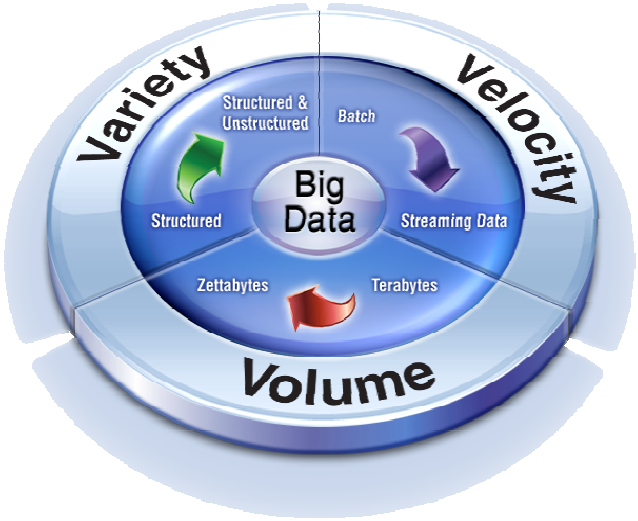
Big Data and NoSQL



Sudsanguan Ngamsuriyaroj
Ekasit Kijsipongse
Putt Sakdhnagool

Semester 1/2019

Contents

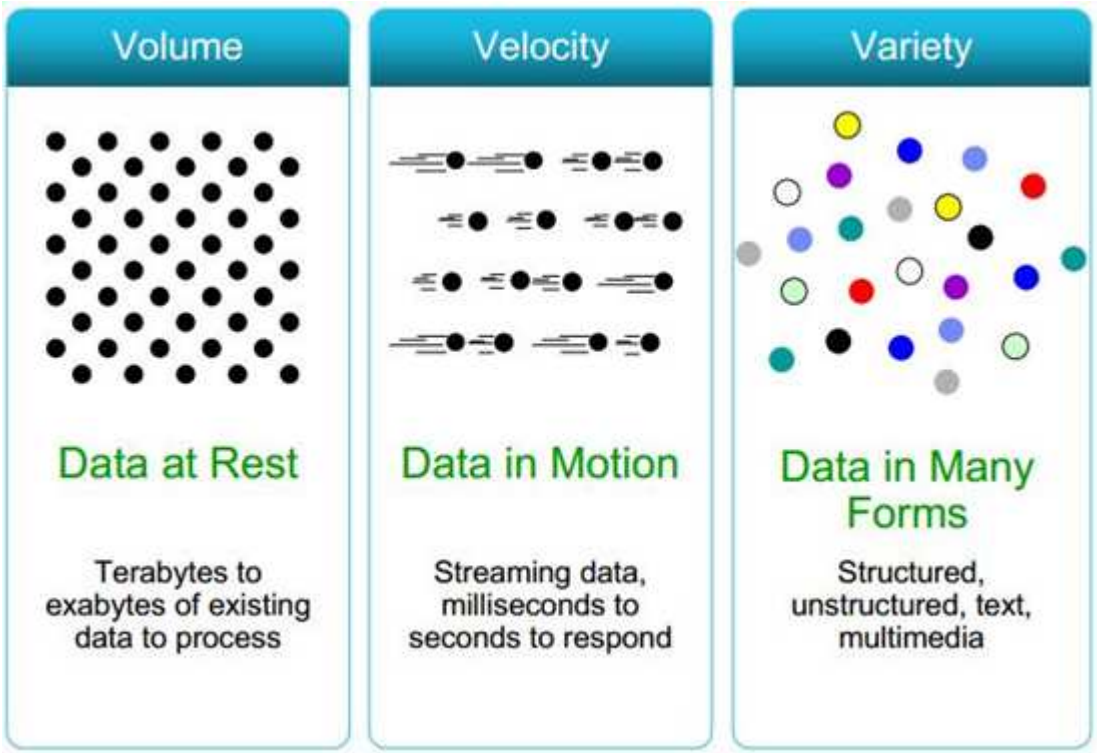
- What is big data?
- Characteristics of big data
- Big data sources
- Applications of big data
- Big data architecture and tools
- NoSQL
- MongoDB

Data is a New Oil?

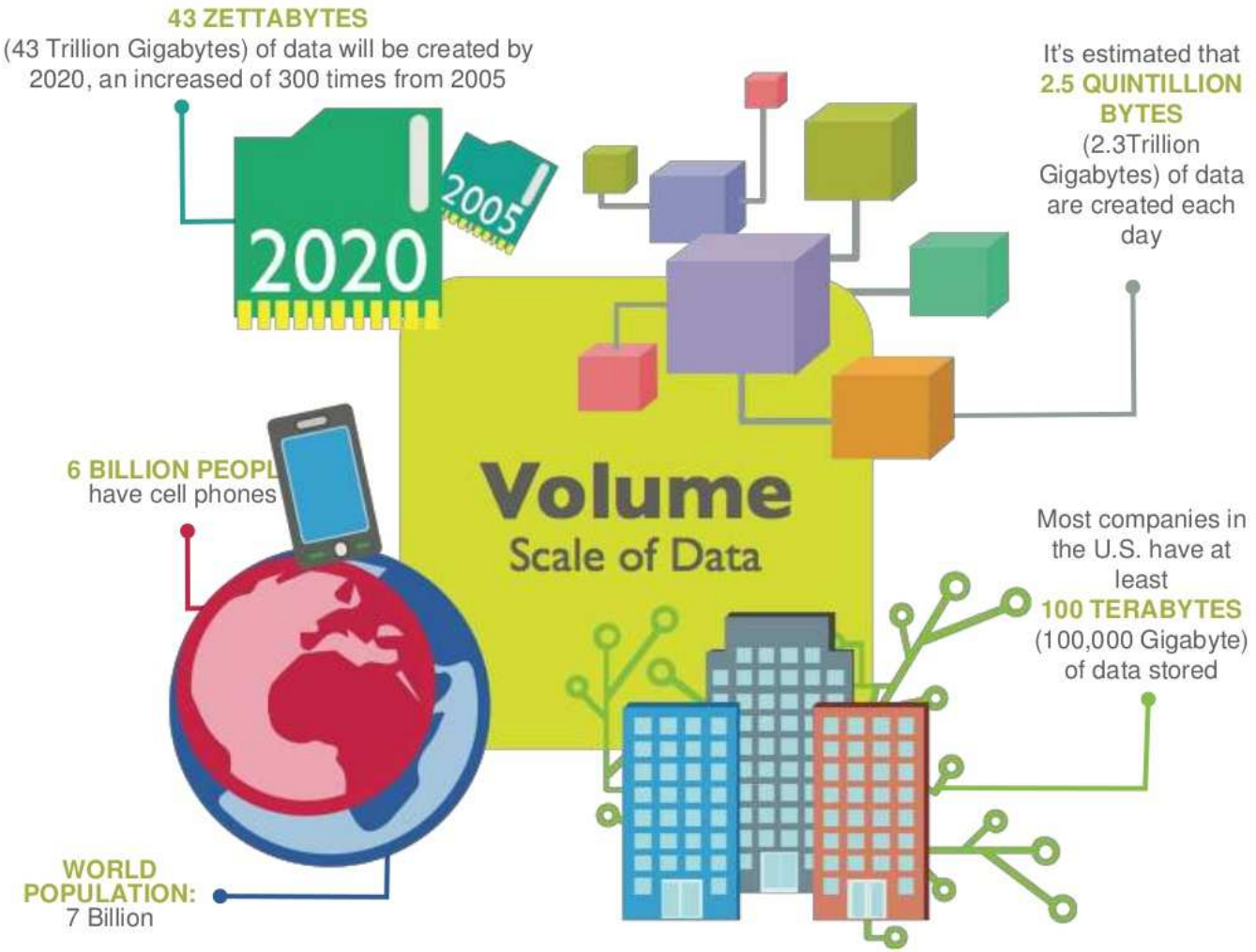
What is BIG DATA?

- 'Big Data' is similar to 'small data', but bigger in size
- Having data bigger requires different approaches: Techniques, tools and architecture
- An aim to solve new problems or old problems in a better way
- Big Data generates value from the storage and processing of very large quantities of digital information that cannot be analyzed with traditional computing techniques.

