Introduction to No SQL Databases

pm_jat @ daiict

Introduction to No SQL Databases

- What are NoSQL databases?
- Why do we need newer database types?
  - Where does our age old relational is failing?
- Different genres of NoSQL databases

SQL Databases

- SQL Databases primarily mean Relational Databases
- Relational databases are predominantly around since
  - its proposal by EF Codd [1970]
  - IBM System R first implementation of SQL [1974]
- In recent decades, the enterprise computing saw many changes in terms of programming languages, architectures, platforms, and processes; however one thing remained same was "relational databases", till the time
- "No SQL" emerged (typically since 2009) as a major challenger.

Relational "Reign"

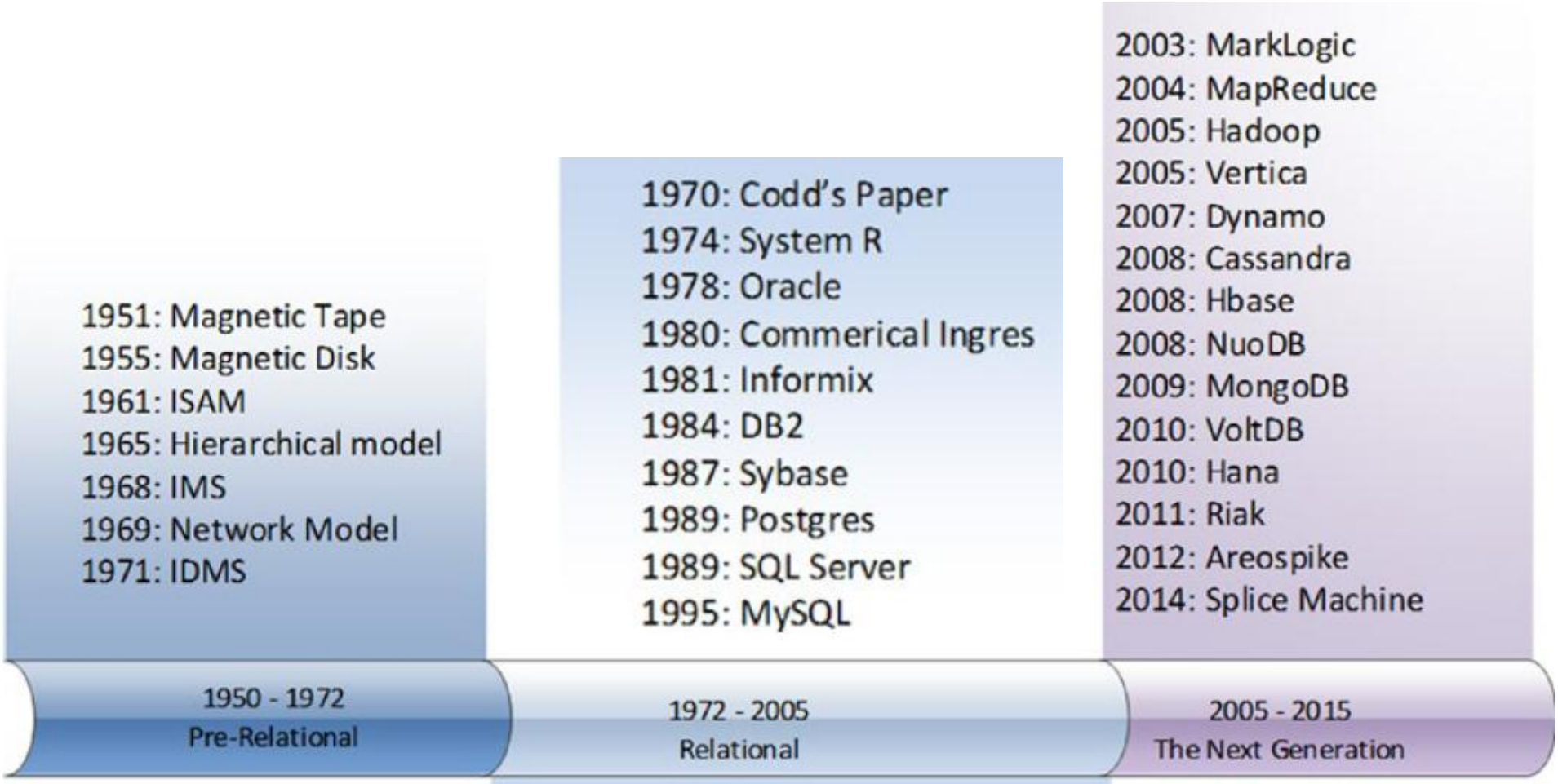
- What has been great about Relational databases that they ruled for decades?
  - Shared Access of Integrated data
  - Data Independence [Three Schema Architecture]
    - Metadata, logical abstraction, internal data independence
  - Standardization [ANSI SQL, ODBC, JDBC, etc]
  - Very robust Query Optimization
  - Concurrent Access of data [ACID compliance]
  - Recovery, Security etc.

