
# INFO90002: Database Systems & Information Modelling

Farah Zaib Khan

Lecture 11

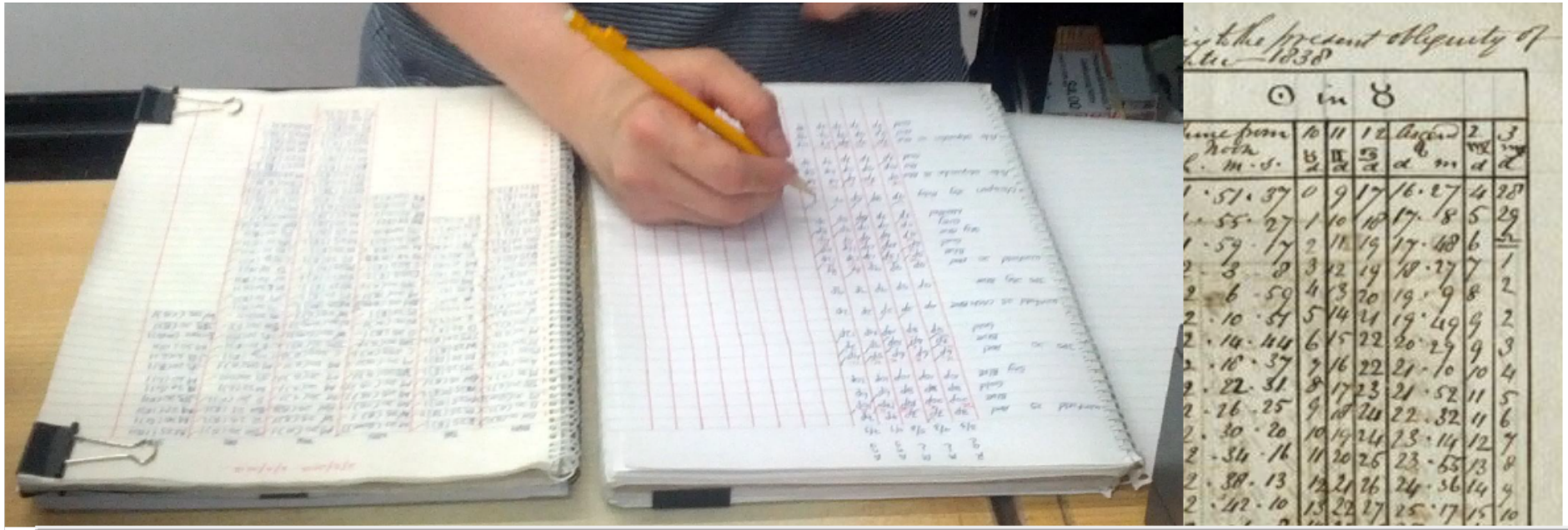
NoSQL Databases

# Learning Objectives

- By the end of this session, you should be able to:
  - Define what Big Data is
  - Describe why databases go beyond relational DBs
  - Understand why we need NoSQL
  - Types of NoSQL
  - MongoDB demo
  - CAP theorem
  - ACID vs BASE

* material in this lecture is drawn from http://martinfowler.com/books/nosql.html, including talk at GOTO conference 2012 and Thoughtworks article at https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Much of business data is tabular



| EMP_RECORD... | EMP_ID | EMP_REGION | EMP_DEPT | EMP_HIRE_D... | E... | E... | EMP_... | EMP_SALARY | EMP_NAME | EMP_SKIL |
|---|---|---|---|---|---|---|---|---|---|---|
| 68 | 3715 | 4 | 153 | 09061987 | 9 | 6 | 1987 | 14000000 | IRENE HIRSH | 041085 |
| 62 | 39412 | 1 | 650 | 03119590 | 3 | 11 | 9590 | 167000000 | ANN FAHEY | 031099 |
| 56 | 1939 | 2 | 265 | 09281988 | 9 | 28 | 1988 | 21300000 | EMILY WILM... | 021077 |
| 50 | 3502 | 2 | 165 | 07041985 | 7 | 4 | 1985 | 19500000 | CATHEZINE ... | 011015 |
| 44 | 4435 | 2 | 117 | 05141989 | 5 | 14 | 1989 | 17000000 | AGNES KING | 00 |
| 68 | 1673 | 3 | 138 | 07021985 | 7 | 2 | 1985 | 16800000 | MARTIN XU | 041033 |
| 62 | 4181 | 3 | 161 | 02031988 | 2 | 3 | 1988 | 15900000 | JOHN DURN | 030045 |
| 56 | 1443 | 1 | 265 | 12028900 | 12 | 2 | 8900 | 6000000 | PAT DUNN | 021055 |
| 50 | 3607 | 3 | 127 | 08072000 | 8 | 7 | 2000 | 18300000 | ANDREA HIN... | 011014 |
| 44 | 1775 | 3 | 288 | 02051989 | 2 | 5 | 1989 | 2700000 | PETER JONES | 00 |
| 68 | 1209 | 2 | 165 | 05121986 | 5 | 12 | 1986 | 17300000 | DIDRA WILK... | 041065 |

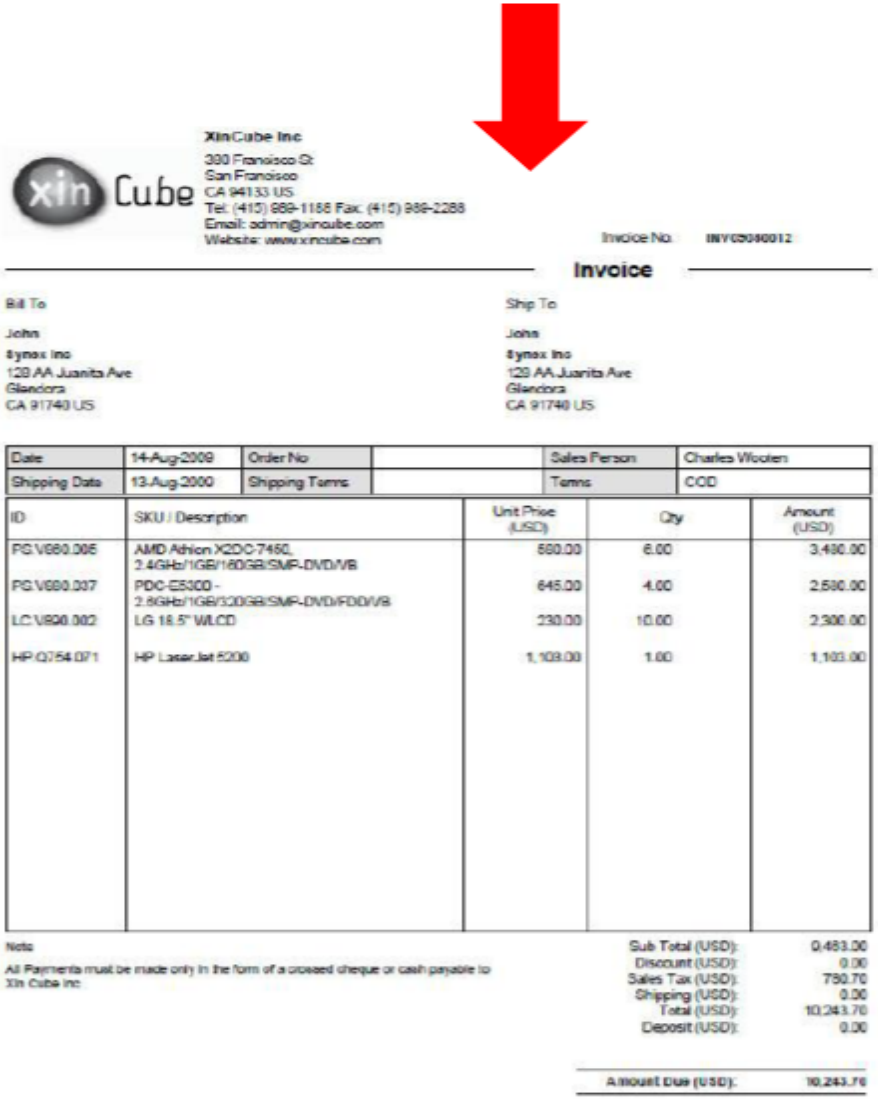
# The dominance of the relational model

- Pros of relational databases
  - simple, can capture (nearly) any business use case
  - can integrate multiple applications via shared data store
  - standard interface language SQL
  - ad-hoc queries, across and within "data aggregates"
  - fast, reliable, concurrent, consistent
- Cons of relational databases
  - Object Relational (OR) impedance mismatch
  - not good with big data
  - not good with clustered/replicated servers

- Adoption of NoSQL driven by “cons” of Relational
- but ‘polyglot persistence‘ = Relational will not go away

# Some data is not inherently tabular

One business object (in aggregate form) is stored across many relational tables.



