

# NoSQL

# NoSQL – O que é?

- Classe de sistemas de bases de dados (vários)
- “Not only SQL”
  - Não usam SQL como linguagem para pesquisas
  - Arquitectura distribuída, tolerante a falhas
  - Não há esquema fixo
  - Não existem joins
    - Operações caras para combinar registos de duas ou mais tabelas num único set
    - Os joins requerem grande consistência e esquemas fixos
    - No entanto são sistemas mais flexíveis
- Não pretendem substituir RDBMS

# Where NoSQL Is Used?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)



# Cronologia do NoSQL

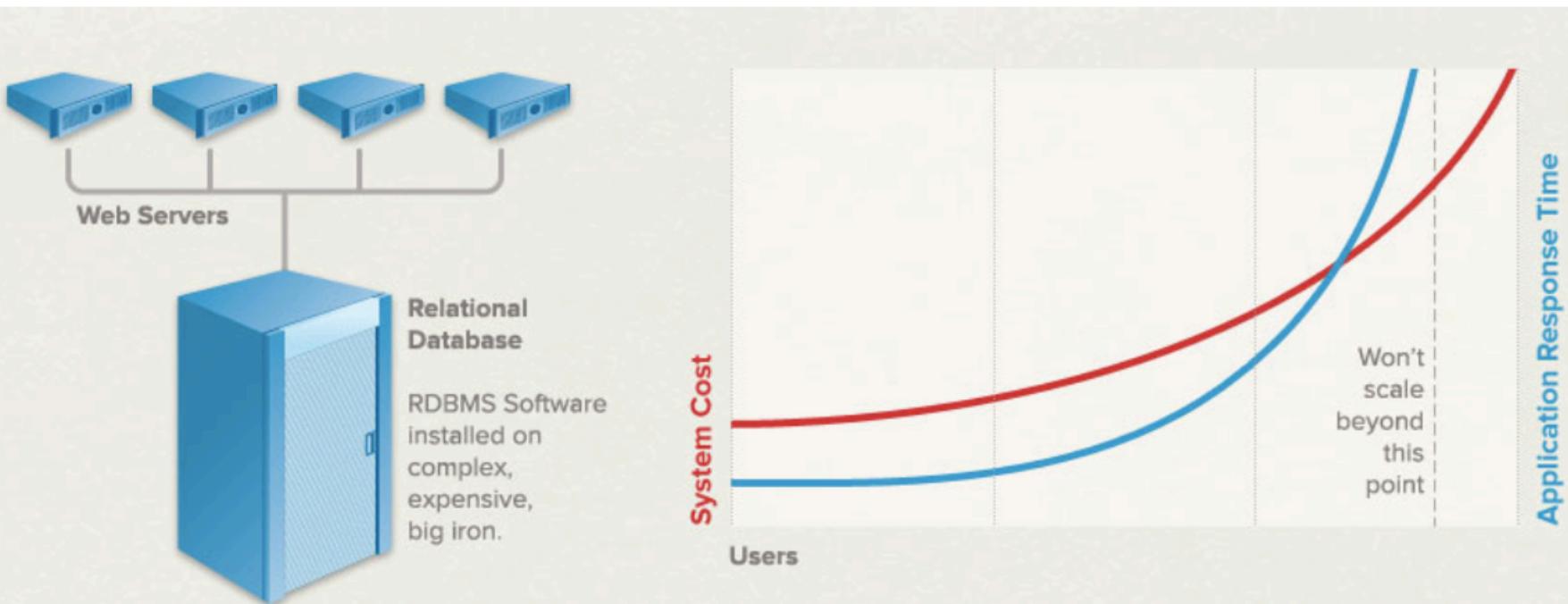
- 1998 – Carlo Strozzi menciona o termo mas com um objectivo bem diferente...
- 2003 – MemCached da LiveJournal
- 2004 – Paper do Google FileSystem
- 2006 – Paper do Google MapReduce
- 2006 – Paper do Google BigTable
- 2007 – Release do Hadoop no Apache
- 2009 – Paper do Amazon Dynamo
- 2010 – MongoDB (1a versão de produção)

# Escalabilidade

- Capacidade de um sistema, rede ou processo para lidar com uma carga crescente de trabalho de uma forma eficaz e eficiente ou a sua elasticidade de forma a lidar com esse mesmo crescimento.
- Por exemplo, pode referir-se à capacidade de um sistema de aumentar o seu output face ao aumento de carga quando se adiciona recursos (geralmente hardware).