NoSQL

- The term NoSQL first appeared in 2009 in name of meet "NoSQL Meet"
- Today, most commonly spelled as "Not Only SQL" – has some critic though
- NoSQL generally applied to a number of recent non-relational databases such as Cassandra, MongoDB, Neo4J, and Riak.
- Primary characteristics are
  - Run on clusters – can scale seamlessly well
  - Flexible Schema – can be even schema less
  - Easier to program

Database Technological Evolutions

- Timeline of major database releases and innovations [text 4]

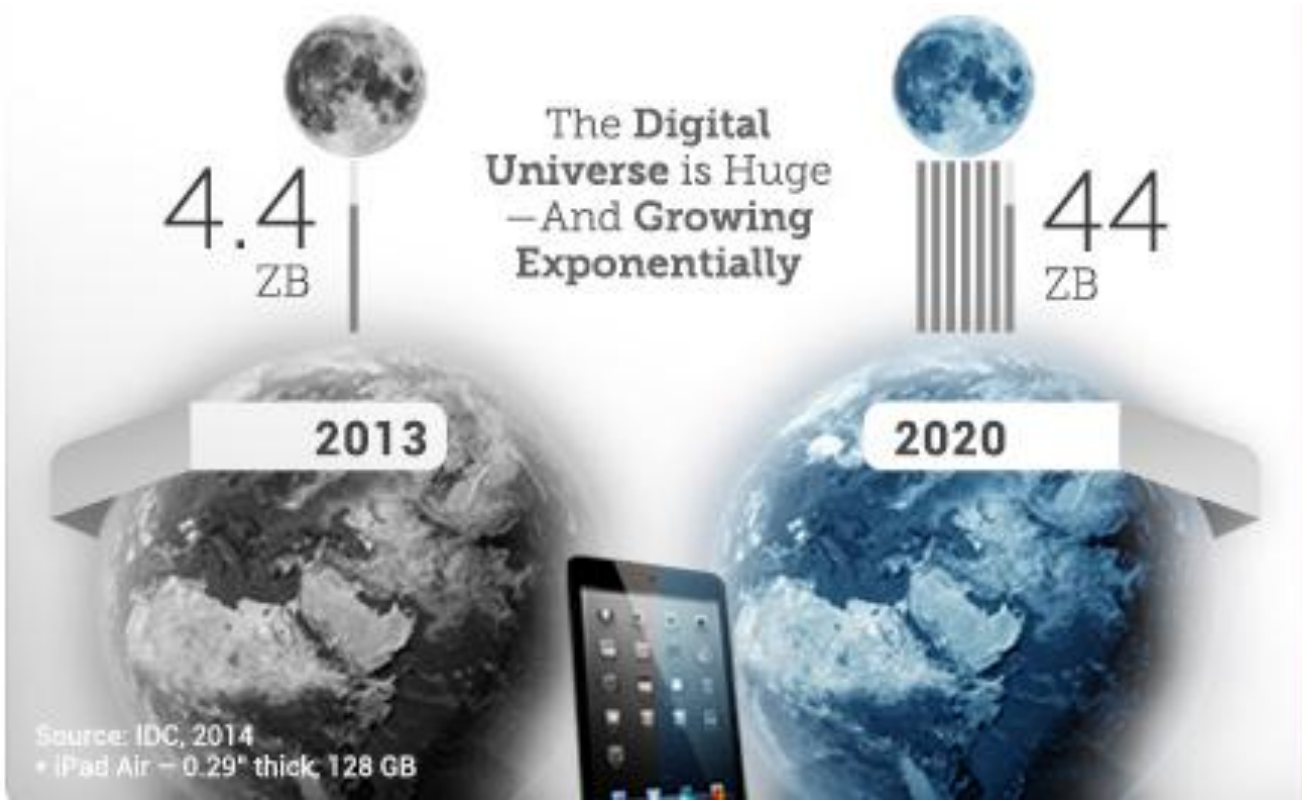


No SQL - Motivations

- Text 1 outlines following three main motivating reasons for the success of NoSQL databases -
  - Scalability – huge data; forcing to “Compute on Clusters”
  - Impedance Mismatch
  - Data Integration

RDBs are not designed for Scale[1]