XinCube Inc
300 Francisco St
San Francisco
CA 94133 US
Tel: (415) 989-1188 Fax: (415) 989-2288
Email: admin@xincube.com
Website: www.xincube.com

Invoice No: INV0000012

**Invoice**

Bill To
John
Synex Inc
123 AA Juanita Ave
Glendora
CA 91740 US

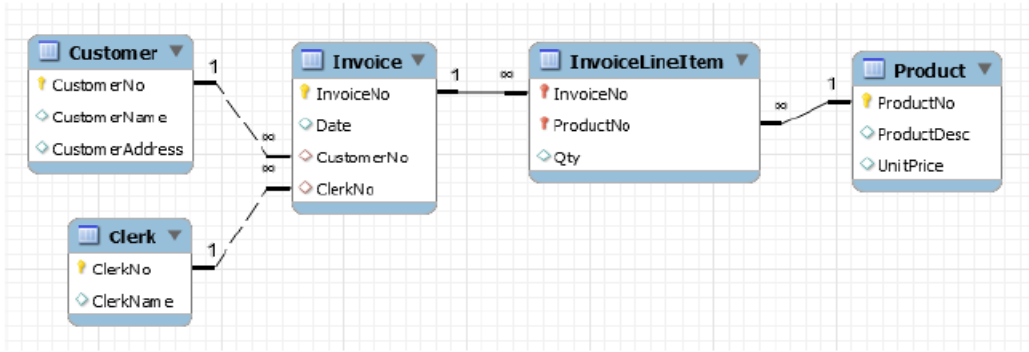
Ship To
John
Synex Inc
123 AA Juanita Ave
Glendora
CA 91740 US

| Date | 14-Aug-2009 | Order No | | Sales Person | Charles Wooten |
|---|---|---|---|---|---|
| Shipping Date | 13-Aug-2009 | Shipping Terms | | Terms | COD |

| ID | SKU / Description | Unit Price (USD) | Qty | Amount (USD) |
|---|---|---|---|---|
| PS:V880.005 | AMD Athlon X2DC-7450, 2.4GHz/1GB/160GB/SMP-DVD/VB | 580.00 | 6.00 | 3,480.00 |
| PS:V880.007 | PDC-E5300 - 2.6GHz/1GB/320GB/SMP-DVD/FDD/VB | 645.00 | 4.00 | 2,580.00 |
| LC:V890.002 | LG 18.5" WLCD | 230.00 | 10.00 | 2,300.00 |
| HP:Q754.071 | HP LaserJet 5200 | 1,103.00 | 1.00 | 1,103.00 |

Note
All Payments must be made only in the form of a crossed cheque or cash payable to Xin Cube Inc

| | |
|---|---|
| Sub Total (USD): | 9,463.00 |
| Discount (USD): | 0.00 |
| Sales Tax (USD): | 780.70 |
| Shipping (USD): | 0.00 |
| Total (USD): | 10,243.70 |
| Deposit (USD): | 0.00 |
| Amount Due (USD): | 10,243.70 |

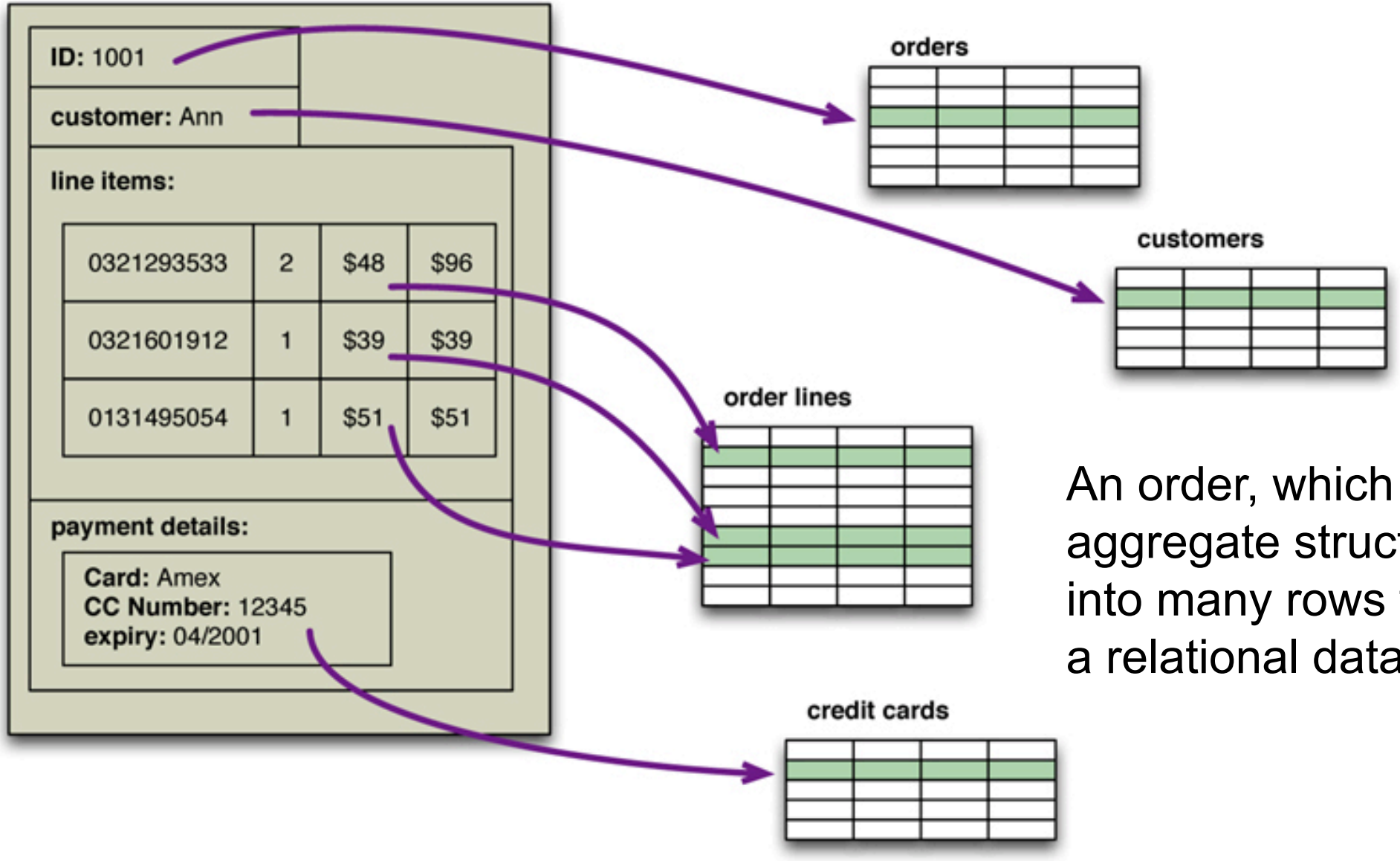



This enables analytical queries like:

```
select productno, sum(qty)
from InvoiceLineItem
group by productno;
```

But there is a lot of work to dissemble and reassemble the aggregate.

