

# NOSQL

MARCIO JASINSKI

FURB

# PLANEJAMENTO

Sábado I – 29/02

**08:00 – 8:30** – Introdução

**08:30 – 10:00** – Bancos NoSQL

**10:00 – 10:15** – Intervalo

**10:15 – 11:00** – Redis (Key/Value)

**11:00 – 12:00** – Prática I – Redis

**13:15 – 15:15** – MongoDB (Document)

**15:15 – 15:30** – Intervalo

**15:30 – 17:00** – Prática II - MongoDB

Sábado II – 14/03

**08:00 – 10:00** – Neo4j (Graph)

**10:00 – 10:15** – Intervalo

**10:15 – 12:00** – Prática II – Neo4j

**13:15 – 15:15** – Hbase (Column Family)

**15:15 – 15:30** – Intervalo

**15:30 – 17:00** – Prática III - HBase



# TRABALHOS E NOTAS

- Nada substitui a prática e o interesse 😊
- Todas as entregas serão via github (se não tiver uma conta, basta criar)
- Adicionar seu nome e sua conta no github na planilha da Turma: <http://bit.ly/furb-nosql-2020>
- Adiciona o nome o email mesmo que ainda não tenha conta no github 😊

Atividade	Peso
Prática Key/Value	2
Prática Document DB	2
Prática Graph Database	2
Prática Column Database	2
Trabalho/Prova	2



# REFERÊNCIAS

- Sadalage P. J., Fowler M. "NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence" 2012
- Perkins L., Redmond E. and Wilson J. R "Seven Databases in seven weeks" 2018
- Das V., "Learning Redis Design efficient web and business solutions with Redis" 2015
- Chodorow K., "MongoDB Definitive Guide" 2013
- Robbinson I., Webber J. and Eifrem E. "Graph databases new opportunities for connect data" 2015



# 1. INTRODUÇÃO

...

# NOSQL

- Criado por Eric Evans (Rackspace) para divulgar um evento sobre armazenamento de dados
- O significado de NoSQL é basicamente: Not Only SQL
- NoSQL é mais um movimento do que uma tecnologia

Não utiliza modelo  
relacional

Open Source

CAP Theorem

Schemaless

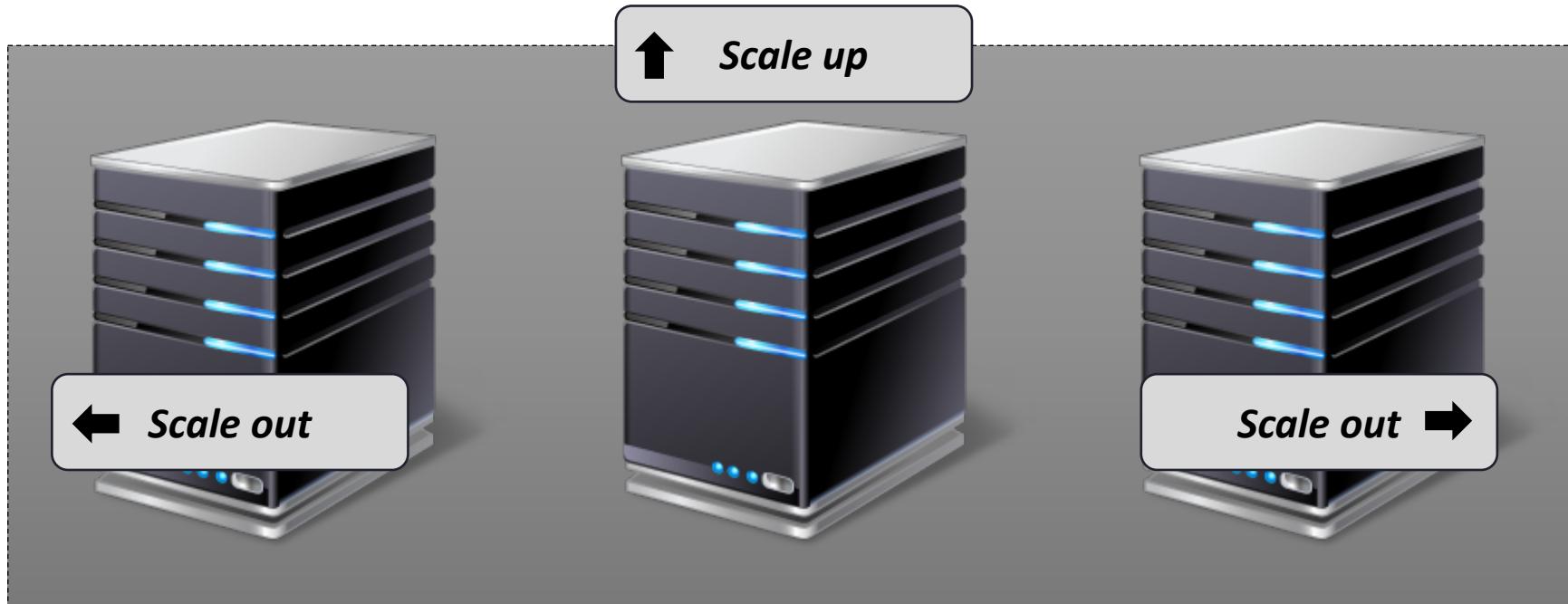
Modelo de Transação

Projetado para rodar  
em clusters

# NOSQL – PRINCIPAIS CARACTERÍSTICAS

1. Escala
2. Distribuição
3. Consistência
4. Estrutura Flexível (Schemaless)
5. Acesso aos dados sem SQL

# 1. ESCALA – POR QUE NÃO ESCALAR BANCOS RELACIONAIS?



Limitação do local

Transações e o tempo de resposta do cluster

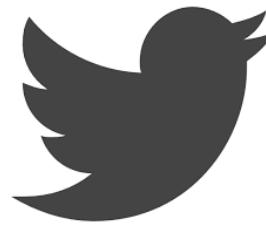
Custo de licenciamento

Quantidade de nós reduzida

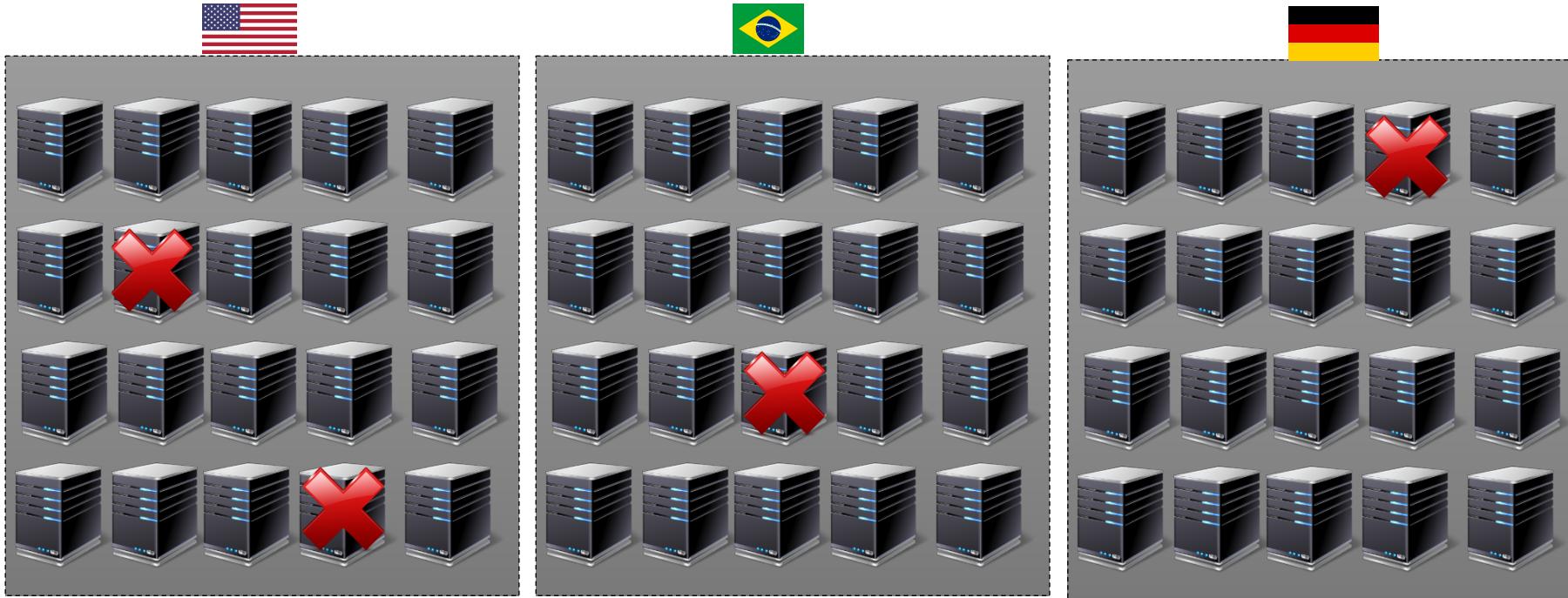
# 1. ESCALA – POR QUE NÃO ESCALAR BANCOS RELACIONAIS?

