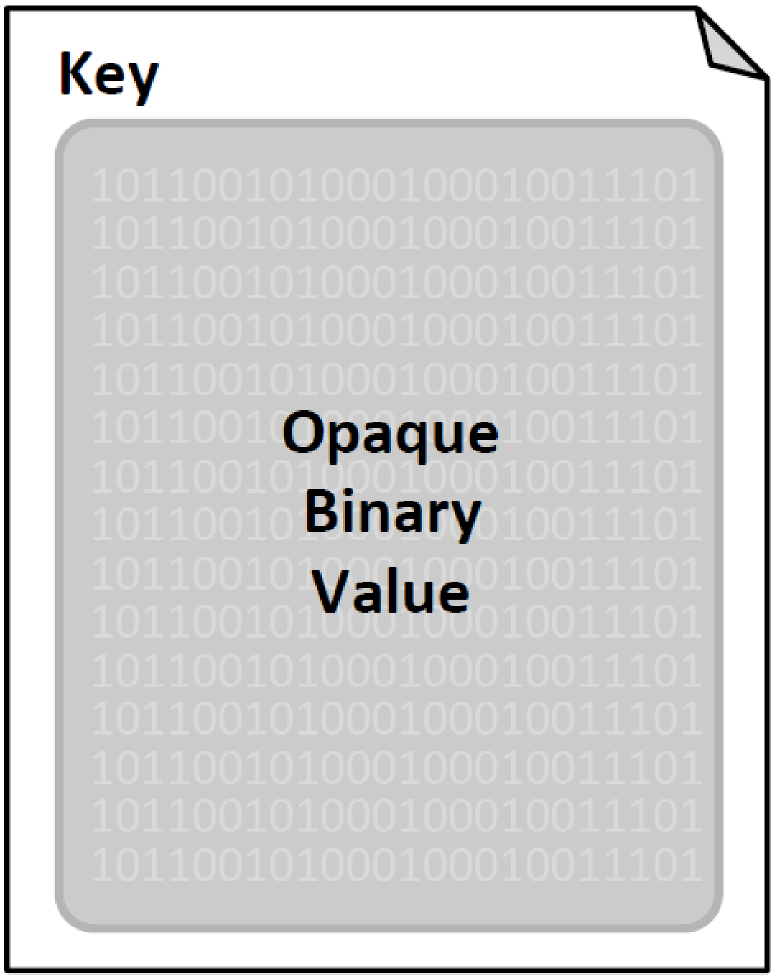
	Key-Value	Data Structure	Document	Column	Graph
Cache (memory only)	memcached	redis			
Database (memory/disk)	membase		couchbase mongoDB	cassandra	Neo4j

Important Design Goals

- **Scale out: designed for scale**
  - Horizontal scaling on commodity hardware
  - Low latency updates
  - Sustain high update/insert throughput

- **High availability – downtime implies lost revenue**
  - Replication (with peer to peer replication)
  - Geographic replication
  - Automated failure recovery

The Key-Value Store – the foundation of NoSQL

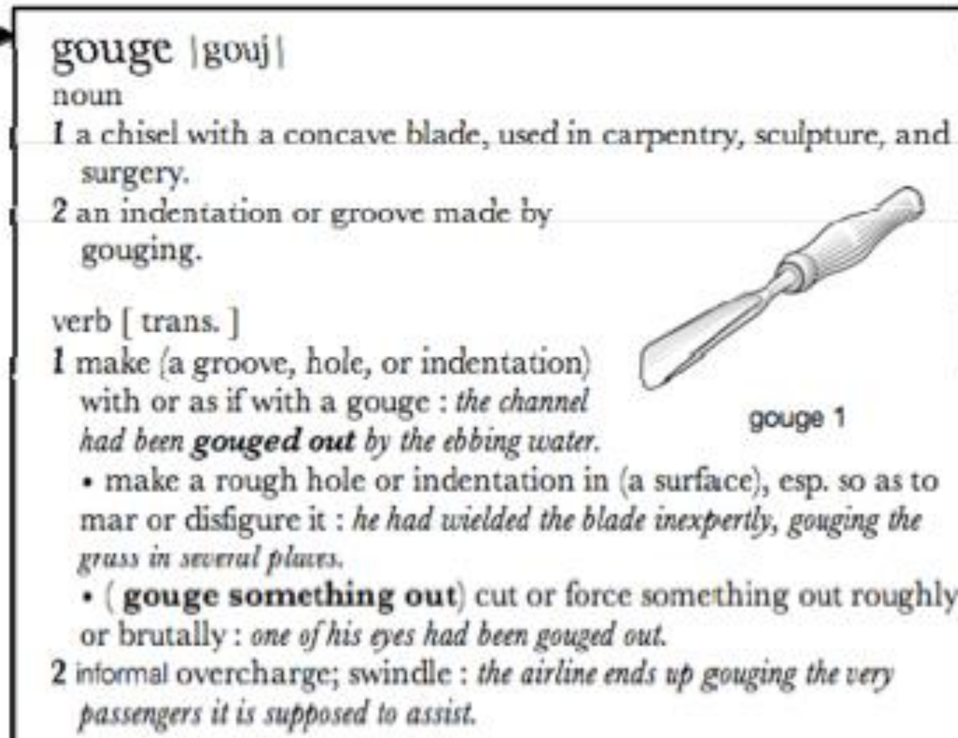


- Idea
  - HashMap
- Data unit: Key/Value pair
  - Key - string
  - Value
    - Basic types: int, string, ...
    - Collections of basic types: set, list, ...

Key-Values Stores are like Dictionaries

The "key" is just the word "gouge"

The "value" is all the definitions and images

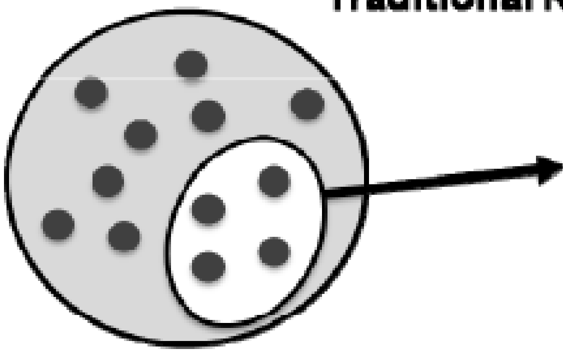


gouge |gouj|
noun
1 a chisel with a concave blade, used in carpentry, sculpture, and surgery.
2 an indentation or groove made by gouging.

verb [trans.]
1 make (a groove, hole, or indentation) with or as if with a gouge : *the channel had been **gouged out** by the ebbing water.*
• make a rough hole or indentation in (a surface), esp. so as to mar or disfigure it : *he had wielded the blade inexpertly, gouging the grass in several places.*
• (**gouge something out**) cut or force something out roughly or brutally : *one of his eyes had been gouged out.*
2 informal overcharge; swindle : *the airline ends up gouging the very passengers it is supposed to assist.*

