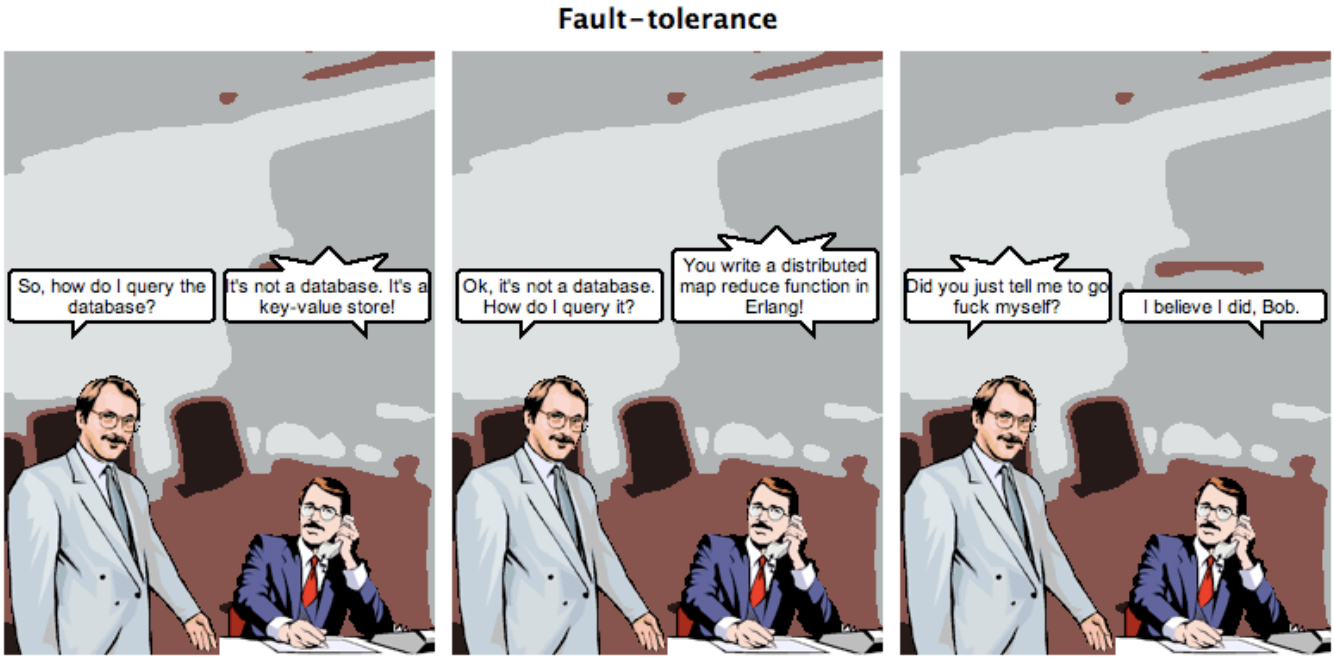
Introduction to NoSQL

Nicolas Travers
CNAM – France

Schedule & Organization

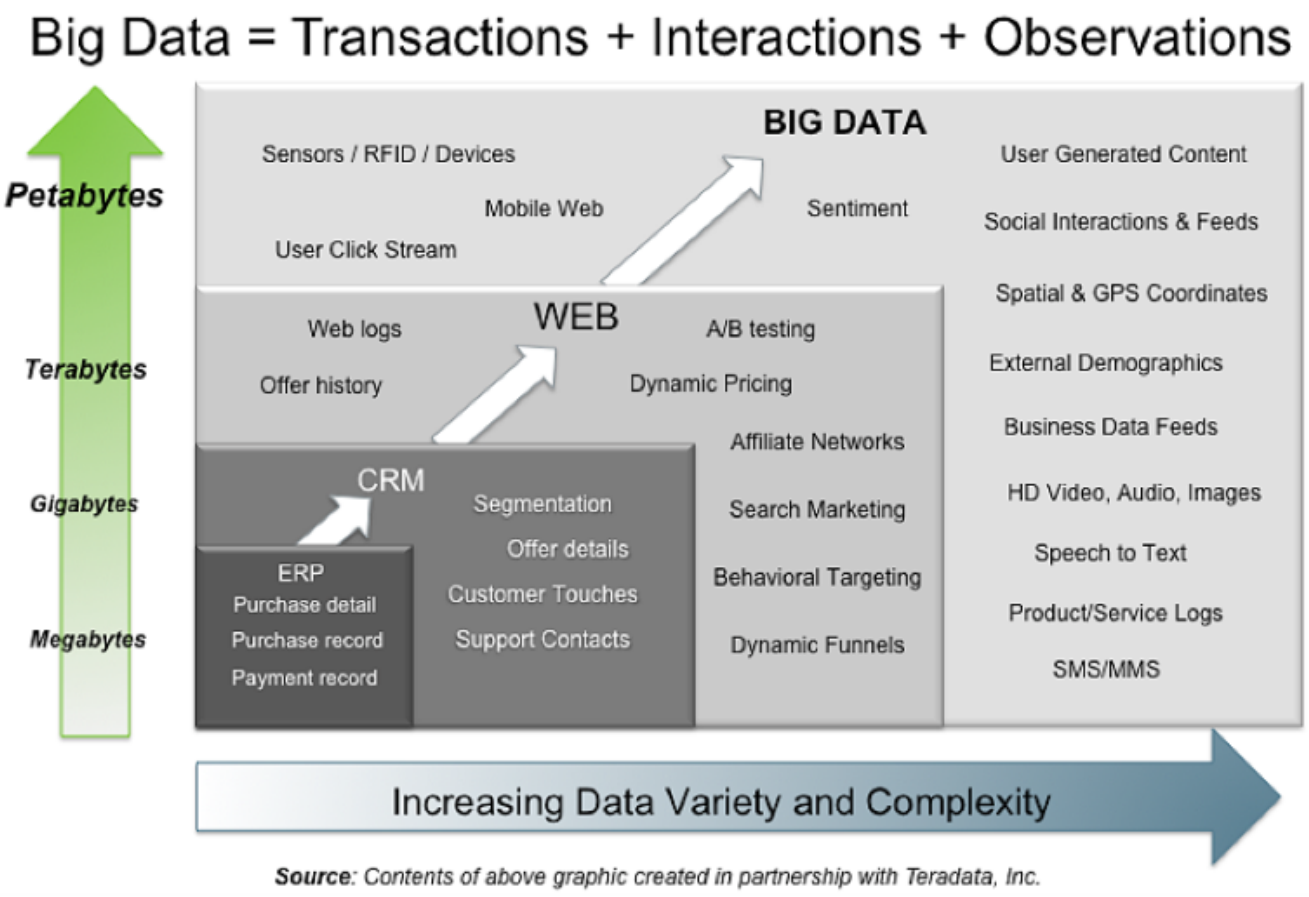
- Introduction to **NoSQL** databases
  - 3V, ACID vs BASE, families, CAP theorem, JSon
- Presentation of **MongoDB**
  - Language, distribution, replication, application
- **Practice Works** on MongoDB
  - Queries : find + aggregate