## Facebook

800 mi de usuários/ dia  
60 mil servidores  
4,5 bi likes/dia  
350 mi de fotos/dia  
300 petabytes de dados



# 1. ESCALA – CLOUD COMPUTING



Datacenters em diferentes locais

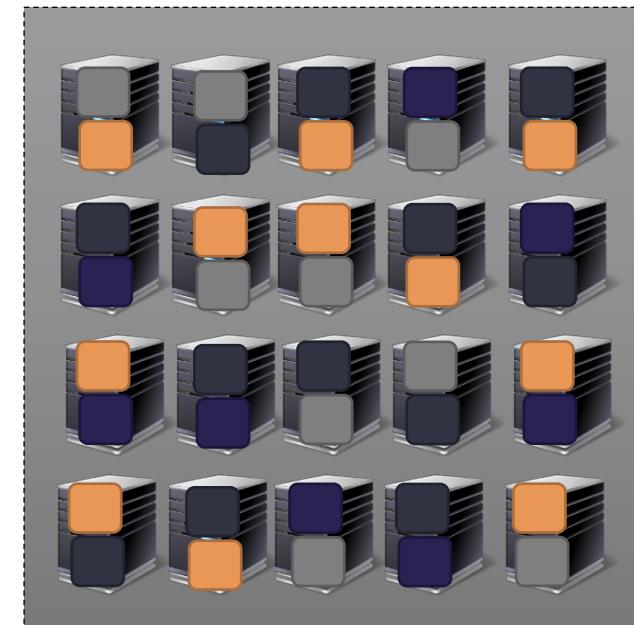
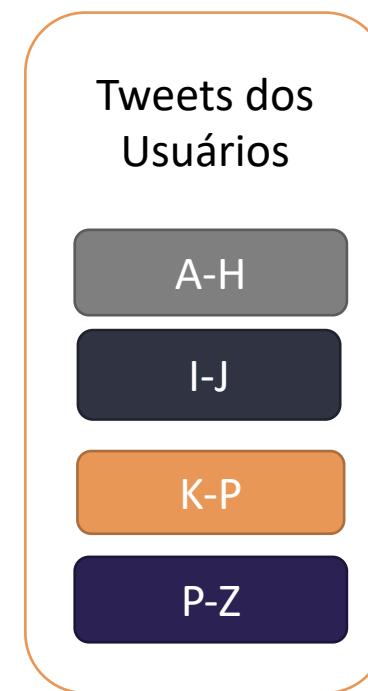
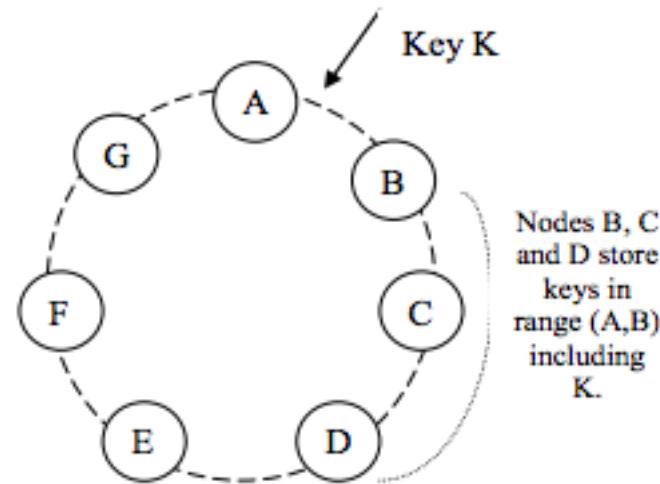
Nem tudo precisa de transação

Sem custo de licença por máquina

1000 servidores e tolerante a falhas

## 2. DISTRIBUIÇÃO DOS DADOS

- Particionamento, Sharding, Replicação



### Particionamento

### Sharding

### 3. CONSISTÊNCIA - ACID

- MongoDB is a *document-oriented* database, not a relational one. The primary reason for moving away from the relational model is to make scaling out easier, but there are some other advantages as well.
- A document-oriented database replaces the concept of a “row” with a more flexible model, the “document.” By allowing embedded documents and arrays, the document- oriented approach makes it possible to represent complex hierarchical relationships with a single record. This fits naturally into the way developers in modern object- oriented languages think about their data.
- There are also no predefined schemas: a document’s keys and values are not of fixed types or sizes. Without a fixed schema, adding or removing fields as needed becomes easier. Generally, this makes development faster as developers can quickly iterate. It is also easier to experiment. Developers can try dozens of models for the data and then choose the best one to pursue



# 3. CONSISTÊNCIA - ACID

Begin

```
update table RX998  
update table RX166  
insert into table RX444  
...  
insert into table RX111  
update table RX998  
update table RX166
```

commit

End

**ACID**

**Atomicity**

**Consistency**

**Isolation**

**Durability**

# 3. CONSISTÊNCIA - ACID

Begin

```
update table RX998  
update table RX166  
insert into table RX444  
...  
insert into table RX111  
update table RX998  
update table RX166
```

```
commit
```

End

## Atomicidade

**Tudo ou nada!**

O trabalho como parte indivisível (atômico). A transação deve ter todas as suas operações executadas em caso de sucesso ou, em caso de falha, nenhum resultado de alguma operação refletido sobre a base.

Após o término de uma transação (*commit* ou *rollback*), a base de dados não deve refletir resultados parciais da transação.



# 3. CONSISTÊNCIA - ACID

Begin

```
update table RX998  
update table RX166  
insert into table RX444  
...  
insert into table RX111  
update table RX998  
update table RX166
```

commit

End

## Consistência

### Manter a “sanidade” do banco

A execução de uma transação deve levar o banco de dados de um estado consistente a um outro estado consistente, ou seja, uma transação deve respeitar as regras de integridade dos dados (como unicidade de chaves, restrições de integridade lógica, etc.).

Todos os dados escritos no banco de dados devem ser válidos de acordo com todas as regras definidas, incluindo restrições (*constraints*), operações em cascata (*cascade*), *triggers* e qualquer combinação destes.

### 3. CONSISTÊNCIA - ACID

Begin

```
update table RX998  
update table RX166  
insert into table RX444  
...  
insert into table RX111  
update table RX998  
update table RX166  
  
commit
```

End

#### Isolamento

**Não interferir em transações alheias**

O *isolamento* é um conjunto de técnicas que tentam evitar que transações paralelas interfiram uma nas outras, fazendo com que o resultado de várias transações em paralelo seja o mesmo resultado se as mesmas transações fossem executadas sequencialmente (uma após a outra). .

### 3. CONSISTÊNCIA - ACID

Begin

```
update table RX998  
update table RX166  
insert into table RX444  
...  
insert into table RX111  
update table RX998  
update table RX166  
  
commit
```

End

#### Durabilidade

**“Não perde mais!”**

Os efeitos de uma transação em caso de sucesso (*commit*) devem persistir no banco de dados mesmo em casos de quedas de energia, travamentos ou erros. Garante que os dados estarão disponíveis em definitivo

### 3. CONSISTÊNCIA - ACID

- Bancos relacionais reforçaram o modelo ACID por décadas, pois isso é bom para aplicações;
- No entanto, novas demandas de volume e velocidade fazem com que bancos que implementem as propriedades ACID, tenham dificuldade de escalar, de ser mais distribuído.
- Para atender o volume e velocidade, as soluções abriram mão do ACID no nível de banco de dados
- É possível contrabalancear com esforço na aplicação (SAGA)
- Nem tudo precisa das garantias do modelo ACID

### 3. CONSISTÊNCIA - BASE

- Nunca pegou de fato, mas foi uma tentativa de brincar com os termos...

**Basic Availability**

BA – Foco em manter a base disponível

**Soft state**

S – O estado pode mudar com o tempo (clusters)

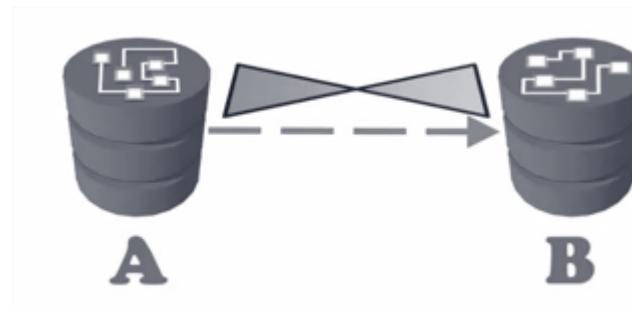
**Eventual Consistency**

E – Em algum momento, tudo estará consistente.