Three Characteristics of Big Data V3s



1st Character: Volume

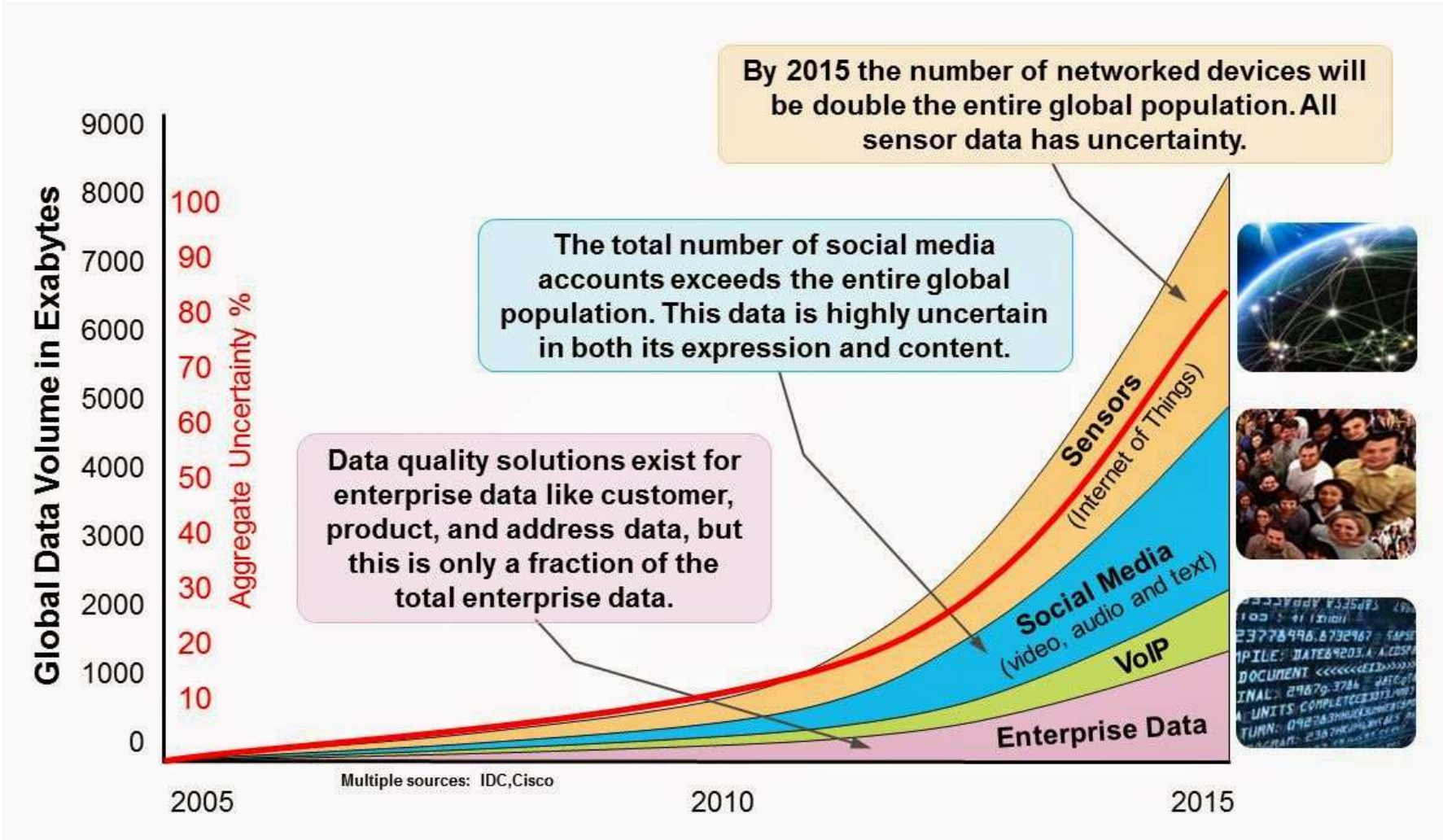


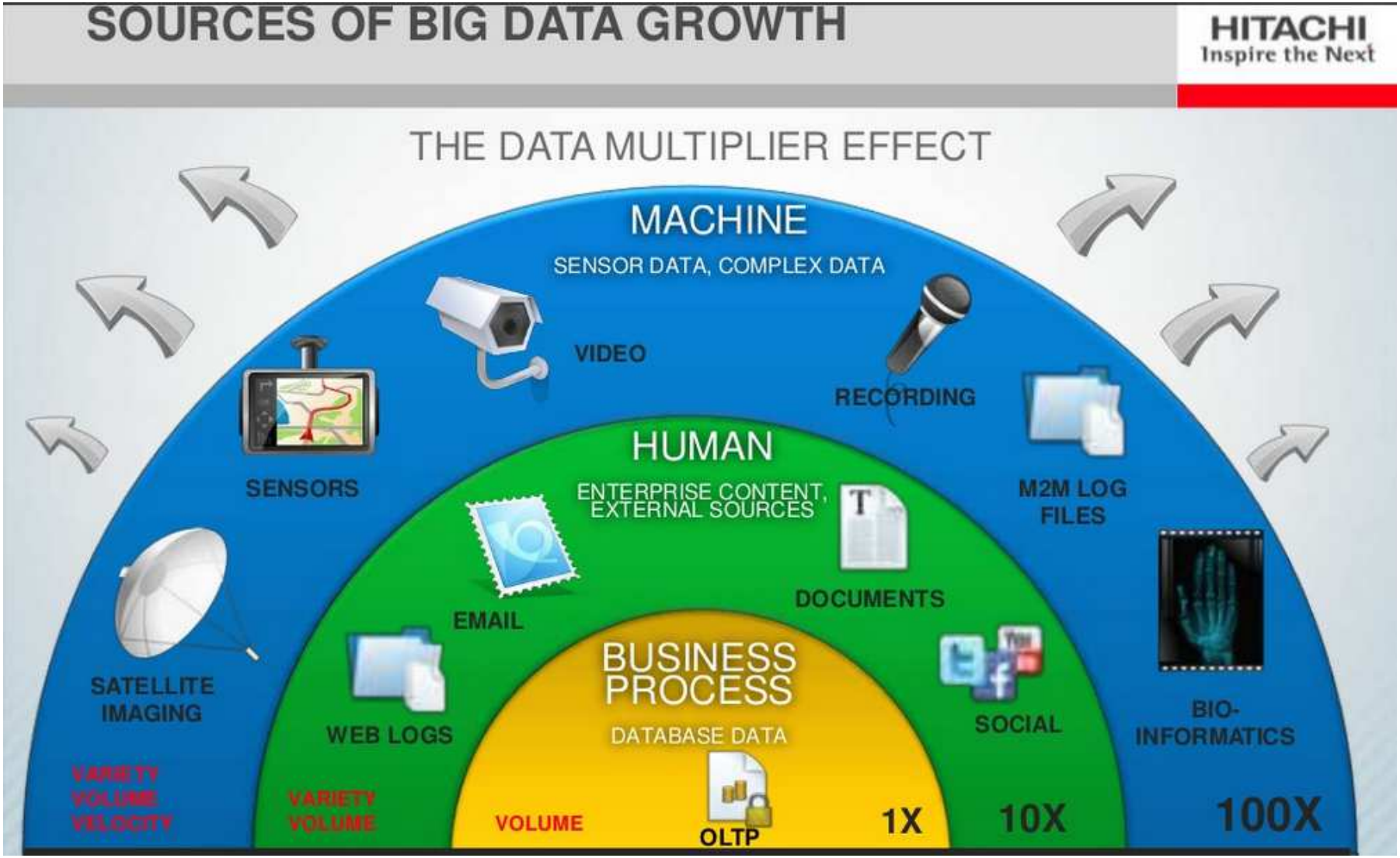
Huge Amount of Data

- From the beginning of recorded time until 2003, we created 5 billion gigabytes (exabytes) of data.
- In 2011, the same amount was created every two days
- In 2013, the same amount of data is created every 10 minutes.

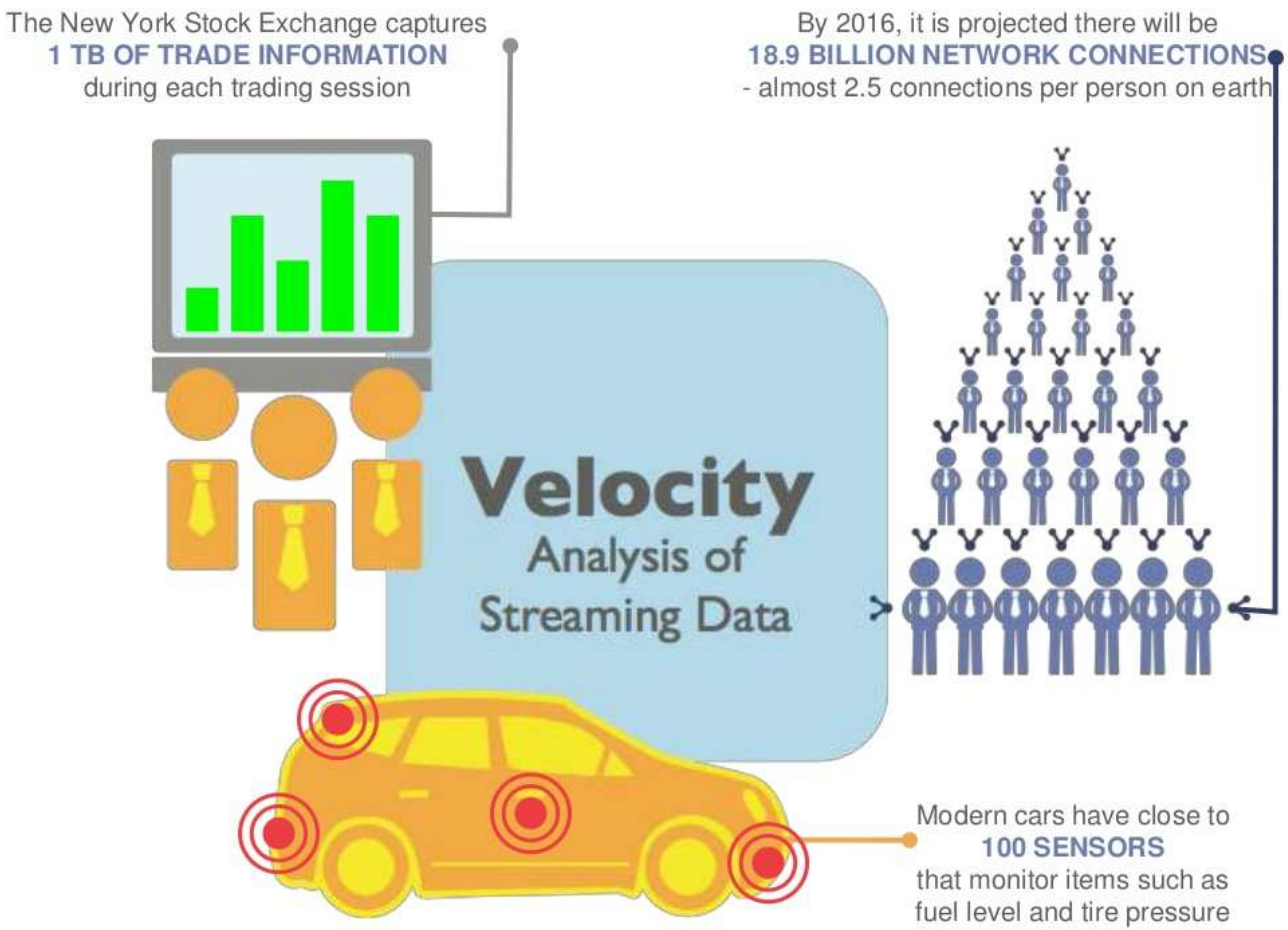
kilobyte (kB/KB)	10^3	2^{10}
megabyte (MB)	10^6	2^{20}
gigabyte (GB)	10^9	2^{30}
terabyte (TB)	10^{12}	2^{40}
petabyte (PB)	10^{15}	2^{50}
exabyte (EB)	10^{18}	2^{60}
zettabyte (ZB)	10^{21}	2^{70}
yottabyte (YB)	10^{24}	2^{80}

Data Growth

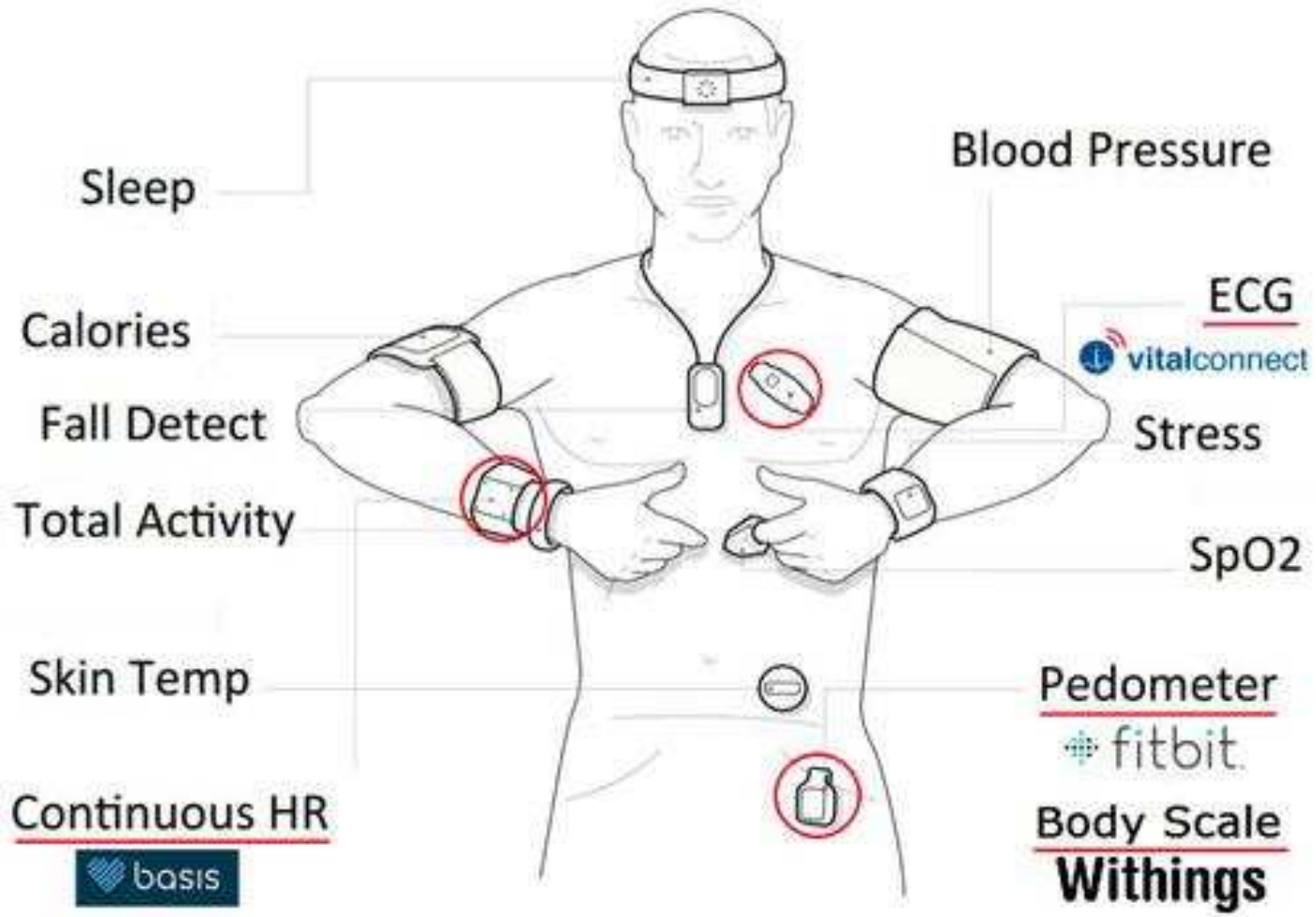

By 2015 the number of networked devices will be double the entire global population. All sensor data has uncertainty.
The total number of social media accounts exceeds the entire global population. This data is highly uncertain in both its expression and content.
Data quality solutions exist for enterprise data like customer, product, and address data, but this is only a fraction of the total enterprise data.
Global Data Volume in Exabytes
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Aggregate Uncertainty %
100
90
80
70
60
50
40
30
20
10
Sensors
(Internet of Things)
Social Media
(video, audio and text)
VoIP
Enterprise Data
Multiple sources: IDC,Cisco
2005
2010
2015


SOURCES OF BIG DATA GROWTH
HITACHI
Inspire the Next
THE DATA MULTIPLIER EFFECT
MACHINE
SENSOR DATA, COMPLEX DATA
VIDEO
RECORDING
SENSORS
M2M LOG FILES
HUMAN
ENTERPRISE CONTENT, EXTERNAL SOURCES
EMAIL
DOCUMENTS
SATELLITE IMAGING
BUSINESS PROCESS
DATABASE DATA
WEB LOGS
SOCIAL
BIO-INFORMATICS
VARIETY
VOLUME
VELOCITY
VARIETY
VOLUME
VOLUME
OLTP
1X
10X
100X