# Aggregates




An order, which looks like a single aggregate structure in the UI, is split into many rows from many tables in a relational database

Data in Aggregate form: Examples of JSON and XML

**JSON Example**

```
{"products": [
        {"number": 1, "name": "Zoom X", "Price": 10.00},
        {"number": 2, "name": "Wheel Z", "Price": 7.50},
        {"number": 3, "name": "Spring 10", "Price": 12.75}
]}
```

**XML Example**

```
<products>
        <product>
                <number>1</number> <name>Zoom X</name> <price>10.00</price>
        </product>
        <product>
                <number>2</number> <name>Wheel Z</name> <price>7.50</price>
        </product>
        <product>
                <number>3</number> <name>Spring 10</name> <price>12.75</price>
        </product>
</products>
```

# Big Data and its 3Vs

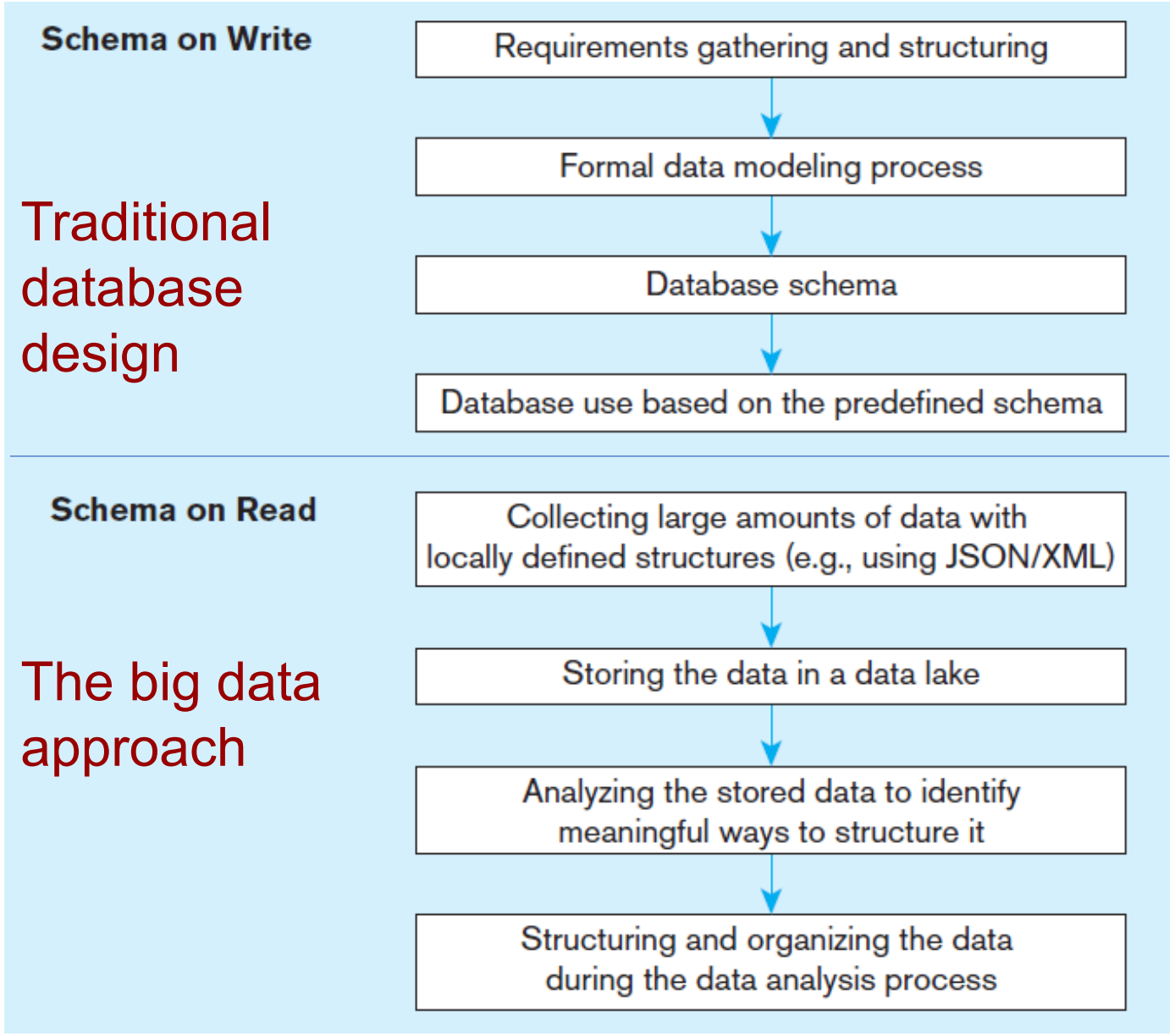
- Data that exist in very large volumes and many different varieties (data types) and that need to be processed at a very high velocity (speed).
  - **Volume** – much larger quantity of data than typical for relational databases
  - **Variety** – lots of different data types and formats
  - **Velocity** – data comes at very fast rate (e.g. mobile sensors, web click stream)

# Big Data Characteristics

- **Schema on Read, rather than Schema on Write**
  - Schema on Write – preexisting data model, how traditional databases are designed (relational databases)
  - Schema on Read – data model determined later, depends on how you want to use it (XML, JSON)
    - Capture and store the data, and worry about how you want to use it later

- **Data Lake**
  - A large integrated repository for internal and external data that does not follow a predefined schema
  - Capture everything, dive in anywhere, flexible access

# Schema on write vs. schema on read



# NoSQL database properties

## Features

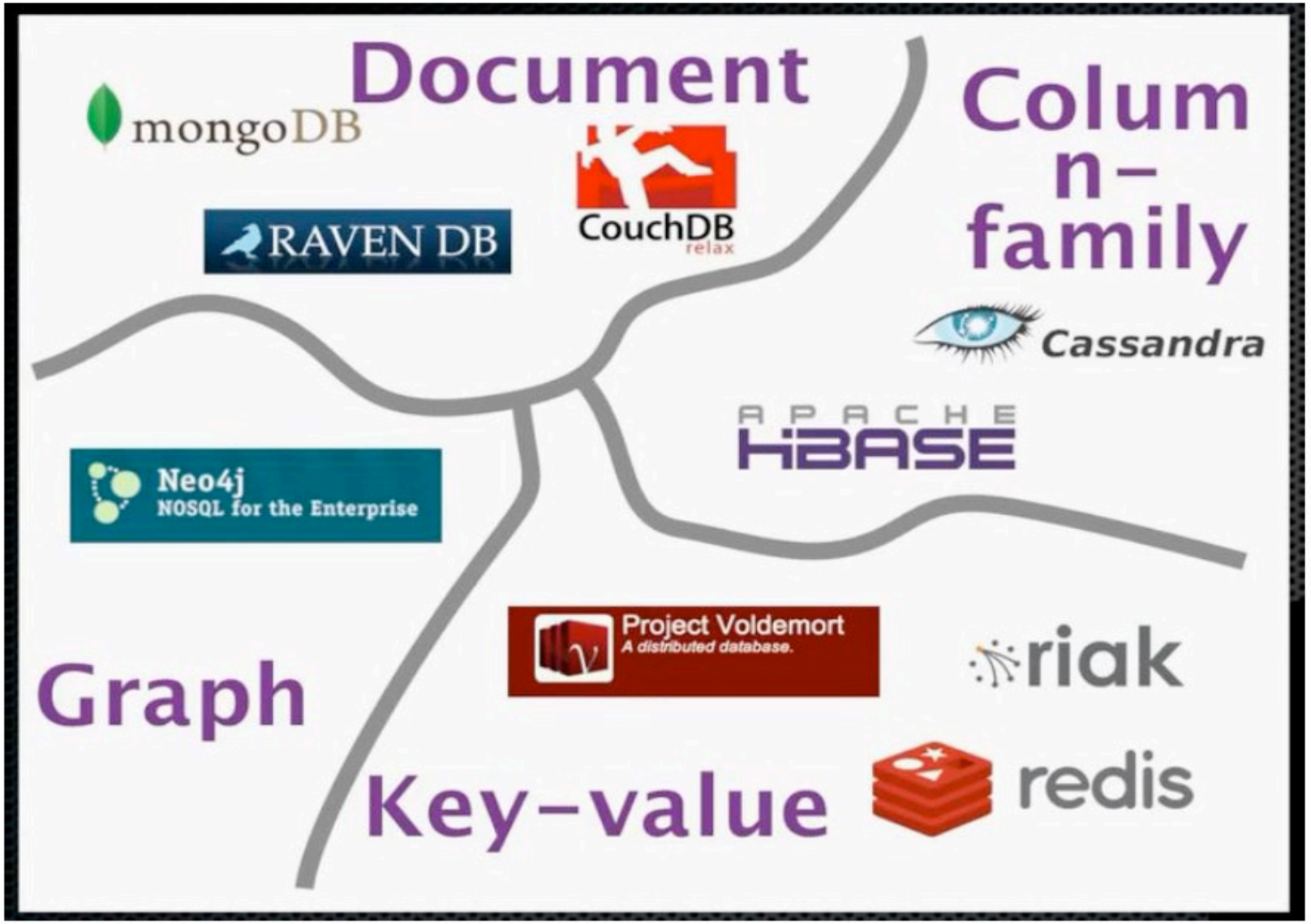
- Doesn’t use relational model or SQL language
- Runs well on distributed servers
- Most are open-source
- Built for the modern web
- Schema-less (though there may be an "**implicit schema**")
- Supports schema on read
- Not ACID compliant – details later in the lecture
- ‘Eventually consistent’ – details later in the lecture