## 4. SCHEMALESS

- Há um foco menor (na maior parte dos bancos NoSQL) sobre a preparação da estrutura dos dados;
- Em bancos relacionais, tudo começa pelo schema e tudo deve respeitar o schema;
- Bancos NoSQL, podem armazenar diferentes schemas em uma mesma estrutura;
- Alterar o schema é em geral uma atividade simples em bancos NoSQL

### Schema on write vs Schema on read



- Estrutura das Tabelas
- Tipos de dados e tamanhos

# 5. DIFERENTES FORMAS DE ACESSO AOS DADOS

- Formas mais comuns em bancos NoSQL:
- API
- Chamadas REST
- Linguagem de consulta própria
- SQL



# Teorema de CAP

---

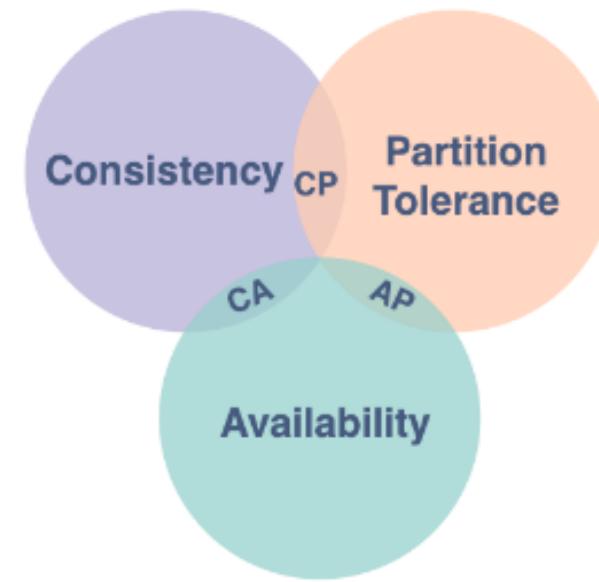
# TEOREMA DE CAP

- Proposto em 2000 por Eric Brewer, foi provado em 2002, por pesquisadores do MIT
- Existem algumas críticas ao modelo como o conceito da consistência e até mesmo a abrangência do modelo.
- O Teorema afirma que não é possível ter todas as características ao mesmo tempo:

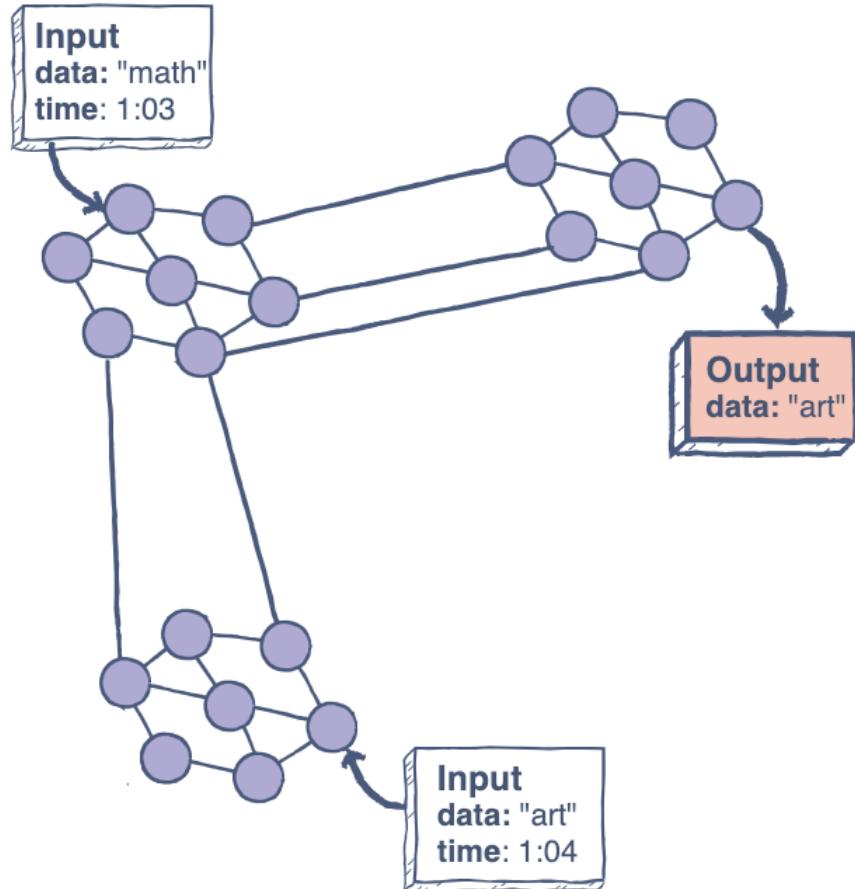
Consistency,

Availability

Partition Tolerance.



# TEOREMA DE CAP

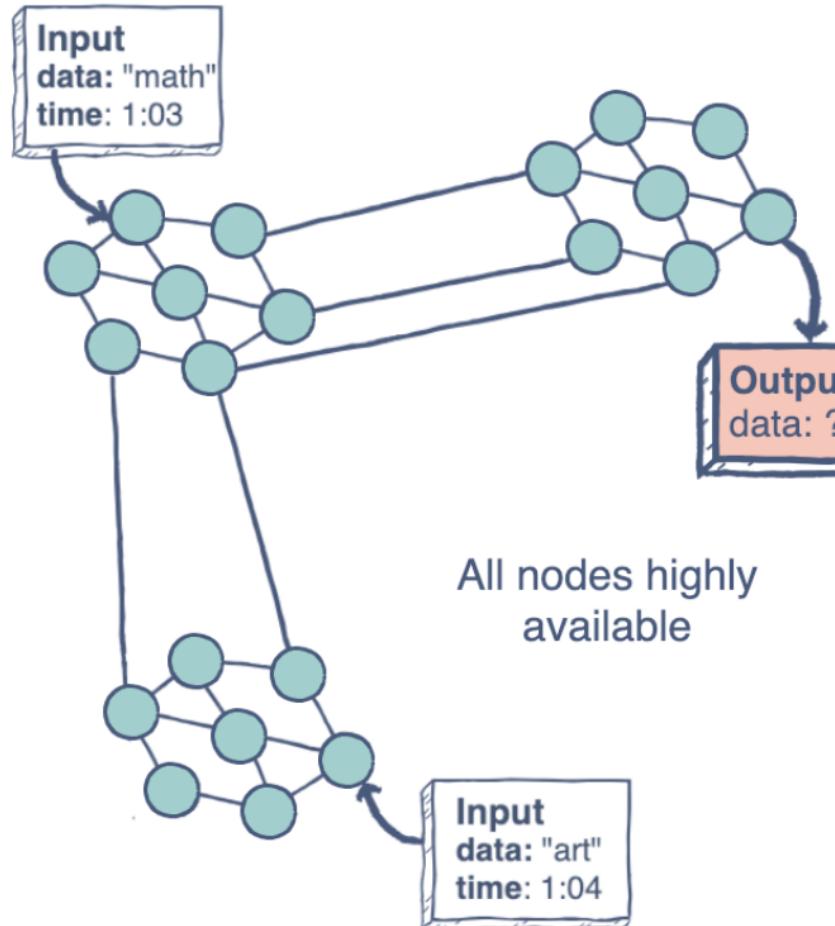


## Consistência

O sistema é dito consistente se todos os nós enxergam todos os dados ao mesmo tempo.

Ou seja, se fazer uma operação de leitura em um sistema consistente, ele deve retornar o valor da operação escrita mais recente. Se esse dado for solicitado a todos os nós, todos ao mesmo tempo devem retornar o mesmo dado.