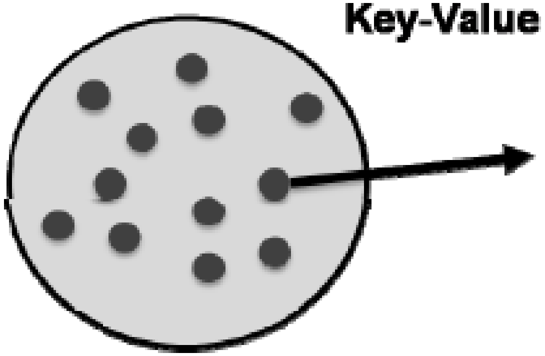
No Subset Queries in Key-Value Stores

Traditional Relational Model

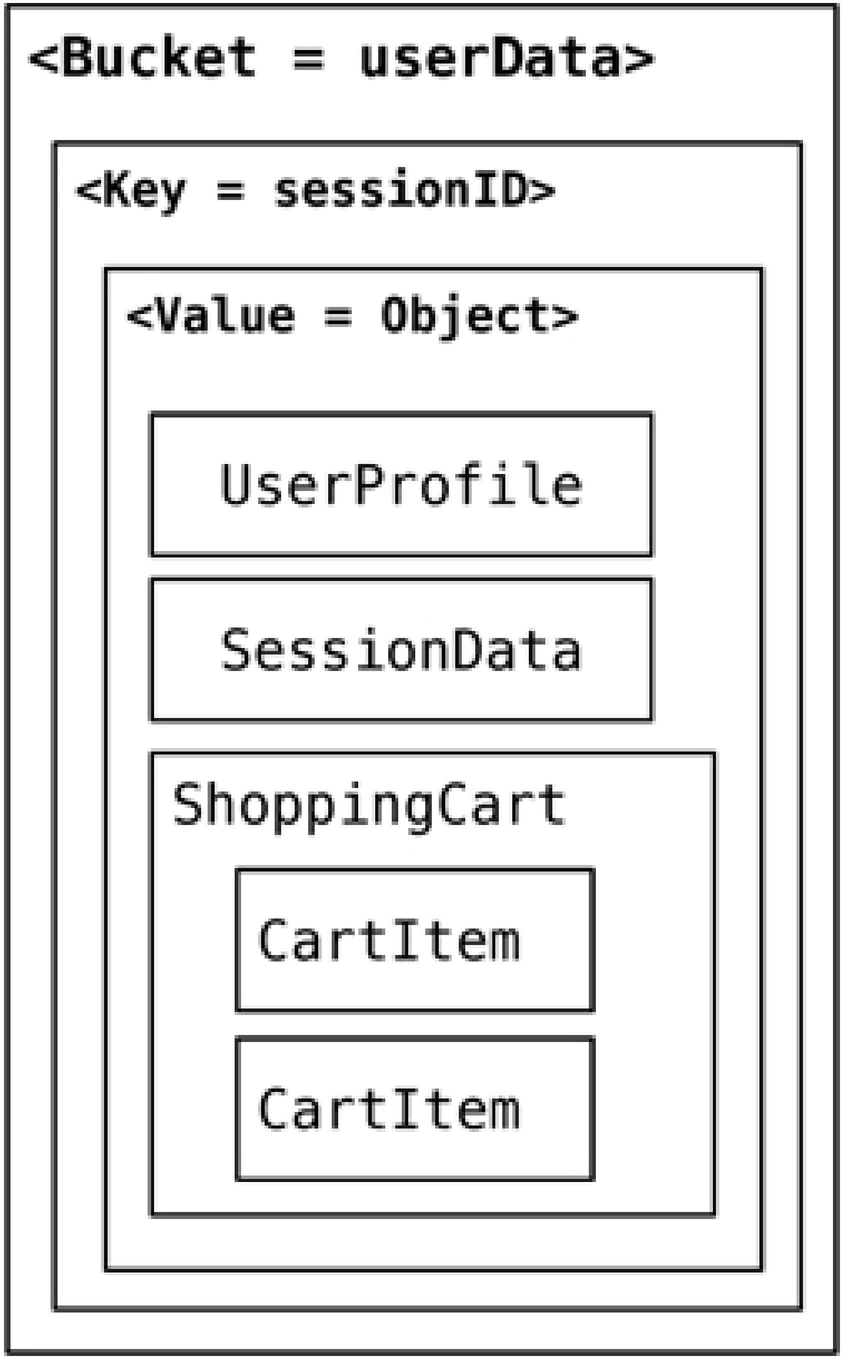


- Result set based on row values
- Value of rows for large data sets must be indexed
- Values of columns must all have the same data type

Key-Value Store Model



- All queries return a single item
- No indexes on values
- Values may contain any data type



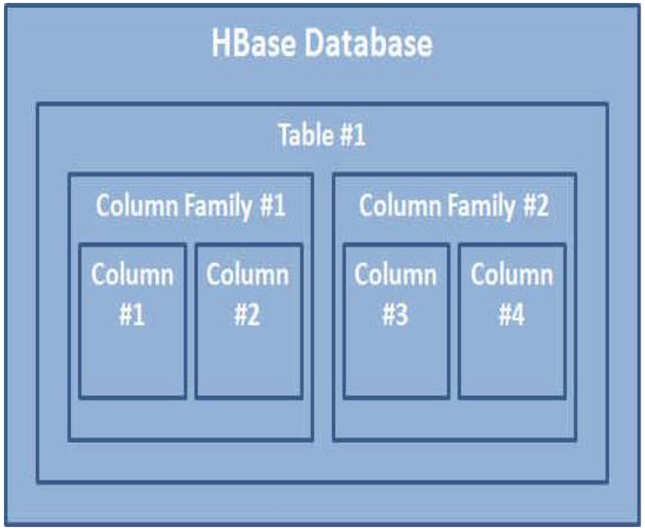
Storing all the User Data in a single bucket

Key/Value Stores

- Membase
  - Used by Zynga, NHN
- Redis
  - Used by Craigslist, Seznam, ALEF
- Dynamo
  - Used by Amazon
- Voldemort
  - Used by LinkedIn

Column-Oriented Stores

- Contrast with row-oriented RDBMS
- Data unit
  - Set of key(column)/value pairs
  - Sorted by row-key (primary key)
- Nulls are not stored
- Columns are organized in column-families
  - Name: FirstName, LastName,
  - Location: Address, State, GPS



Column-Oriented Stores

- Bigtable
  - Used by Google
- HBase
  - Used by Facebook, Yahoo!, Mahalo
- Hypertable
  - Used by Zvents, Baidu, Redif
- Cassandra
  - Used by Facebok, Twitter, Digg