## Goals – Main Drivers

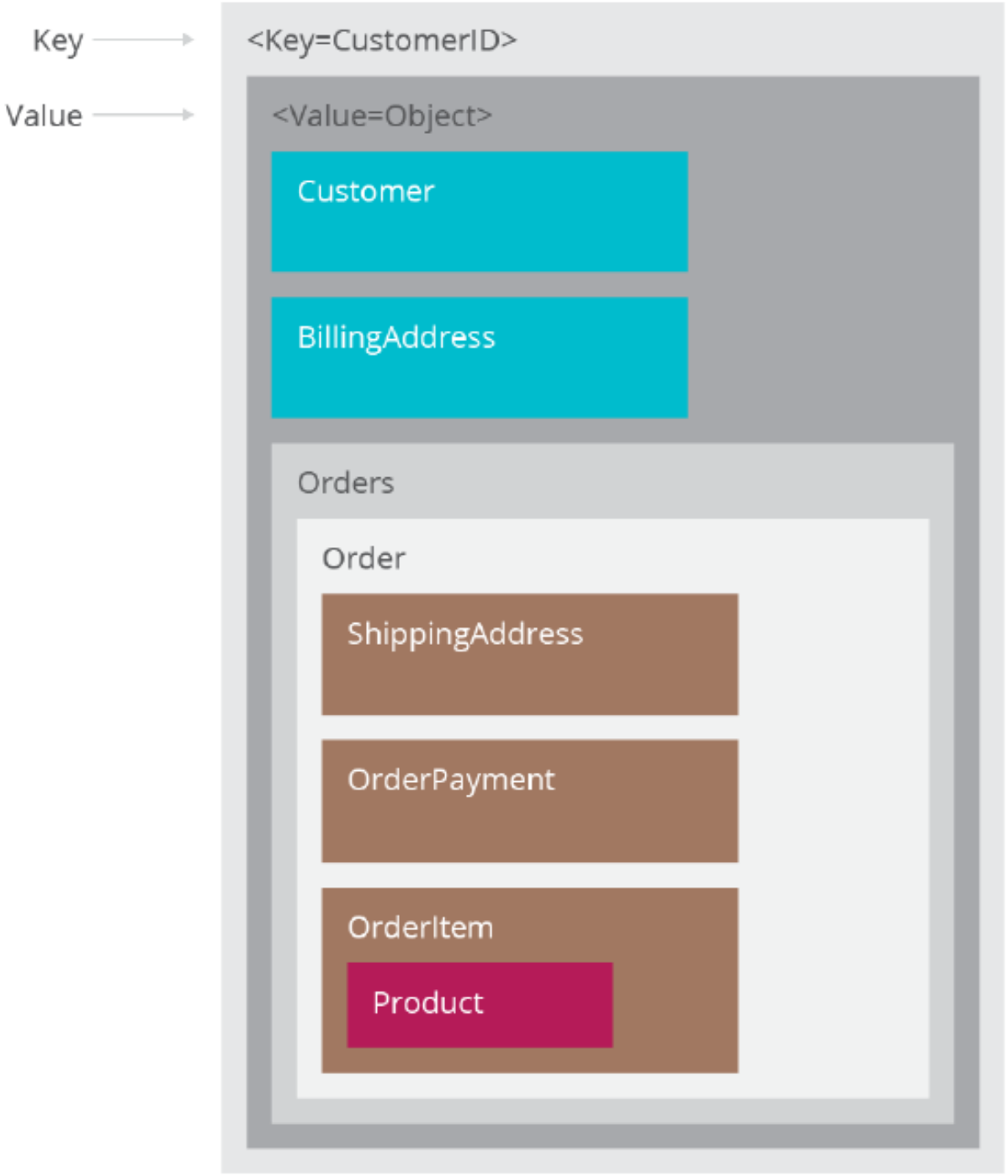
- to improve programmer productivity (OR mismatch)
- to handle larger data volumes and throughput (big data)

from NoSQL Databases: An Overview by Pramod Sadalage, Thoughtworks(2014)

# Types of NoSQL databases




(diagram from Martin Fowler)




# Types of NoSQL: key-value stores

- Key = primary key
- Value = anything (number, array, image, JSON)
  - *–the application is in charge of interpreting what it means*
- Operations: *Put* (for storing), *Get* and *Update*



levelDB










Key-value store representatives

Ranked list: http://db-engines.com/en/ranking/key-value+store

# K-V Stores: Suitable Use Cases

**Storing Web Session Information**

- Every web session is assigned a unique session_id value
- Fast, everything is stored in a single object

**User Profiles, Preferences**

- Every user has a unique user_id/user_name + preferences (language, time zone, design, access rights, … )
- As in the previous case: Fast, single object, single GET/PUT

**Shopping Cart Data**

- Similar to the previous cases

# K-V Stores: When Not to Use

## Relationships among Data

- Relationships between different sets of data
  - Some key-value stores provide link-walking features

## Multi-operation Transactions

- Saving multiple keys

  Failure to save any of them → revert or roll back the rest of the operations

## Query by Data

- Search the keys based on something found in the value part
  - Additional indexes needed (some stores provide them)

# Types of NoSQL: document databases

- Basic concept of data: ***Document***
- Documents are self-describing pieces of data
  - Hierarchical tree data structures
  - Nested associative arrays (maps), collections, scalars
  - XML, JSON (JavaScript Object Notation), BSON, …
- Documents in a collection should be “similar”
  - Their schema can differ
- Often: Documents stored as values of key-value
  - Key-value stores where the values are examinable

<Key=CustomerID>

```
{
    "customerid": "fc986e48ca6"
    "customer":
    {
    "firstname": "Pramod",
    "lastname": "Sadalage",
    "company": "ThoughtWorks",
    "likes": [ "Biking","Photography" ]
    }
    "billingaddress":
    { "state": "AK",
       "city": "DILLINGHAM",
       "type": "R"
     }
}
```

# Document Databases: Representatives





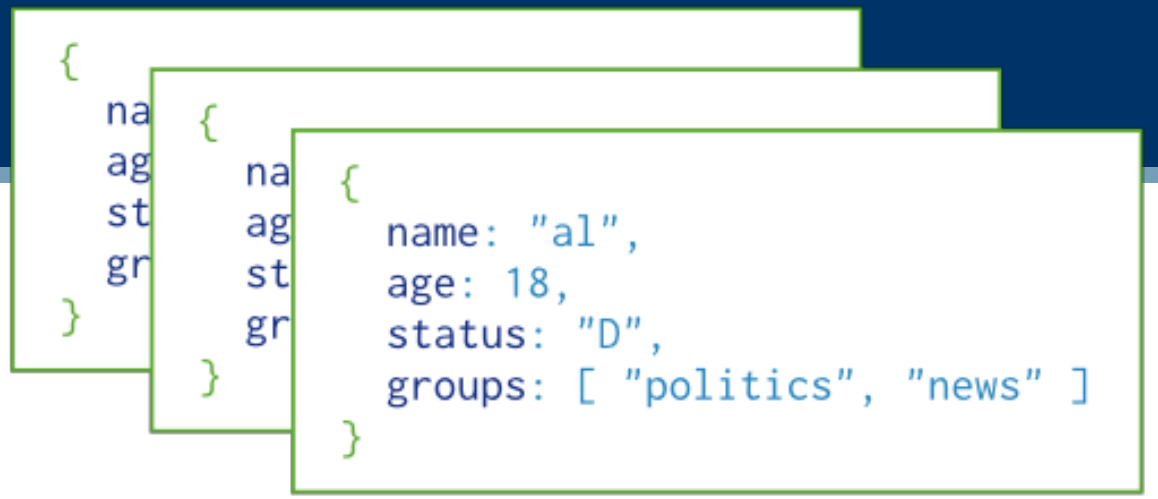



**MS Azure DocumentDB**

Ranked list: http://db-engines.com/en/ranking/document+store

# MongoDB Terminology

- each JSON document:
  - belongs to a collection
  - has a field _id
    - unique within the collection
- each collection:
- belongs to a “database”