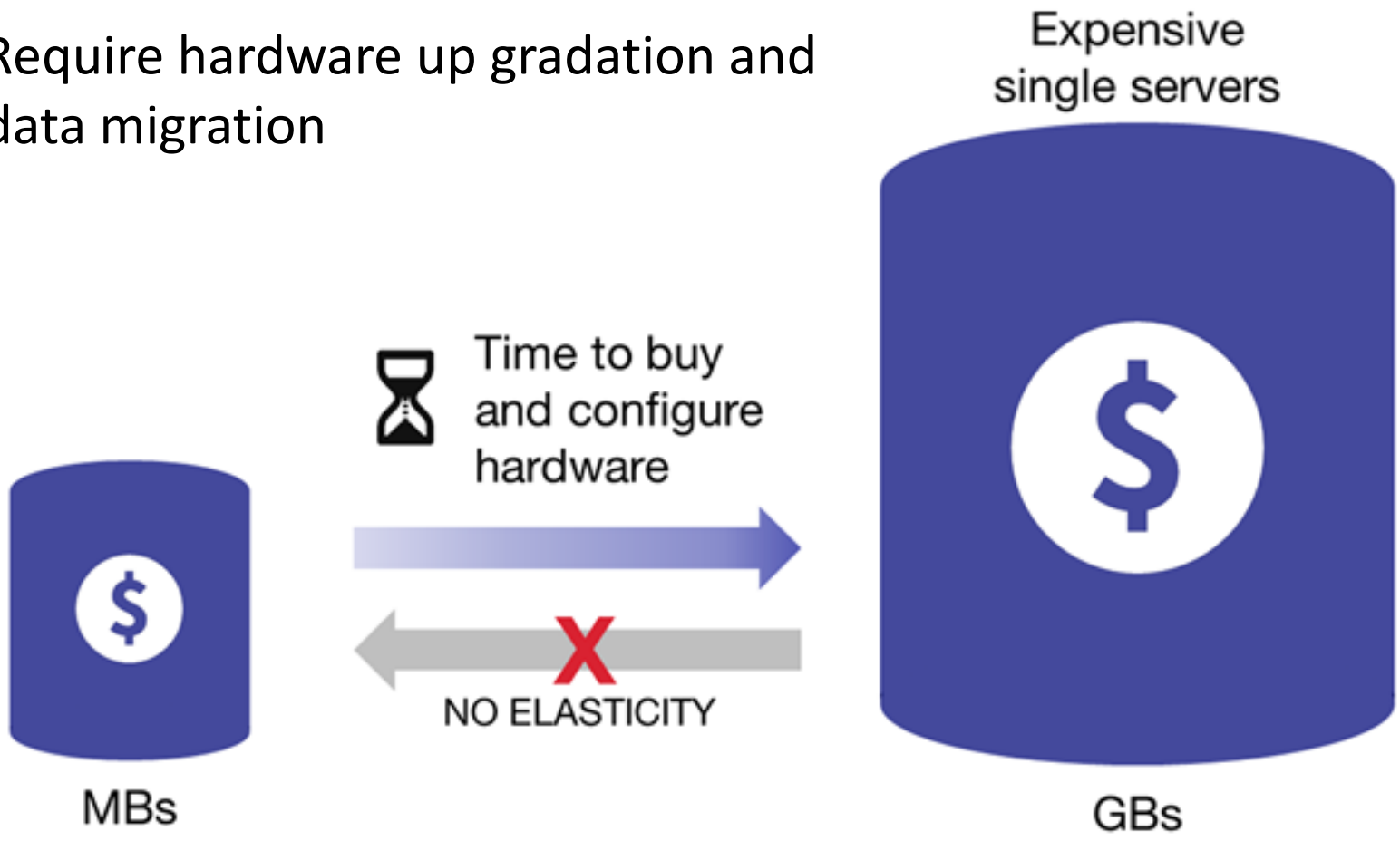
- Data to be stored and processed are seeing exponential growth[2] from 4.4ZB in 2013 to 44ZB in 2020



[1] https://www.marklogic.com/blog/relational-databases-scale/
[2] https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

RDBs are not designed for Scale[1]

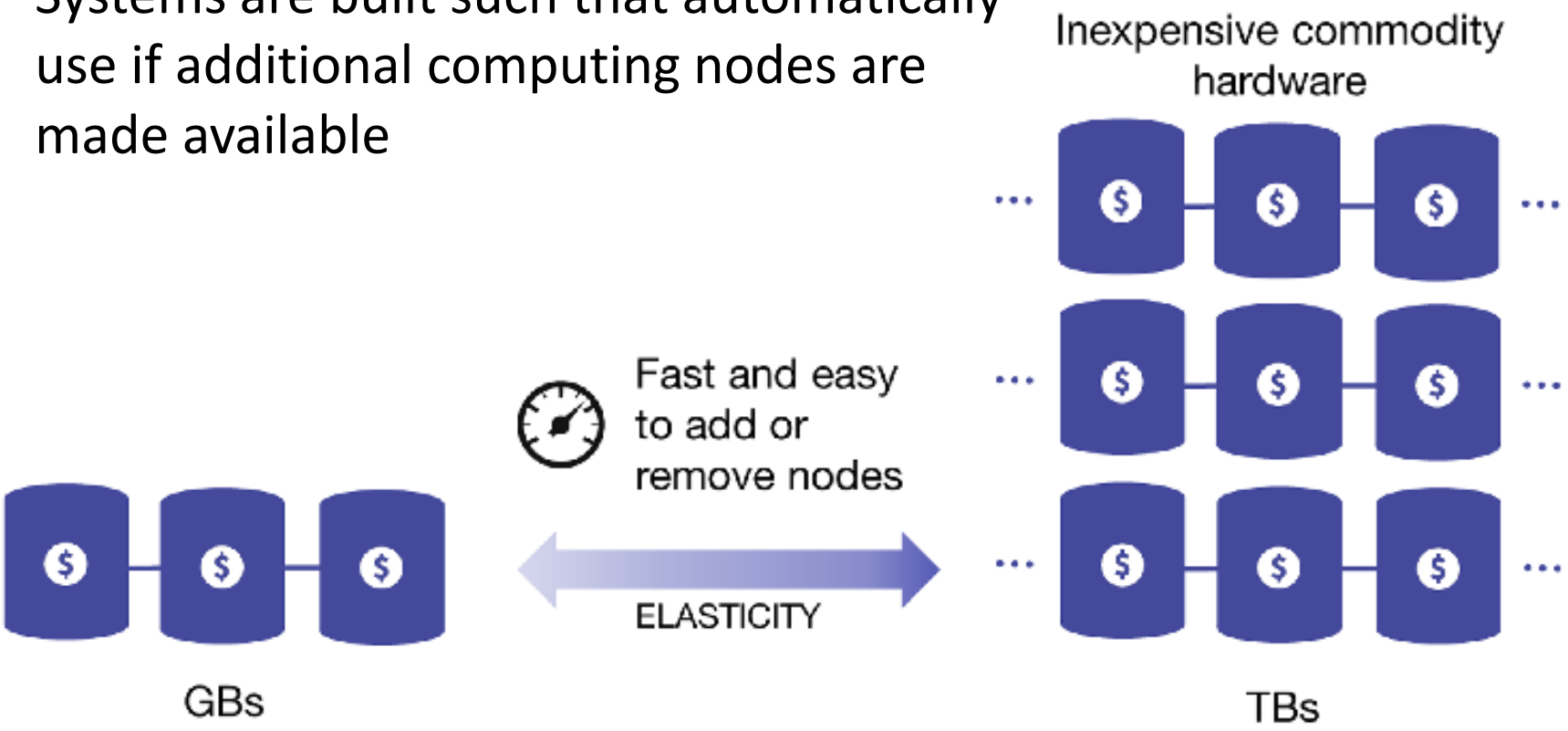
- Scaling Relational Databases Is Hard
- Require hardware up gradation and data migration