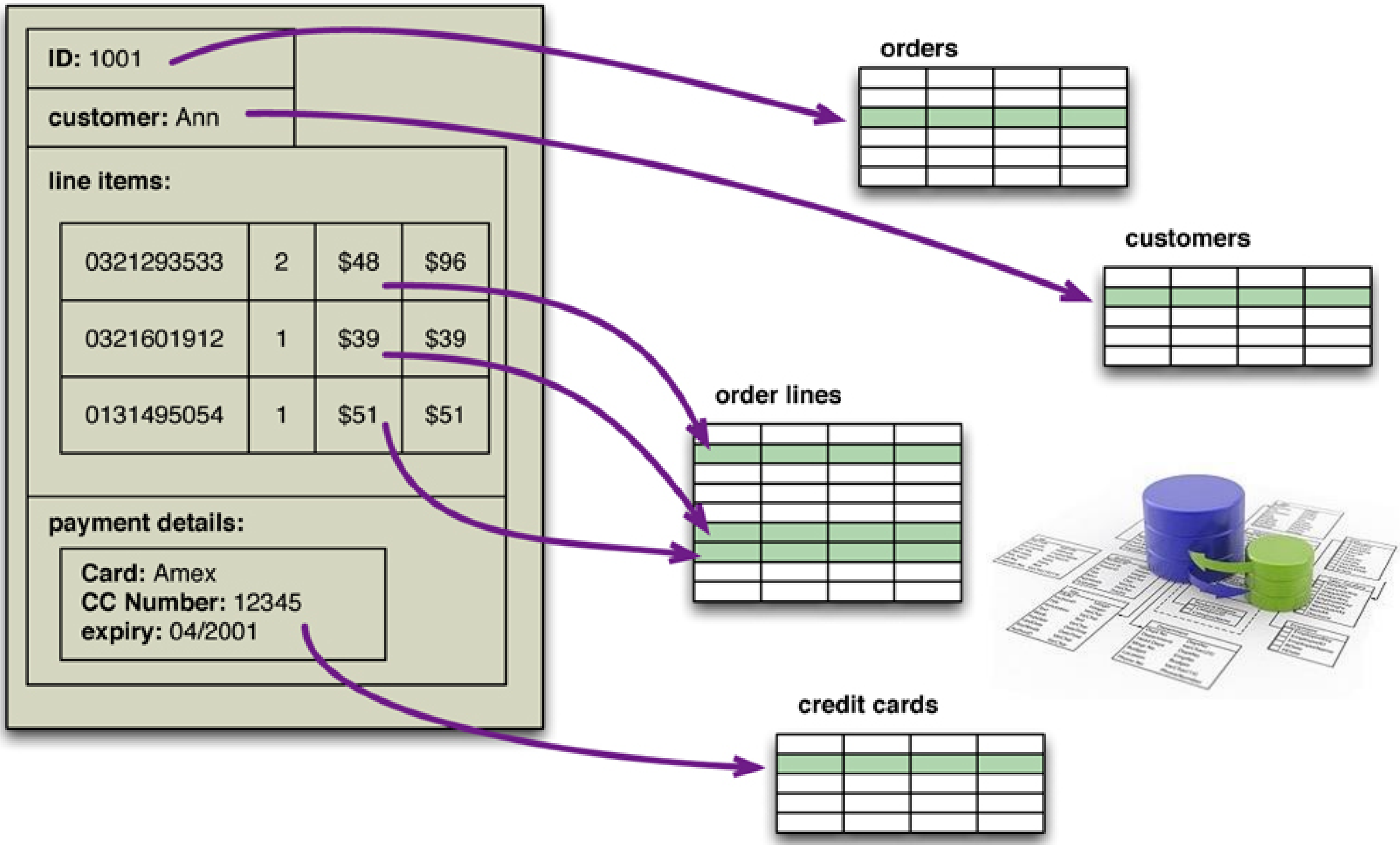
Document Databases

- Data unit:
  - Document = Object
  - Stored as a whole (not fragmented)
- JSON (BSON) notation
- Allows indexes on attributed

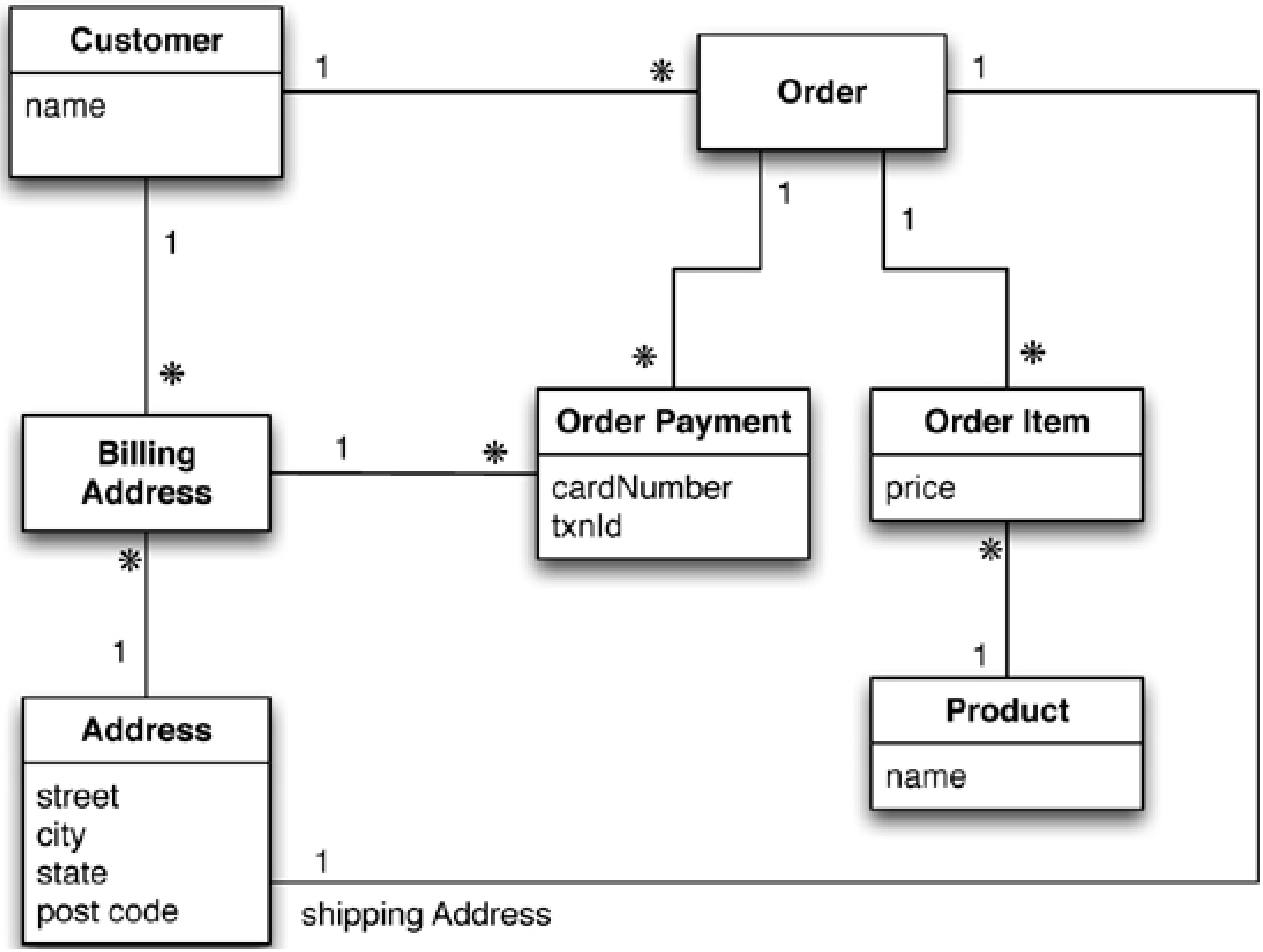
Document Databases

- CouchDB
  - Used by Apple, BBC, Cern, PeWeProxy
- MongoDB
  - Used by Github, ForSquare, Shutterfly, Sourceforge