DBMS vs NoSQL



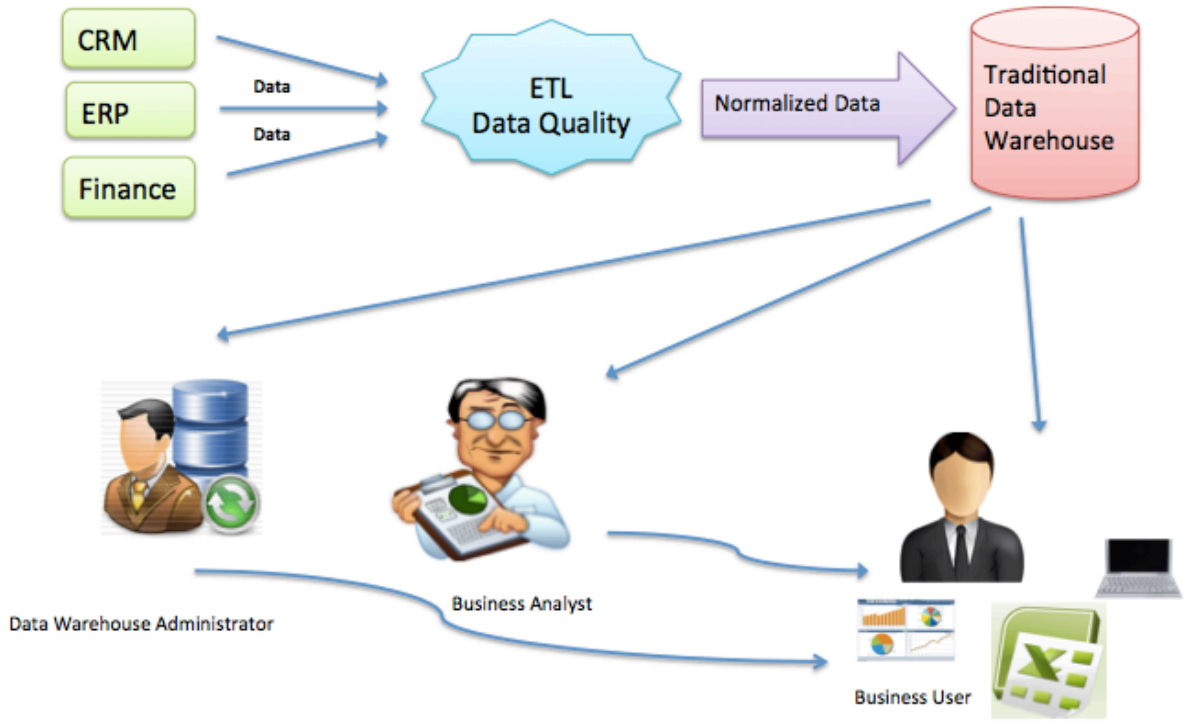
Context

- Applications and Web platforms
  - Exponential growth of the amount of Data (x2 / 2 years)
  - Unprecedent management of this volume
    - Need to distribute both computation and data
    - Huge number of servers
    - Heterogeneous data, maybe complex and often linked
- Ex :
  - Google, Amazon, Facebook
  - Google DataCenter :
    - 5000 servers/data center, ~1M de servers
  - Facebook :
    - 1 PetaBytes of data



***Source**: Contents of above graphic created in partnership with Teradata, Inc.*

BI : Traditional methods



Decisional vs 3V

Incompatible classical approach with the **3V** of *BigData* :

- **Volume**: Designed to store GB/TB of data, but needs PB (maybe EB).
- **Variety**: Heterogeneous and variable types of data, text, semi-structured
- **Velocity**: Data are produced more and more quickly

DBMS: Limitations

- Standard databases
  - Functionalities
    - Joins between tables
    - Complex queries
    - Strong coherency of data

➢ Requirements in a distributed context
  - Links between entities => same server
  - ++ links => difficulties for data organization

DBMS: Limitations (2)

- **ACID** properties for transactions
  - Set of operations
  - **Atomicity** (integral completion or none)
  - **Consistency** (consistent at start and end)
  - **Isolation** (no communication between them)
  - **Durability** (an operation cannot be reversed)
  - Pessimistic view on consistency

➢ Requirements in a distributed context
  - Difficulties in insuring those properties
  - Conflict with efficiency / performances

ACID vs BASE

- Modern systems use the BASE model
  - Optimistic view on consistency
  - ***Basically Available***:
    - Any request => An answer
    - Even in a changing state
  - ***Soft State***:
    - Opposite to Durability.
    - System's state (servers or data) could change over time (without any update)
  - ***Eventually consistent***:
    - With time, data can be consistent
    - Updates have to be propagated