# Escalabilidade BD Relacionais

- A certo ponto não escalam (elevados custos)



**Web Application - RDBMS Scales Up.** To support more users, you must get a bigger database server for your RDBMS. As a result, system cost grows exponentially with linear increases in users, and application response time degrades asymptotically.

# Escalabilidade - opções

- Escalabilidade tradicional (vertical)
  - bases de dados na cloud escalam comprando servidores “maiores” (hardware) e mais caros
  - Nem sempre é possível
  - Em muitos casos não é fiável
- Paradigma de arquitectura
  - escalabilidade horizontal baseada na partição dos dados, i.e., dividir a base de dados em vários servidores “pequenos” e baratos.

# Sharding

- Técnica
  - data sharding
    - partição horizontal de dados em diversas máquinas
- Consequências
  - Gerir o acesso paralelo da aplicação
  - Escalar tanto para as leituras como escritas
  - Não é transparente, aplicação tem que ser partition-aware

# Escalabilidade NoSQL

- Escalabilidade horizontal é possível
- Lida bem com ‘picos’ na rede
- Possibilidade de scale up e scale down consoante a procura

# Escalabilidade NoSQL

The diagram illustrates a system architecture for [www.wellsfargo.com](http://www.wellsfargo.com). It shows three user silhouettes at the top, connected to a Load Balancer. The Load Balancer connects to a pool of Web Servers, which in turn connect to a pool of NoSQL Database Servers. This architecture demonstrates how a single application can scale out by adding more commodity servers at different layers.