[1] https://www.marklogic.com/blog/relational-databases-scale/

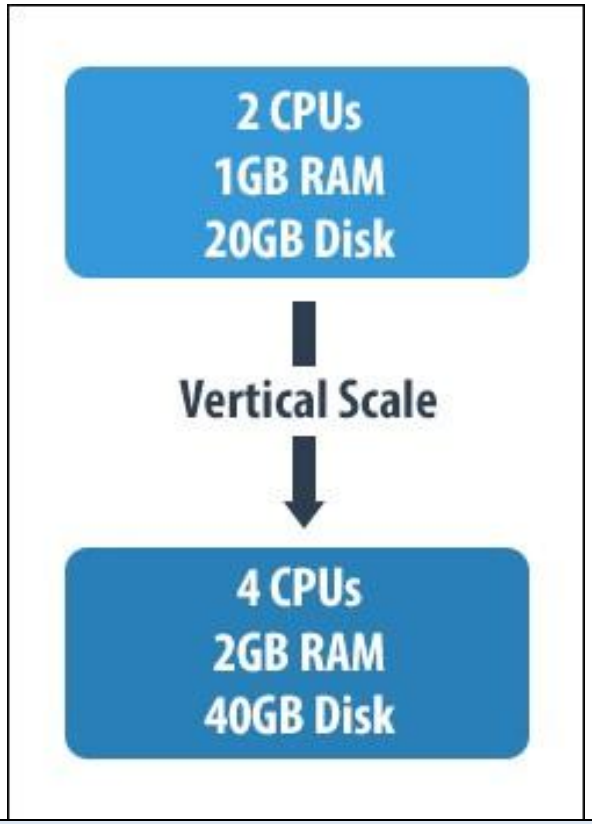
NoSQL databases are designed for Scale[1]

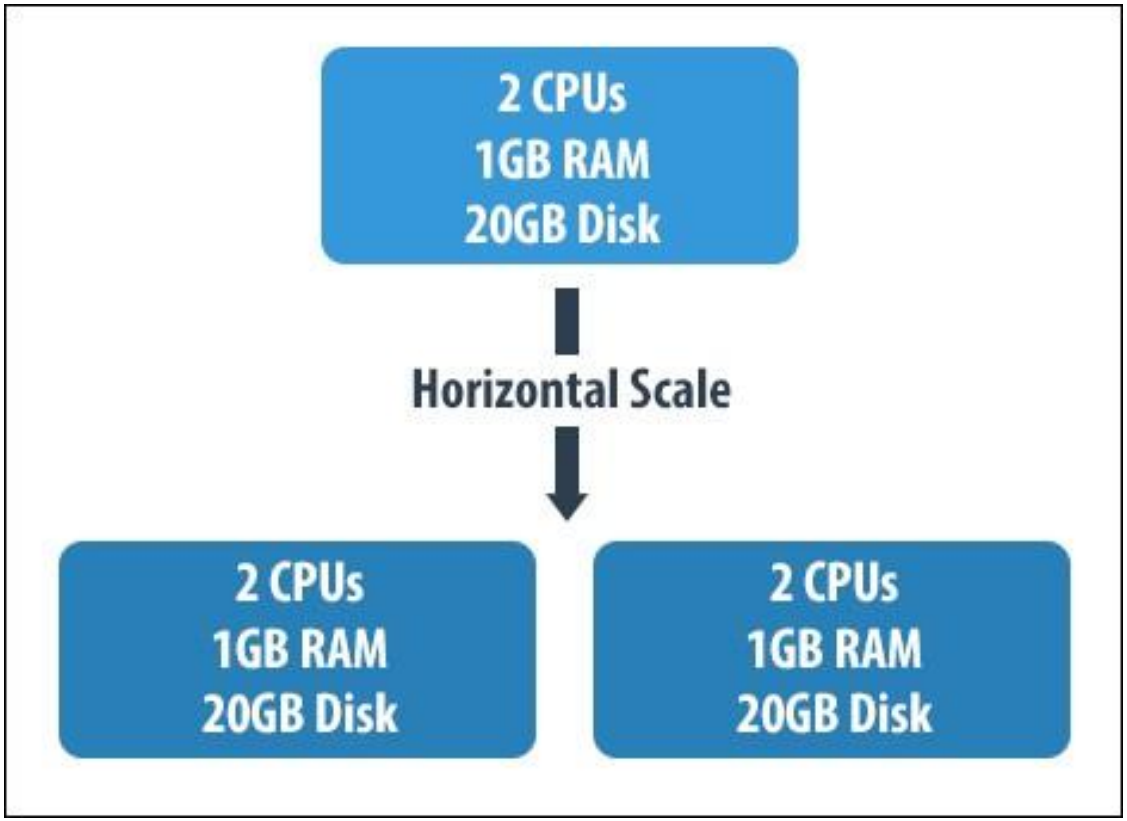
- Diagram below, Though taken from MarkLogic Server, captures the concept
- Systems are built such that automatically use if additional computing nodes are made available



[1] https://www.marklogic.com/blog/relational-databases-scale/

Vertical vs Horizontal Scale





https://hackernoon.com/database-scaling-horizontal-and-vertical-scaling-85edd2fd9944