2nd Character: Velocity



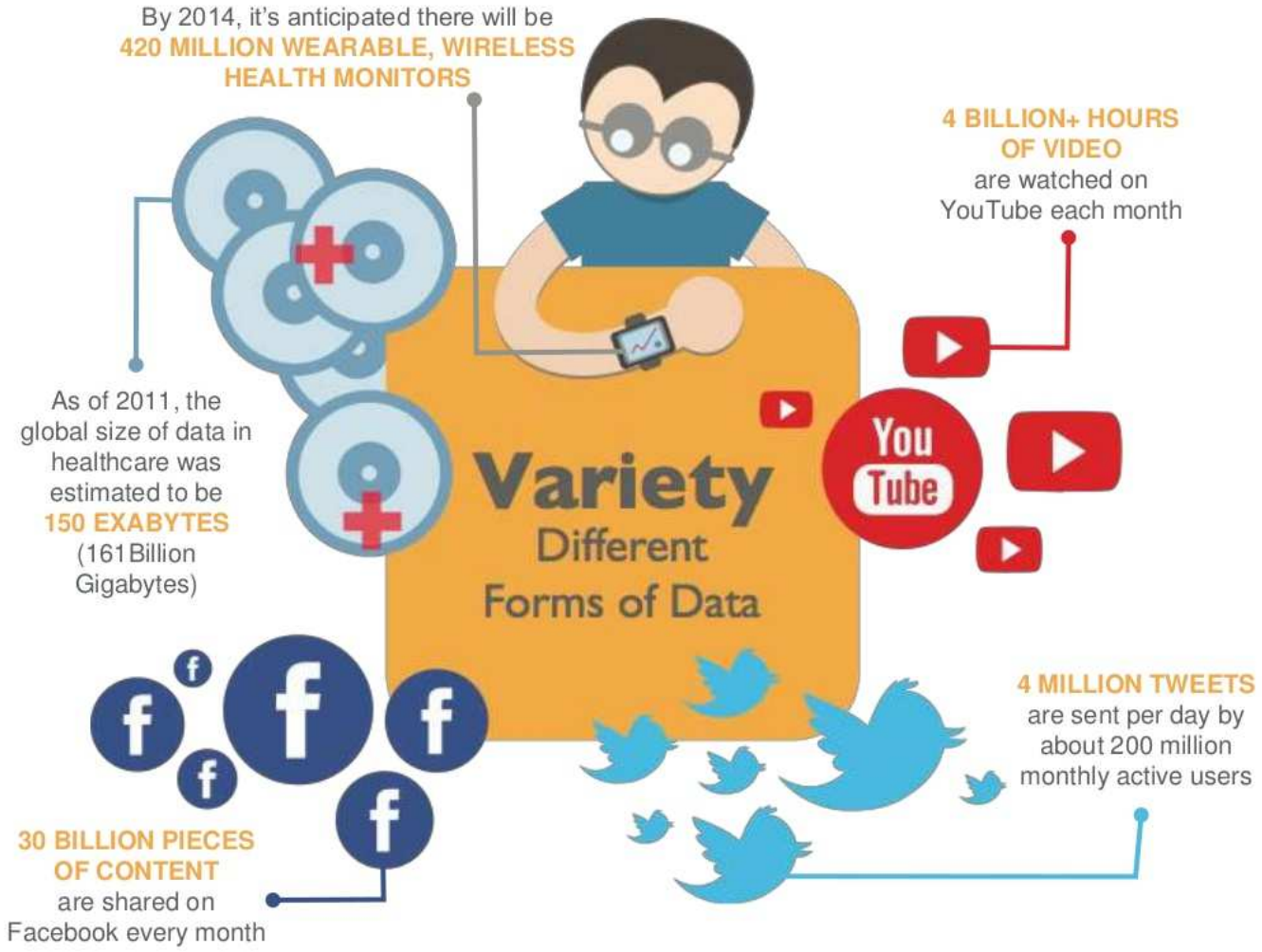
Health Sensors



Data at Rest vs Data in Motion

- Data at rest
  - Finding stats about group in a closed room
  - Analyzing sales data for last month to make strategic decisions
- Data in motion
  - Finding stats about group in a marathon
  - e-commerce order or RFID processing
- Difference lies in when are you analyzing your data?
  - after the event occurs ⇒ at rest
  - as the event occurs ⇒ in motion
- Velocity brings challenges to applications to process continuously incoming data in real-time (stream processing)

3rd Character: Variety



Types of Data

- Big Data isn't just numbers, dates, and strings stored in tables (structured data)

- Big Data includes different types of data

- Big Data is also geospatial data, 3D data, audio and video, and unstructured text, including log files and social media

Structured Data

- Table

Location	Temperature	Humidity	Rain
Colombo	32 C	77 %	22 mm
Kandy	28 C	79 %	12 mm
Galle	29 C	80 %	2 mm
Jaffna	33 C	76 %	0 mm
Badulla	27C	69 %	2 mm

Semi-structured Data

- XML (eXtensible Markup Language) is a markup language that describe data in a text format which is both human-readable and machine-readable.

```
<empinfo>
  <employees>
    <employee>
      <name>Scott Philip</name>
      <salary>£44k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Tim Henn</name>
      <salary>£40k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Long yong</name>
      <salary>£40k</salary>
      <age>28</age>
    </employee>
  </employees>
</empinfo>
```

Semi-structured Data

- JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

```
{ "empinfo" :
    {
          "employees" : [
          {
            "name" : "Scott Philip",
            "salary" : £44k,
            "age" : 27,

          },

          {
            "name" : "Tim Henn",
            "salary" : £40k,
            "age" : 27,
          },

          {
           "name" : "Long Yong",
           "salary" : £40k,
           "age" : 28,

          }
                    ]
    }
}
```

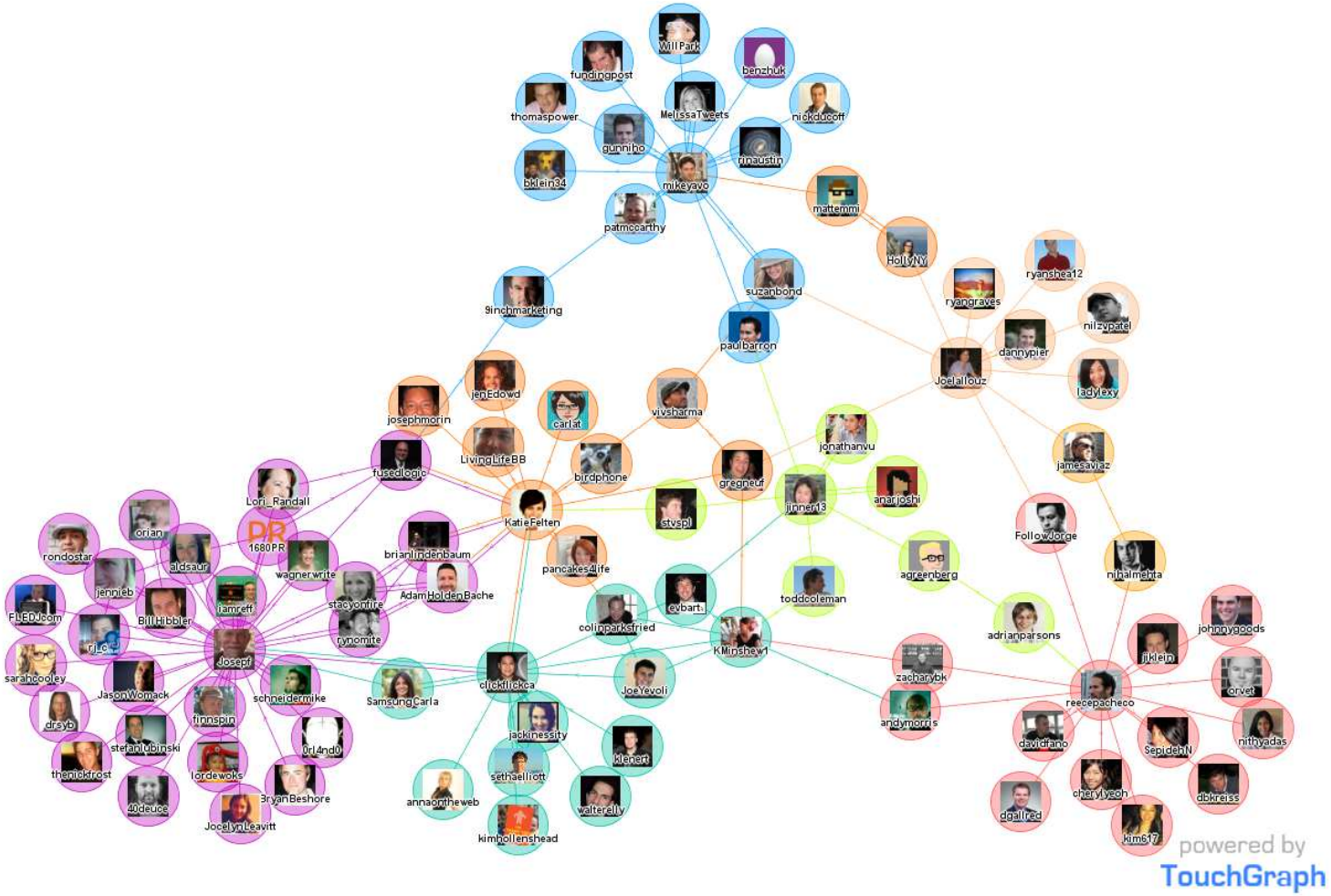
Unstructured Data

- E.g. text, images, videos or music

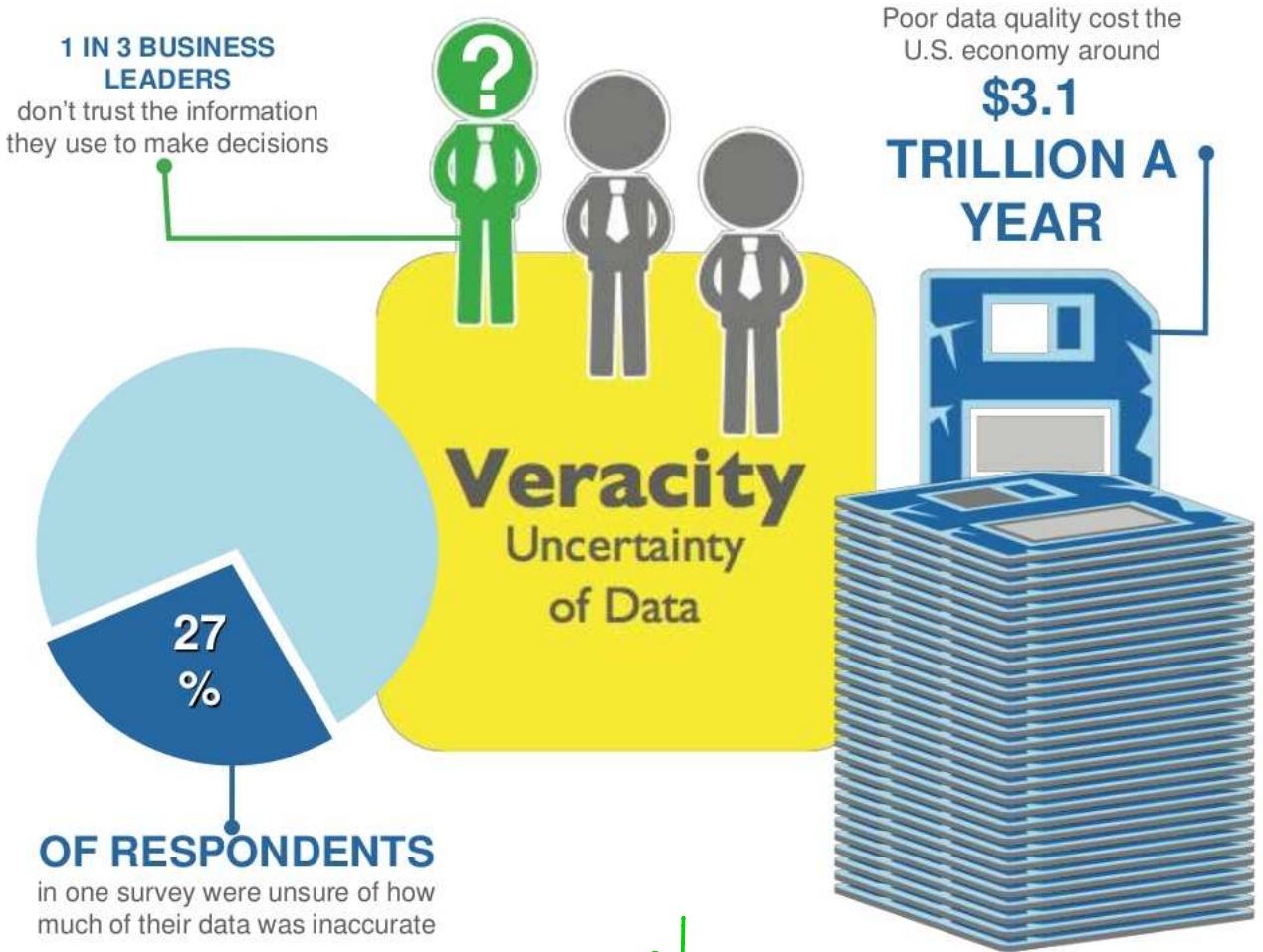
(WSO2 CEP), is an enterprise grade server that integrates to various systems to collect, analyse, and notify meaningful patterns in real time. The core back end runtime engine that powers WSO2 CEP 2.x server is WSO2 Siddhi which is a very high performing Java event processing engine A join query always has two Handler processors, one for each input stream it joins. Here, when an event from one stream reaches the In-Stream Join Processor, it is matched against all the available events of the other stream's Window Processor. When a match is found, those matched events are then sent to the Query Projector to create the output in-events; at the same time, the original event will be added to the Window Processor and it will remain there until it expires. Similarly, when an event expires from its Window Processor, it is matched against all the available events of the other stream's Window Processor; when a match is found, those matched events are sent to the Query Projector to create the output expired-events.