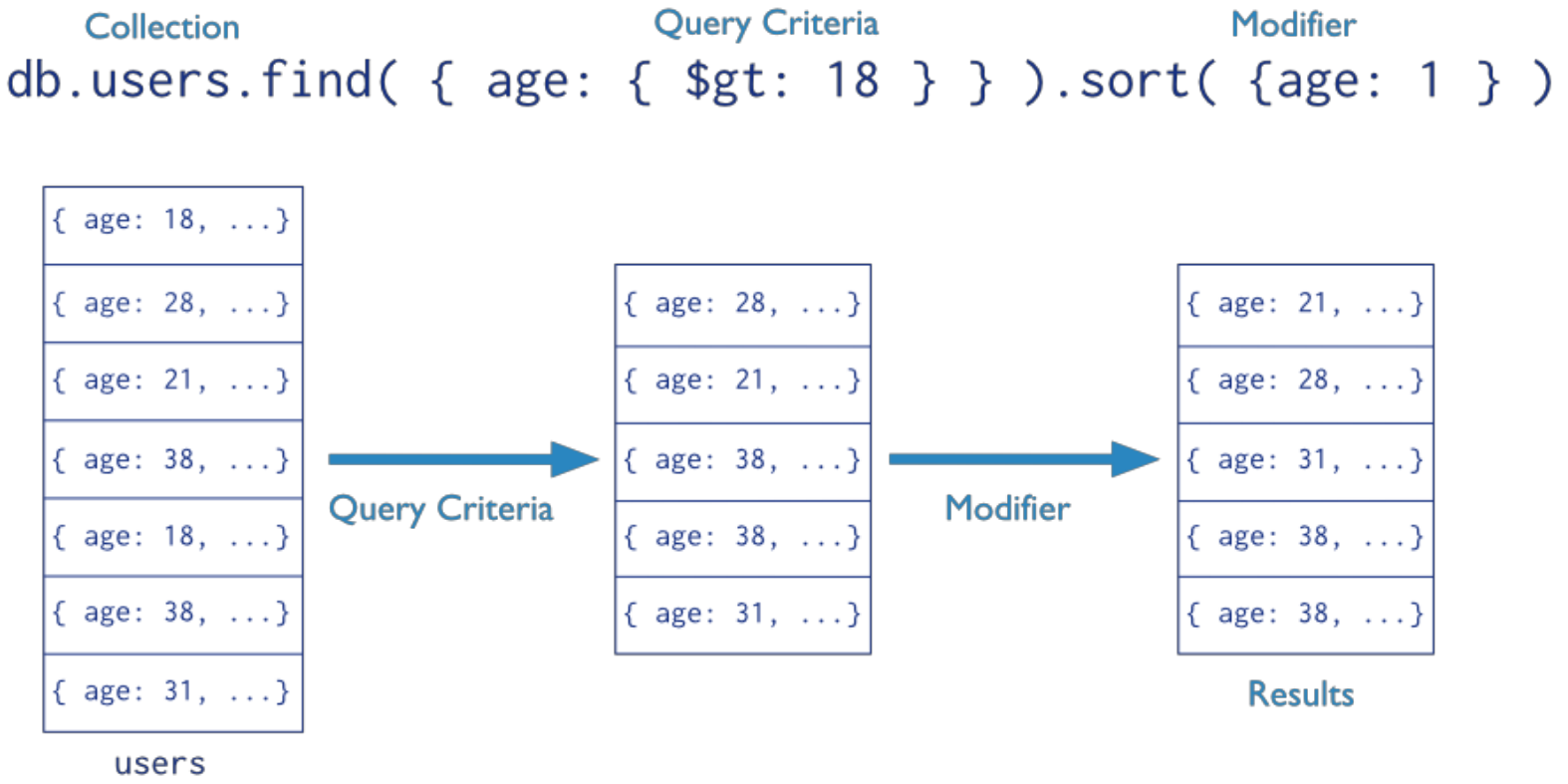



Collection

| RDBMS | MongoDB |
|---|---|
| database instance | MongoDB instance |
| schema | database |
| table | collection |
| row | document |
| rowid | _id |

# Querying MongoDB

- Mongo query language
- A MongoDB query:
  - Targets a specific collection of documents
  - Specifies criteria that identify the returned documents
  - May include a projection to specify returned fields
- Basic query - all documents in the collection:
  - **db.users.find()**
  - **db.users.find( {} )**



# MongoDB Querying: Selection

```
db.inventory.find({ type: "snacks" })
```

- All documents from collection inventory where the type field has the value snacks

```
db.inventory.find({ type: { $in: [ 'food', 'snacks' ] } } )
```

- All inventory docs where the type field is either food or snacks

```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

- All docs where the type field is  food and the price is less than 9.95

# MongoDB Inserts

```
db.inventory.insert( { _id: 10, type: "misc", item: "card", qty: 15 } )
```

- Inserts a document with three fields into collection inventory
- User-specified _id field

```
db.inventory.insert( { type: "book", item: "journal" } )
```

- The database generates _id field

```
$ db.inventory.find()
```

- { "_id": ObjectId("58e209ecb3e168f1d3915300"), type: "book", item: "journal" }

# MongoDB Updates