Elasticity

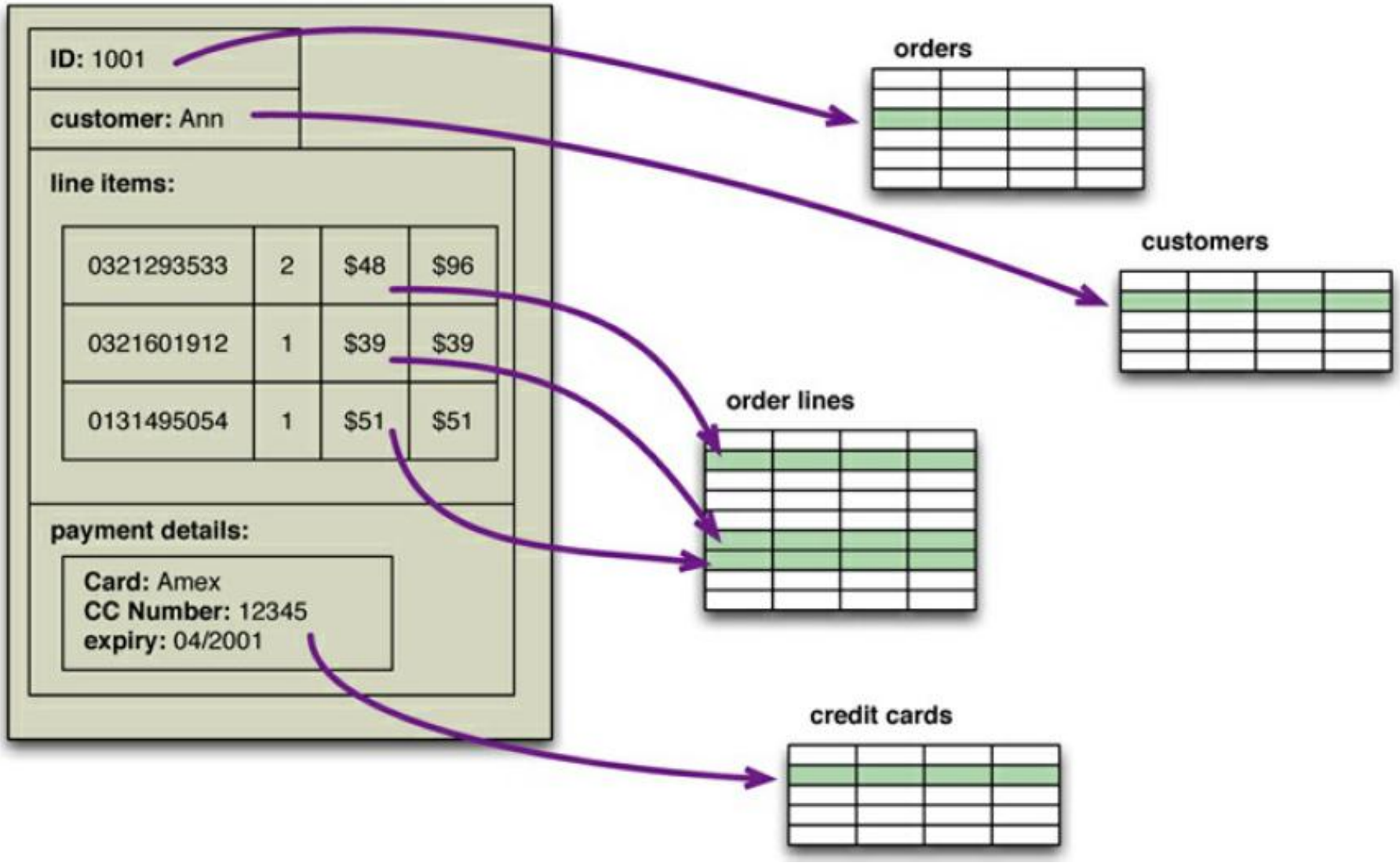
- Definition from [3]

  Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible

Impedance Mismatch

- While programming relational databases, biggest headache is to deal with structural differences of “memory data structures” and “database tables”.
- The problem is popularly known as “Impedance Mismatch”
- Figure next hopefully depicts the problem!
  - An order maps to tables: orders, orderlines, customers, credit cards
  - The task of saving an order involves translating order data into respective tuples, and so.
  - While reading the order, reverse is done!

Impedance Mismatch [text 1]



Impedance Mismatch

- People hoped that some thing like Object Oriented Databases would be replacing relational databases but it did not happen
  - Probability OODB could become success due to its own complexities, and could not provide many of RDB features
- Though most translations (Object to Relations and Relations to Objects) are automated by Object Relational Mapping (ORM) tools like Hibernate or so; they bring in their own complexity, maintenance and computational cost.
- A nice critic of ORM is available from Martin Flower at https://martinfowler.com/bliki/OrmHate.html

Data Integration

- "Integrated Databases" has been buzz word for relational databases in 70s!
- This basically advocates all data for an organization are stored as "single" and "integrated" database.
- This led to
  - database becoming complex!
  - Some sort of dependency among applications (A change in database structure for one application might crash another application, and so)

Data Integration

- Newer software architecture like SOA brings in a concepts of "Application Database" and data integration happens through web services (typically REST)
- Implementation of SOA applications requires lots (not exactly data volume but also data variety) of pulling of data and consuming them together.
- That is where various data integration formats like JSON, XML became (of course for other reasons)
- Performance of Relational databases through ORM and then bringing them to a common format like JSON becomes bottleneck!