Note: Inspite of the optimizations, a join query is quite expensive when it comes to performance, and this is because the Window Processor will be locked during the matching process to avoid race conditions and to achieve accuracy in joining process; therefore, users should avoid matching huge windows in high volume streams. Based on the user scenario, using appropriate window sizes (by time or length) or using within keywords will help to achieve maximum performance. Pattern and sequence queries can have many Handler Processors; here, they will have a Handler Processor for each incoming event stream. After events are received by the Handler Processor, it passes them to the Inner Handler Processors; these Inner Handler Processors are responsible for processing the states in pattern and sequence queries. Here, the Inner Handler Processors contain all the events that are partially matched up to its state level, and when a new event arrives it tries to match whether it satisfies its Filter condition along with the partially matched events. If there is a match, it passes the corresponding previously matched events and the current event to the next state (Inner Handler Processor).

Graph and Linked Data

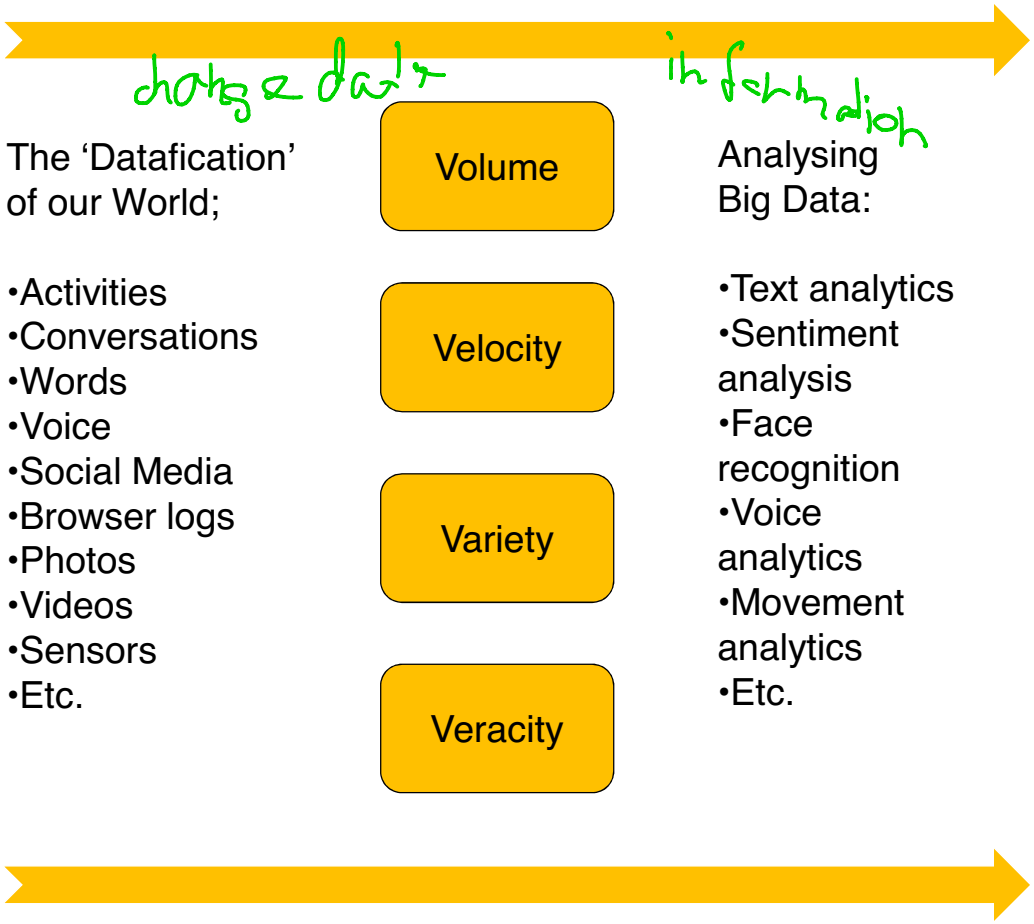


4th Character: Veracity



Turn Big Data Into Value

Big Data + Analytics = Value



The 'Datafication' of our World;

- Activities
- Conversations
- Words
- Voice
- Social Media
- Browser logs
- Photos
- Videos
- Sensors
- Etc.

Volume

Velocity

Variety

Veracity

Analysing Big Data:

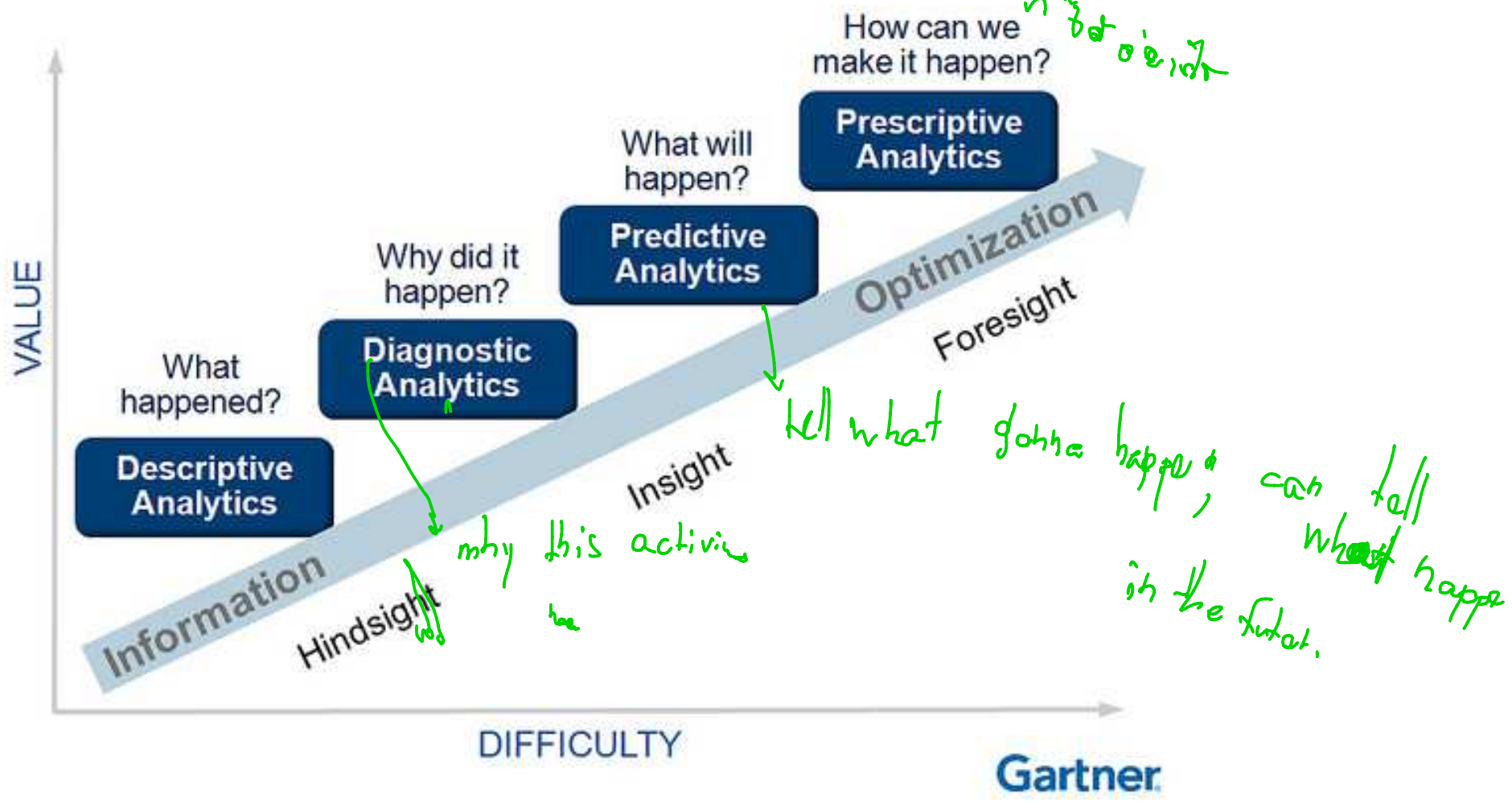
- Text analytics
- Sentiment analysis
- Face recognition
- Voice analytics
- Movement analytics
- Etc.

Product recommendation
Customer churn prevention

Value

Credit risk scoring
Cost reduction

Type of Data Analytics



Big Data Use Cases

Big Data Use Case

COMPANY
Mastercard

INDUSTRY
Finance

EMPLOYEES
67,000

TYPE
Behavioral Analytics

PURPOSE:

With 1.8 billion customers, MasterCard is in the unique position of being able to analyze the behavior of customers in not only their own stores, but also thousands of other retailers. The company teamed up with Mu Sigma to collect and analyze data on shoppers' behavior, and provide the insights it finds to other retailers in benchmarking reports.

tweet this

Big Data Use Case

Starbucks collects data on its customers' purchasing habits in order to send personalized ads and coupon offers to the consumers' mobile phones. The company also identifies trends indicating whether customers are losing interest in their product and directs offers specifically to those customers in order to regenerate interest.

Big Data Use Case

PURPOSE:

Shell uses sensor data to map its oil and gas wells in order to increase output and boost the efficiency of its operations. The data received from the sensors is analyzed by artificial intelligence and rendered in 3D and 4D maps.

Big Data Use Case

COMPANY
Union Pacific Railroad

INDUSTRY
Transportation

EMPLOYEES
44,000

TYPE
Predictive Support

PURPOSE:

With predictive analytics and tools such as visual sensors and thermometers, Union Pacific can detect imminent problems with railway tracks in order to predict potential derailments days before they would likely occur. So far the sensors have reduced derailments by 75 percent.

Big Data Use Case

PURPOSE:

Kayak uses big data analytics to create a predictive model that tells users if the price for a particular flight will go up or down within the next week. The system uses one billion search queries to find the cheapest flights, as well as popular destinations and the busiest airports. The algorithm is constantly improved by tracking the flights to see if its predictions are correct.

Big Data Use Case

PURPOSE:

Customer complaints and PR crises have become more difficult to handle thanks to social media. To better keep track of customer sentiment and what is being said about the company online, Nestle created a 24/7 monitoring centre to listen to all of the conversations about the company and its products on social media. The company will actively engage with those that post about them online in order to mitigate damage and build customer loyalty.