**Application Scales Out**  
Just add more commodity web servers

System Cost

Users

Application Response Time

**Database Scales Out**  
Just add more commodity data servers

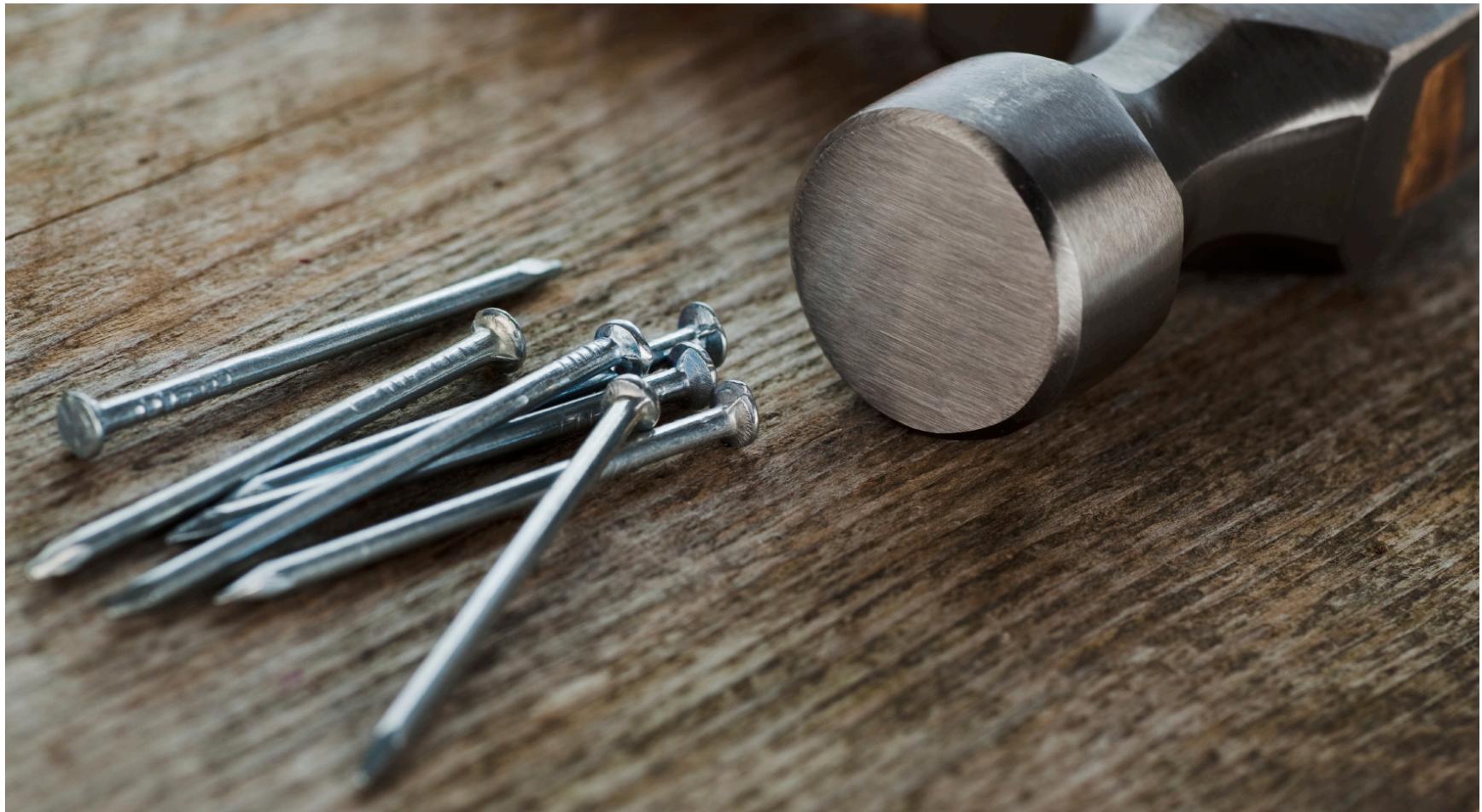
System Cost

Users

Application Response Time

In contrast to the non-linear increase in total system cost and asymptotic degradation of performance previously seen with RDBMS technology, NoSQL database technology flattens both curves.

# Nem todos os problemas são pregos



# ACID

- **Atomicidade**
  - ou todo o trabalho na transação é realizado ou nada
- **Consistência**
  - As transações alteram o estado da base de dados de um estado consistente para outro estado consistente (implica restrições)
- **Isolamento**
  - Os resultados de uma transação apenas são disponibilizados após commit.
- **Durabilidade**
  - Os resultados de uma transação após commit sobrevivem a falhas de sistema.

# Bases de dados Relacionais

- Economia mundial é relacional
- SQL é uma query language rica, declarativa
- Bases de dados forçam integridade
- ACID
- Developers estão bem familiarizados
- Grande suporte de ferramentas (JDBC, ...)

# Desafios das BD relacionais

- Desalinhamento
  - Dificuldade em mapear modelos de domínios complexos num esquema relacional
- Schema relacional é rígido
  - Dificuldade em lidar com dados semi-estruturados (alteração de atributos)
  - Alterações ao schema = downtime / prejuízo
- Extremamente difícil escalar escrita
  - Escalabilidade vertical é limitada / cara
  - Escalabilidade horizontal é limitada / cara

# Teorema CAP

- Eric Brewer, 1998
- **Consistência** (*consistency*)
- **Disponibilidade** (*availability*)
- **Tolerância particional** (*partitioning tolerance*)

# Consistência

- “Consistência Atómica”
- Diferente de consistência ACID
- Todos os nós do cluster vêm os mesmos dados a cada momento
- Cliente percebe que um grupo de operações ocorreu todo ao mesmo tempo