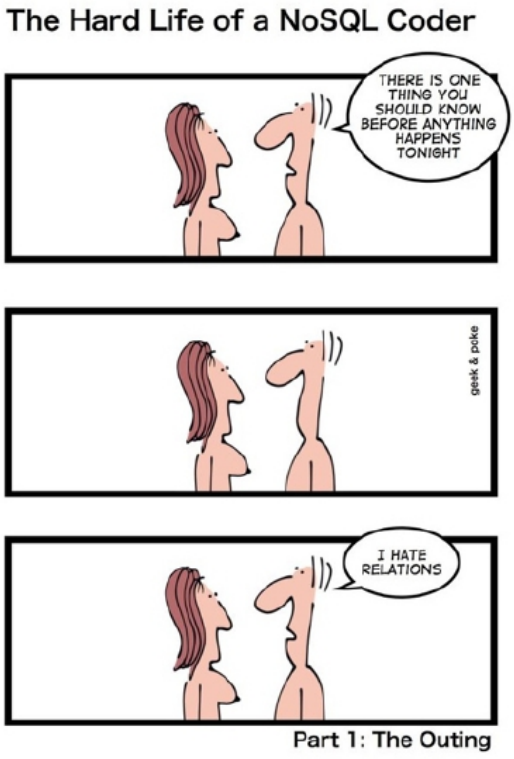
Solution: NoSQL

- NoSQL : **Not Only SQL**
  - New data storage/management approach
  - Scales up the system (through distribution)
  - Complex metadata management
    - No schema

➢ Do not substitute DBMS, dedicated to:

  - Very huge volume of data (PetaBytes)
  - Very short response time
  - Consistency is not mandatory

Databases and NoSQL



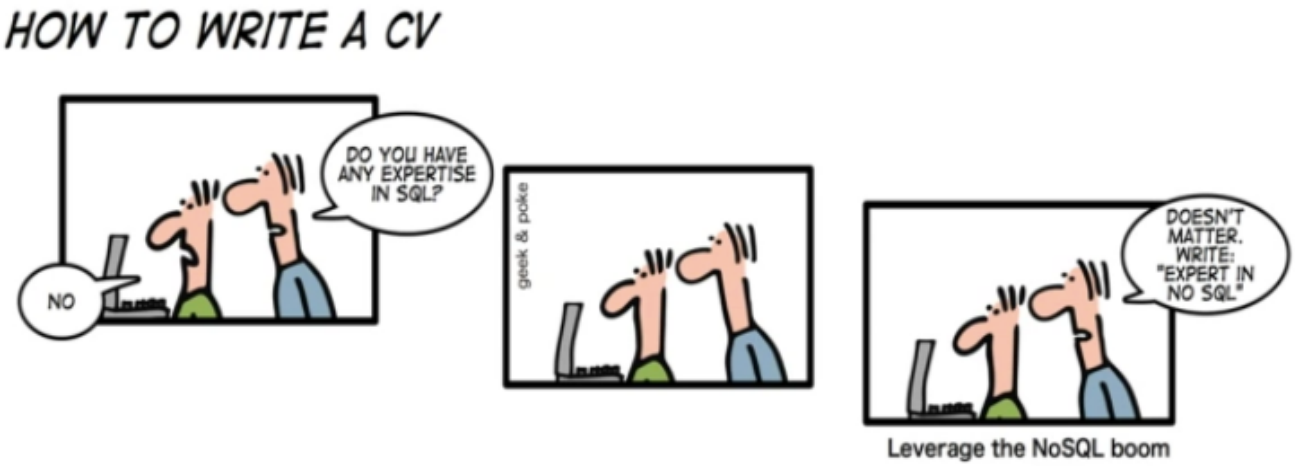
NoSQL DB: Characteristics

- **No relations** => *Collections*
  - No fix structures (nay none)
- **Complex data** (e.g. *documents*)
  - Objects, nesting, arrays
- **Data distribution**
  - High parallelization (Map/Reduce)
- **Data replication**
  - Disponibility vs Consistency (no transactions)
  - Few writes, many reads

Sharding : Scalability

- Datablocks are distributed in a cluster of servers
- Horizontal partitioning
- 3 types of technics:
  1. Resource allocation based: *HDFS*
  2. Tree-based structure: *Clustered index* (sort)
  3. Hash-based structure: *Consistent Hashing*