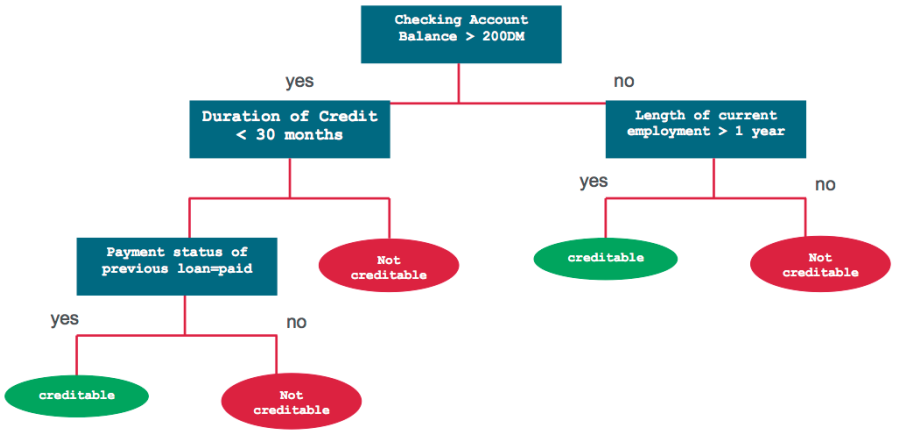
Big Data Use Case

Product Recommendation

Big Data Use Case

Label → creditable: 0 or 1
Features → {"balance", "duration", "history", "purpose", "amount", "savings",
"employment", "instPercent", "sexMarried", "guarantors", "residenceDuration",
"assets", "age", "concCredit", "apartment", "credits", "occupation", "dependents", "hasPhone", "foreign"}

creditability	balance	duration	history	purpose	amount	s
1.0	0.0	18.0	4.0	2.0	1049.0	
1.0	0.0	9.0	4.0	0.0	2799.0	
1.0	1.0	12.0	2.0	9.0	841.0	
1.0	0.0	12.0	4.0	0.0	2122.0	
1.0	0.0	12.0	4.0	0.0	2171.0	
1.0	0.0	10.0	4.0	0.0	2241.0	
1.0	0.0	8.0	4.0	0.0	3398.0	
1.0	0.0	6.0	4.0	0.0	1361.0	



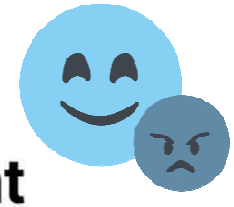
Predict Loan Credit Risk

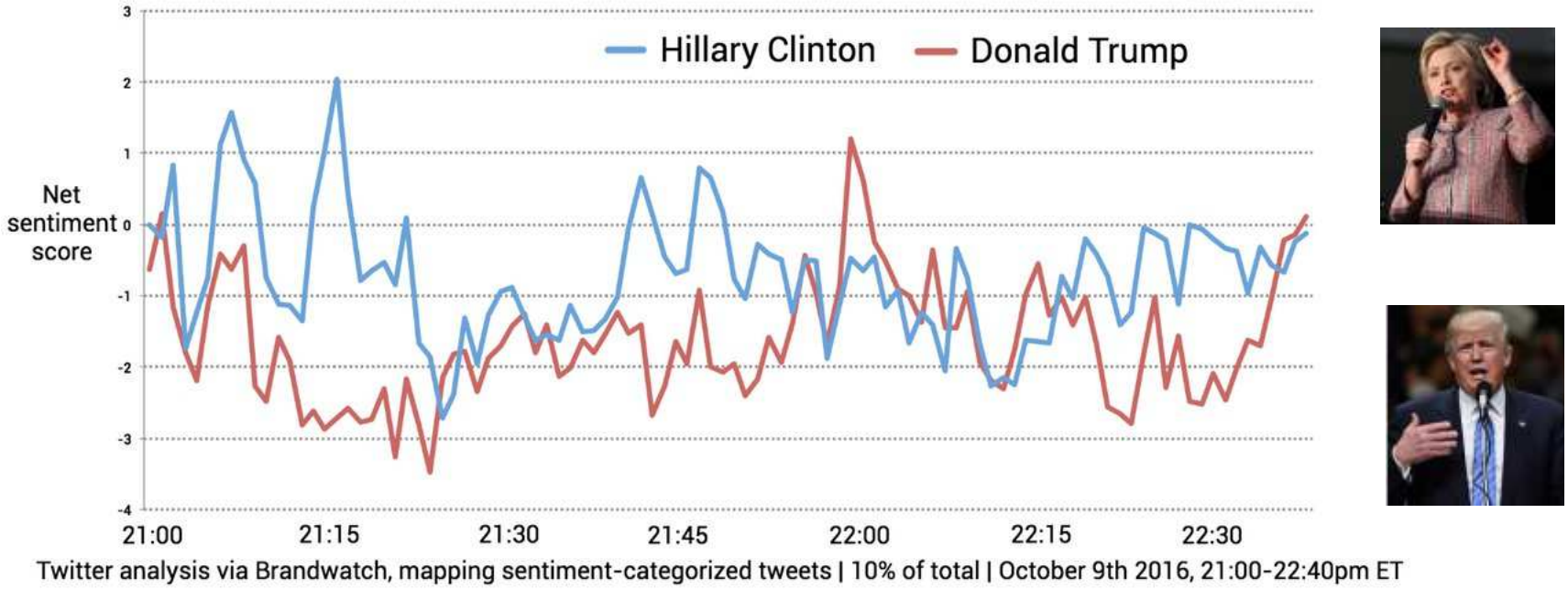
Big Data Use Case

Walmart combines public data, social data and internal data to monitor what customers and friends of customers are saying about a particular product online. The retailer uses this data to send targeted messages about the product, and to share discount offers. Walmart also uses data analysis to identify the context of an online message, such as if a reference to “salt” is about the movie or the condiment.

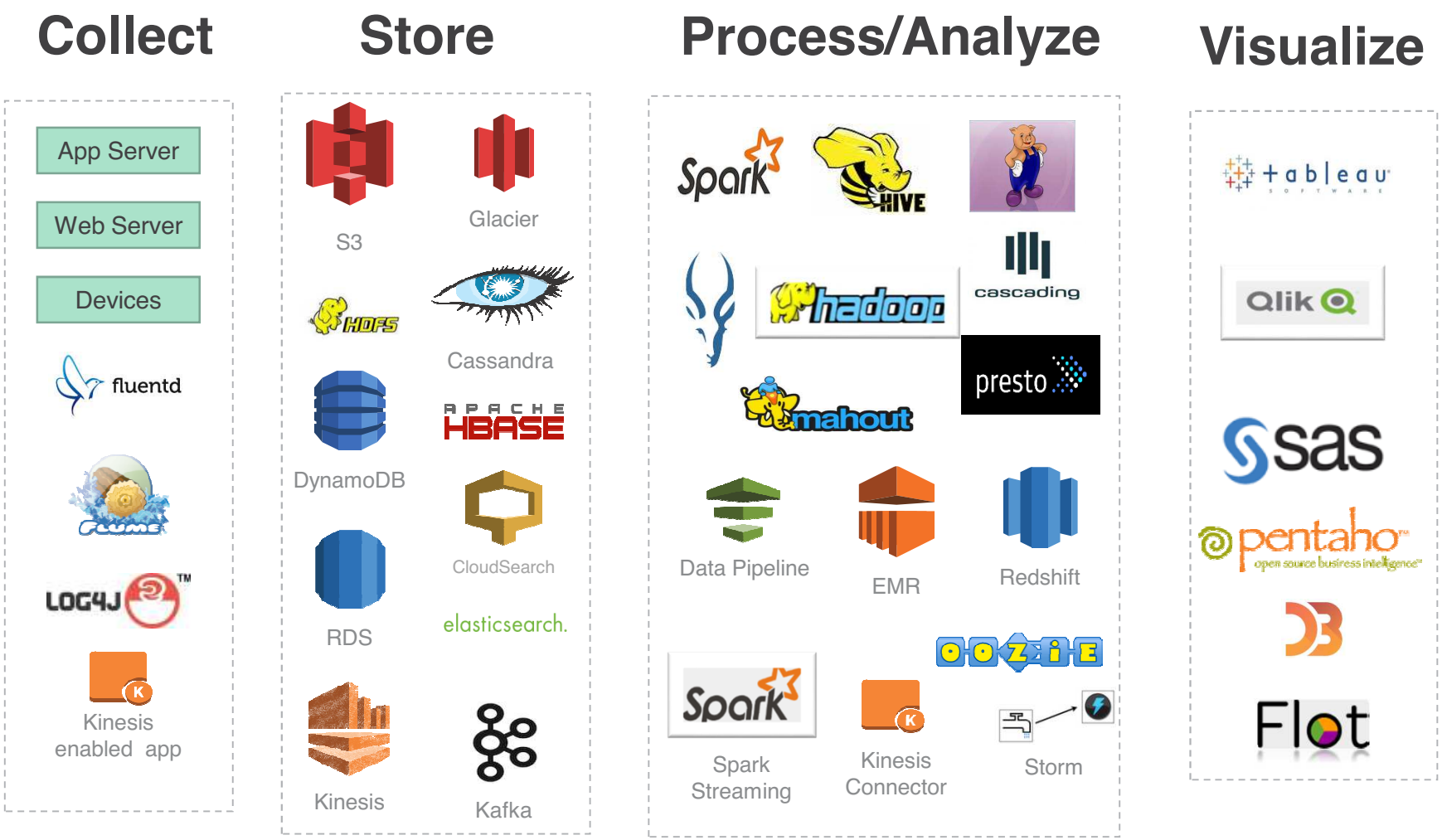
Big Data Use Case

- Analyzing tweets to determine whether they are most likely positive, negative or neutral





Big Data Tools

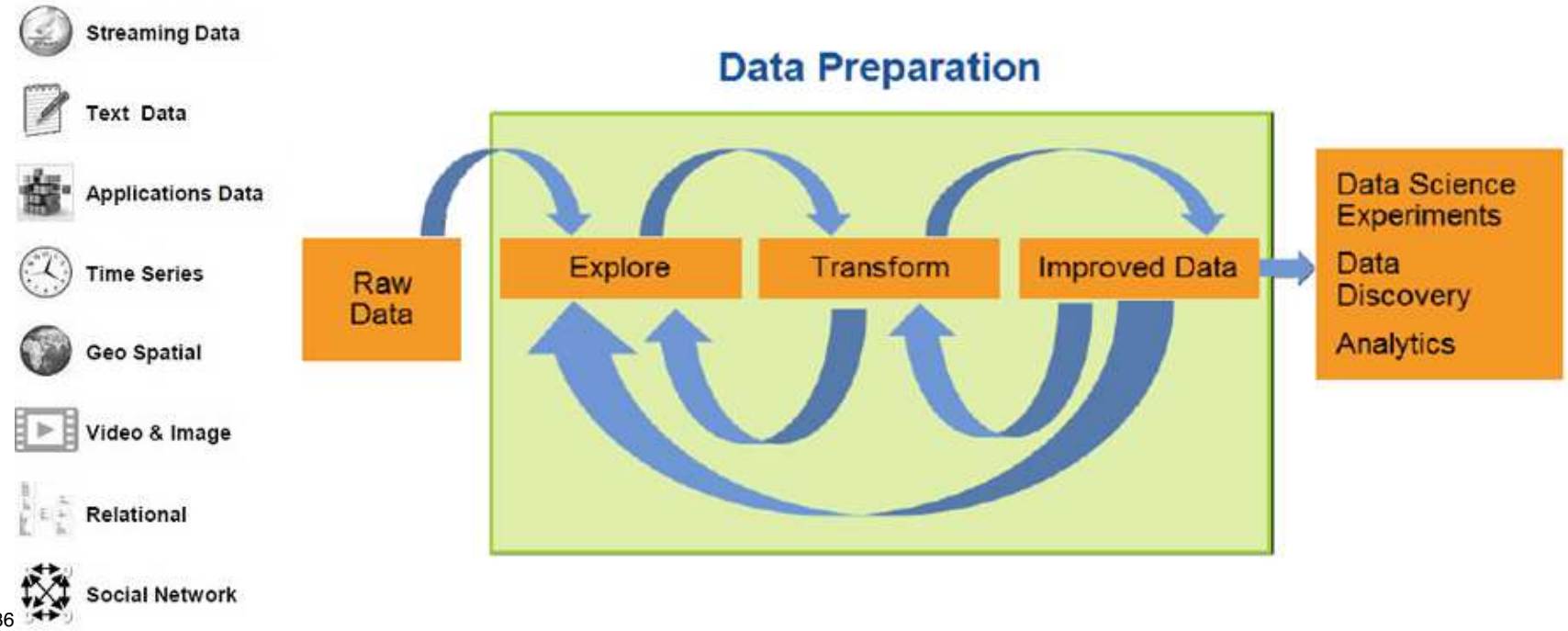


Big Data Tools

- Where data is **stored**?
  - Distributed Storage (e.g. Amazon S3)
- How data is **stored & indexed**?
  - Distributed File System (e.g. HDFS)
  - High-performance schema-free databases (e.g. MongoDB)
- Where processing is **hosted**?
  - Distributed Servers / Cloud (e.g. Amazon EC2)
- What **processes** are performed on data?
  - Analytic / Semantic Processing (e.g. Data Mining)
- What is the **processing model**?
  - Distributed Processing (e.g. MapReduce)
- How data is **viewed**?
  - Big data visualization (e.g. Polymaps)