```
db.inventory.update({ type: "book", item : "journal" },
                    { $set: { qty: 10 } },
                    { upsert: true } )
```

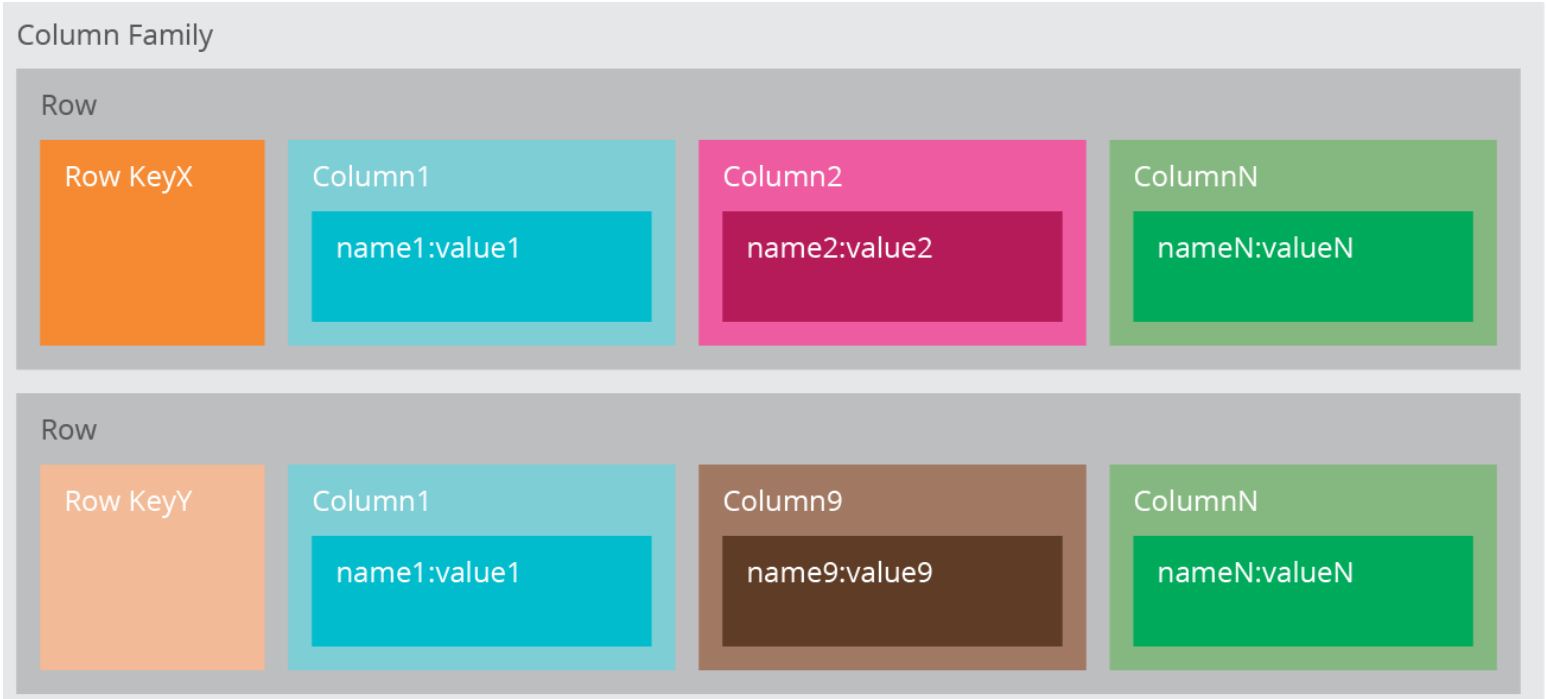
- Finds all docs matching query { type: "book", item : "journal" } and sets the field { qty: 10 }

**upsert: true**

- if no document in the inventory collection matches
- creates a new document (generated _id)
- it contains fields _id, type, item, qty

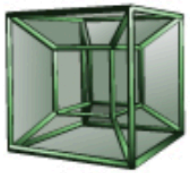
# Types of NoSQL: column families

- Columns rather than rows are stored together on disk.
- Makes analysis faster, as less data is fetched.
- This is like automatic vertical partitioning.
- Related columns grouped together into ‘families’.




https://www.youtube.com/watch?v=8KGVFB3kVHQ



HYPERTABLE


# Column families representatives

- Ranked list: http://db-engines.com/en/ranking/wide+column+store

# Column families

## Column-family stores

- are worth only for large data and large query throughput
- two ways to see the data model:
  - large sparse tables or multidimensional (nested) maps
- data distribution is via row key
  - analogue of document ID or key in document or key-value stores

## Cassandra

- CQL: structured after SQL, easy transition from RDBMS

# Aggregate-oriented databases

- *Key-value*, *document store* and *column-family* are “aggregate-oriented- store business object in its entirety” databases (in Fowler’s terminology)

- **Pros:**
  - *entire* aggregate of data is stored together (no need for transactions)
  - efficient storage on clusters / distributed databases

- **Cons:**
  - hard to analyse across subfields of aggregates
  - e.g. sum over products instead of orders

# Types of NoSQL: graph databases

- A ‘graph' is a node-and-arc network