# TEOREMA DE CAP

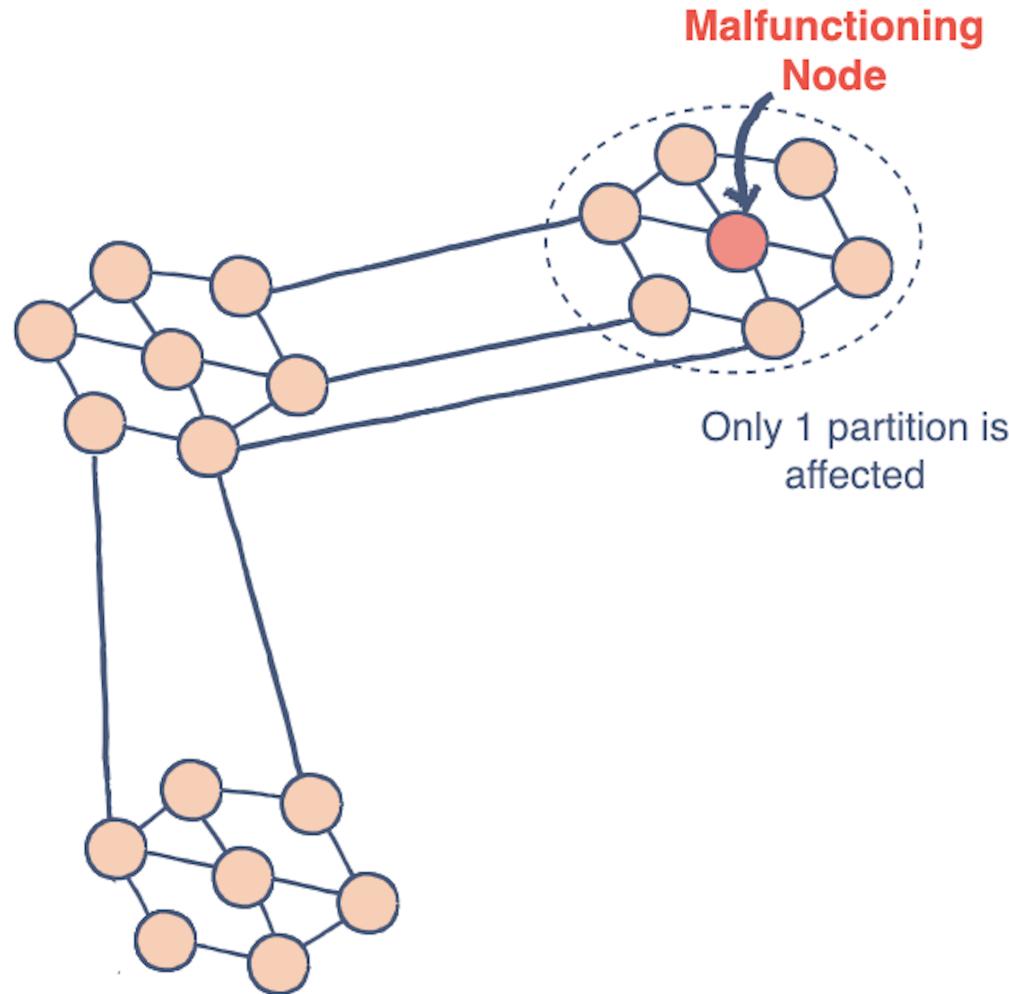


## Disponibilidade

Em sistemas distribuídos, a disponibilidade garante que o sistema permaneça operacional 100% do tempo. Cada requisição recebe uma resposta (que não seja erro do sistema) independente do estado de um nó.

Observe que aqui não há a garantia de que a resposta será a escrita mais recente.

# TEOREMA DE CAP



## Tolerância de particionamento

Essa condição indica que o sistema não vai falhar, independente da mensagem ser atrasada ou mesmo perdida entre nós do sistema.

Tolerância a partição é basicamente um modelo de replicação de dados com redundância, que garante que a queda de um nó não será sentida pela solução.

# TEOREMA DE CAP

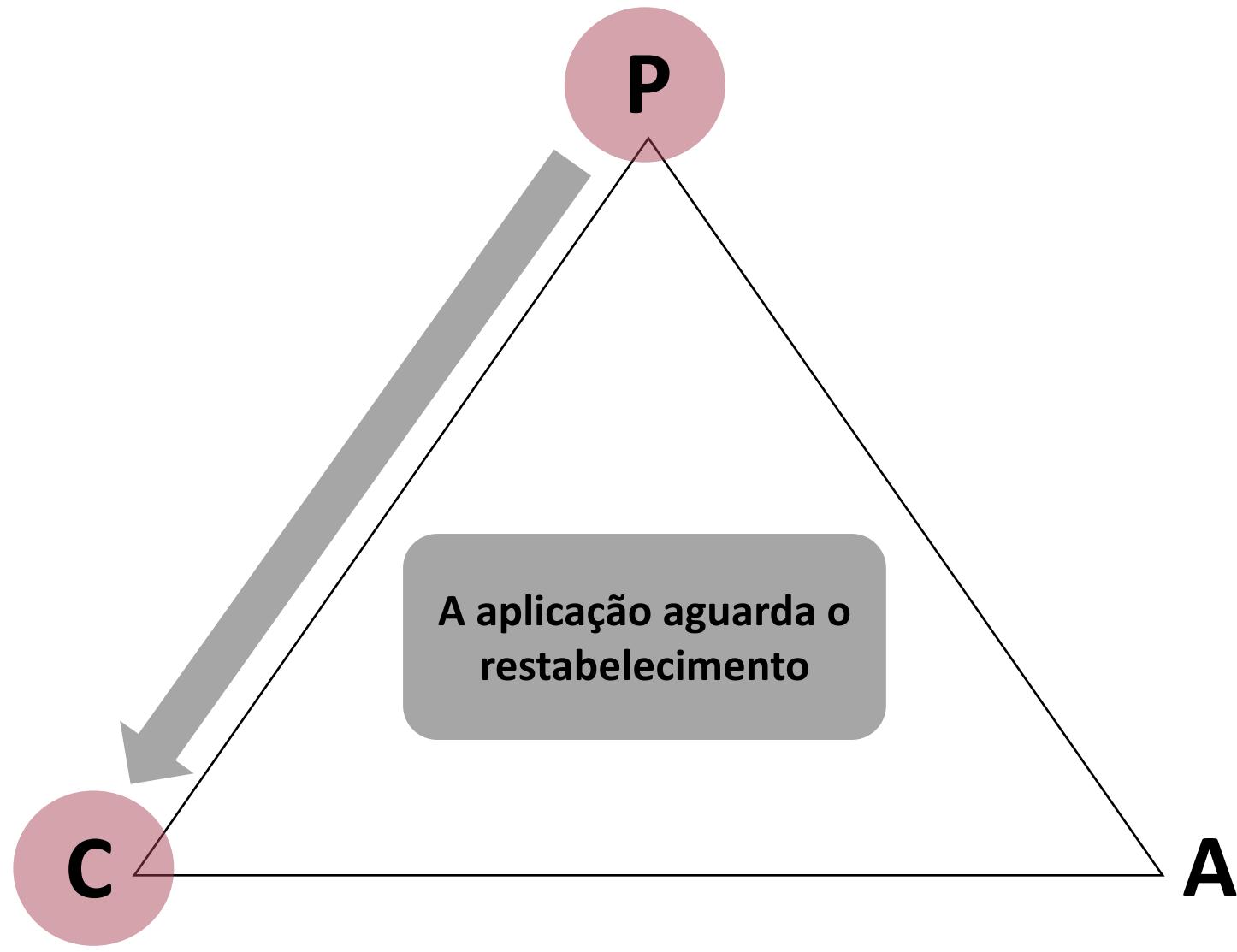


## Compreendendo o CAP

Uma das formas mais simples de compreender o CAP é reconhecendo a essência dos sistemas cloud: estar preparado para falhas.

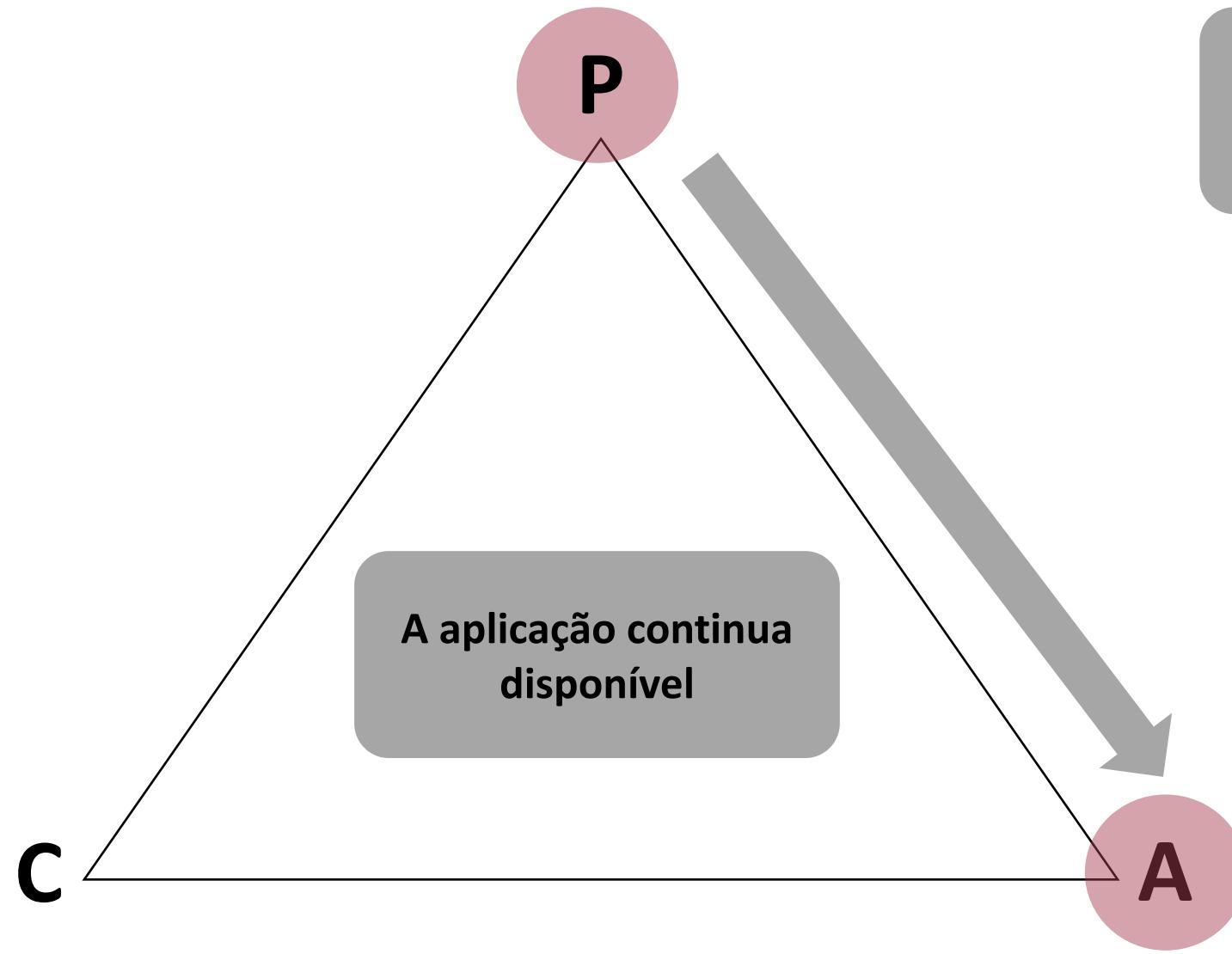
Há apenas uma certeza, nós vão falhar ao longo do tempo e a decisão é o que fazer quando isso acontecer...