Collecting

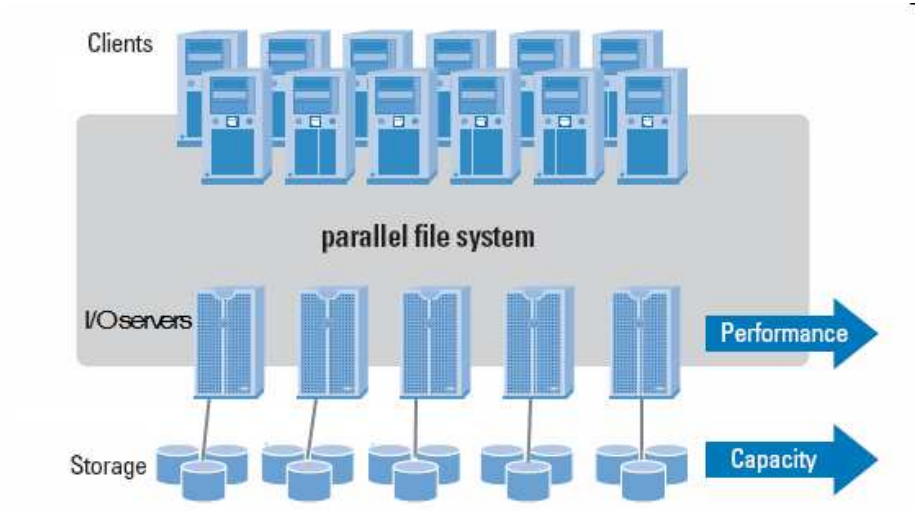
- Extract / Load / Crawl
- Data cleansing and preparation is a highly iterative and time-consuming process (~80% of the work on data analytics)



Garbage In, Garbage Out

Parallel/Distributed File Systems

- Storing unstructured data
- Permanently store data
- Divide data into logical units (files, blocks)
- Support concurrency
- Support replication provide reliable, performance.
- Example: CephFS, GlusterFS, HDFS, GPFS



RDMBS

- A relational database is a set of tables containing data fitted into predefined categories
- Each table contains one or more data categories in columns
- Each row contains a unique instance of data for the categories defined by the columns
- Use SQL as a query language

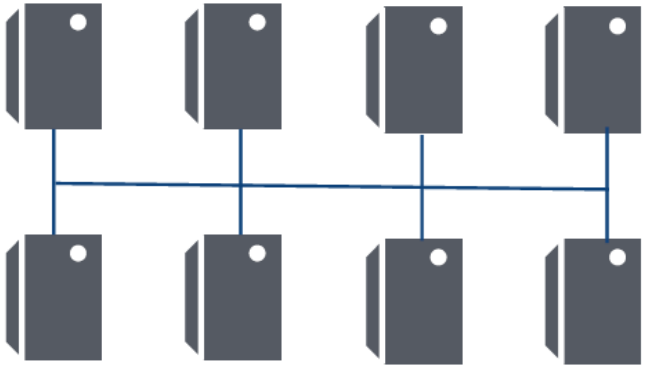
- RDBMS has a problem with Unstructured or Semi-Structured Data
- RDBMS were not designed to be highly distributed

NoSQL Databases

- NoSQL is a broad class of database that differs from the classic RDBMS in some significant ways, most important being they do not use SQL as their primary query language.
- NOSQL means **Not Only** SQL
- It's not about saying that SQL is dead
- It's about other database solutions are better suited for some problems
  - Semi-structured and unstructured data

NoSQL Characteristics

- NoSQL is all about scalability
- Scale-out architecture
- Deliver Heavy R/W workloads
- Schemaless
- High consistency not required
- There is no universal query language like SQL



- NoSQL Categories
  - Key Value stores
  - Document Databases
  - Column Databases
  - Graph Databases

MongoDB

- Open source
- Document-oriented storage
- Full index support
- Replication & high availability
- Auto-Sharding
- Simple query language
- Fast in-place updates
- MapReduce functionality
- Scalable

DB rankings

357 systems in ranking, November 2019

Rank Nov 2019	Rank Oct 2019	Rank Nov 2018	DBMS	Database Model	Score Nov 2019	Score Oct 2019	Score Nov 2018
1.	1.	1.	Oracle	Relational, Multi-model	1336.07	-19.81	+34.96
2.	2.	2.	MySQL	Relational, Multi-model	1266.28	-16.78	+106.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1081.91	-12.81	+30.36
4.	4.	4.	PostgreSQL	Relational, Multi-model	491.07	+7.16	+50.83
5.	5.	5.	MongoDB	Document, Multi-model	413.18	+1.09	+43.70
6.	6.	6.	IBM Db2	Relational, Multi-model	172.60	+1.83	-7.27
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	148.40	-1.77	+4.94
8.	8.	↓ 7.	Redis	Key-value, Multi-model	145.24	+2.32	+1.06
9.	9.	9.	Microsoft Access	Relational	130.07	-1.10	-8.36
10.	10.	↑ 11.	Cassandra	Wide column	123.23	+0.01	+1.48

From http://db-engines.com/en/ranking

Document Databases

- Each record is called a “Document”
- Documents are semi-structured data, encoded in a standard data exchange format such as JSON (JavaScript Object Notation)
- An example record from MongoDB, might look like

```
{
“_id” : ObjectId(“4fccbf281168a6aa3c215443”),

“first_name” : “Thomas”,
“last_name” : “Jefferson”,
“address” : {
      “street” : “1600 Pennsylvania Ave NW”,
      “city” : “Washington”,
      “state” : “DC”
    }
}
```

Document Data Model

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	Rome

CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bently	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

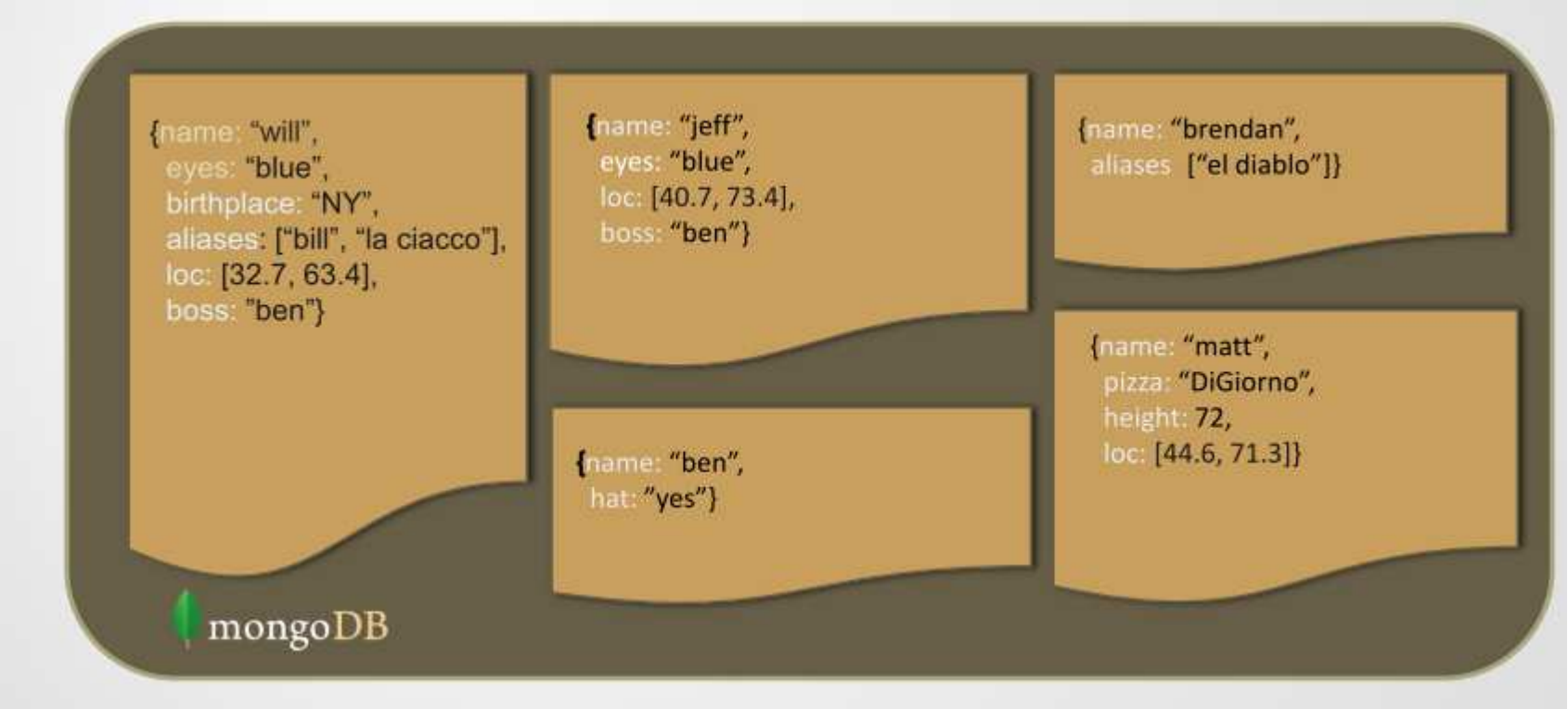
NO RELATION

MongoDB

```
{
  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, … }
  ]
}
```

Schema Free

- MongoDB does not need any pre-defined data schema
- Documents in the same table could have different data



Query Language

- CRUD Operations – Create, Read, Update, Delete
- MongoDB Interactive shell

./bin/mongo

>

- **Case sensitive**
- Some useful commands
  - db To check which database you're using
  - show dbs Show all databases
  - use <dbname> Switch db's/make a new one
  - show collections See what collections exist

From SQL to MongoDB

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Collection is not strict about what it Stores

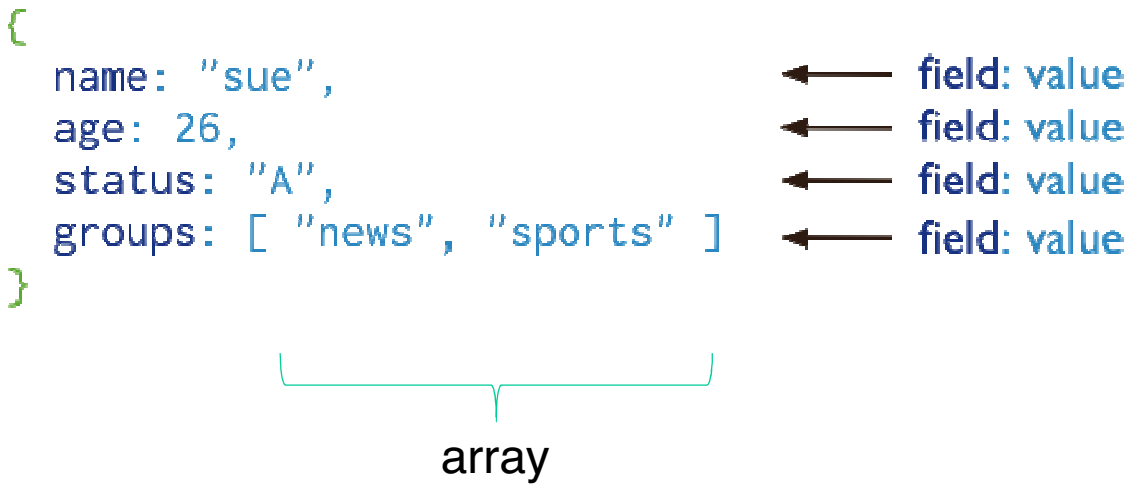
Schema-less

Hierarchy is evident in the design

Embedded Document ?

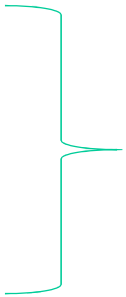
BSON Document

- MongoDB stores data in the form of BSON documents, which are JSON-like field and value pairs.
- Both keys and values are fully searchable
- The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.