Invoice Example



Relational Model



Relational Model – Example data

Customer	
Id	Name
1	Martin

Orders		
Id	CustomerId	ShippingAddressId
99	1	77

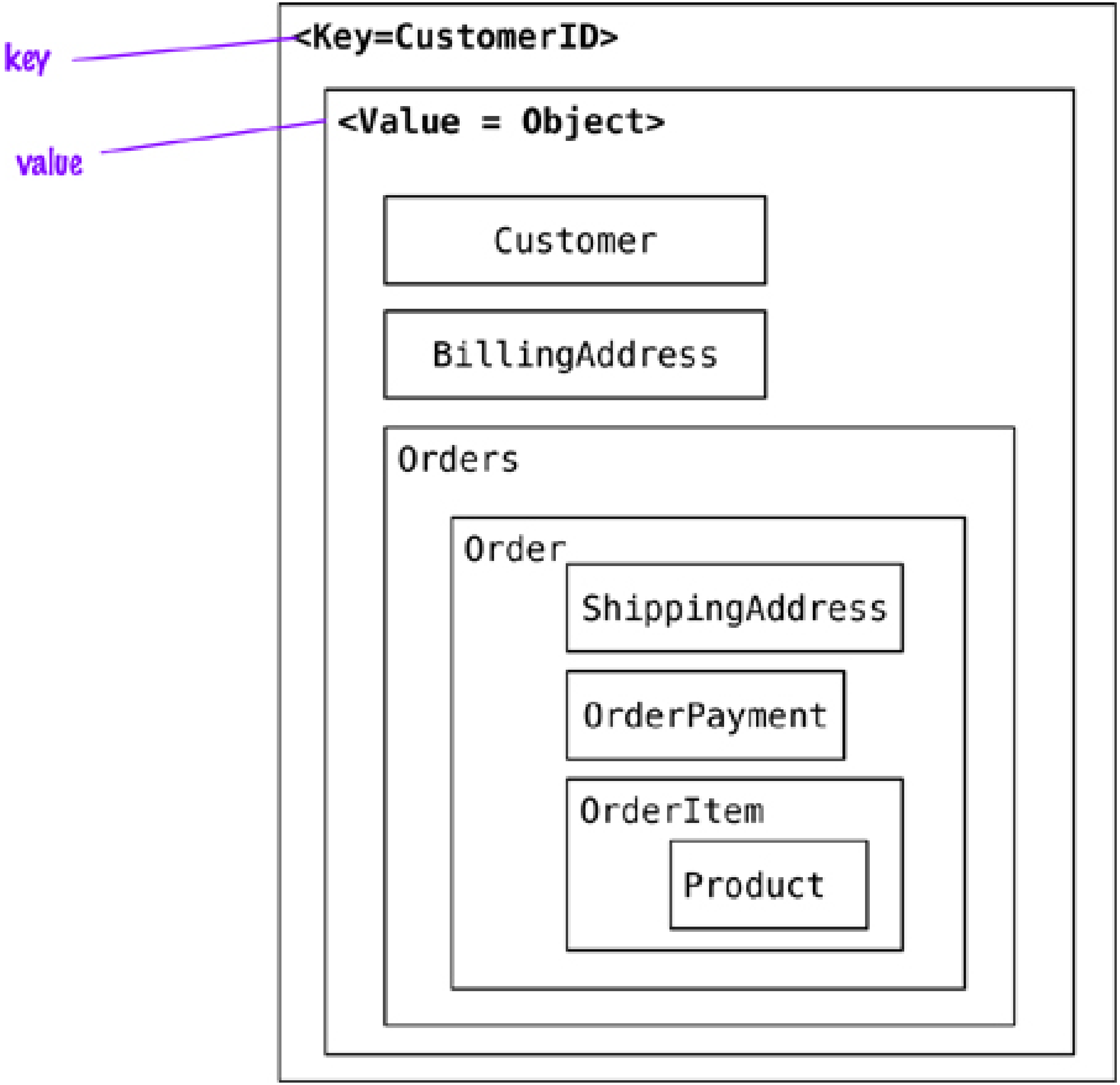
Product	
Id	Name
27	NoSQL Distilled

BillingAddress		
Id	CustomerId	AddressId
55	1	77

OrderItem			
Id	OrderId	ProductId	Price
100	99	27	32.45

Address	
Id	City
77	Chicago

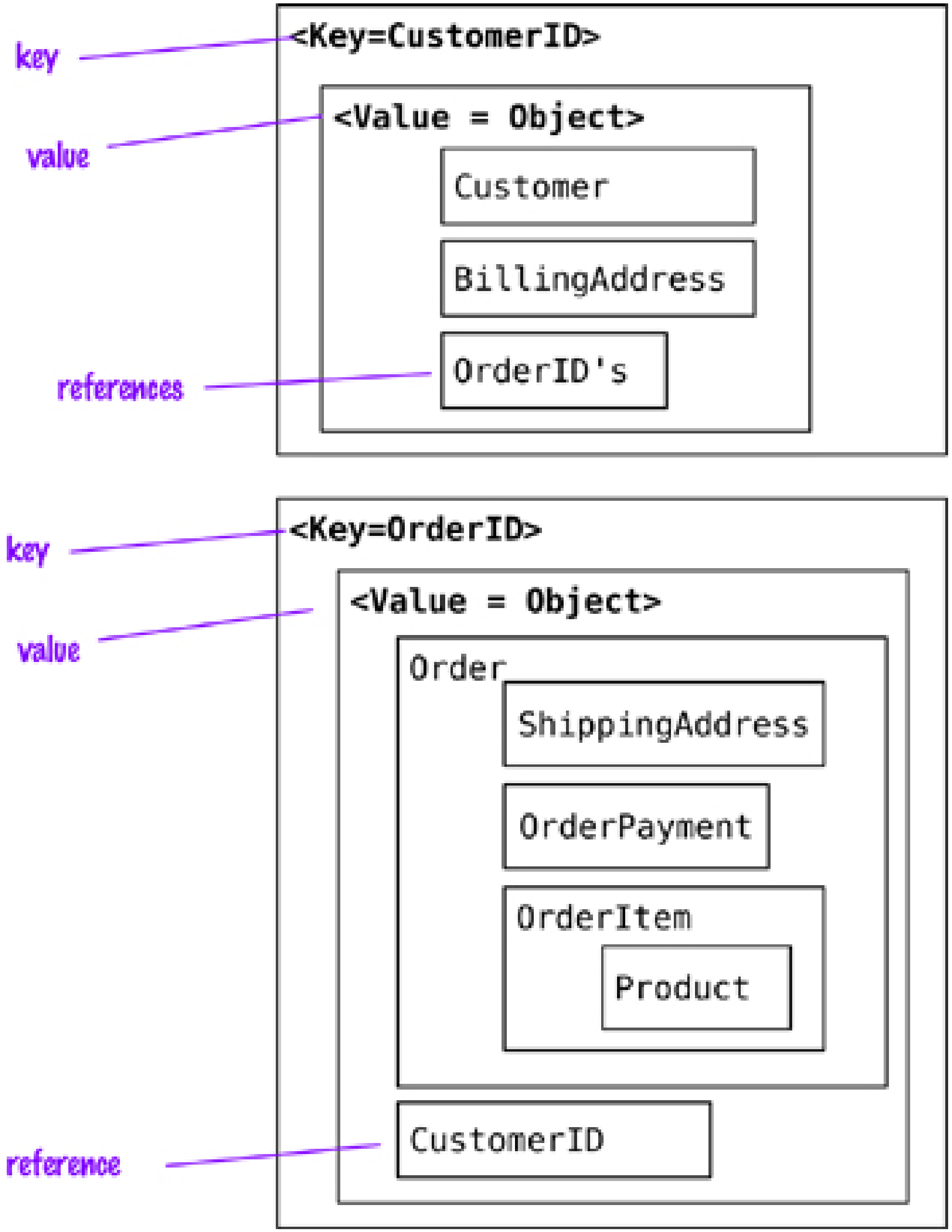
OrderPayment				
Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft



Embed all the objects for customer and their orders

JSON Document

```
// in customers
{
"customer": {
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
  {
    "id":99,
    "customerId":1,
    "orderItems":[
    {
    "productId":27,
    "price": 32.45,
    "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
  "orderPayment":[
    {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
    }],
  }]
}
}
```



Customer is stored separately from Order

JSON Document

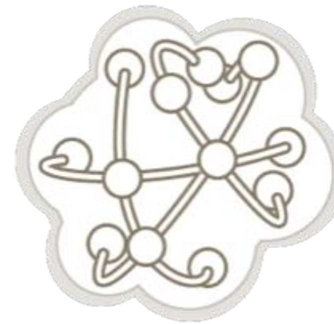
```
# Customer object
{
"customerId": 1,
"customer": {
  "name": "Martin",
  "billingAddress": [{"city": "Chicago"}],
  "payment": [{"type": "debit","ccinfo": "1000-1000-1000-1000"}],
  "orders":[{"orderId":99}]
 }
}

# Order object
{
"customerId": 1,
"orderId": 99,
"order":{
  "orderDate":"Nov-20-2011",
  "orderItems":[{"productId":27, "price": 32.45}],
   "orderPayment":[{"ccinfo":"1000-1000-1000-1000",
           "txnId":"abelif879rft"}],
   "shippingAddress":{"city":"Chicago"}
   }
}
```

Graph Databases

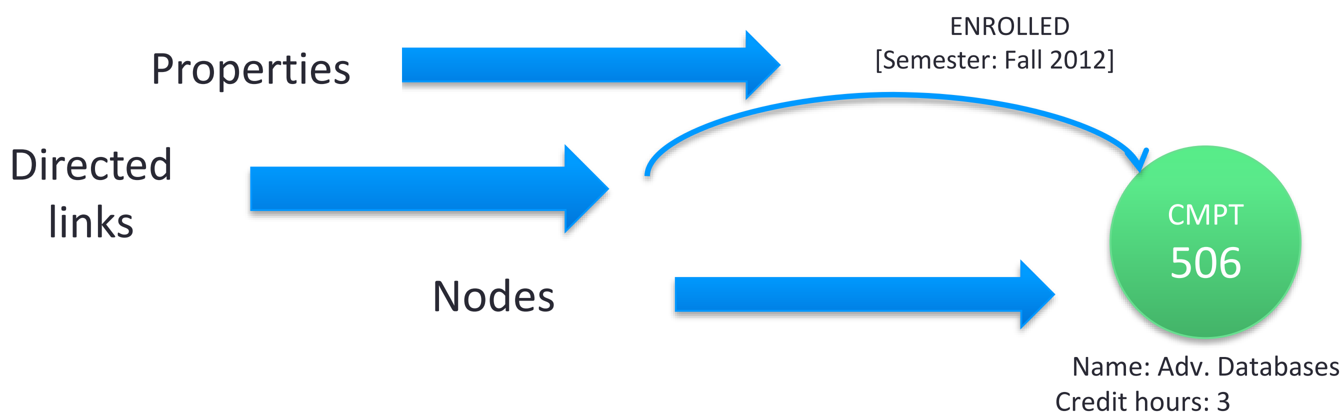
- Based on Graph Theory
- ACID Database
- You can use graph algorithms easily

Graph Databases

Graph DB



- Data model:
  - Nodes with properties
  - Named relationships with properties



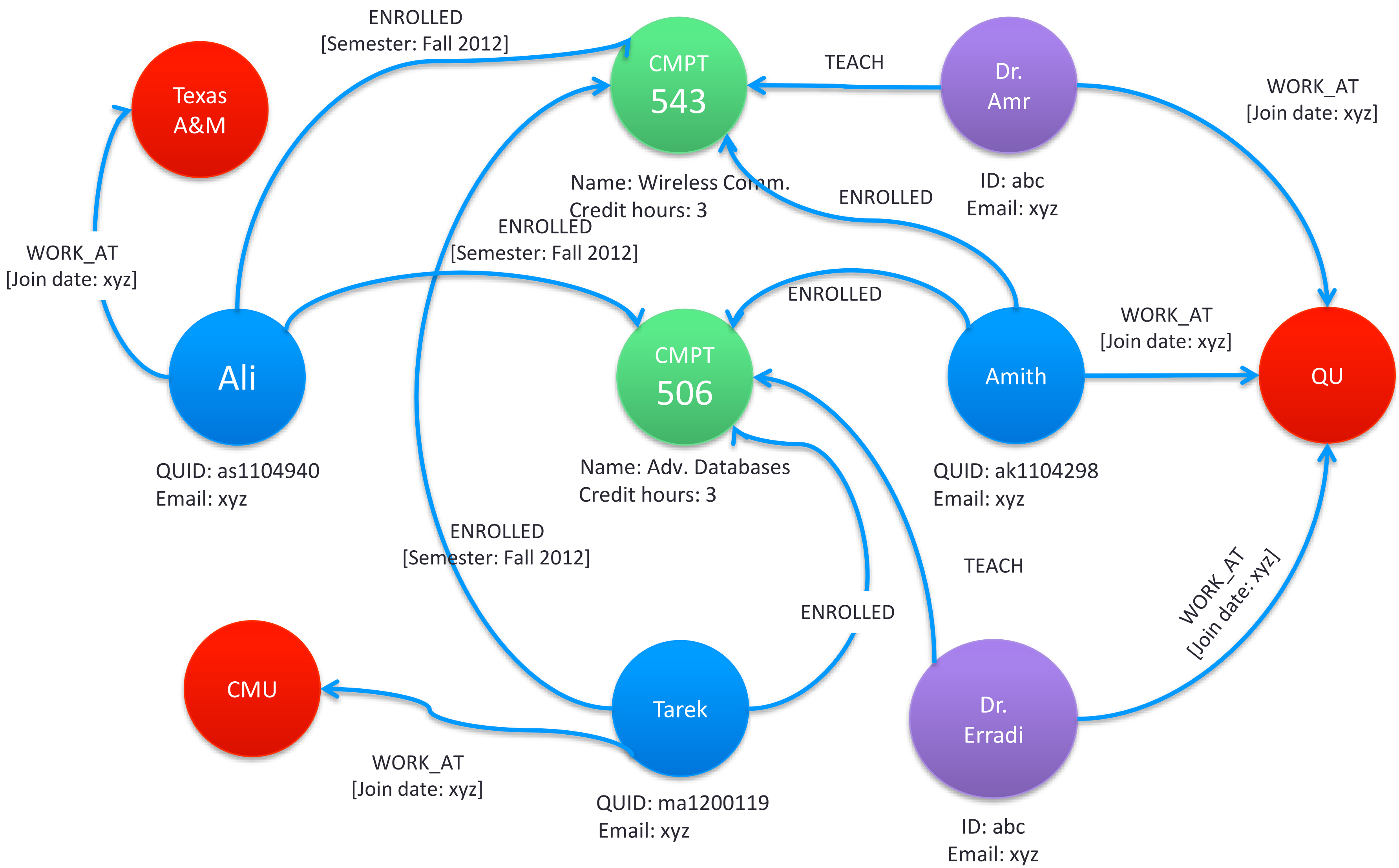
https://github.com/jimwebber/neo4j-tutorial/tree/master/presentation

Pros and Cons

- Strengths
  - More natural and Powerful data model
  - Fast
    - For connected data, can be many orders of magnitude faster than RDBMS
  - High performance graph operations
    - Traverses 1,000,000+ relationships / second on commodity hardware

- Weaknesses:
  - Sharding not easy
  - Schema-less makes difficult to query
  - Does not work well with Range Queries like the RDBMS.

