Recap: XML

Tags (<foo>)

Attributes (<foo bar="buzz" />)

Text nodes (<foo>bar</foo>)

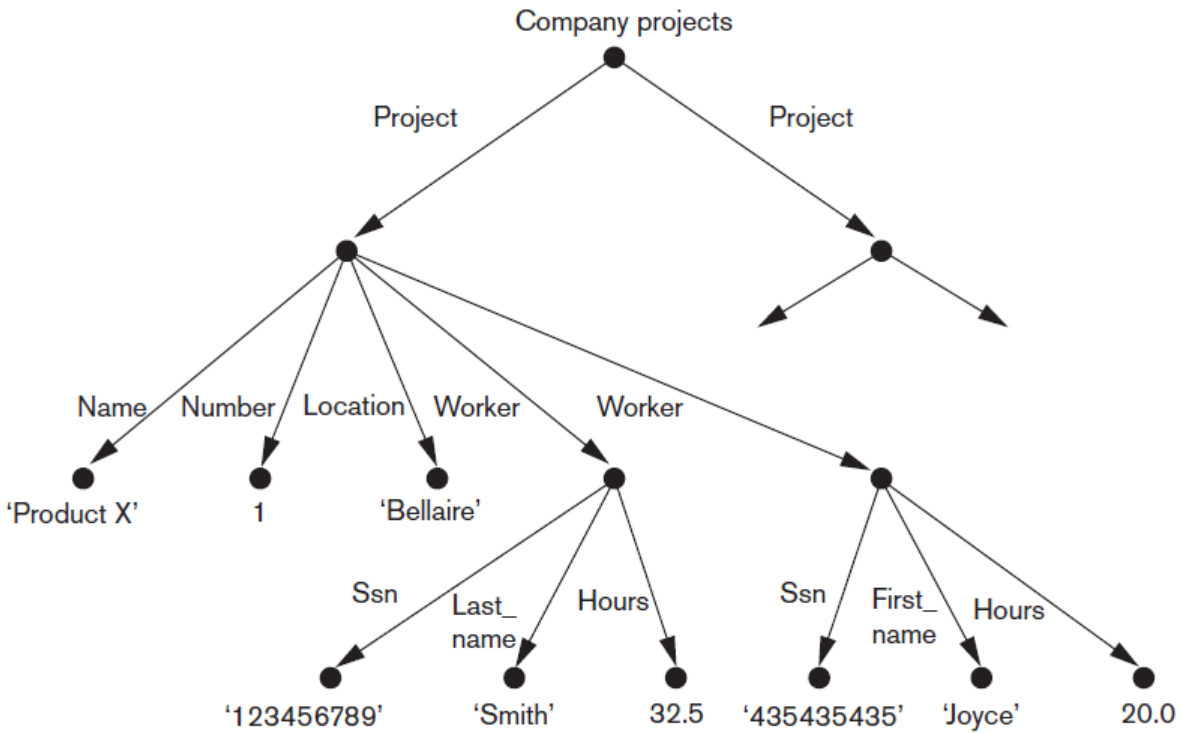
Wellformedness versus Validity

Validator:

https://www.w3schools.com/Xml/xml_validator.asp

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Example XML Document -->
<config >
    <program_version> 16 </program_version>
    <item item_date=" January 2009">
        <mode>1</mode>
        <item_unit>900</item_unit>
    </item>
    <item date="February 2009" date2="March 2009">
        <mode>2</mode>
        <unit/>
    </item>
    <item date="December 2009" date2="December 2009">
        <mode>9</mode>
        <unit>5</unit>
        <current>100</current>
        <interactive>3</interactive>
    </item>
    <ITEM date="Jan 2010">
        <mode>9</mode>
        <unit>5</unit>
        <current>100</current>
        <interactive>3</interactive>
    </ITEM>
    <item date="Jan 2011">
        <mode>9
        </mode>
    </item>
    <unit>5</unit>
</config>
```

Hierarchical Data Model



XML - Advantages and Disadvantages

XML is **very** powerful, but this comes with a price:

- Verbose (start and close tags, lots of metadata)
- Not particularly easy to read for humans
- Complexity
  - Of XML itself (namespaces, entities, …)
  - And of some support tech (namely XML Schema and XQuery)

Lightweight Semi-Structured Data

Given these problems with XML, some alternatives have been proposed in the last years

- Keep the core ideas (e.g., hierarchical structure)
- But remove most of the heavy-weight support technology

Examples:

- JSON
- YAML

JSON

JSON (JavaScript Object Notation)

Basically serialized JavaScript objects

Particularly easy to use from JavaScript, but nowadays libraries for most programming languages

JSON Syntax

Basic syntax:

```
{
    KEY1: VALUE1, KEY1: VALUE2, … KEYn: VALUEn,
}
```

`KEY`: string

`VALUE`: one of simple type, list (`[ ]`), complex (`{ }`), or `null`

JSON Syntax

Supported simple types:

- Number
- String
- Boolean

No support for attributes, namespaces, entities, …

JSON Example

```
{
        "firstName": "John",
        "lastName": "Smith",
        "isAlive": true,
        "age": 27,
        "address": {
          "streetAddress": "21 2nd Street",
          "city": "New York"
        },
        "phoneNumbers": [
          {
            "type": "home",
            "number": "212 555-1234"
          },
          {
            "type": "office",
            "number": "646 555-4567"
          }
        ],
        "children": [],
        "spouse": null
}
```

JSON - Advantages and Disadvantages

JSON is:

- Much simpler than XML
- (A little) easier to read

But still a big soup of brackets and special characters

YAML

YAML:

- "YAML ain't a markup language" *or*
- "Yet another markup language"

Superset of JSON

- All legal JSON documents are also legal YAML documents
- But: removes need for most brackets, significant whitespace

Core usage:

- Human-readable config files

YAML

Basic structure is still

```
key: value
```

But quotes are now optional, and whitespace is used to indicate nesting

Dash symbol (-) used to indicate list entries