# Disponibilidade

- Todas as operações resultam numa resposta de sucesso ou falha
- Falha(s) de um ou mais nós não impedem a operação dos restantes nós

# Tolerância particional

- Operações completam-se mesmo quando componentes individuais não estão disponíveis
- Sistema continua funcional apesar de perda arbitrária de mensagens (erros de comunicação)

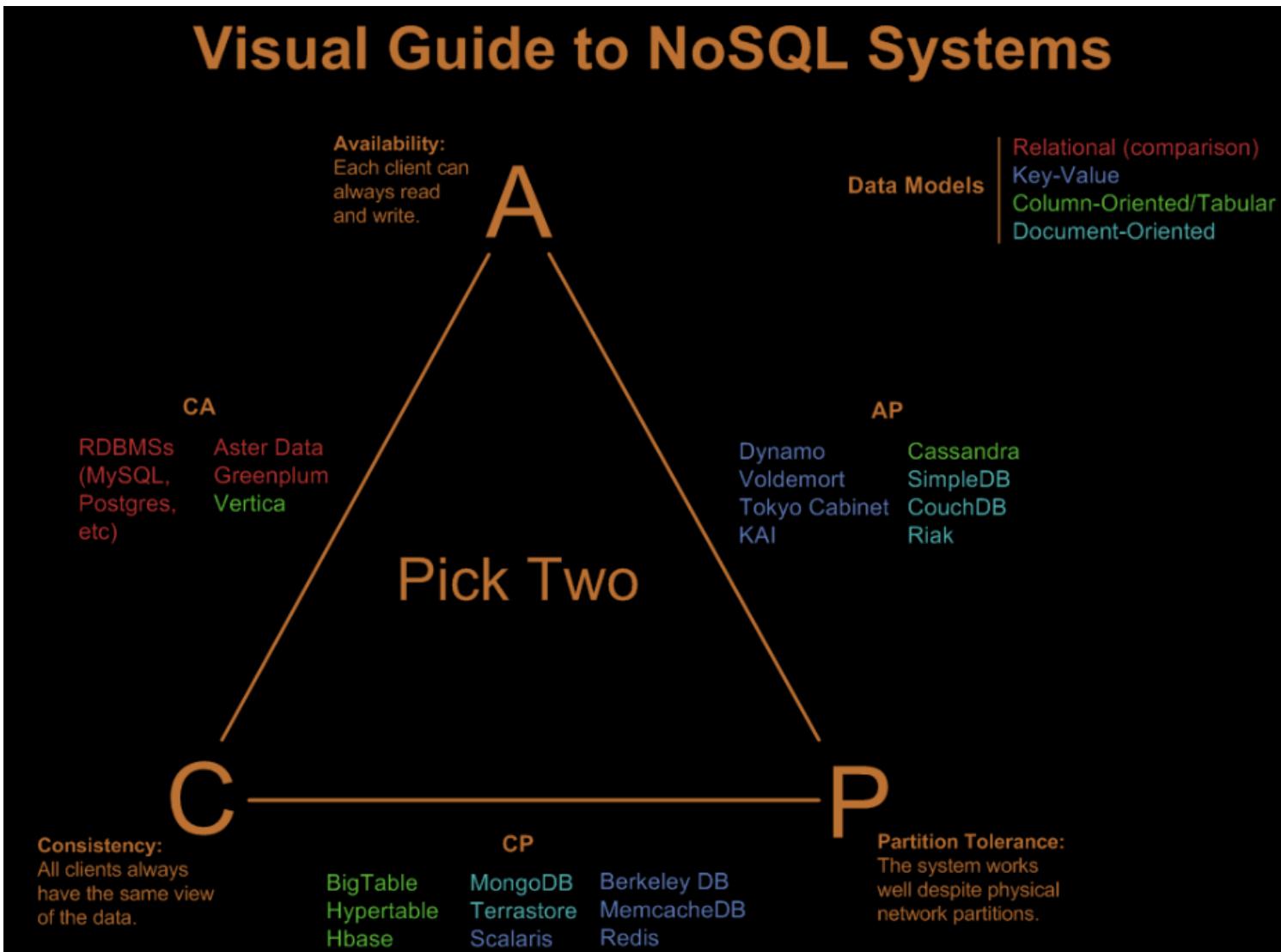
# Teorema CAP

- Definição: qualquer sistema com dados distribuidos não é capaz de garantir as 3 propriedades em simultâneo