Example Graph Model



Graph Databases

- AllegroGraph
  - Used by TwitLogic, Pfizer
- FlockDB
  - Used by Twitter
- Neo4j
  - Used by Box.net

CAP Theorem

< Back

RDMS ACID properties

- **Atomicity**: Everything in a transaction succeeds or the entire transaction is rolled back (All or Nothing)
- **Consistency**: A transaction that runs on a correct database leaves it in a correct ("consistent") state
- **Isolation**: Transactions cannot interfere with each other => The updates of a transaction must not be made visible to other transactions until it is committed
- **Durability**: Once a transaction commits, updates can't be lost or rolled back

Transactions

Consistency of Data
Saveguard against undesired/wrong Manipulation of Data

Durability of Data
Saveguard against System Failure
Support Recovery

- Consistency: Saveguard against undesired / wrong manipulation of data
- Durability: Saveguard against loss of data
- A Transaction Manager must guarantee that
  - Either the entire transaction is carried out and the result is recorded in permanent storage or not at all (atomic actions)
  - In case of system failure, data can be recovered
  - Concurrently executing transactions are isolated from each other so that (a) their write operations do not overwrite each others' results and (b) that one transaction is protected from reading partial / intermediate and uncommitted results of other transactions.

Atomicity

- Consider a bank transaction T:

T:	A=A+100,	B=B-100

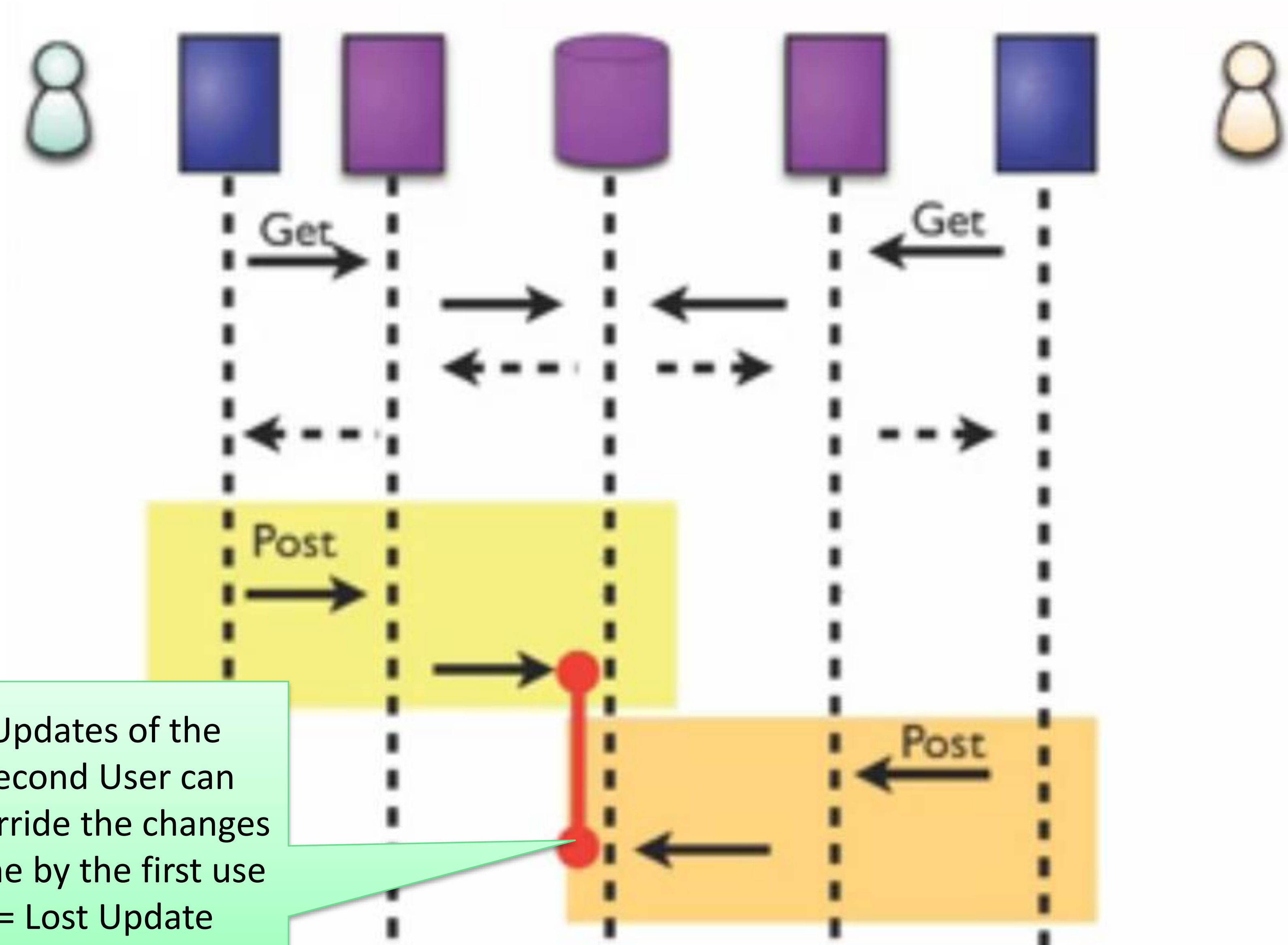
→ time

- T is transferring $100 from B's account to A's account.
- What if there is an error right after the first statement of T has been executed, i.e. the second statement is not executed?