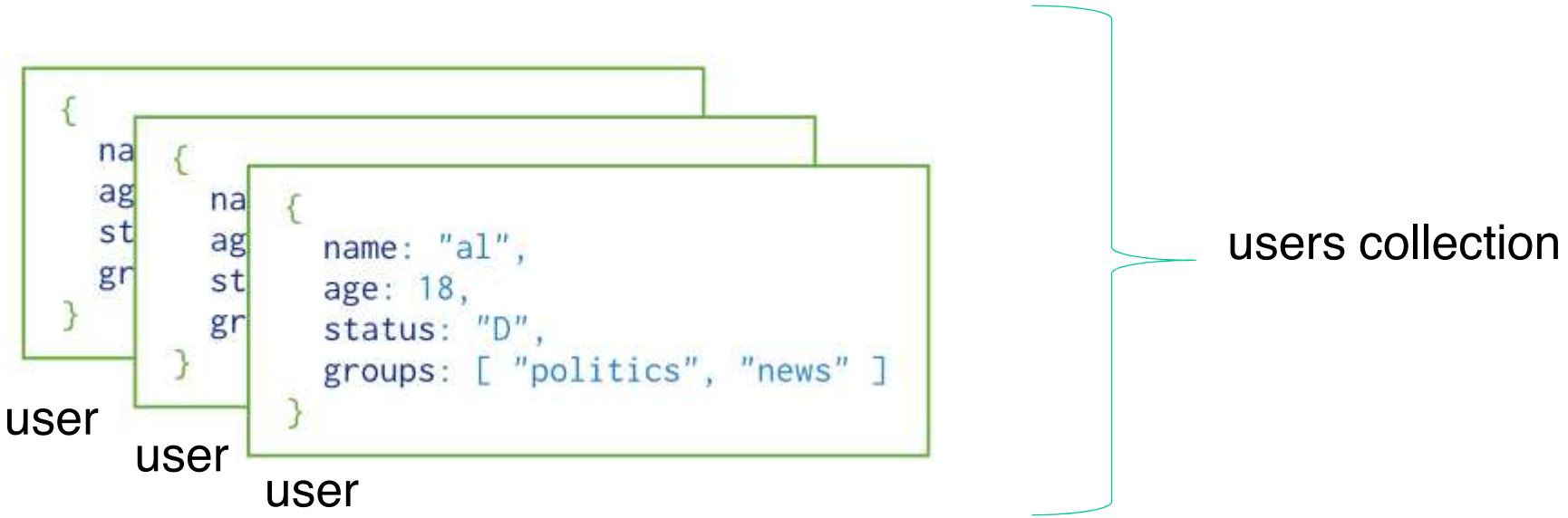
Embedded Document

```
{
   name: “Joe",
   address:
      {
         street: "123 Fake Street",
         city: "Faketon",
         state: "MA", zip: 12345
      }
}
```



Embedded Document

Documents and Collections



Create a Document

Collection

Document

```
db.users.insert(
                {
                  name: "sue",
                   age: 26,
                status: "A",
                groups: [ "news", "sports" ]
                }
               )
```

Document

```
{
  name: "sue",
  age: 26,
  status: "A",
  groups: [ "news", "sports" ]
}
```

Collection

{ name: "al", age: 18, ... }
{ name: "lee", age: 28, ... }
{ name: "jan", age: 21, ... }
{ name: "kai", age: 38, ... }
{ name: "sam", age: 18, ... }
{ name: "mel", age: 38, ... }
{ name: "ryan", age: 31, ... }
{ name: "sue", age: 26, ... }

Create a Document

```
> db.users.insert(
...    {
...       name: "sue",
...       age: 26,
...       status: "A“,
...       groups: ["news", "sports"]
...    }
... )
WriteResult({ "nInserted" : 1 })
```

Multi-line Input

Output

```
db.users.insert({name: "sue", age: 26, status: "A", groups: ["news",
"sports"]} )
WriteResult({ "nInserted" : 1 })
```

Create a Document

- **insert()** is the primary method to insert document(s) into MongoDB collection.
- If the collection does not exist, insert() creates the collection during the 1st insert.
- The created document contains an **_id** field with the generated **ObjectId** value

```
{ "_id" : ObjectId("562bae7b696d40e58758b8a1"), "name" : "sue", "age" : 26,
    "status" : "A", "groups" : [ "news", "sports" ] }
```

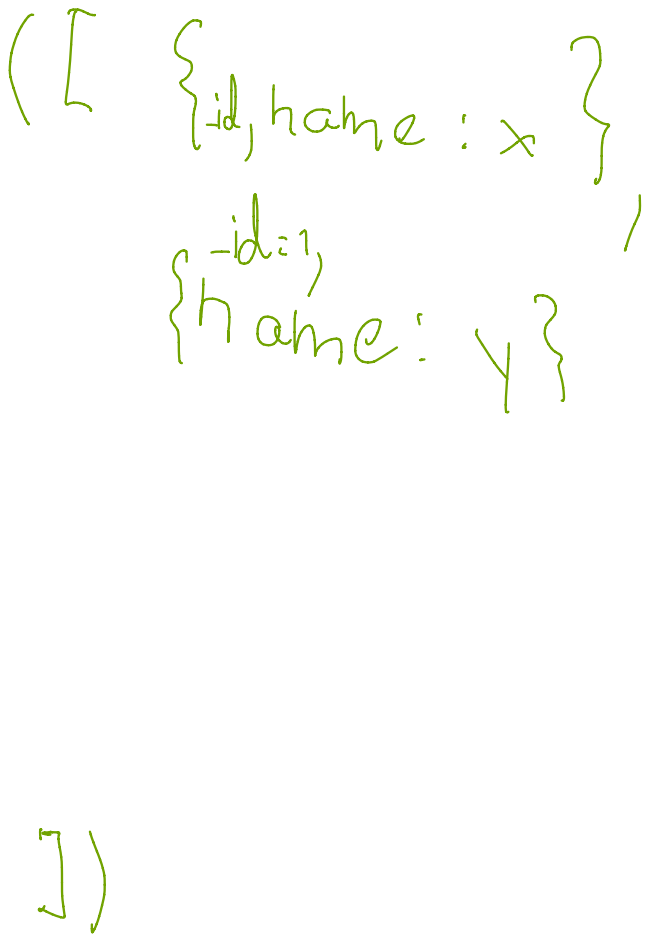
- You can insert a document with specifying _id field. The value must be unique within the collection

```
db.users.insert({ _id: 1, name: "sue", age: 26, status: "A", groups: ["news", "sports"]
    } )
```

Create Multiple Documents

- Insert an array of documents

```
db.users.insert([
  { _id: 3,
    name: "Damon",
    address:{street:"Kastorias",
             city: "Athens",
             zipCode:1534},
    age:35
  },
  { _id: 4,
    name: "John",
    status: "A"
  }
])
```



Read

- Return all documents in a collection

SQL

```
SELECT * FROM users
```

MongoDB

```
db.users.find()

db.users.findOne()    # return the first document
```

- Count

```
db.users.count();
```

Read

- Distinct

SQL

```
SELECT DISTINCT(name) FROM users;
```

MongoDB

```
db.users.distinct("name");
```

- To select documents that match selection criteria

SQL

```
SELECT * FROM users WHERE age=30
```

MongoDB

```
db.users.find({age:30})
```

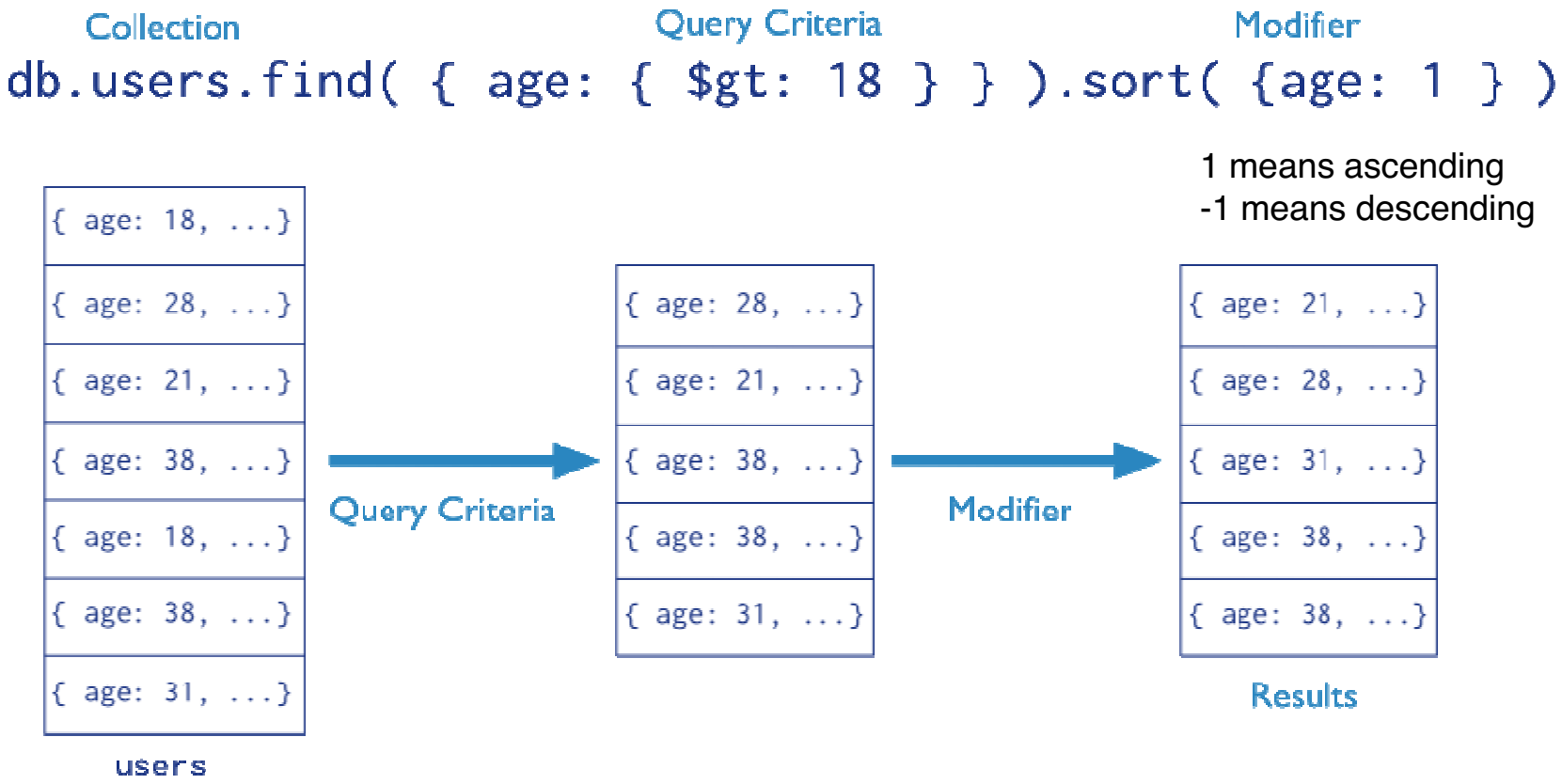
Conditional Operators

WHERE key Condtion value .find({key:{Operator:value})

SQL Conditions	MongoDB Operators
key > value	key:{$gt:value}
key < value	key:{$lt:value}
key >= value	key:{$gte:value}
key <= value	key:{$lte:value}
key <> value	key:{$ne:value}
key IN(value)	key:{$in:[value]}
key NOT IN(value)	key:{$nin:[value]}
key EXIST(value)	key:{$exist:value}

Read

Find the users of age greater than 18 and sort by age.