```
firstName: John
lastName: Smith
age: 25
address:
  streetAddress: 21 2nd Street
  city: New York
  state: NY
  postalCode: '10021'
phoneNumber:
- type: home
  number: 212 555-1234
- type: fax
  number: 646 555-4567
gender:
  type: male
```

Key Takeaways (1)

Semi-structured data mixes data and meta-data in a document

Difference between textual and binary data

Different types of semi-structured data:

- XML
- JSON
- YAML

Key Takeaways (2)

XML well-formedness and validity

Core XML syntax rules:

- Tags, attributes, and text nodes
- Every start tag needs an end tag
- Correct nesting of tags

Examples of languages implemented on top of XML

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

“Big Data” and Map/Reduce

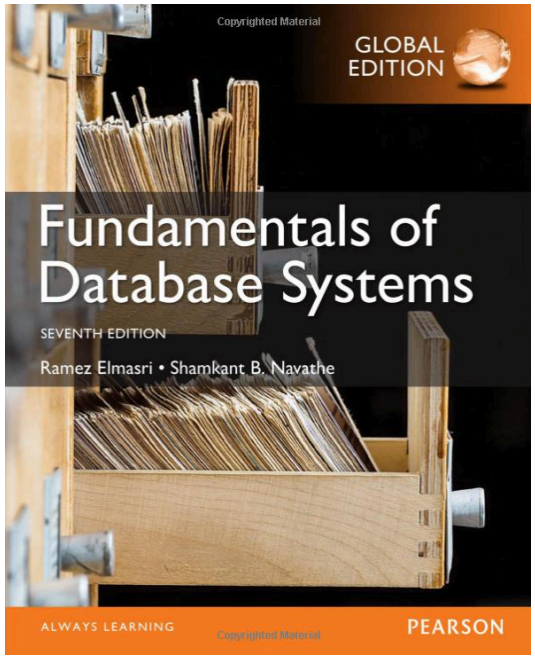
LECTURE 13

Dr. Philipp Leitner

philipp.leitner@chalmers.se

@xLeitix

LECTURE 13



Covers …

Chapter 25

(without the Hadoop parts)

What we will be covering

Basic idea and definitions of Big Data

Map/Reduce as a common data processing model for Big Data

Big Data

Tremendous growth of data generation in the last decade

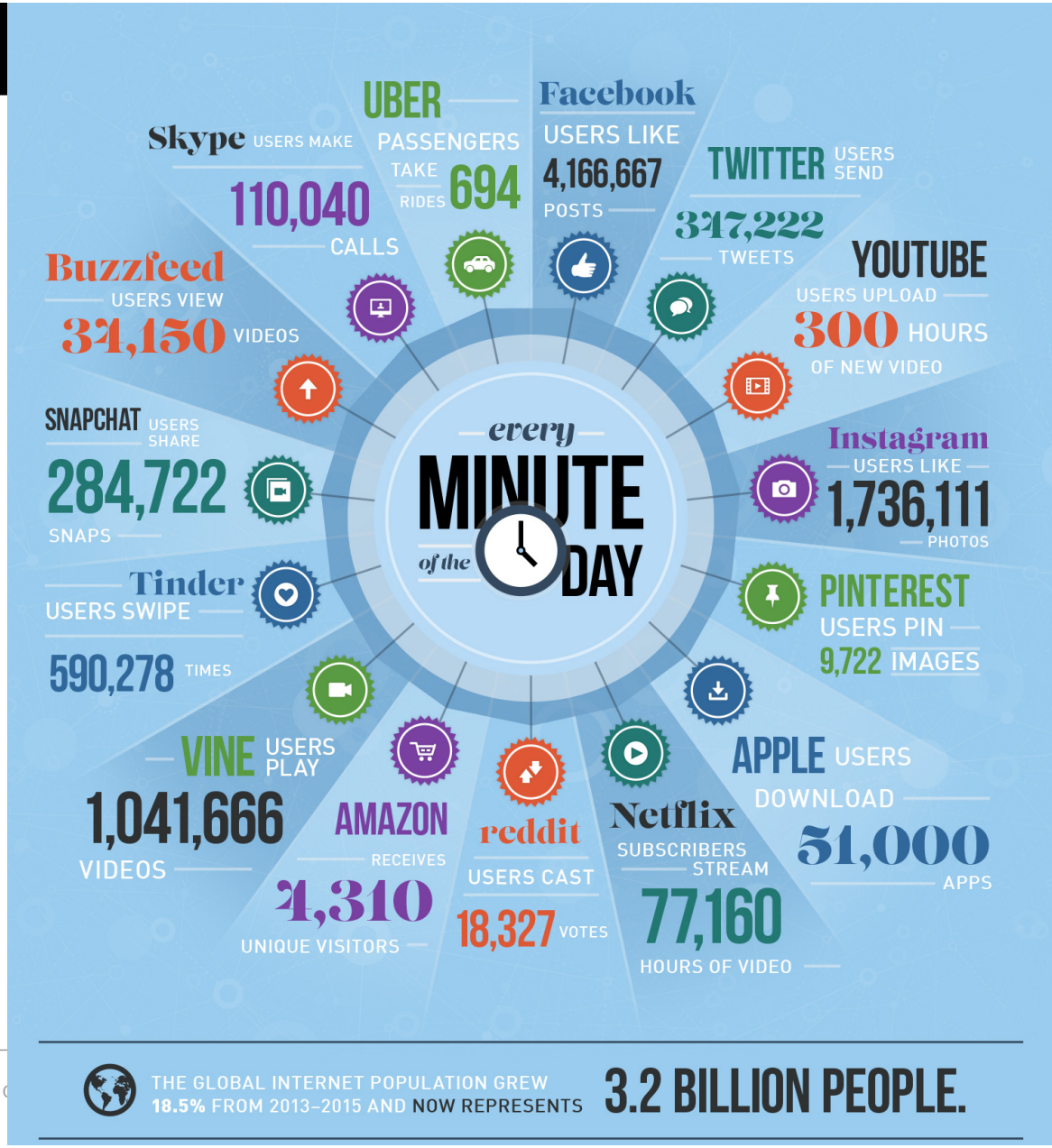
- Social media
- Mobile computing
- Sensors / Internet-of-Things
- Communication networks and satellite imagery
- …

Example: Social Media

Facebook users “like” over 4M posts per minute.

Close to 600.000 “swipes” on Tinder every minute.

….



Example: Sensors and the Internet-of-Things

Internet-of-Things

Catchy name for the trend to connect all kinds of devices to the Web, not only computers and phones

Examples:

- Amazon's Alexa
- The "intelligent" fridge