=> You will get partial update = inconsistent state

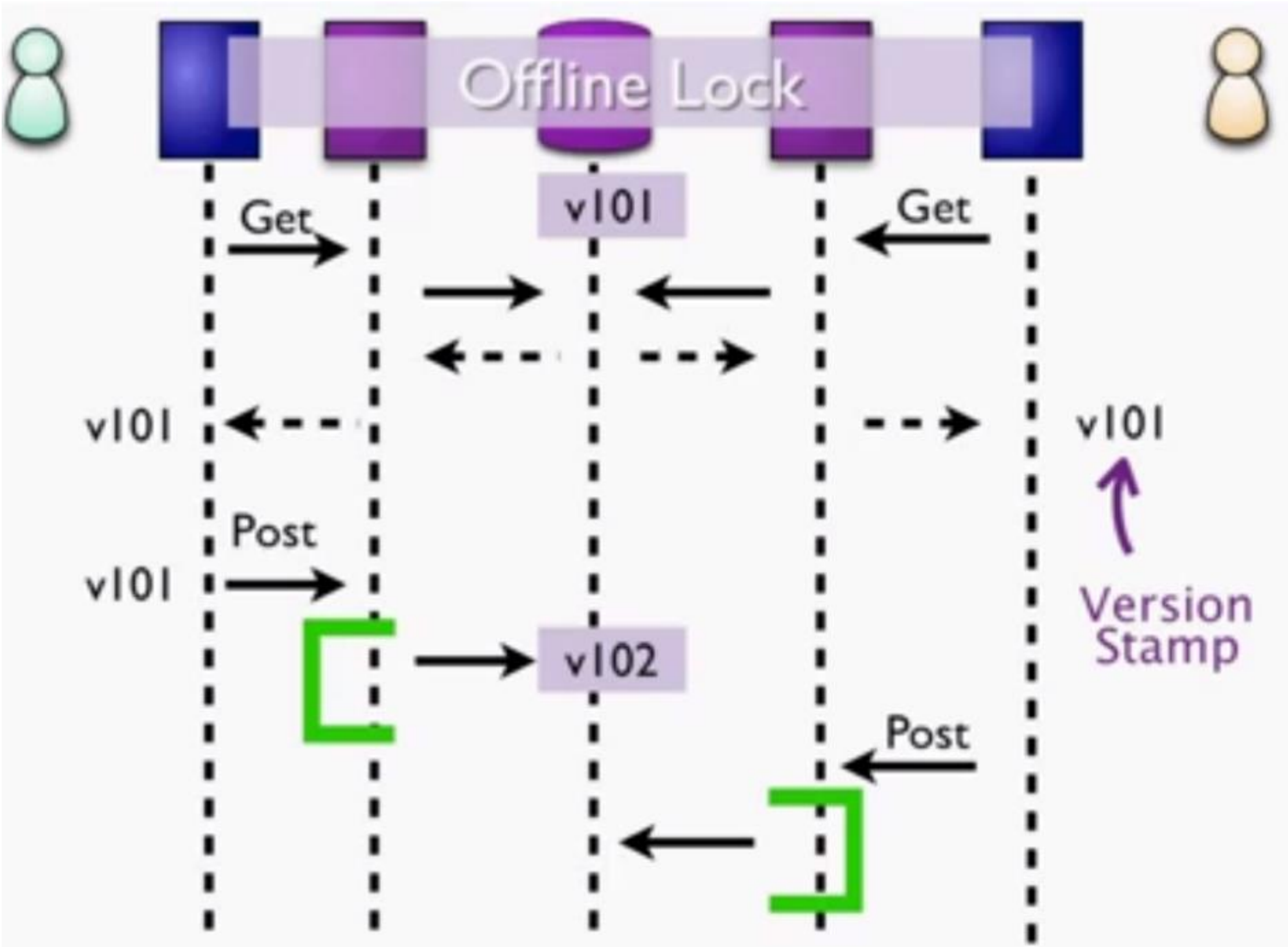
- The DBS has to ensure that every transaction is treated as an *atomic* unit, i.e. either all or none of its SQL statements are executed.

Even with Transactions Lost Update Can still happen



Updates of the Second User can override the changes done by the first use = Lost Update

Lost Update Solution Without Transactions



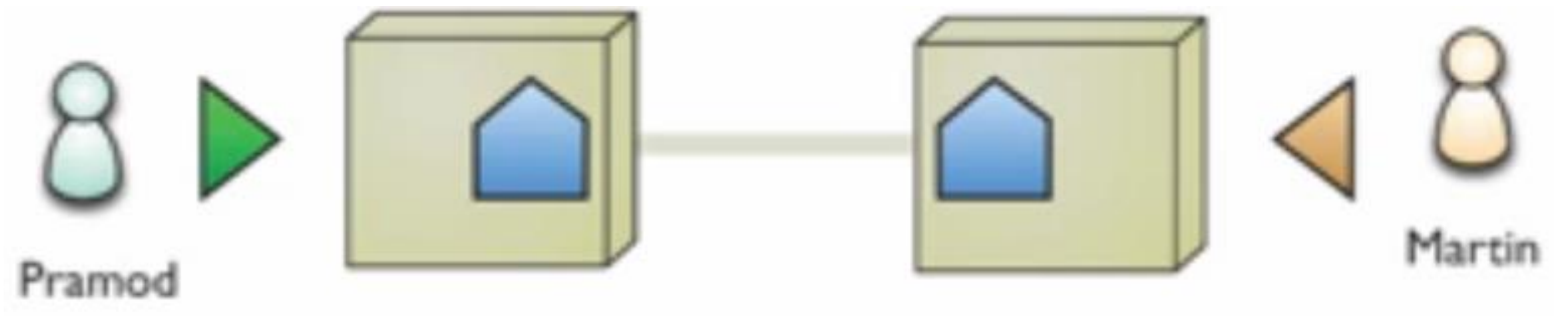
ACID Critics

- Any data store can achieve Atomicity, Isolation and Durability but do you always need consistency?

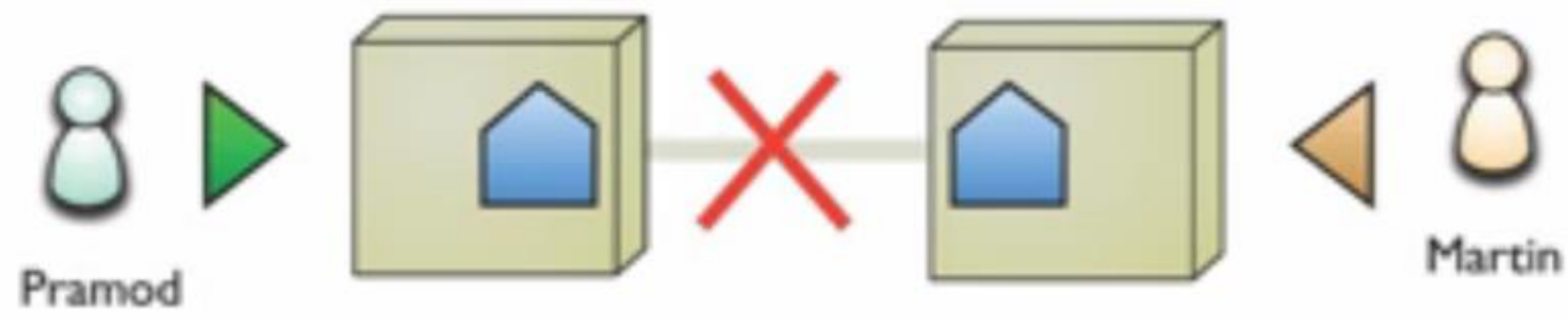
No.

- By giving up ACID properties, one can **achieve higher performance, higher availability and higher scalability**