- Social graphs (e.g. friendship graphs) are common examples
- Graphs are difficult to program in relational DB
- A *graph DB* stores entities and their relationships

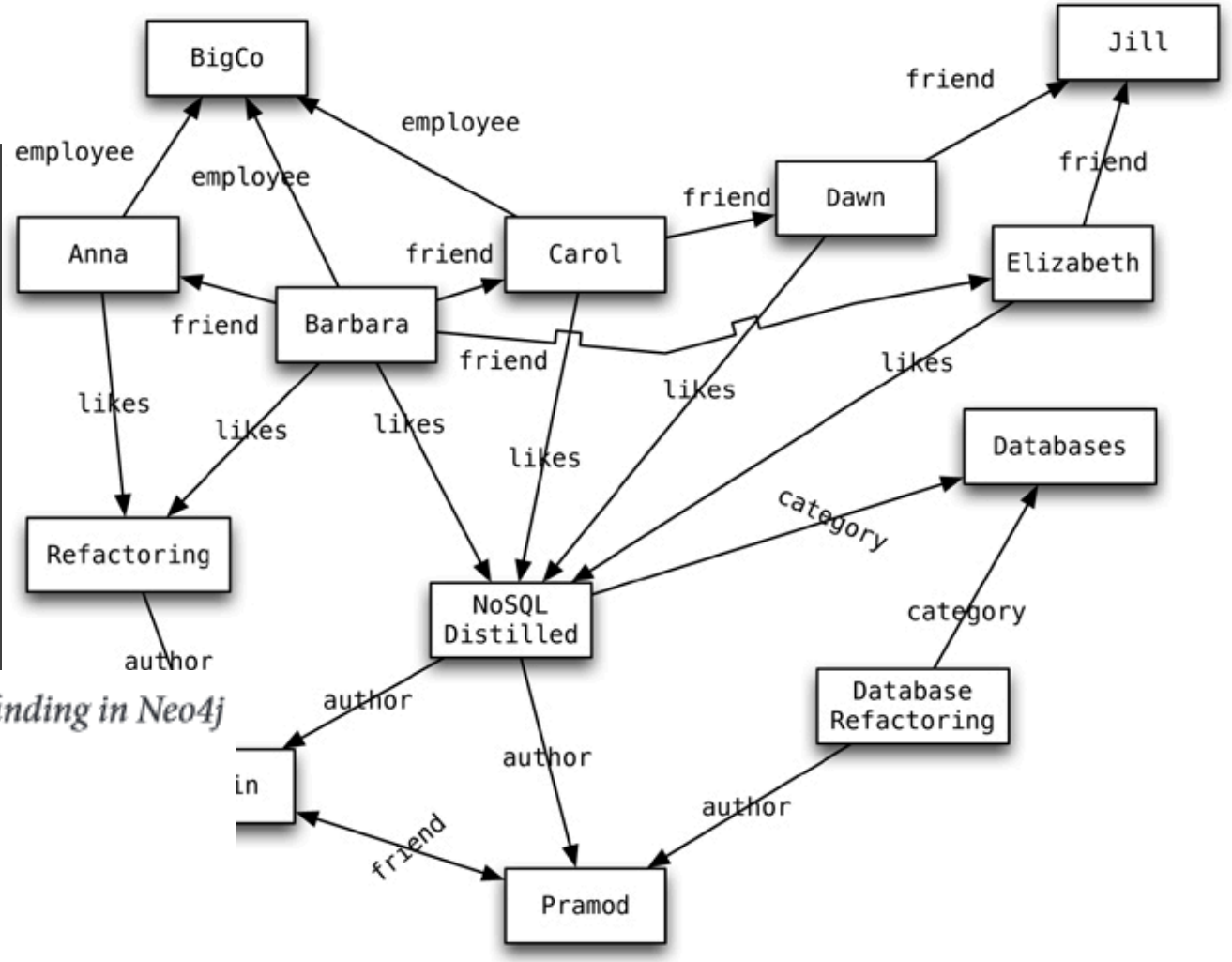



*Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j*

| Depth | RDBMS execution time(s) | Neo4j execution time(s) | Records returned |
|---|---|---|---|
| 2 | 0.016 | 0.01 | ~2500 |
| 3 | 30.267 | 0.168 | ~110,000 |
| 4 | 1543.505 | 1.359 | ~600,000 |
| 5 | Unfinished | 2.132 | ~800,000 |

# Graph databases: representatives






Ranked list: http://db-engines.com/en/ranking/graph+dbms

# Types of graphs

## Single-relational graphs

- Edges are homogeneous in meaning
  - e.g., all edges represent friendship

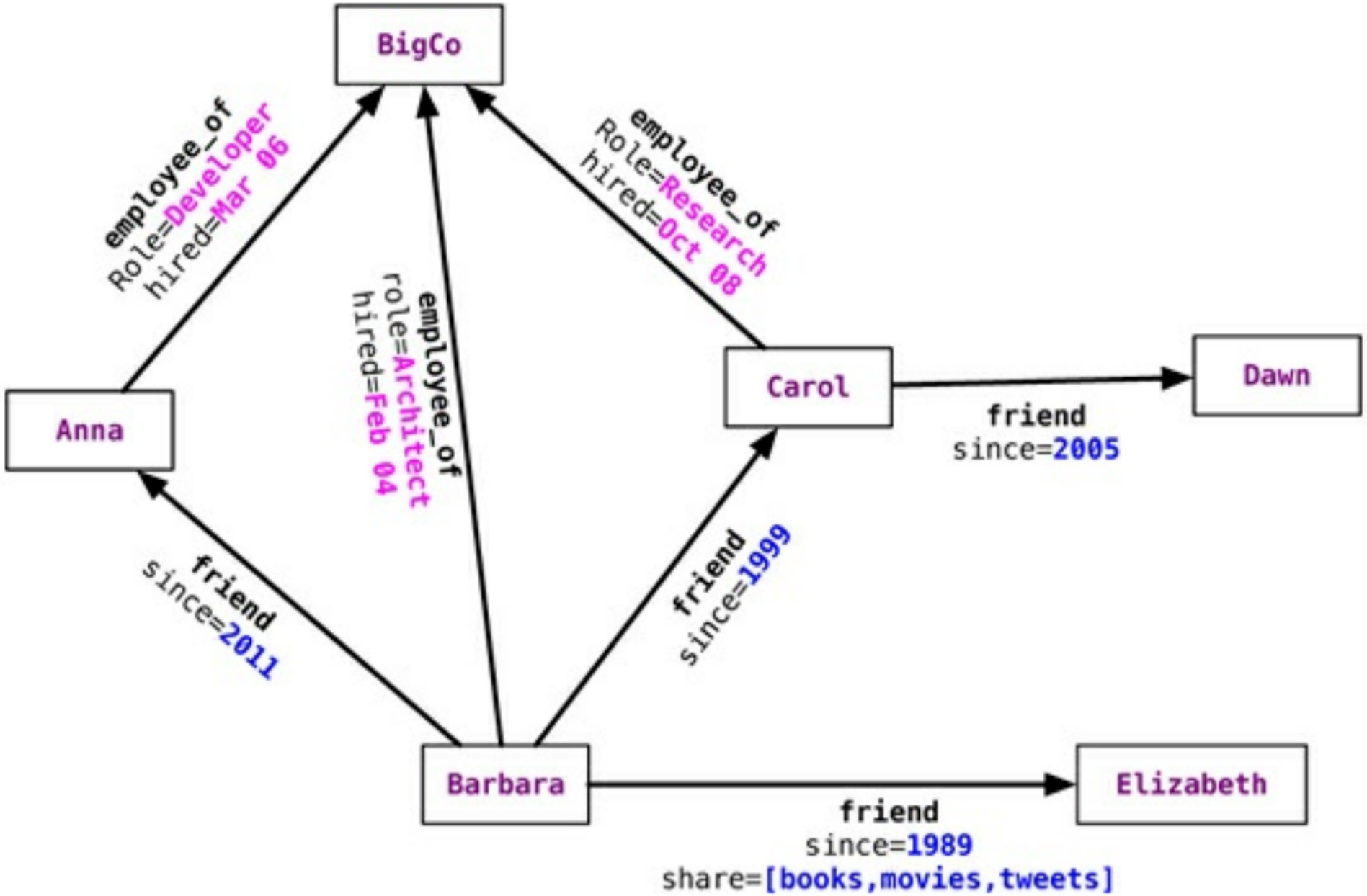
## Multi-relational (property) graphs

- Edges are typed or labelled
  - e.g., friendship, business, communication
- Vertices and edges maintain a set of key/value pairs
  - Representation of non-graphical data (properties)
  - e.g., name of a vertex, the weight of an edge

# Graph databases: Relationships

- Different types of relationships between nodes
  - To represent relationships between domain entities
- No limit to the number and kind of relationships
- Relationships have: type, start node, end node, own properties
  - e.g., “since when” did they become friends

# Graph databases: Relationship properties



# Summary: NoSQL Classifications

## Key-value stores

- A simple pair of a key and an associated collection of values. Key is usually a string. The database has no knowledge of the structure or meaning of the values.

## Document stores

- Like a key-value store, but “document” goes further than “value”. The document is structured, so specific elements can be manipulated separately.