(but also cars, cameras, agricultural sensors, etc.)

Example: Sensors and the Internet-of-Things

Following IDC, there will be **80B devices** connected to the Internet by 2025.

These devices are expected to produce a total of **180 Zettabytes** (that's 180 * 10^12 GB) of data by then.

Example: Sensors and the Internet-of-Things

Why so much?

Curated content << user-generated content << machine-generated signals

Machine-generated signals:

- Status, keep-alive messages
- Sensory data (temperature, speed, humidity, …)
- Photo and video material
- Location updates

Challenges - The 5Vs of Big Data

Volume

Massive amounts of data

Velocity

New data is incoming at high speed

Variety

Data comes in various forms

Veracity

Data points differ in accuracy and trustworthiness

Value

Data needs to be made useful

Solution Approaches

Variety
- NoSQL databases

Volume
- Sharding and eventual consistency

Velocity
- New processing models (e.g., Map/Reduce, stream processing)
- Edge computing / data aggregation close to data sources

Veracity
- E.g., stochastic event correlation

Value
- *(out of scope in this course)*

Map/Reduce

Programming model to write **data analysis programs**

- Intended to operate on very large, sharded datasets
- Inherently parallel and fault-tolerant

Developed by Google researchers in 2004

- First use case: (re-)build the Google index from a very large database of websites

Map/Reduce

Super-short intro video:

https://www.youtube.com/watch?v=gI4HN0JhPmo

(note that the video at the end says that all pairs with the same key get sent immediately to the same reducer - that's not necessarily true)