NoSQL Systems



Several NoSQL systems

- **Key-Value Store**
  - Data are identified by a unique key (used for querying)
  - *DynamoDB, Voldemort, Redis, Riak, MemcacheDB*
- **Column data**
  - Relation 1-n “one-to-many” (messages, posts)
  - *HBase, Hypertable, Spark, Elasticsearch*
- **Documents**
  - Complexes data, attributes/values
  - *MongoDB , Cassandra, CouchDB, Terrastore*
- **Graphs**
  - Highly connected entities, Social Networking
  - *Neo4j, OrientDB, FlockDB*

I - NoSQL & Key-Value store

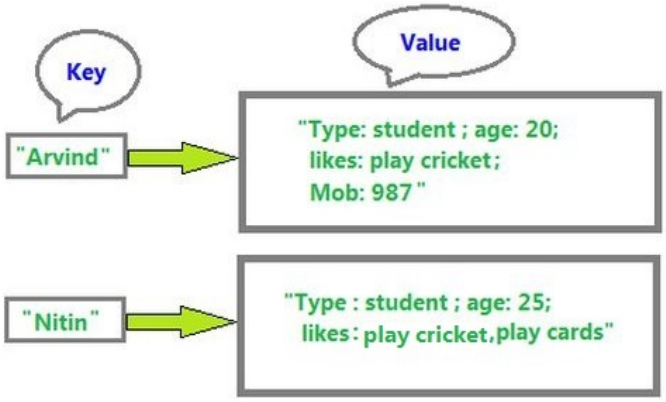
- Similar to a distributed "*HashMap*"
- Key + Value
  - No fixed schema on values (strings, object, integer, binaries…)
- Drawbacks:
  - No structures nor typing
  - No structured-based queries

➢ DynamoDB (Amazon), Redis (VMWare), Voldemort (LinkedIn)

I - NoSQL & Key-Value store (2)

- CRUD Operations (HTTP)
  - Create(key,value)
  - Read(key)
  - Update(key,value)
  - Delete(key)
- Horizontal scaling (partitionning/distribution)
- No vertical distribution (data segmentation)

I - NoSQL & Key-Value store (3)



II – NoSQL & Columns

- Column-based storage
  - DBMS: tuples (lines)
- Easy to insert a new column
  - Dynamic schema

➢ BigTable/Hbase (Google), Cassandra (Facebook&Apache), SimpleDB (Amazon)

II – NoSQL & Columns (2)

- **Advantages**:
  - XML/JSon support
  - Column indexing
  - Horizontal scaling
- **Drawbacks**:
  - Hard to query complex data
  - Difficult for linked data (distances, paths, time)
  - Pre-defined queries (not on the fly)

II – NoSQL & Columns (3)

Row Oriented
(RDBMS Model)

Multi-valued

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

null

Column Oriented
(Multi-value sorted map)

id	Name
1	Ricky
2	Ankur
3	Sam

id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

III – NoSQL & Documents

- Based on the key-value store
  - Add semi-structured data (JSon/XML)
- HTTP API
  - More complex than CRUD

➢MongoDB, CouchDB (Apache), RavenDB, Terrastore

III – NoSQL & Documents (2)

- Document management
  - Simple types (Int, String, Date)
  - No fix schema (docs may vary)
  - Nested data
- **Advantages:**
  - Richness for queries
  - Indexing several attributs
  - Easy to scale up
- **Drawbacks:**
  - Difficulties for data interconnexions
  - Dedicated to key-value (id)

III – NoSQL & Documents (3)

Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.

Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

IV – NoSQL & Graph

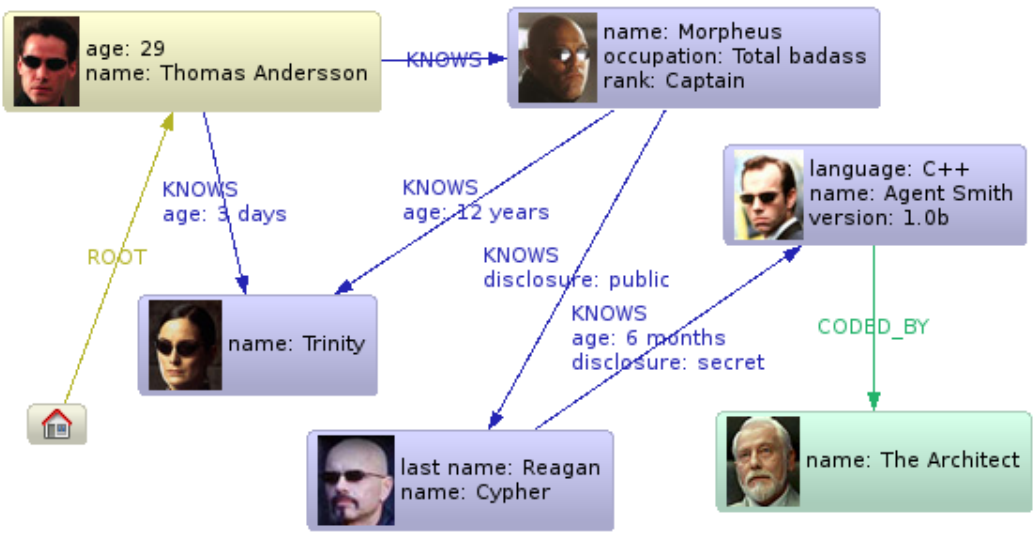
- Storage: nodes, relations and properties
  - Graph Theory
  - Path querying on the graph
  - Data are loaded on demand
  - Difficulties for modeling

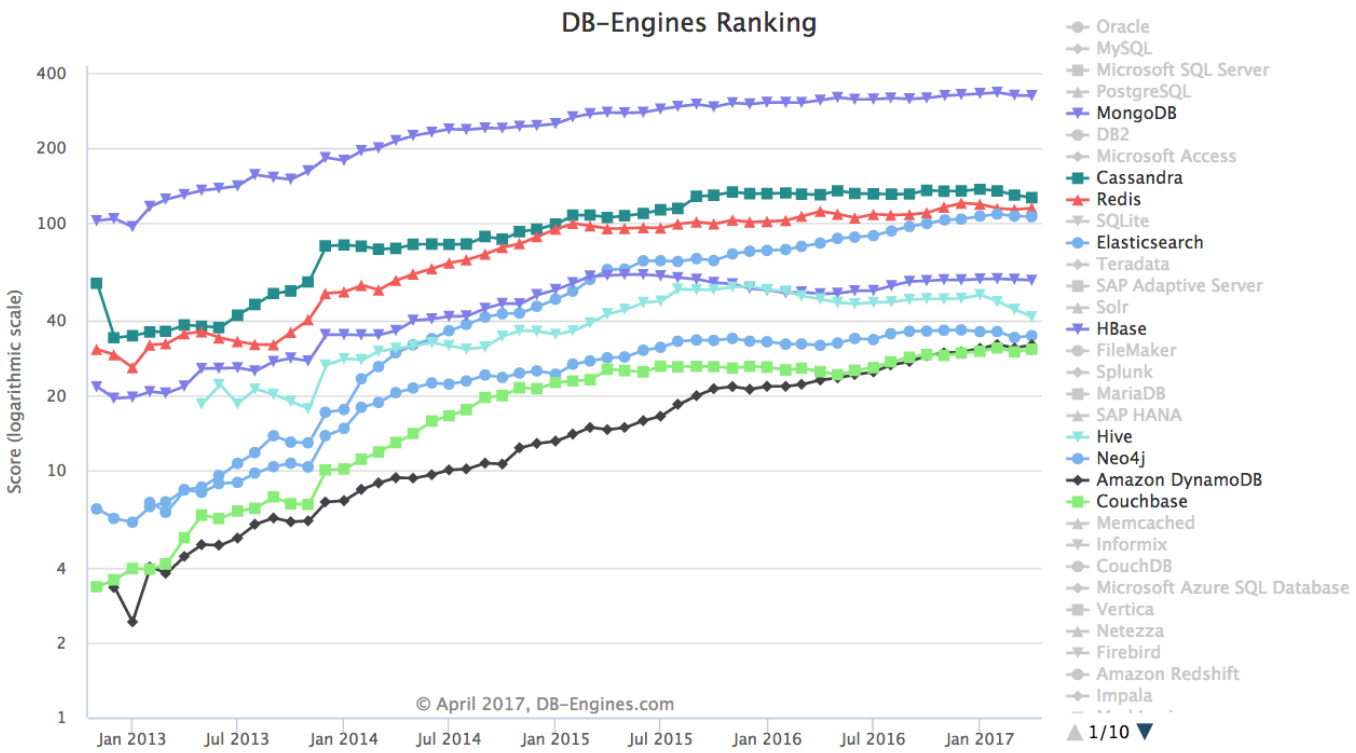
➢ Neo4j, OrientDB (Apache), FlockDB (Twitter)