# Teorema CAP e ACID

- Em NoSQL queremos garantir P
- Abandonar A or C do ACID
  - Relaxar C facilita a replicação e tolerância a falhas
  - Relaxar A reduz (ou elimina) a necessidade de controlo

# CAP e sistemas NoSQL



# Transações BASE

- **Basically Available**
  - não garante a disponibilidade dos dados
    - teorema de CAP
  - mas todos os pedidos são respondidos
    - mesmo que seja mensagem de erro
- **Soft State**
  - o estado do sistema pode mudar ao longo do tempo através de operações em background para eventual consistência
- **Eventual consistency**
  - o sistema não verifica a consistência depois de cada operação e continua a responder a pedidos sem verificar consistência. Eventualmente tratará de garantir consistência no futuro.

# Transações BASE

- Características:
  - Fraca consistência
  - Prioridade à disponibilidade
  - Melhor esforço
  - Respostas aproximadas permitidas
  - Simples e rápido

# NoSQL

- Definição ([www.nosql-database.org](http://www.nosql-database.org)):

Next Generation Databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable**. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly.

Often more characteristics apply such as: **schema-free, easy replication support, simple API, eventually consistent** / BASE (not ACID), a huge amount of data and more. So the misleading term "nosql" (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above.

## *HOW TO WRITE A CV*



Leverage the NoSQL boom

# Bases de dados NoSQL

- Grandes volumes de dados ('Big Data')
- Replicação e distribuição escaláveis
  - Potencialmente milhares de máquinas
  - Potencialmente distribuídas no mundo inteiro
- As pesquisas exigem resposta rápida
- Sobretudo operações de pesquisa, poucos updates
- Inserts e updates assíncronos
- Não existe esquema (schema-less)
- BASE - Não são obrigatórias as propriedades ACID
- Desenvolvimento open source

# Tipos de bases de dados NoSQL

- Wide column
- Key Value / Tuple stores
- Document
- Graph
- Multimodal
- Object
- Outros...

# O ‘foco’ actualmente

- Key Value store
  - hash tables de keys
- Column oriented
  - cada bloco de storage contém apenas dados de uma coluna
- Document Store
  - armazena documentos constituídos por links para elementos
- Graph
  - focam-se em relações e não só em entidades

# Key Value Store

- Tabela com 2 colunas contendo chave e valor associado a essa mesma chave
- A chave funciona como índice e o valor pode ser apontado como um look up.
- Exemplo
  - projecto Voldemort
  - Linkedin
  - Eventual consistência de chave valor, auto escalável

# Column Oriented

- Armazena dados ordenados por coluna
- Permitem pares key value num sistema paralelo
  - Modelo de dados
    - define-se atributos num esquema e permite actualizações
  - Princípio de armazenamento
    - grandes hash tables distribuídas
  - Propriedades
    - particionamento vertical/horizontal, grande disponibilidade
  - Completamente transparente para a aplicação
  - Permite compressão à coluna