Fundamental Data Structure

Map/Reduce programmes operate on a very simple data structure:

Key/Value pairs (similar to Key/Value Stores)

Keys don’t need to be unique after mapping, but data items with the same key are typically merged during reducing

Values can be arbitrary documents

Basic Model

A Map/Reduce programme needs two definitions:

- A **mapping function** which reads items from the data store and emits key/value pairs
- A **reduce function** which accepts a key and some or all values of this key, and emits a new key/value pair

Model makes no assumptions about the order of mapping

Typical Use Case

Map/Reduce is **not a general query language** like SQL

Instead, it is useful to write **aggregate queries** over large data sets:

Similar to SQL aggregate queries with grouping

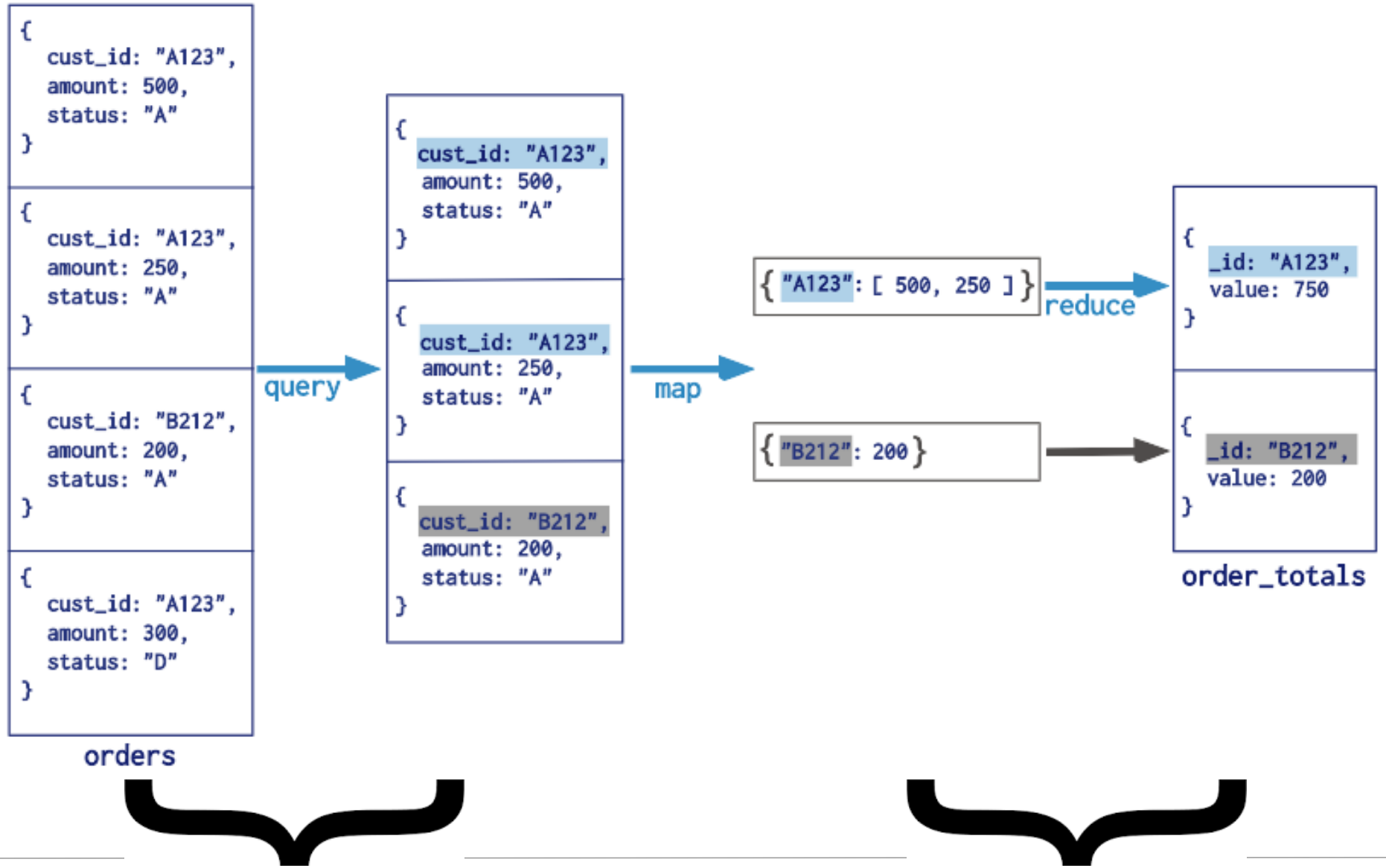
Example

Let’s assume we have a big MongoDB database

All documents are customer orders

Have a customer ID `cust_id` and order amount `amount`

We want a Map/Reduce program that gets us a list of total amounts per customer


{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{ "A123": [500, 250] }
{ "B212": 200 }
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals
Mapping
Reducing

Implementation in MongoDB

The book presents Map/Reduce based on **Hadoop**

- A standard stand-alone system for big data processing
- We are **not** using Hadoop in this course

However, MongoDB also natively supports Map/Reduce queries

- Defined through **JavaScript** functions operating on JSON documents

Implementation in MongoDB - Mapping