IV – NoSQL & Graph (2)

- Available storage
  - Object (cf. documents)
  - Edges (with properties)
- Difficult for Sharding

IV – NoSQL & Graph (3)

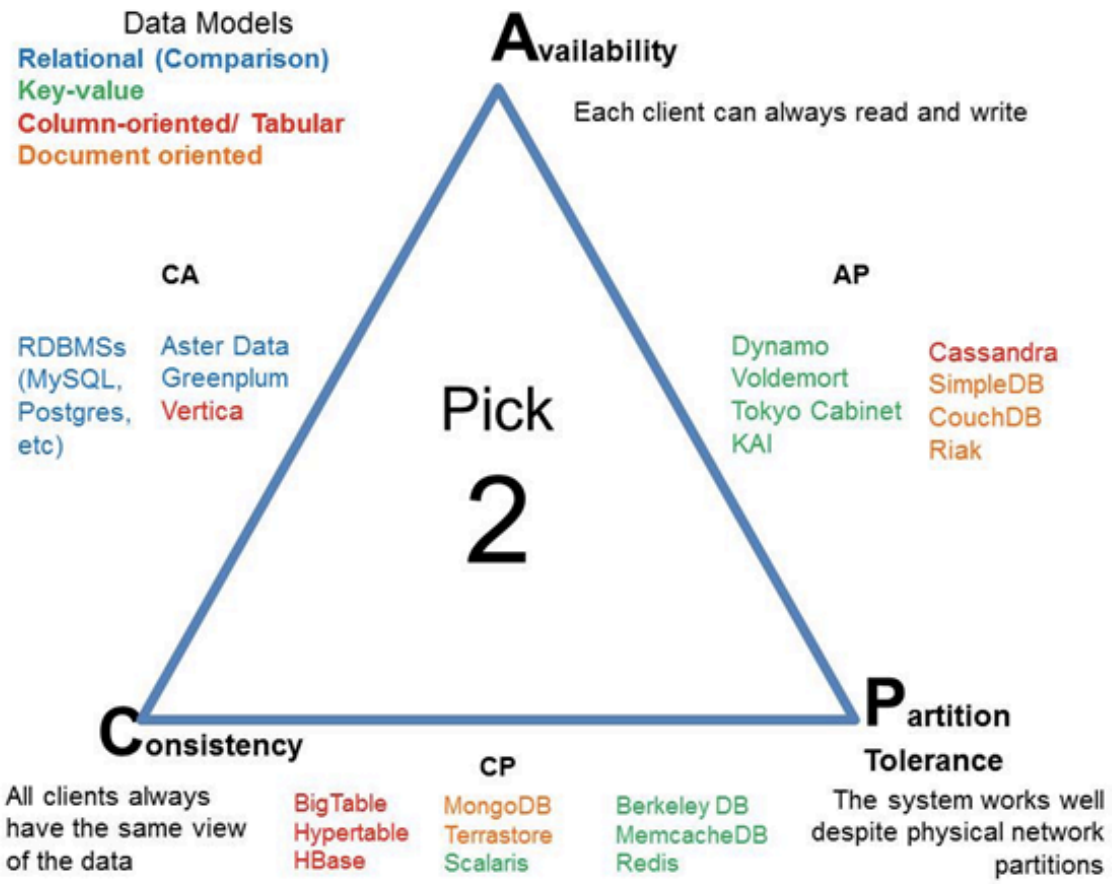




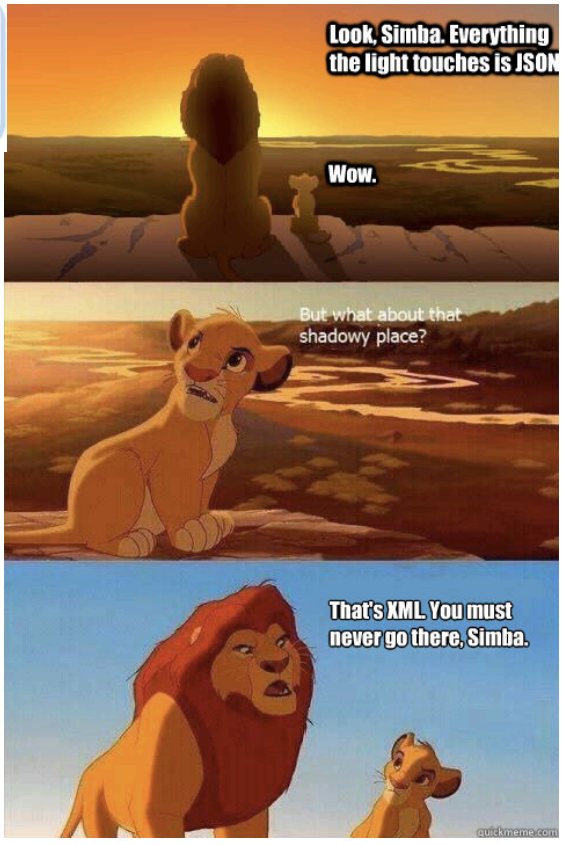
Brewer's CAP Theorem (2000)

- 3 main properties for distributed management
  1. ***Consistency***:
     - A data have the same value at the same time (coherency)
  2. ***Availability***:
     - Even if a server is down, data is available
  3. ***Partition Tolerance***:
     - Even if the system is partitioned, a query must have an answer (unless for global failures)

➢ Theorem: *A distributed, networked system can have only two of these three properties.*


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
AP
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Consistency
All clients always have the same view of the data
Partition Tolerance
The system works well despite physical network partitions
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis


JSON
JavaScript Object Notation


Look, Simba. Everything the light touches is JSON
Wow.
But what about that shadowy place?
That's XML. You must never go there, Simba.

- Initially XML used for complex internet communications (Web Services)