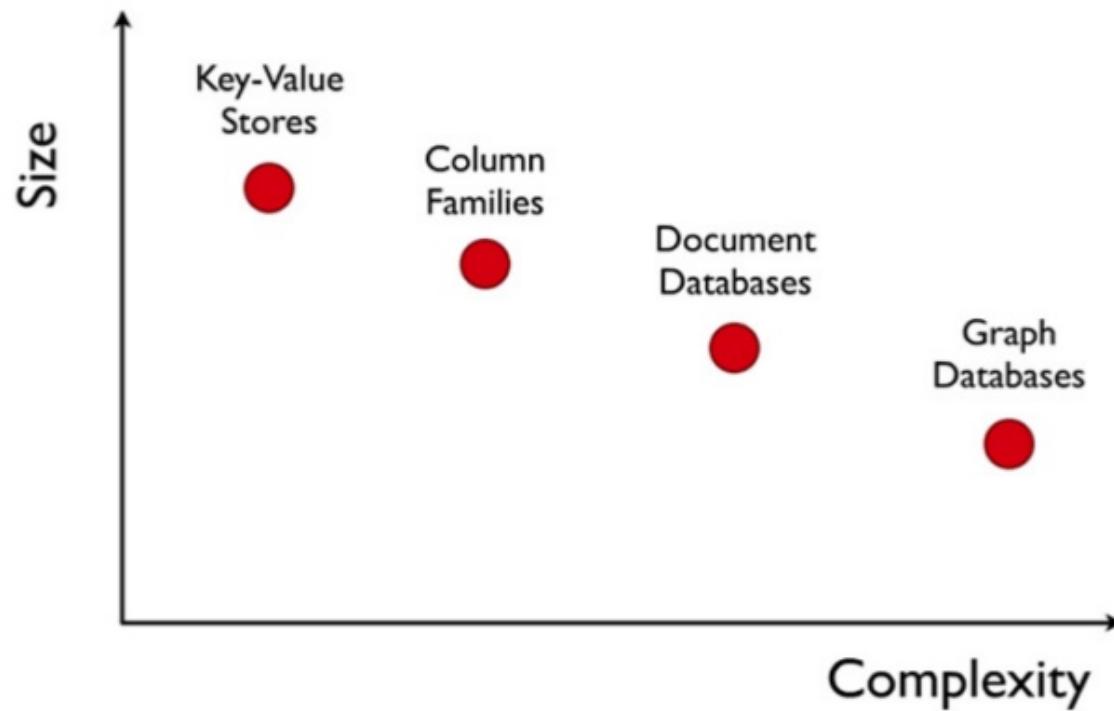
# Document

- Coleção de documentos
- Não existe qualquer schema
- Baseadas em JSON
  - permite listas, datas, booleanos, etc...
- Indexa documentos semi estruturados

# Graphs

- São criadas com nós, relações entre nós e propriedades dos nós
  - Nós representam entidades (CR7, Real Madrid)
    - Natureza similar aos objectos de programação OOP
  - Propriedades são informação sobre nós (idade)
  - Arestas representam relações (joga em)
- Escalabilidade em bases de dados de grafos é problemática (sharding...)

# Complexidade



# MongoDB

- Base de dados orientado ao documento
  - JSON (BSON)
  - Documentos organizados em collections (tabelas)
- Updates totais ou parciais de documentos
  - Update transacional em cada documento
  - Modificadores atómicos
- Linguagem de pesquisa rica e flexível
- Suporta índices
  - primários e compound
- GridFS para armazenar grande ficheiros
- MAP / REDUCE

# MongoDB – casos de uso

- Real time analytics
- Sistemas de gestão de conteúdos
- Update parcial de um único documento
- Caching
- Grande volume de escritas

# Quem usa?

- Foursquare
- Bit.ly
- NY Times
- SourceForge
- Shutterfly

# BSON

- Binary JSON

```
{"hello": "world"}
```

→    \x16\x00\x00\x00  
      \x02  
      hello\x00  
      \x06\x00\x00\x00world\x00  
      \x00

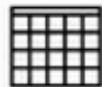
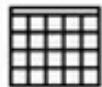
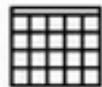
# MongoDB vs RDBMS