# TEOREMA DE CAP

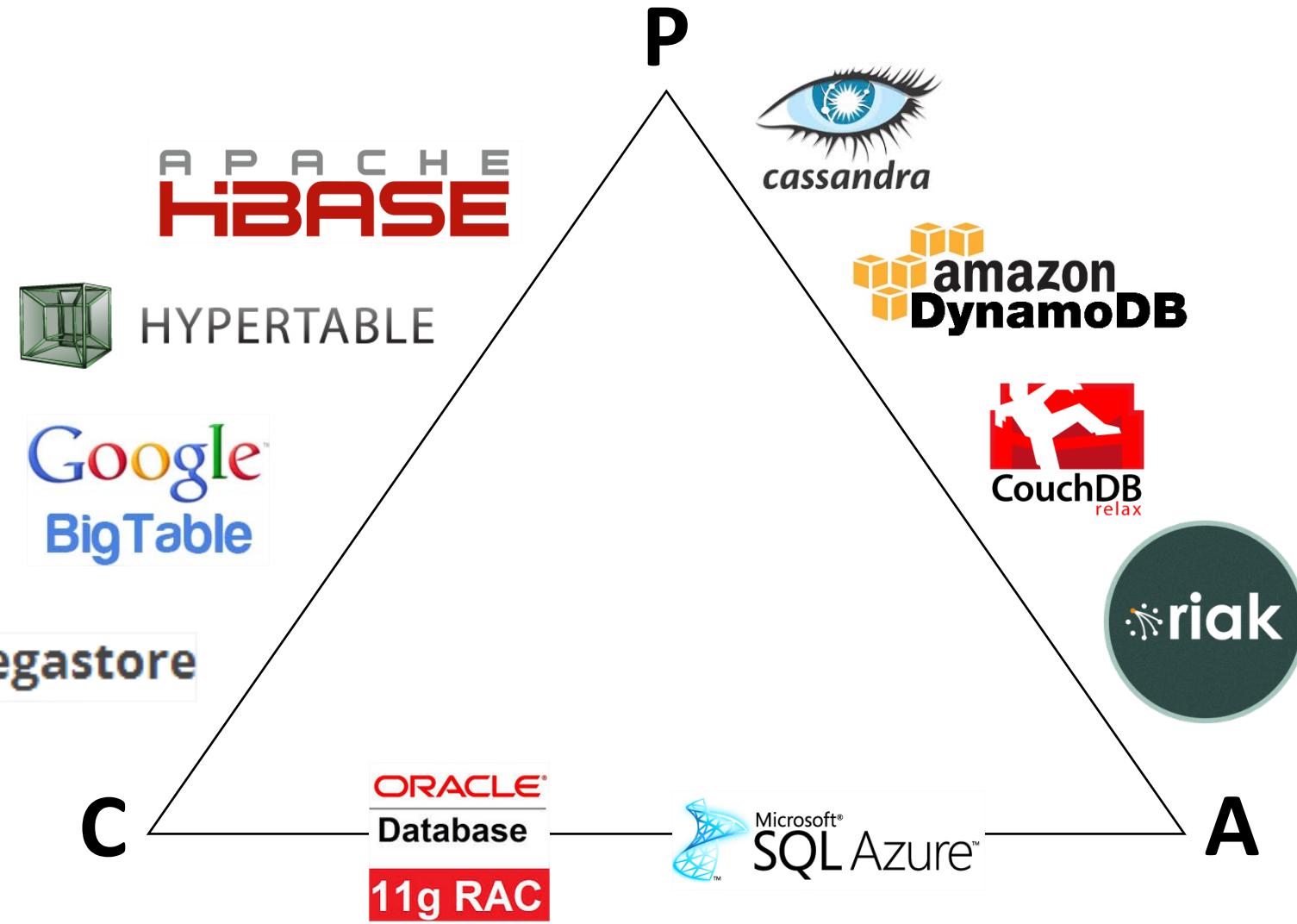


Se houver problemas de comunicação em 2 nós do cluster?