## Column-family stores

- Data is grouped in “column groups/families” for efficiency reasons.

## Graph-oriented databases

- Maintain information regarding the relationships between data items. Nodes with properties.

# Demo

mongoDB

Data files are uploaded on LMS “Resources

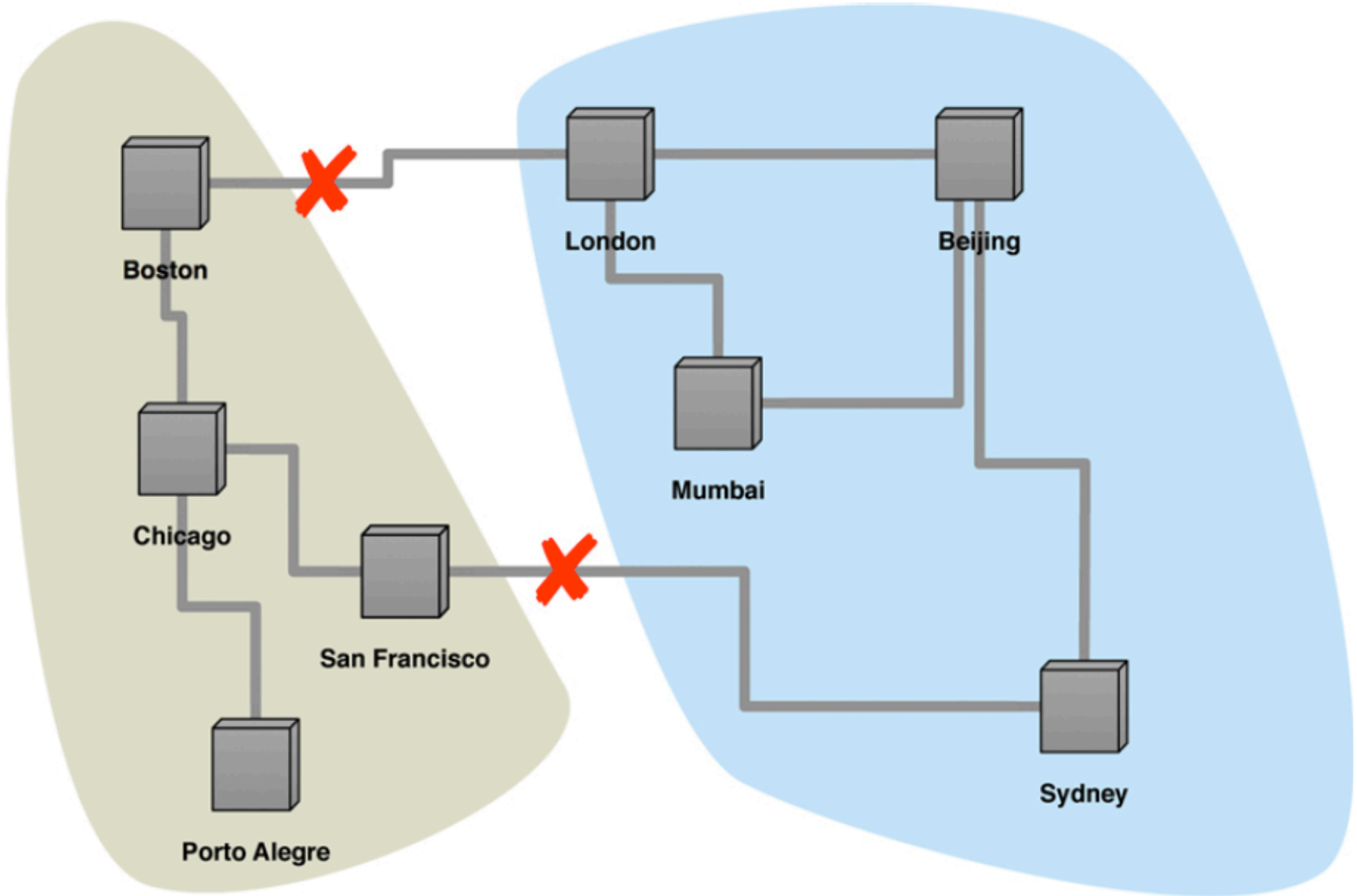
# Key terms

**Consistency:** After an update, all readers in a distributed system (assuming replication) see the same data

**Availability:** If a node (server) is working, it can read and write data

**Partition Tolerance:** System continues to operate, even if two sets of servers get isolated

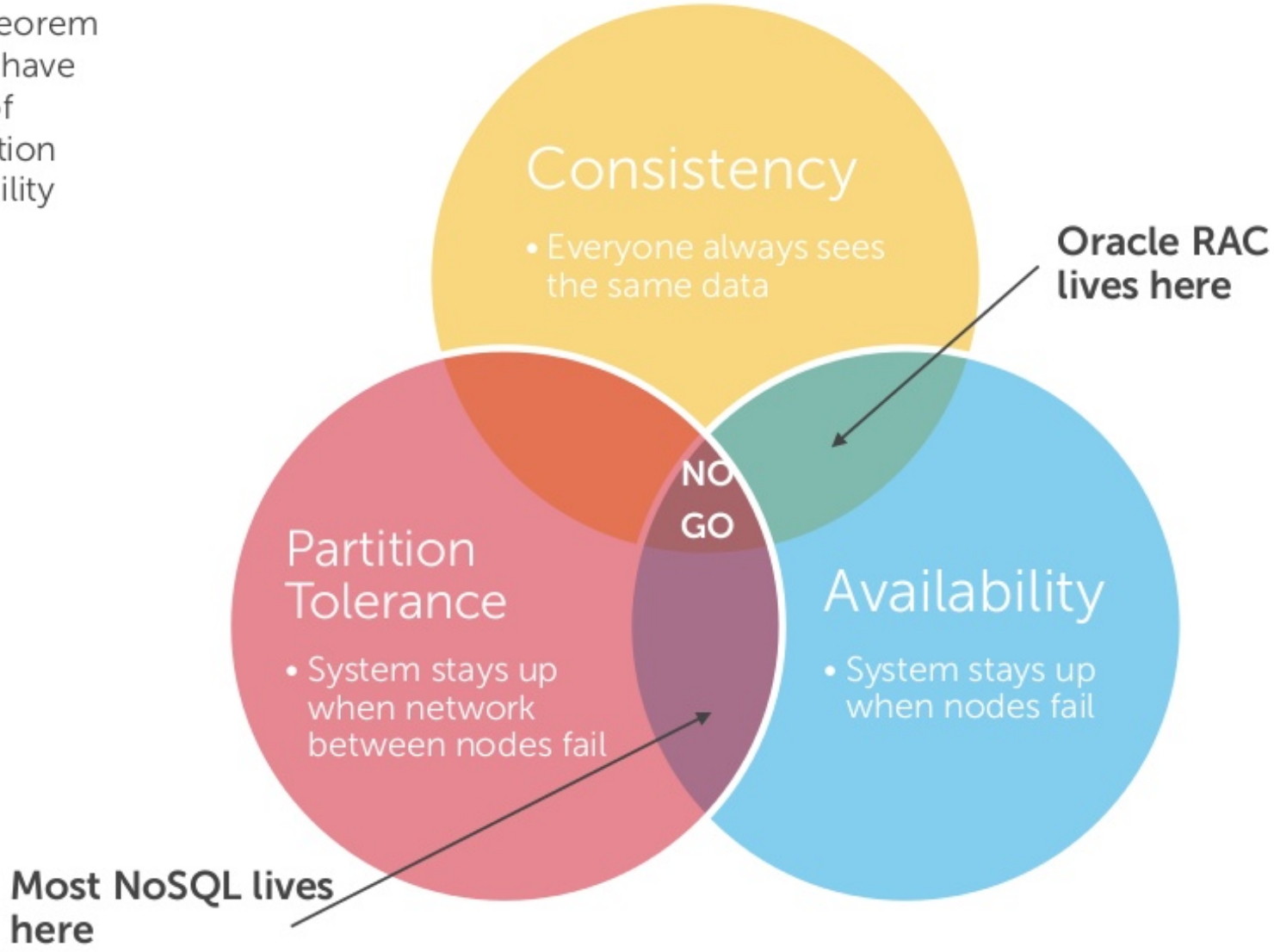
With two breaks in the communication lines, the network partitions into two groups.



Distributed data: the CAP theorem

# CAP Theorem says something has to give

- CAP (Brewer's) Theorem says you can only have two out of three of Consistency, Partition Tolerance, Availability



# CAP Theorem: Real application

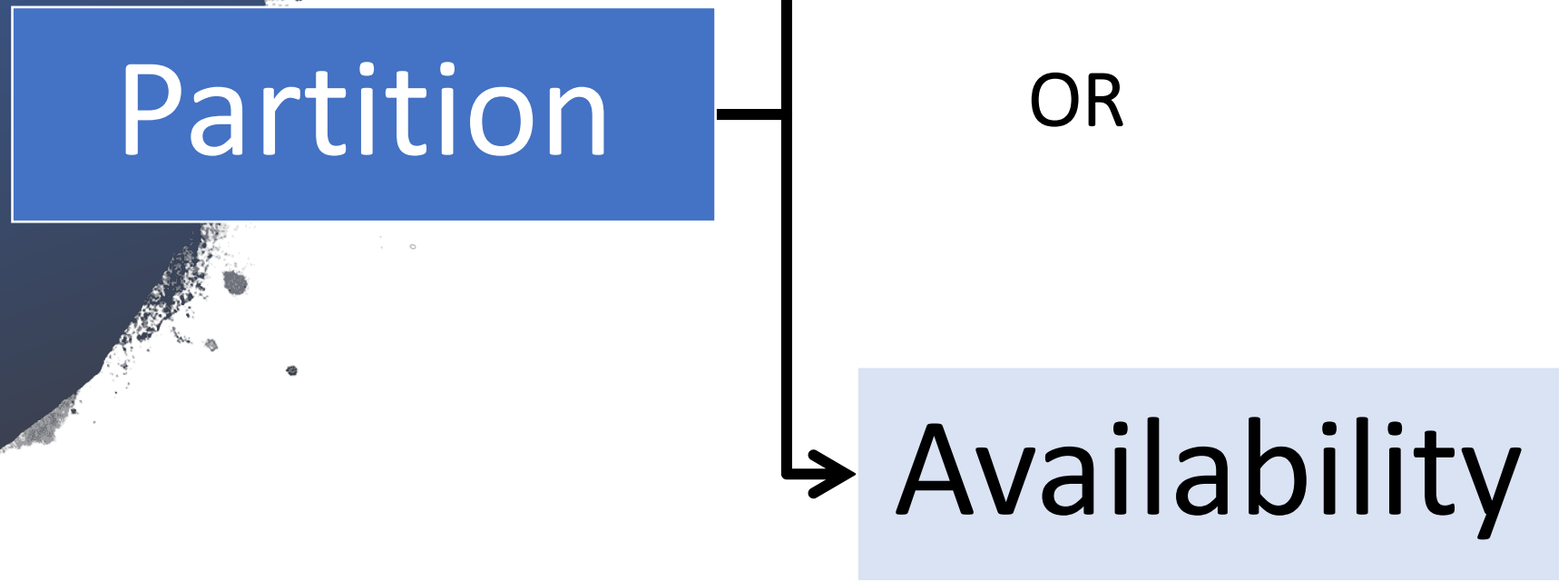
A distributed system practically has to be tolerant of network Partitions (P)

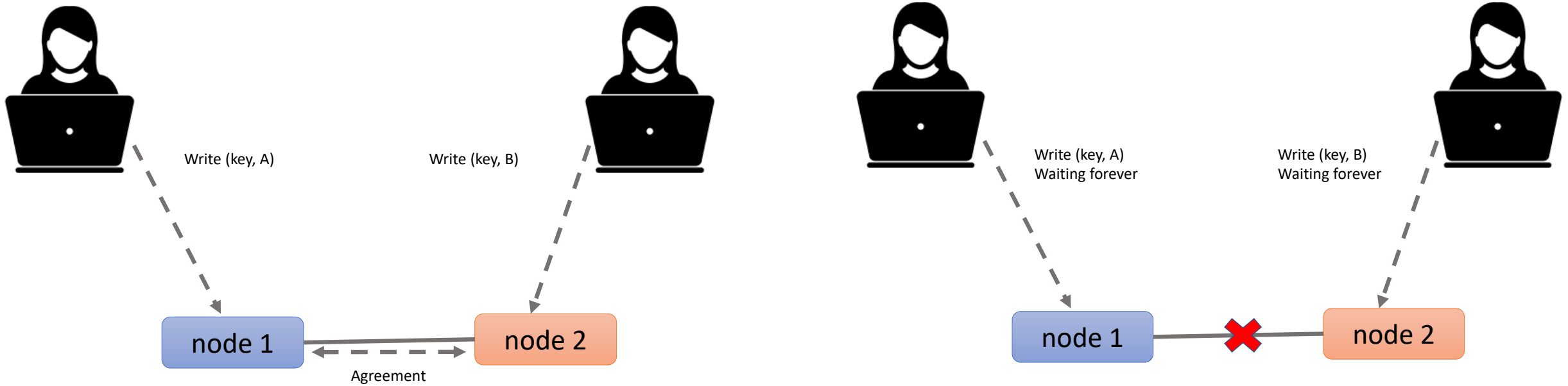
So, tradeoff between Consistency and Availability

# CAP theorem: Alternative presentation

Fowler's version of CAP theorem:

*if you have a distributed database, when a partition occurs, you have a choice between consistency OR availability.*






# Partition tolerance & Consistency

# ACID vs BASE

**Basically Available**: The system does guarantee the *availability* of the data; there will be a response to any request. But 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'.

**Eventual consistency**: The system will eventually become consistent once it stops receiving input. The data will propagate to everywhere it needs to, sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.