```
var map = function() {
    emit(this.cust_id, this.amount);
};
```

(use the `emit(key,val)` function to produce the mapping, use `this` to access the document that `map` is called on)

Implementation in MongoDB - Reducing

```
var reduce = function(key, amounts) {
    return Array.sum(amounts);
};
```

(function is invoked once or multiple times per key, values per key are contained in an array as second param, use `return` to return the reduced result; if there is just one value for a key `reduce` will **not** be invoked)

Implementation in MongoDB - Starting a Query

```
db.orders.mapReduce(
   map, reduce,
   { out: “aggregated_orders" }
)
```

(save results to a new collection `aggregated_orders`)

Full Spec for Map/Reduce in MongoDB

https://docs.mongodb.com/manual/reference/command/mapReduce/

Some useful additional features of Map/Reduce in MongoDB

query:

Pre-select only documents matching a specific query before even invoking `map`.

Good for performance optimization.

finalize:

Another function, invoked exactly once after all reducing is done.

Makes writing clean `reduce` functions easier.

Another example

Assume documents of the following structure:

```
{
     _id: ObjectId("50a8240b927d5d8b5891743c"),
     cust_id: "abc123",
     ord_date: new Date("Oct 04, 2012"),
     status: 'A',
     price: 25,
     items: [ { sku: "mmm", qty: 5, price: 2.5 },
              { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

Another example

```
{
        _id: ObjectId("50a8240b927d5d8b5891743c"),
        cust_id: "abc123",
        ord_date: new Date("Oct 04, 2012"),
        status: 'A',
        price: 25,
        items: [ { sku: "mmm", qty: 5, price: 2.5 },
                 { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

We now want to calculate **how often each item was ordered in total** and **on average per order since the beginning of the year**.

In-class exercise:

Think about what steps are needed.

Sketch the mapper, reducer, and finalizer if needed (in JS or pseudo-code)

Another example

We now want to calculate how often each item was ordered in total and on average per order since the beginning of the year.

Steps:

1. Map only orders with `ord_date > 2018/01/01`
2. For each order, emit once per item with how often it was ordered.
3. In reducing, tally this for each item
4. In finalizing, calculate the average for each item

Invoking Map/Reduce

```
db.orders.mapReduce(map, reduce,
    {
        out: { out: “order_example" },
        query: {
            ord_date: { $gt: new Date('01/01/2018') }
        },
        finalize: finalize
    }
)
```

Mapping

```
var map = function() {
    for (var idx = 0; idx < this.items.length; idx++) {
        var key = this.items[idx].sku;
        var value = {
            count: 1,
            qty: this.items[idx].qty
        };
        emit(key, value);
    }
};
```

Reducing

```
var reduce = function(keySKU, countObjVals) {
    reducedVal = { count: 0, qty: 0 };

    for (var idx = 0; idx < countObjVals.length; idx++) {
        reducedVal.count += countObjVals[idx].count;
        reducedVal.qty += countObjVals[idx].qty;
    }

    return reducedVal;
};
```

Finalizing

```
var finalize = function (key, reducedVal) {

   reducedVal.avg = reducedVal.qty/reducedVal.count;
   return reducedVal;

};
```

Formal Rules of Reducing

Commutative:

```
reduce(key,[ A, B ] ) == reduce( key, [ B, A ] )
```

Idempotent:

```
reduce( key, [ reduce(key, valuesArray) ] ) ==
reduce( key, valuesArray )
```

Formal Rules of Reducing

Associative:

```
reduce(key, [ C, reduce(key, [ A, B ]) ] ) ==
reduce( key, [ C, A, B ] )
```

Formal Rules of Reducing

As a developer **you** need to make sure that your JS function follows these rules

Not enforced by MongoDB

Leads to nasty, hard-to-detect bugs

A larger MR Example

Assume a MongoDB database of all ZIP codes in the United States