# TEOREMA DE CAP



# TEOREMA DE CAP

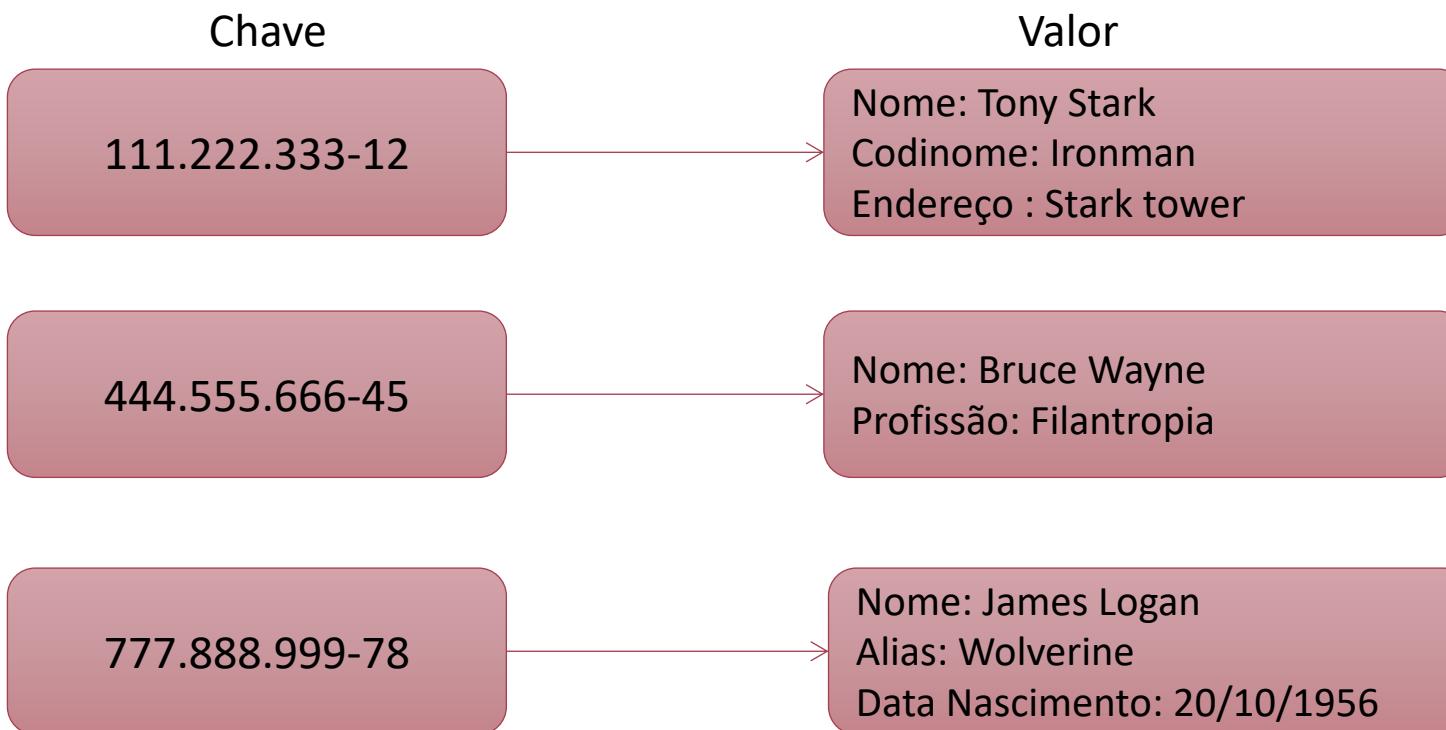


# Banco Chave-Valor

---

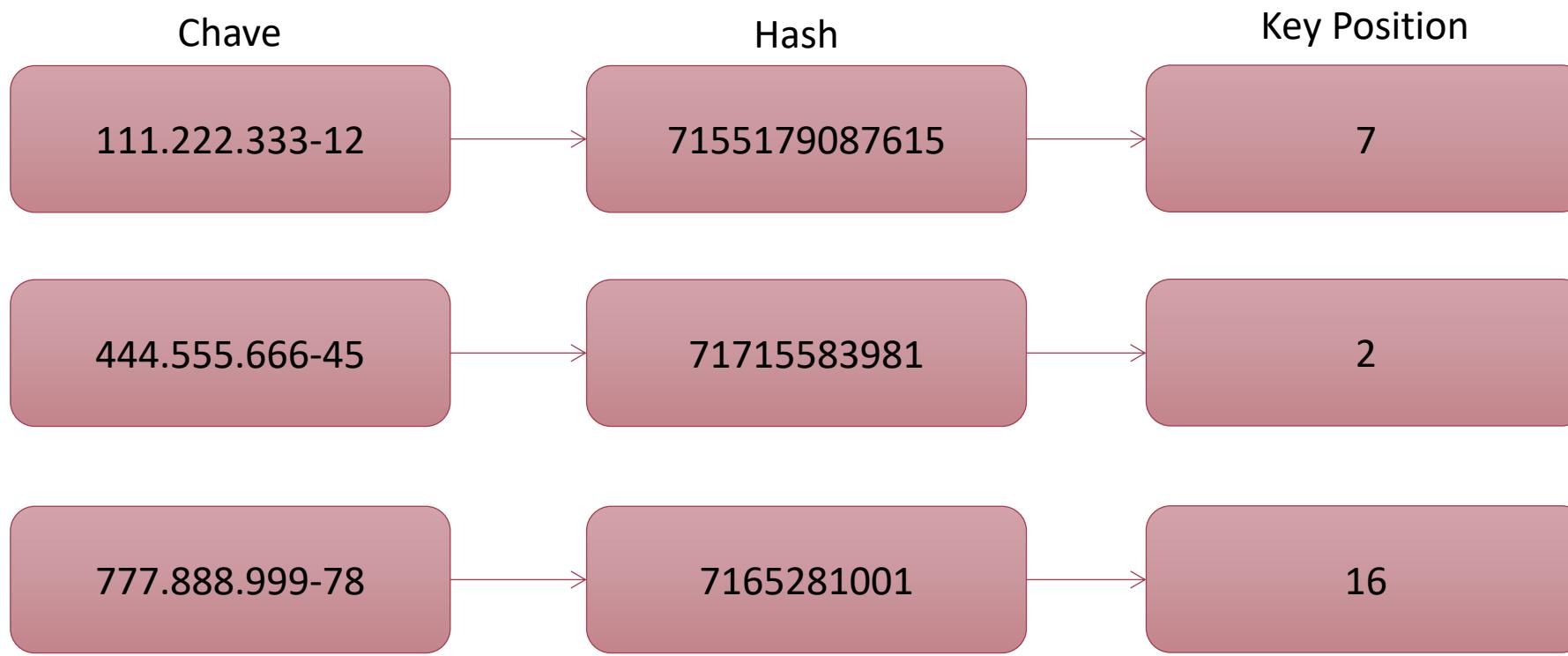
# CHAVE-VALOR

- Modelo mais simples de banco e muito utilizado para alta performance
- Utiliza o conceito Mapeamento Hash Distribuído



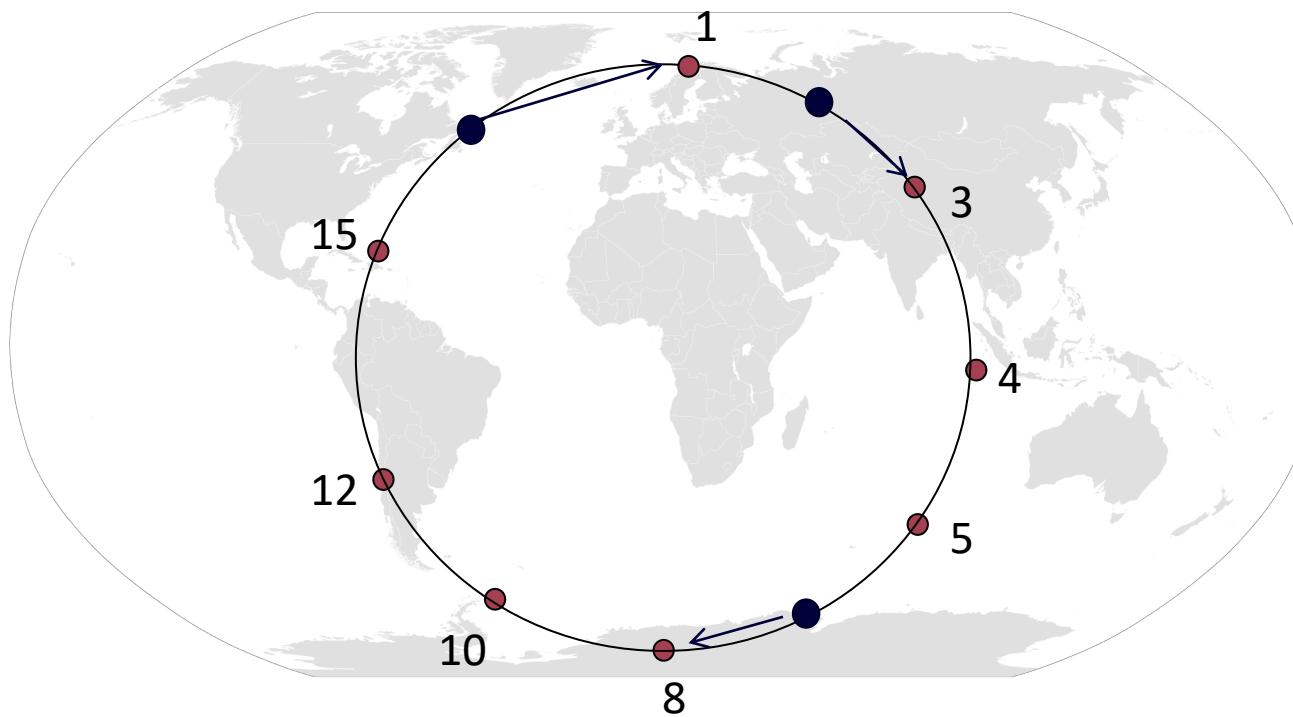
# CHAVE-VALOR

- Mapeamento Hash Distribuído (Consistent Hash)
  - Cada nó tem um pouco da informação
  - Poucas funções, em geral o principal é `get(key)`, `put(key, value)`



# CHAVE-VALOR

- Mapeamento Hash Distribuído
  - Cada servidor recebe um conjunto de chaves
  - Cada item é direcionado ao servidor responsável



## Circular DHT

Conceito de distribuir dados com virtualização para não depender de apenas um nó.

Reducir o impacto da entrada ou saída de novos nós

# CHAVE-VALOR

- **Características**
- Alta escalabilidade
- Alta produtividade (API Simples)
- Alta performance (geralmente opera em memória)
- Baixa complexidade
- Schemaless
- Funcionalidade mais limitada (chave->valor)
- Altamente particionável



# CHAVE-VALOR

- **Evitar utilizar**
- Dados do valor em geral não são indexados
- Não é indicado se existem relacionamentos entre os dados
- Não é uma boa opção se precisar de transações (fechamento de notas, estoque, etc)
- Evite depender de procurar sobre o conteúdo dos dados

**O que acontece se você fizer busca sobre conteúdo dos dados em um banco chave valor?**

# CHAVE-VALOR

- **Usos indicados**
- Armazenar dados de sessão do usuário
- Carrinho de compras
- Perfil de usuário e preferências