The term “NoSQL”

- The term “NoSQL” was first used for a meet call “NoSQL Meetup” in 2009 in San Francisco organized by Johan Oskarsson
- By then products like BigTable and Dynamo were already attractions; and objective of meet was to share ideas on these paradigm shifting products!
- The call was for “**open-source, Distributed, non-relational databases**.”
- There were presentation in Meet from Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB, and MongoDB

How do we Spell "NoSQL"

- Do NoSQL databases not use SQL?
- Most commonly spelled as "Not Only SQL"
- Does it mean NoSQL systems include relational as well (if SQL is Relational)?
- But many NoSQL database systems do have some query language inspired from SQL for example Cassandra has CQL
- Any way "NoSQL" be just name; and define certain characteristics that a NoSQL typically has!

"NoSQL" database characteristics

- Let us list down characteristics of NoSQL Database Systems
  - Most NoSQL databases run on clusters
  - Do not implement ACID but have different notion of consistency [ensuring consistency is hard on clusters]
  - NoSQL systems do not include database systems that are Relational, and before!
  - NoSQL databases operate on either No Schema or has notion of Flexible Schema

Why Relational is hard on clusters!

- May not really hard, if we do not need
  - Referential Integrity
  - ACID properties
- Moreover “row based sharding” turn out to be inefficient for query execution or so
  - There is an attempt doing this is “YugaByte” - relational over clusters!

- Will see more insight later!

Is RDB getting phased out?

- Answer is definitely NO!
- RDBs are still going to be as the most common form of database in use [text 1]
- RDB still stands out as better option where
  - **Transaction databases** where ACID compliance is required and cluster distribution is not required
- Or we say other way round; actually there are only two loudly known reasons of choosing NoSQL databases are-
- (1) Huge Data Size
- (2) Ease of development (bypassing ORM and so, however there is a cost associated with, when we do this)

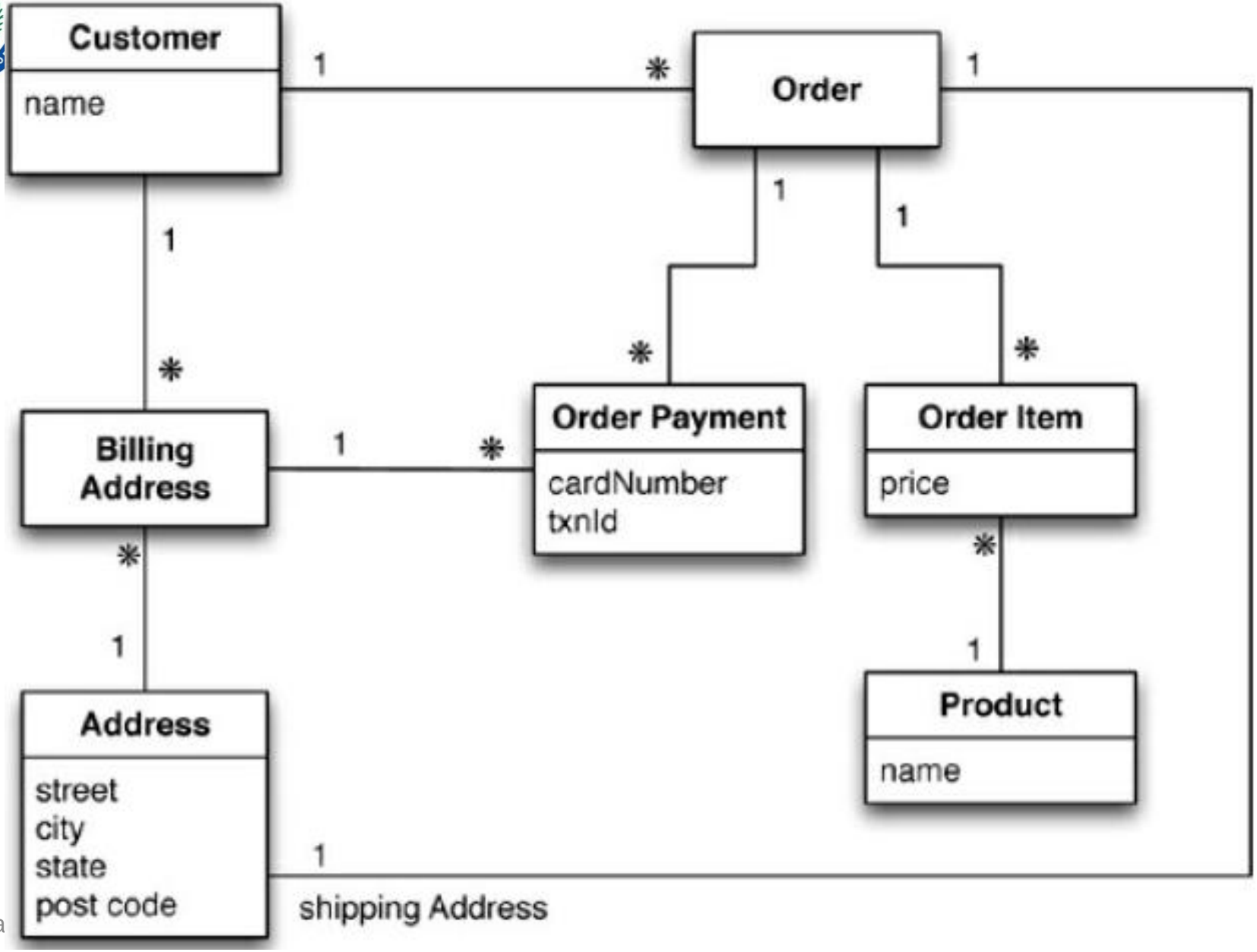
Genres of NoSQL

- "genre" mean here is "Database Type"
- Following are common NoSQL database types
  - Key value databases - Dynamo DB, Redis
  - Document Databases – CouchDB, MongoDB
  - Column Oriented databases- MonetDB, Cassandra]
  - Graph Databases- Neo4J
- Each database type is suited for certain "use cases"; let us see one here is "Data Aggregation" discussed in [text 1]