  - Too verbose
- **JSON (JavaScript Object Notation)**
  - Lightweight, text-oriented, language independent
  - Used for several Web services (Google API, Twitter API)

JSon : Structures

- Key + Value
  - "lastname" : "Travers"
  - Keys with quotations
- Objects/documents
  - Collection of key/values
  - { "lastname" : "Travers",
    "firstname" : "Nicolas",
    "kind" : 1}

Data types

- Scalar : String, Integer, float, boolean, null…
- List : arrays […]
- Documents : objetcs {…}

Arrays

- No typing inside arrays
  - “lessons” : [“SQL”, 1, 4.2, null, “NoSQL”]
- Can nest documents
  - “doc” : [{“test” : 1},
    {“test” : {“nesting” : 1.0}},
    {“key” : “text”, “value” : null}]

JSon : Identifiers

- Key « _id » commonly used to identify documents
  - Overwrite already stored ids
  - Can be automaticaly generated
    - Ex MongoDB : "_id" : ObjectId(1234567890)

JSon : complete example

```
{
    “_id” : 1234,
    “lastname” : “Travers”, “firstname” : “Nicolas”,
    “work” : {
        “company” : “Cnam”,
        “location” : {
            “street” : “2 rue conté”,
            “city” : “Paris”,
            “zip” : 75141
        },
    },
    “fields” : [ “DB” , “DB tuning”, “XML”, “NoSQL”, “IR” ]
}
```