# Banco de Documentos

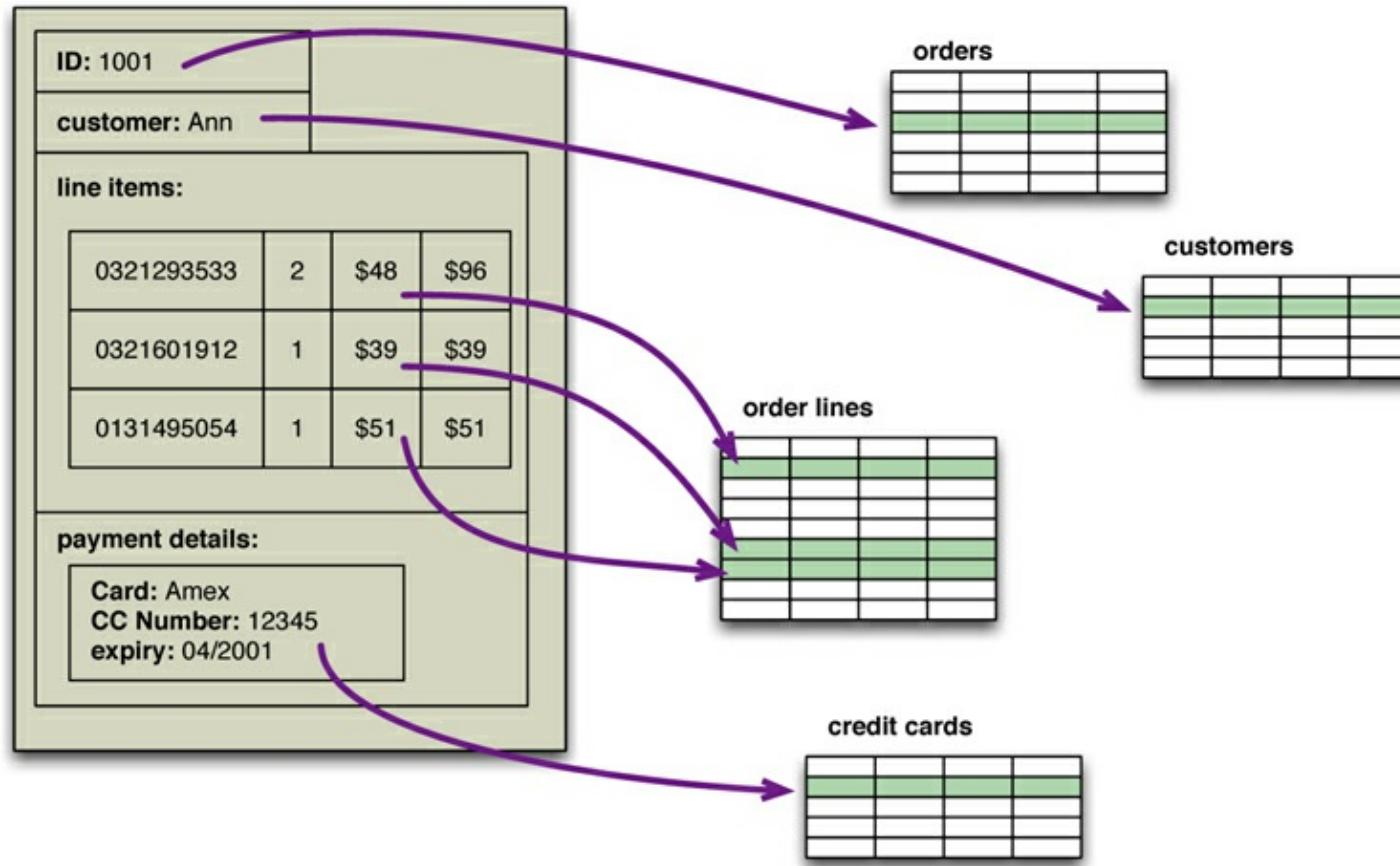
---

# BANCO DE DOCUMENTOS

- Tem o documento como centro conceitual do banco.
- O documento é uma string ou objeto de texto estruturado, geralmente em JSON (mas pode ser XML, BSON, etc)
- Cada documento, pode ser associado a uma linha de um banco relacional
- Mas considere essa linha, com agregada de todos os joins que formam a informação

Oracle	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document
rowid	_id
join	DBRef

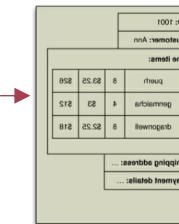
# BANCO DE DOCUMENTOS



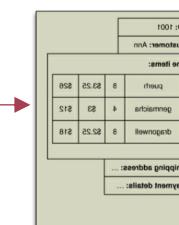
# BANCO DE DOCUMENTOS

Operações sobre os documentos

Document Key

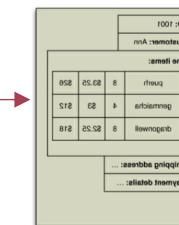


Document Key



⋮

Document Key



{index, keys, values}

# BANCO DE DOCUMENTOS

- Aggregates
  - Conjunto de dados como unidade
  - Limite da operação de atomicidade

```
{ "_id" : ObjectId("53a4773e561b180034b6ed95"),
  "user" : {
    "id" : "marcio.jasinski" },
  "sessions" : [
    {"title" : "Demonstração DOJO - 1a turma" },
    {"title" : "Arquitetura orientada a serviço" },
    {"title" : "Persistência Poliglota" },
    {"title" : ".NET - Como desenvolver para Cloud" }
  ]
}
```



# BANCO DE DOCUMENTOS

- **Características**
- Alta escalabilidade (mas alguns casos o master write pode ser gargalo)
- Alta produtividade (armazena objetos de forma natural)
- Boa performance
- Baixa complexidade
- Schemaless
- Escala bem nós para leitura



# BANCO DE DOCUMENTOS

- **Usos indicados**
  - Armazenamento de notas
  - Armazenamento de eventos com detalhes
  - Ambientes de gestão de conteúdo como blogs
  - Soluções com esquemas bem dinâmicos
  - Detalhamento de produtos para e-commerce



# BANCO DE DOCUMENTOS

- **Evitar utilizar**
- Não é indicado se existem relacionamentos entre os dados
- Não é uma boa opção se precisar de transações (fechamento de notas, estoque, etc)
- Se há necessidade de buscas complexas com joins

# Banco de Grafos

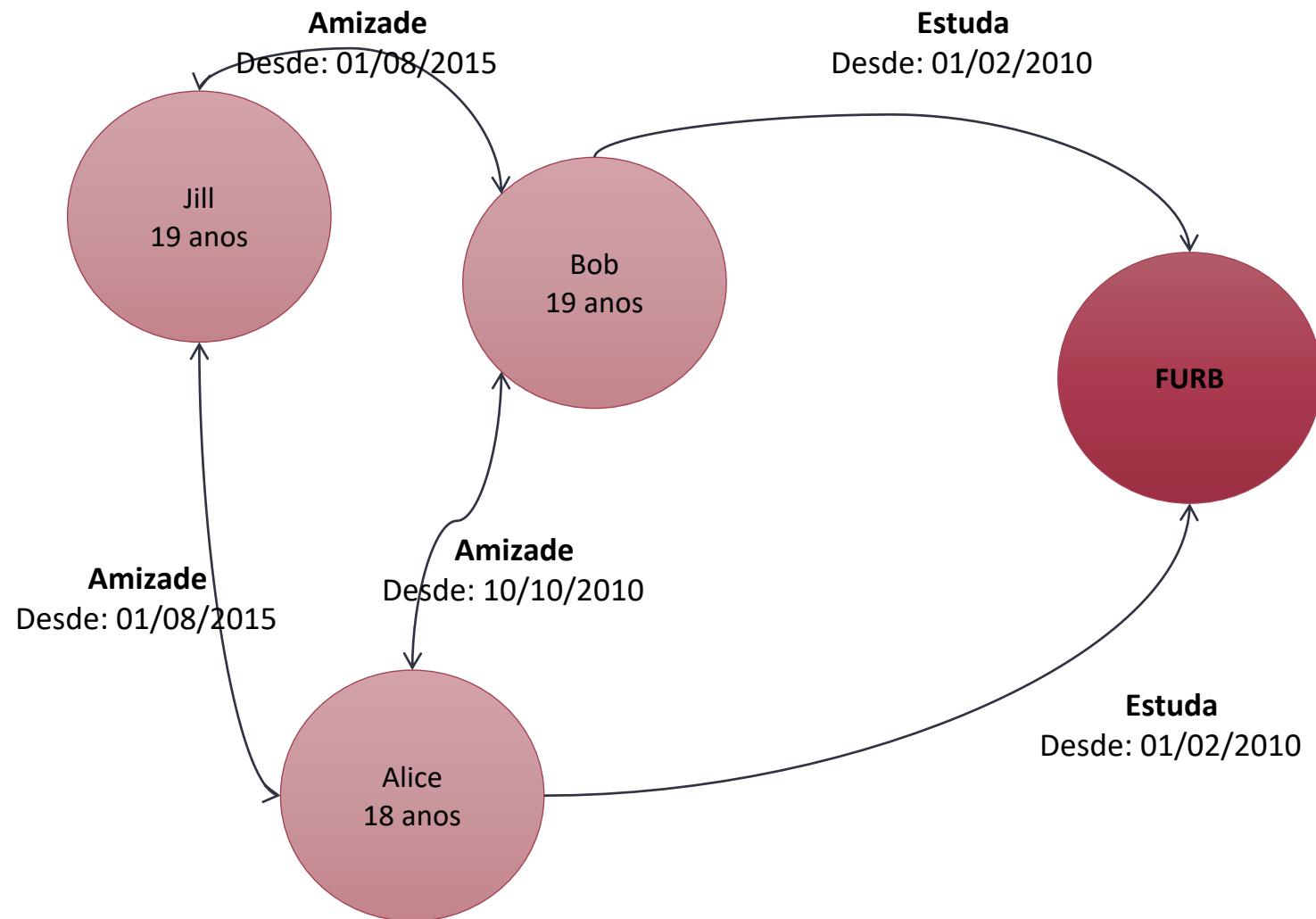
---

# BANCO DE GRAFOS

- Utiliza o conceito de grafos armazenado em disco
- É um modelo mais natural para uma série de aplicações como redes sociais
- Tem vantagens de ser uma área bem explorada da computação como algoritmos
  - Caminho mínimo/máximo
  - Identificação de ciclos
  - Busca em profundidade
  - Busca em largura
  - Componentes conexas/Desconexas
  - Djikstra
  - Custo mínimo/máximo