Read

- Select certain fields

SQL

```
SELECT name, age  FROM users where age = 30
```

MongoDB

```
db.users.find({age:30}, {name:1,age:1})
```

1 means include
0 means exclude

- Query fields in embedded documents

```
db.users.find({"address.city" : "Athens"})
```

- Query with arrays

```
db.users.find({groups: "sports"})
```

Read

- Match an embedded/nested document

```
db.users.find({address : {street:"Kastorias",
                          city: "Athens",
                          zipCode:1534}} )
```

Equality matches on the whole embedded document require an exact match of the specified document, including the field order. The following doesn't match.

```
db.users.find({address : {street:"Kastorias",
                          zipCode:1534,
                          city: "Athens"}} )
```

Logical Operators

- AND conditions

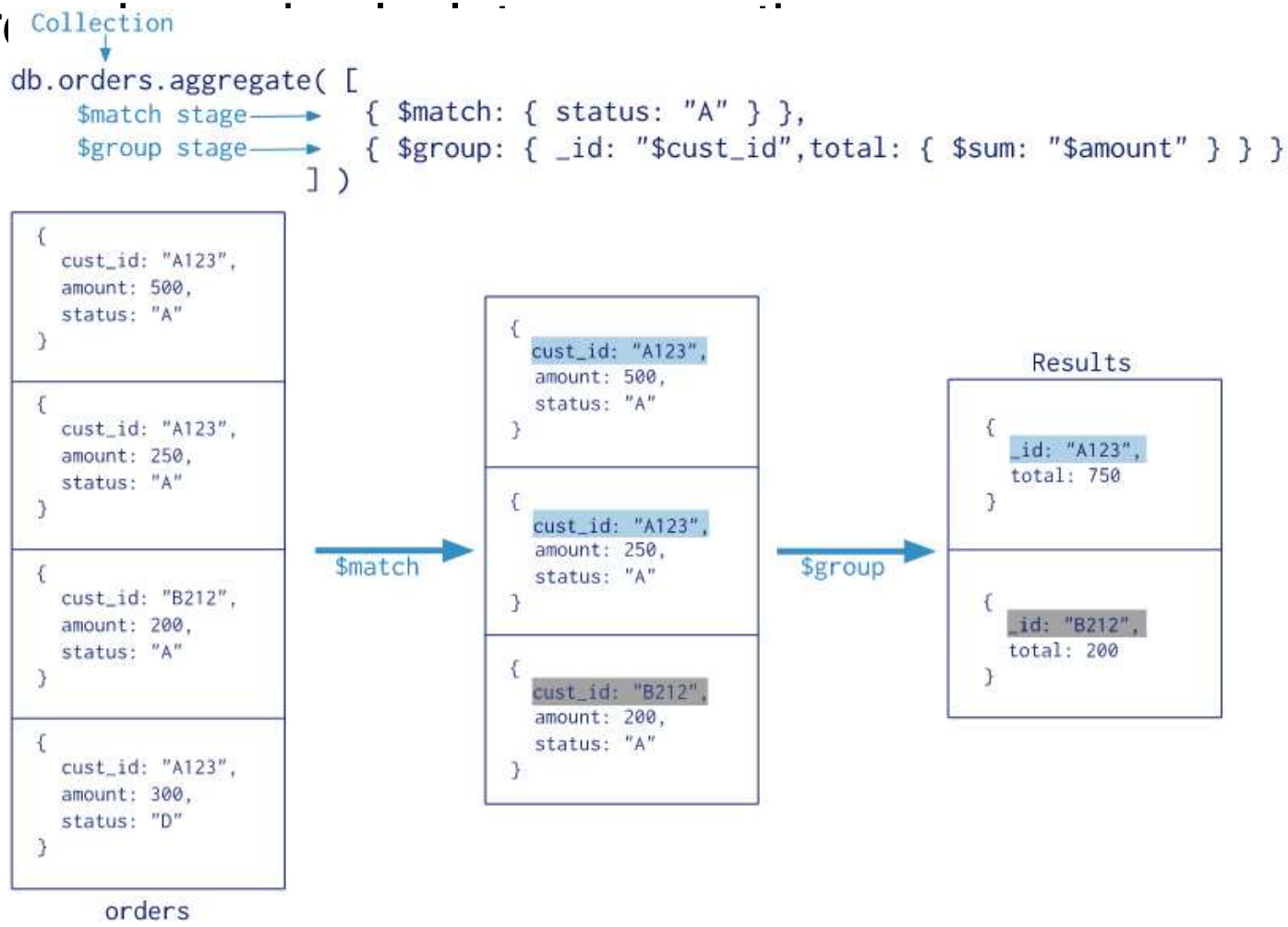
```
db.users.find( { status: “A”, age: { $lt: 30 } } )
```

- OR conditions

```
db.users.find({
    $or: [ { status: “A”}, { age: { $lt: 30 } } ]
 })
```

Aggregation Operations

- Gr



Aggregation Operations

- The _id field is *mandatory*; or the value of *null* for all documents
- Support $sum, $min, $max, and $avg operators

- With sort

```
db.orders.aggregate([
                { $match: { status: "A" } },
                { $group: { _id: "$cust_id", total: { $sum: "$amount" } } },
                { $sort: { total: -1 } }
               ])
```

Can have multiple fields

- The expression { $match: ... } can be omitted if all orders are used

Update

Update the users of age greater than 18 by setting the status field to A.

SQL

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

MongoDB

```
db.users.update(                      <-- collection
   { age: { $gt: 18 } },              <-- update criteria
   { $set: { status: "A" } },         <-- update action
   { multi: true }                    <-- update option
)
```

do multiple doc.

Delete

Delete the users with status equal to D.

SQL

```
DELETE FROM users      <-- table
WHERE  status = 'D'    <-- delete criteria
```

MongoDB

```
db.users.remove(       <-- collection
   { status: "D" }     <-- remove criteria
)
```

Delete

- Remove All Documents

  db.users.remove({})

- Remove Documents that Matches a Condition

  db.users.remove({age:30})

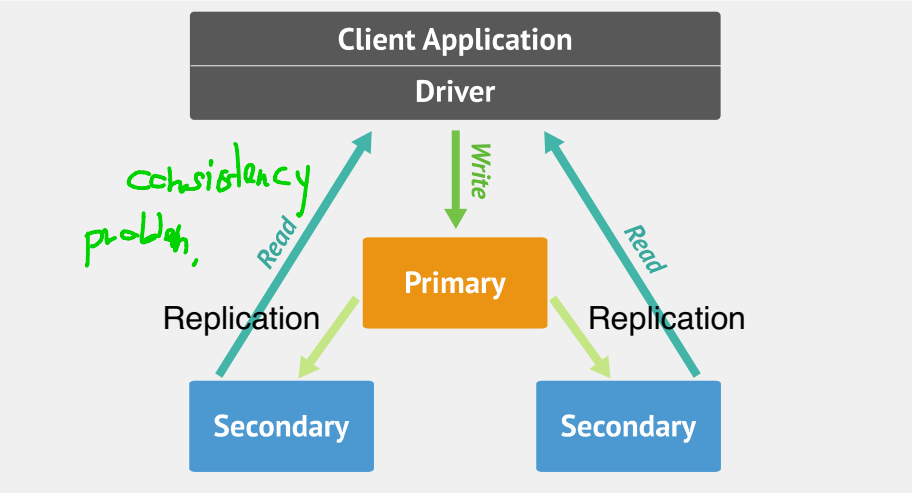
Indexing

- Create an index on a single field
  db.users.createIndex({name: 1})

- Create a compound index
  db.users.createIndex({name: 1, age: 1})

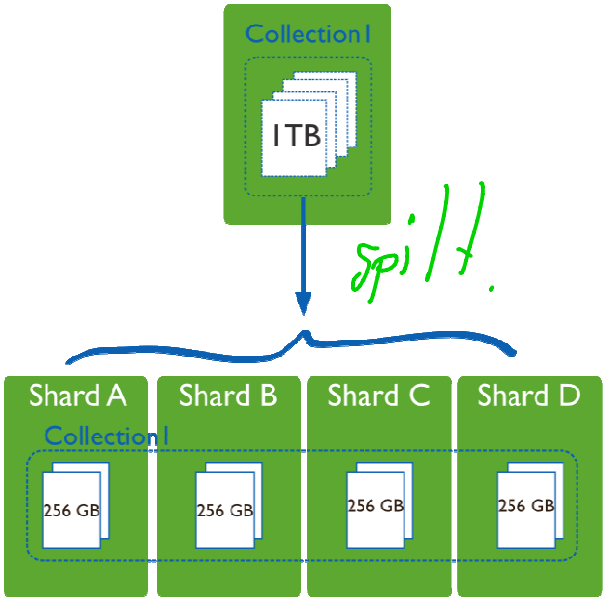
MongoDB Replication

- Keep copies of data on difference servers
- **Replication set** is a group of servers: one primary and many secondary servers
- Client can reads/writes to the primary server but only reads from secondary servers
- Replication reduces response time and downtime
- Recommended for productions



MongoDB Sharding

- Sharding is a method for storing data across multiple machines
- MongoDB uses sharding to support deployments with very large data sets

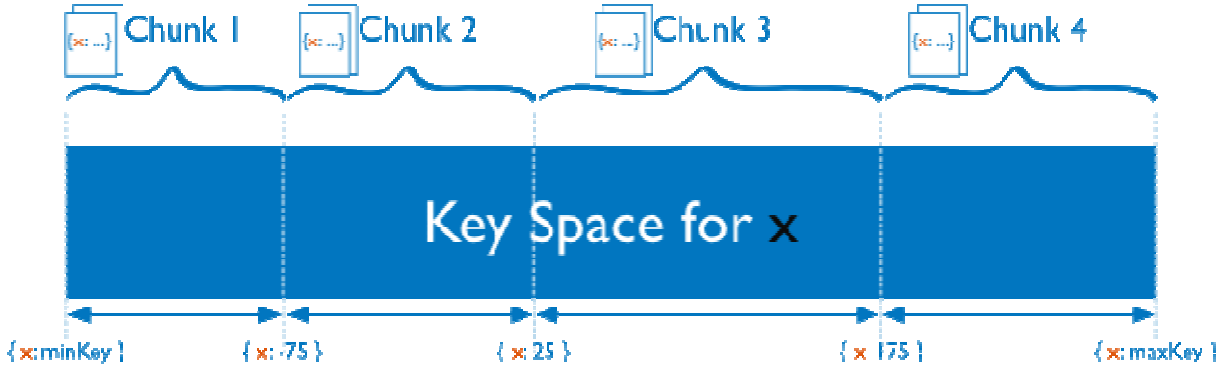


Sharding Methods

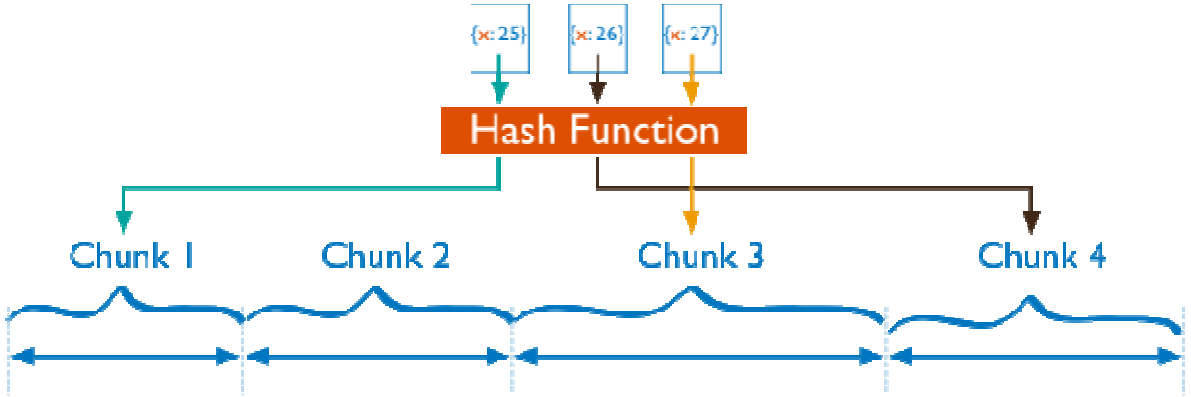
- MongoDB supports two sharding methods
- **Ranged sharding** involves dividing data into ranges based on the shard key values. Each chunk is then assigned a range based on the shard key values
- **Hashed Sharding** involves computing a hash of the shard key field's value. Each chunk is then assigned a range based on the hashed shard key values

histogram.

Range VS Hashed Sharding



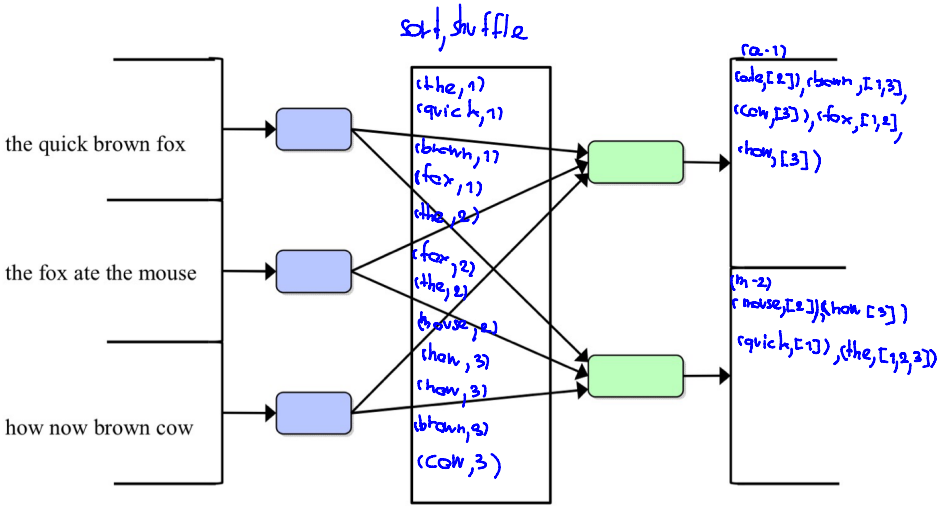
Specify the range



Selecting Big Data stores

- Different storage for each data type
- Choosing the correct data stores based on your data characteristics
- Implementing polyglot data store solutions
  - There is no best one
  - All databases are welcome
  - http://nosql-database.org


Q
A


the quick brown fox
the fox ate the mouse
how now brown cow