```
> db.zips.find()
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.108354, 42.409698 ], "pop" : 4546, "state" : "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51565, 42.377017 ], "pop" : 36963, "state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.410953, 42.275103 ], "pop" : 10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240, "state" : "MA" }
{ "_id" : "01011", "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688, "state" : "MA" }
{ "_id" : "01010", "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706, "state" : "MA" }
{ "_id" : "01012", "city" : "CHESTERFIELD", "loc" : [ -72.833309, 42.38167 ], "pop" : 177, "state" : "MA" }
{ "_id" : "01013", "city" : "CHICOPEE", "loc" : [ -72.607962, 42.162046 ], "pop" : 23396, "state" : "MA" }
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA" }
{ "_id" : "01020", "city" : "CHICOPEE", "loc" : [ -72.576142, 42.176443 ], "pop" : 31495, "state" : "MA" }
{ "_id" : "01028", "city" : "EAST LONGMEADOW", "loc" : [ -72.505565, 42.067203 ], "pop" : 13367, "state" : "MA" }
{ "_id" : "01027", "city" : "MOUNT TOM", "loc" : [ -72.679921, 42.264319 ], "pop" : 16864, "state" : "MA" }
{ "_id" : "01030", "city" : "FEEDING HILLS", "loc" : [ -72.675077, 42.07182 ], "pop" : 11985, "state" : "MA" }
{ "_id" : "01031", "city" : "GILBERTVILLE", "loc" : [ -72.198585, 42.332194 ], "pop" : 2385, "state" : "MA" }
{ "_id" : "01032", "city" : "GOSHEN", "loc" : [ -72.844092, 42.466234 ], "pop" : 122, "state" : "MA" }
{ "_id" : "01033", "city" : "GRANBY", "loc" : [ -72.520001, 42.255704 ], "pop" : 5526, "state" : "MA" }
{ "_id" : "01034", "city" : "TOLLAND", "loc" : [ -72.908793, 42.070234 ], "pop" : 1652, "state" : "MA" }
{ "_id" : "01022", "city" : "WESTOVER AFB", "loc" : [ -72.558657, 42.196672 ], "pop" : 1764, "state" : "MA" }
{ "_id" : "01026", "city" : "CUMMINGTON", "loc" : [ -72.905767, 42.435296 ], "pop" : 1484, "state" : "MA" }
{ "_id" : "01035", "city" : "HADLEY", "loc" : [ -72.571499, 42.36062 ], "pop" : 4231, "state" : "MA" }
```

ZIP Example 1

```
> db.zips.find()
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.108354, 42.409698 ], "pop" : 4546, "state" : "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51565, 42.377017 ], "pop" : 36963, "state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.410953, 42.275103 ], "pop" : 10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240, "state" : "MA" }
{ "_id" : "01011", "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688, "state" : "MA" }
{ "_id" : "01010", "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706, "state" : "MA" }
{ "_id" : "01012", "city" : "CHESTERFIELD", "loc" : [ -72.833309, 42.38167 ], "pop" : 177, "state" : "MA" }
{ "_id" : "01013", "city" : "CHICOPEE", "loc" : [ -72.607962, 42.162046 ], "pop" : 23396, "state" : "MA" }
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA" }
{ "_id" : "01020", "city" : "CHICOPEE", "loc" : [ -72.576142, 42.176443 ], "pop" : 31495, "state" : "MA" }
{ "_id" : "01028", "city" : "EAST LONGMEADOW", "loc" : [ -72.505565, 42.067203 ], "pop" : 13367, "state" : "MA" }
{ "_id" : "01027", "city" : "MOUNT TOM", "loc" : [ -72.679921, 42.264319 ], "pop" : 16864, "state" : "MA" }
{ "_id" : "01030", "city" : "FEEDING HILLS", "loc" : [ -72.675077, 42.07182 ], "pop" : 11985, "state" : "MA" }
{ "_id" : "01031", "city" : "GILBERTVILLE", "loc" : [ -72.198585, 42.332194 ], "pop" : 2385, "state" : "MA" }
{ "_id" : "01032", "city" : "GOSHEN", "loc" : [ -72.844092, 42.466234 ], "pop" : 122, "state" : "MA" }
{ "_id" : "01033", "city" : "GRANBY", "loc" : [ -72.520001, 42.255704 ], "pop" : 5526, "state" : "MA" }
{ "_id" : "01034", "city" : "TOLLAND", "loc" : [ -72.908793, 42.070234 ], "pop" : 1652, "state" : "MA" }
{ "_id" : "01022", "city" : "WESTOVER AFB", "loc" : [ -72.558657, 42.196672 ], "pop" : 1764, "state" : "MA" }
{ "_id" : "01026", "city" : "CUMMINGTON", "loc" : [ -72.905767, 42.435296 ], "pop" : 1484, "state" : "MA" }
{ "_id" : "01035", "city" : "HADLEY", "loc" : [ -72.571499, 42.36062 ], "pop" : 4231, "state" : "MA" }
```

How many distinct ZIP codes are there? Solve using Map/Reduce.

ZIP Example 2