# BANCO DE GRAFOS



# BANCO DE GRAFOS

- **Características**
- É um modelo onde a escala chegou depois (neo4j já suporta); Em geral usa sharding
- Entrega um modelo ACID-compliant
- Entrega alta disponibilidade sem dificuldade com réplicas
- Entrega de por padrão muitos dos algoritmos clássicos de grafos



# BANCO DE GRAFOS

- **Utilização recomendada**
  - Implementação de redes sociais
  - Implementação de problemas de logística
  - Qualquer problema onde grafos pode ser aplicado
- **Utilização não recomendada**
  - Se for difícil modelar como grafo
  - Soluções clássicas bem resolvidas em bancos relacionais
  - Relatórios que em geral são bem tratados por tabelas



# BANCO DE GRAFOS

- 1.000.000 usuários com 50 amigos
- Neo4j:



Depth	Execution time (seconds) for 1 million users	Records returned
2	0.01	~2500
3	0.168	<b>30.267</b>
4	1.359	<b>1543.505</b>
5	2.132	<b>desistiu</b>

Fonte: Partner 2013

# Banco de Família de Colunas

---

# FAMÍLIA DE COLUNAS

- Modelo “parecido” com o relacional
- Flexível quanto à estrutura nas colunas

**Característica do modelo relacional**

<b>id</b>	<b>Nome</b>	<b>Poder</b>	<b>Alias</b>	<b>Inimigo</b>
1	Peter Parker	Aranha	Spiderman	Dr. Octopus
2	Bruce Wayne	NULL	Batman	Pinguim
3	Tony Stark	NULL	Ironman	NULL
4	Orin	Telepatia	Aquamen	NULL



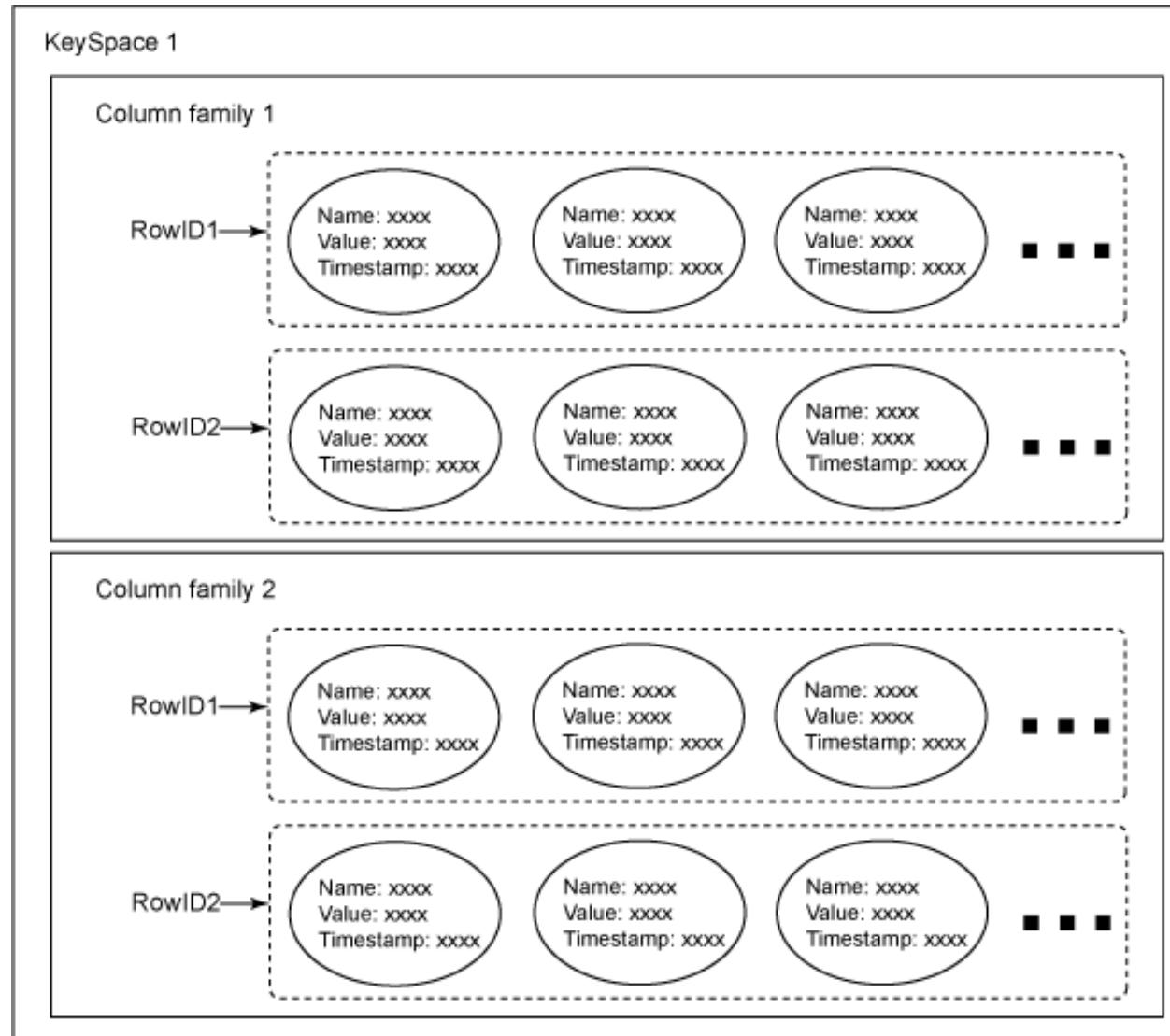
# FAMÍLIA DE COLUNAS

- Comparação com um RDBMS

RDBMS	Cassandra
database instance	cluster
database	keyspace
table	column family
row	row
column (same for all rows)	column (can be different per row)

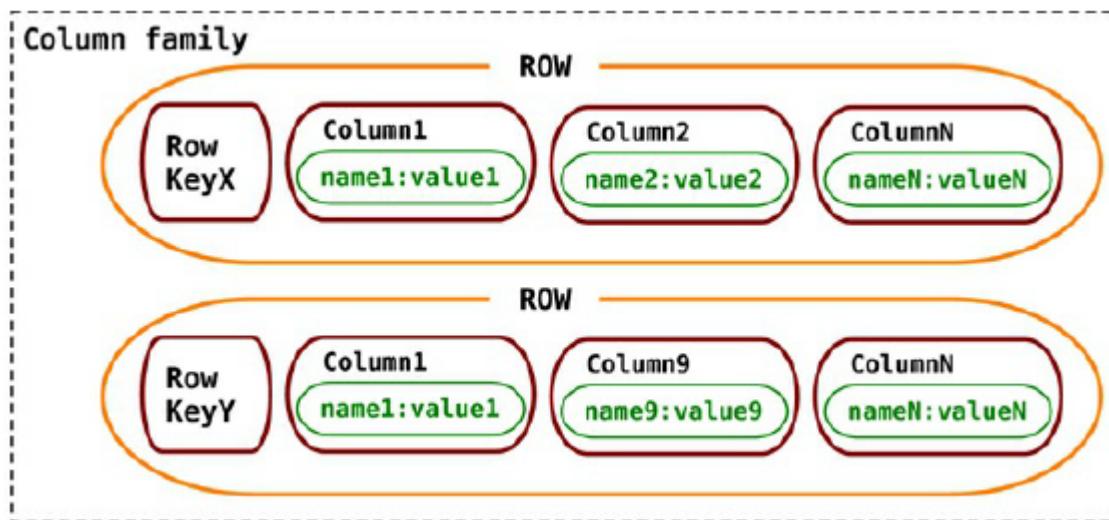


# FAMÍLIA DE COLUNAS



# FAMÍLIA DE COLUNAS

- A principal vantagem é manter a coluna apenas para os registros que fazem sentido, evitando tabelas esparsas



# FAMÍLIA DE COLUNAS

- Características
  - Em geral são desenhados para alta escalabilidade
  - Fazer queries é mais simples para quem conhece RDBMS
  - Modelo parecido com bancos SQL
  - Não é indicado para prototipagem
  - Modelo de transações é geralmente limitado no nível de linha
  - Schemaless
    - Mais flexível que o modelo relacional
    - Menos que Documentos e Chave Valor



# FAMÍLIA DE COLUNAS

- **Onde utilizar**

- Onde há demanda de escala, alta necessidade de escala
- Soluções analíticas
- Necessidade de alto desempenho de análise
- Sistemas de gestão de conteúdo
- Contabilização de visitantes

- **Evitar**

- Soluções com transações
- Soluções com pouca escala



# Bancos NoSQL

---

# BANCOS NOSQL

- Existem mais tipos de bancos que não estão no grupo NoSQL;
  - Livros Contábeis (QLDB)
  - Blockchain (Hyperledger, Ethereum)
  - Temporal Databases/Séries Temporais
  - In Memory Databases\*
  - Modelos de FileSystem (HDFS)
- Os bancos relacionais recuperaram terreno com opções open source

# BANCOS NOSQL

- **Persistência poliglota**

- Conceito que considera o uso de um banco de dados específico para cada problema
- Não é simples, mas tende a ser uma estratégia que usa o melhor de cada oferta