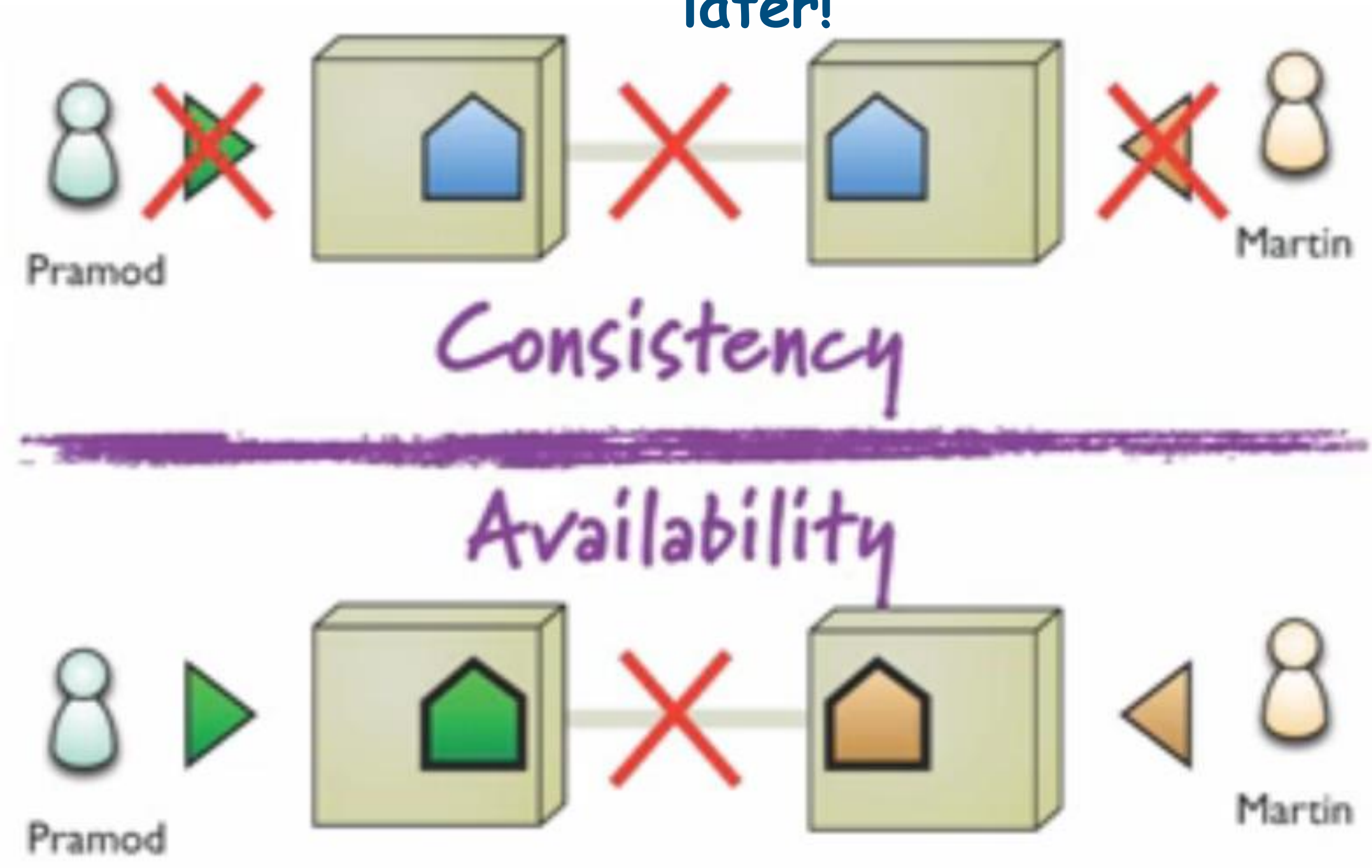
Booking Last Room Scenario by two customers interacting with replicated data in 2 different servers



But the network connection between the two nodes is lost… what is the solution?



Option 1: Preserve **Consistency** (not book the room twice) by saying to the customers 'System is down please try again later!'



Option 2: Sacrifice consistency (let the room be booked twice) but get higher availability

CAP-Theorem

- When data is partitioned, in case of network/node failure, we can only maintain consistency or availability but NOT both at the same time

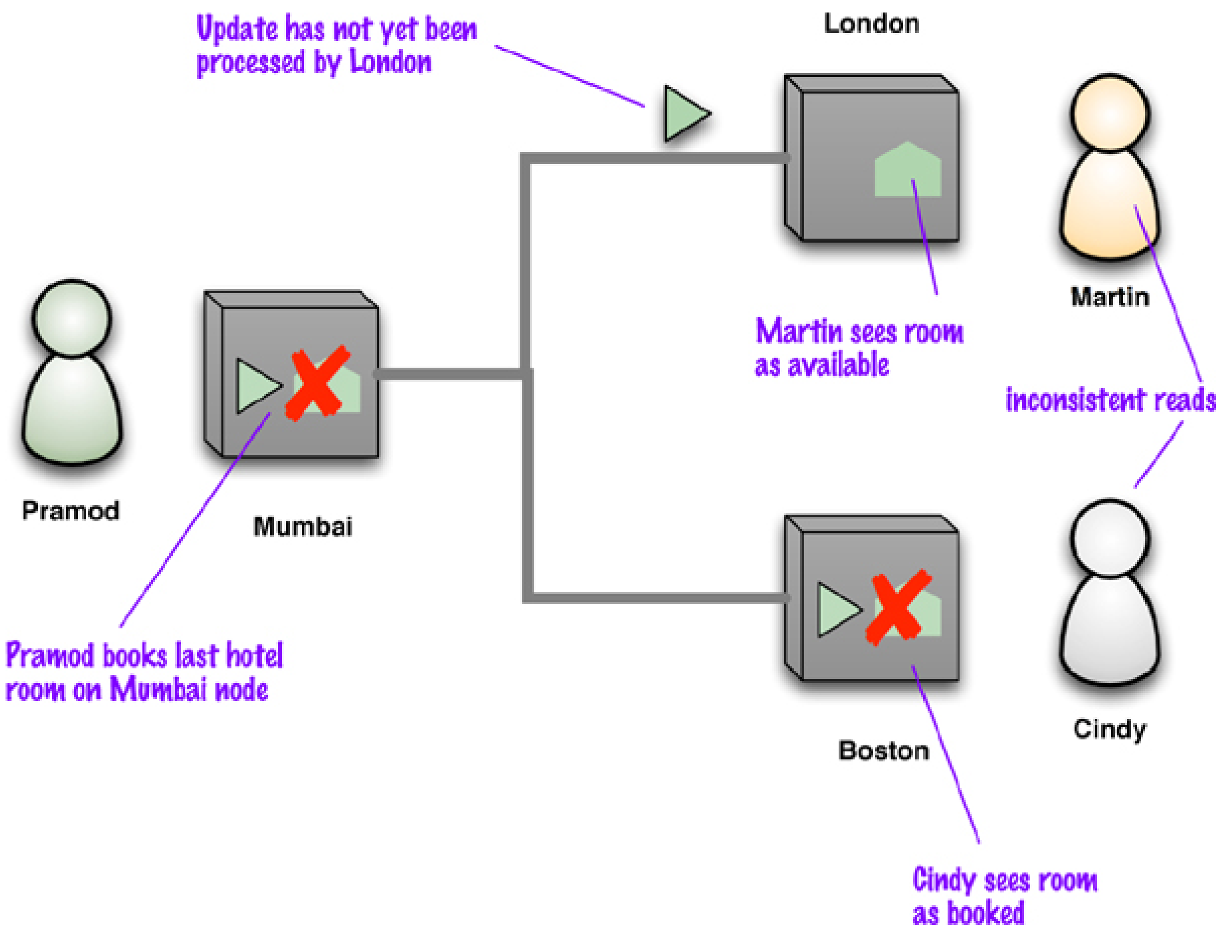
CAP Theorem

- Also known as Brewer's Theorem by Prof. Eric Brewer, published in 2000 at University of Berkeley.
- "Of three properties of a shared data system: data consistency, system availability and tolerance to network partitions, only two can be achieved **at any given moment**."
- Proven by Nancy Lynch et al. MIT labs.

http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

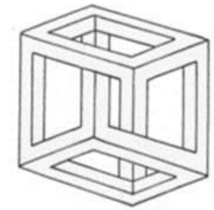
An example of replication inconsistency

- Replication can introduce Replication Consistency. Here is an example:



CAP-Theorem: Consequences

Partition



Drop Availability

Wait until data is consistent and therefore remain unavailable during that time.

Drop Consistency

Accept that things will become "Eventually consistent"
(e.g. bookstore: If two orders for the same book were received, one of the orders becomes a back-order)

This is a Business Decision NOT a technical one

Summary - NoSQL

- NoSQL movement produced **Polyglot Persistence**
- Observation: no single system is ideal to meet all requirements
- Solution: use right storage backend for each use case!
- Many projects use already two data stores: file system and database
- Cost of complexity: increased complexity may lead to increased costs
- Derived from polyglot programming (writing code in several languages)