```
> db.zips.find()
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.108354, 42.409698 ], "pop" : 4546, "state" : "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51565, 42.377017 ], "pop" : 36963, "state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.410953, 42.275103 ], "pop" : 10579, "state" : "MA" }
{ "_id" : "01008", "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240, "state" : "MA" }
{ "_id" : "01011", "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688, "state" : "MA" }
{ "_id" : "01010", "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706, "state" : "MA" }
{ "_id" : "01012", "city" : "CHESTERFIELD", "loc" : [ -72.833309, 42.38167 ], "pop" : 177, "state" : "MA" }
{ "_id" : "01013", "city" : "CHICOPEE", "loc" : [ -72.607962, 42.162046 ], "pop" : 23396, "state" : "MA" }
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA" }
{ "_id" : "01020", "city" : "CHICOPEE", "loc" : [ -72.576142, 42.176443 ], "pop" : 31495, "state" : "MA" }
{ "_id" : "01028", "city" : "EAST LONGMEADOW", "loc" : [ -72.505565, 42.067203 ], "pop" : 13367, "state" : "MA" }
{ "_id" : "01027", "city" : "MOUNT TOM", "loc" : [ -72.679921, 42.264319 ], "pop" : 16864, "state" : "MA" }
{ "_id" : "01030", "city" : "FEEDING HILLS", "loc" : [ -72.675077, 42.07182 ], "pop" : 11985, "state" : "MA" }
{ "_id" : "01031", "city" : "GILBERTVILLE", "loc" : [ -72.198585, 42.332194 ], "pop" : 2385, "state" : "MA" }
{ "_id" : "01032", "city" : "GOSHEN", "loc" : [ -72.844092, 42.466234 ], "pop" : 122, "state" : "MA" }
{ "_id" : "01033", "city" : "GRANBY", "loc" : [ -72.520001, 42.255704 ], "pop" : 5526, "state" : "MA" }
{ "_id" : "01034", "city" : "TOLLAND", "loc" : [ -72.908793, 42.070234 ], "pop" : 1652, "state" : "MA" }
{ "_id" : "01022", "city" : "WESTOVER AFB", "loc" : [ -72.558657, 42.196672 ], "pop" : 1764, "state" : "MA" }
{ "_id" : "01026", "city" : "CUMMINGTON", "loc" : [ -72.905767, 42.435296 ], "pop" : 1484, "state" : "MA" }
{ "_id" : "01035", "city" : "HADLEY", "loc" : [ -72.571499, 42.36062 ], "pop" : 4231, "state" : "MA" }
```

What is the name of the city with the highest population for each state? Note that some cities have more than one ZIP code, which you need to add up. The response should be a list of states, city names, and populations.

Advantages of Map/Reduce

The Map/Reduce model is inherently well-suited for distributed databases:

Fault-tolerant - failed map or reduce tasks can easily be repeated without changing the result or crashing other parts of the program

Parallelizable - all map and reduce tasks can be done in parallel; reduce can be split into arbitrarily many separate reduce tasks

Sharding-ready - it's easy to run reduce tasks first per shard, and then combine all subresults with another reduce

Key Takeaways

The **5Vs of Big Data**

Volume, Velocity, Variety, Veracity, Value

Map/Reduce:

Is a programming model that is **inherently suitable** for distributed database processing

Needs the definition of a **map** and **reduce** function

Reduce functions should be **idempotent**, **commutative**, **associative**