Data Aggregation

- In OOP world there is a relationship called HAS-A
  - ORDER HAS [MANY] ITEM
  - CUSTOMER HAS [MANY] BILLING ADDRESS
  - CUSTOMER HAS [MANY] ORDERS
- Aggregation is a speccial type of HAS-A where “containment” is involved
- Example: Consider a online ecommerce site that has following entities
  - Customer, Addresses, Order, Items, Order-Items, etc
- Let us see the relational design for this vis-à-vis other solution and feel some pros and cons of different database types!

Relational Solution in UML notation



Relational Solution

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

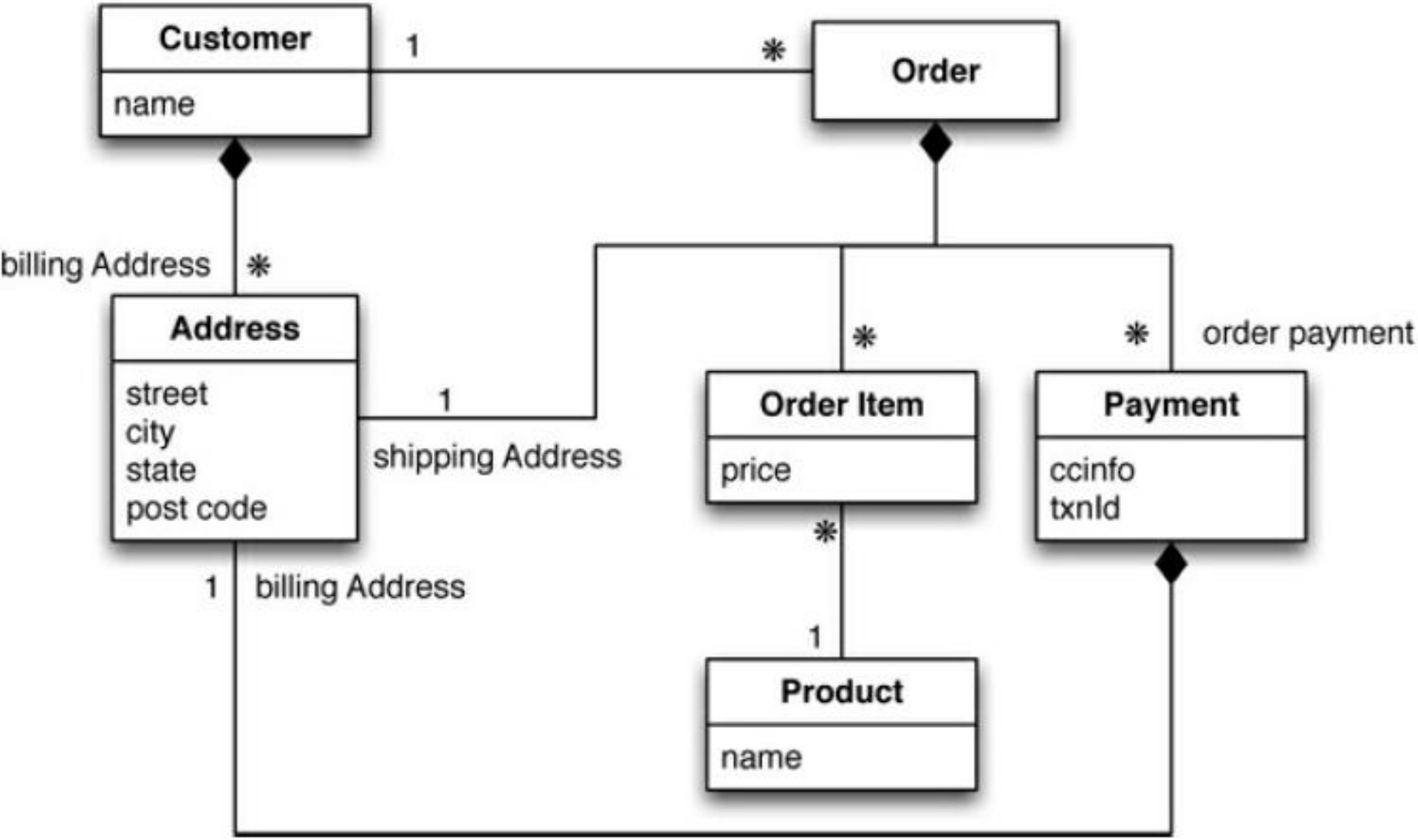
Address

Id	City
77	Chicago

OrderPayment

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

Typical in memory data model [class diagram]