Database



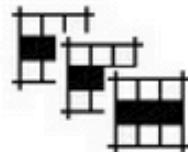
Database



Table

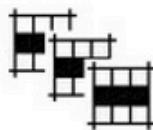


Collection



Document

# MongoDB vs RDBMS



Rows

Id	Username
1	Ridho

Id user	Street Name	House Number
1	Hirschbergerstrasse	62
1	Roemerstrasse	118



Document

```
{  
  Username: "Ridho",  
  Adresses: [{  
    StreetName: "Hirschbergerstrasse",  
    HouseNumber: "62"  
  }, {  
    StreetName: "Roemerstrasse",  
    HouseNumber: "118"  
  }]  
}
```

# MongoDB vs RDBMS

**Books**

Book_id	Title	ISBN	Year
1	Dragon Ball	120-99277	1989
2	Doraemon	220-99187	1995
3	Karate Kid	339-33111	1980

**Authors**

Author_id	Name
1	Akira Toriyama
2	Fujiko Fujio

**Books Authors relation**

id	Book	Author
1	1	1
2	2	2
3	3	1
4	3	2

# MongoDB vs RDBMS

```
{  
    title:"Dragon Ball",  
    ISBN:"120-99277",  
    authors:[ "Akira Toriyama" ], year: 1989  
}  
  
{  
    title:"Doraemon",  
    ISBN:"220-99187",  
    authors:[ "Fujiko Fujio" ] , year: 1995  
}  
  
{  
    title:"Karate Kid",  
    ISBN:"339-33111",  
    authors:[ "Akira Toriyama", "Fujiko Fujio" ] , year: 1980  
}
```

# ObjectID (`_id`)

- Id único para todos os documentos
- 12 bytes
- Consiste em
  - 0-3 timestamp
  - 4-6 machine
  - 7,8 PID
  - 9-11 increment

# MongoDB vs SQL

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
} )
```

# MongoDB vs SQL

```
ALTER TABLE users  
ADD join_date DATETIME
```

```
db.users.update(  
    { },  
    { $set: { join_date: new Date() } },  
    { multi: true }  
)
```

---

# MongoDB vs SQL

```
CREATE INDEX idx_user_id_asc  
ON users(user_id)
```

---

```
db.users.createIndex( { user_id: 1 } )
```

# MongoDB vs SQL

```
INSERT INTO users(user_id,  
                  age,  
                  status)
```

```
VALUES ("bcd001",  
       45,  
       "A")
```

```
db.users.insert(  
    { user_id: "bcd001", age: 45, status: "A" }  
)
```

# MongoDB vs SQL

```
SELECT *
```

**FROM** users

```
db.users.find()
```

# MongoDB vs SQL

```
SELECT *
```

```
FROM users
```

```
WHERE status = "A"
```

```
db.users.find(
```

```
  { status: "A" }
```

```
)
```

---

# MongoDB vs SQL

```
SELECT *
FROM users
WHERE status != "A"
```

```
db.users.find(
    { status: { $ne: "A" } }
```

---

# MongoDB vs SQL

```
SELECT *
FROM users
WHERE age > 25
```

```
db.users.find(
  { age: { $gt: 25 } }
)
```

# MongoDB vs SQL

```
SELECT *
FROM users
WHERE status = "A"
ORDER BY user_id ASC
```

```
db.users.find( { status: "A" } ).sort( { user_id: 1 } )
```

# MongoDB vs SQL

```
SELECT COUNT(*)  
FROM users  
WHERE age > 30
```

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

# MongoDB vs SQL

```
UPDATE users  
SET status = "C"  
WHERE age > 25
```

```
db.users.update(  
    { age: { $gt: 25 } },  
    { $set: { status: "C" } },  
    { multi: true }  
)
```

---

# Desvantagens do NoSQL

- Nova tecnologia logo buggy
- Dados são geralmente duplicados, potencial inconsistência
- Não existe schema standard
- Não existe standard de formato de queries
- Não existe linguagem standard
- Depende da camada de aplicação para garantir integridade dos dados
- Demasiadas opções e soluções