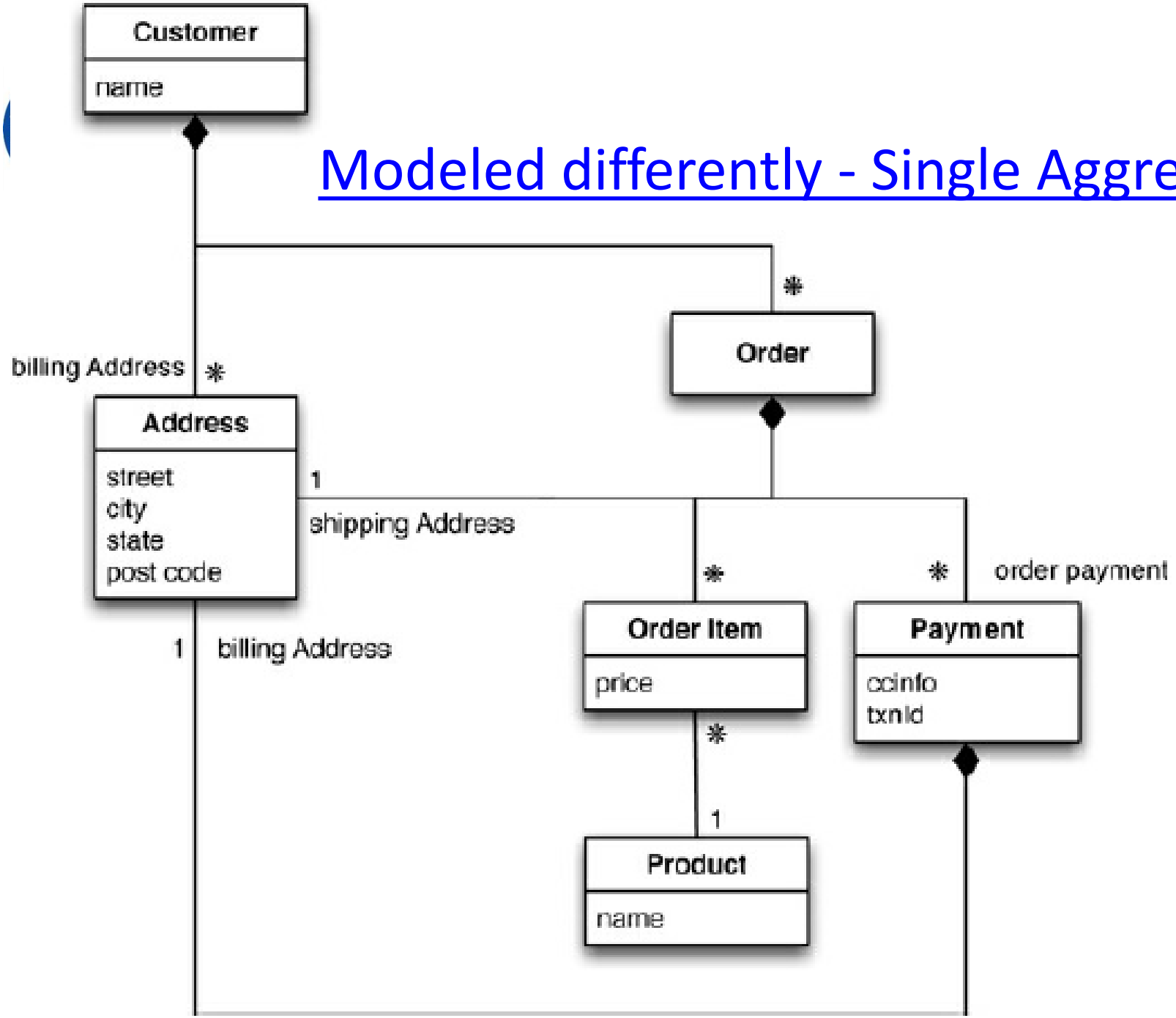


In JSON

```
//in customers
{
    "id":1,
    "name":"Martin",
    "billingAddress":[{"city":"Chicago"}]
}

// in 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"}
        }
    ],
}
```

Modeled differently - Single Aggregation



JSON – Single Aggregation

```
// 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"}
                }],
        }]
    }
}
```

"Aggregate Orientation"

- Let us call non relational solutions as "Aggregation Oriented"
- Data in memory are "Aggregation Oriented" where as relational representation is not!
- Consider a Customer object in memory, and
  - Find out where all data of customer in tables
  - How do we compute that
- Again note the "impedance mismatch"
- Which is good "Aggregate Oriented" or "Aggregate Ignorant" ?

Is “Aggregate Orientation” always good?

- Make sure you understood “Aggregate Orientation”!
- Aggregate Orientation is good for developers; it may not always be good for all sort of queries.
- Considering same example it might be great to find all orders of a customer but miserably bad when looking for sales summary over a period of time?

Key Value and Document Databases

- Key Value and Document Databases are highly "Aggregation Oriented" that's what makes these more program friendly than relational model
- Both are similar in the sense that both are key-value.
- Where Values are typically aggregated objects
- However both differs in terms of content of Key and Value, and particularly value (aggregate)

Key Value and Document Databases

- Difference can be seen in term of following two parameters
- Content of Value
  - Key Value: content of value is just binary block.
  - Document: content of value has to be in some structure like JSON objects
- Operations
  - Key Value: basically just GET and PUT on key; for example for CUSTID = 1234
  - Document: Queries can be specified on part of value; for example ORDER_DATE in between of some date range
- In practice, however, the line between key-value and document gets a bit blurry [text 1]

References/Further Readings

[1] Chapter 1 of book *NoSQL distilled*.

[2] Relational Databases Are Not Designed For Scale
https://www.marklogic.com/blog/relational-databases-scale/

[3] Herbst, Nikolas Roman, Samuel Kounev, and Ralf Reussner. "Elasticity in cloud computing: What it is, and what it is not." *Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13)*. 2013.

[4] History of Data Modeling:
http://graphdatamodeling.com/GraphDataModeling/History.html