



# BANCOS DE DADOS NoSQL

---

*Vinícius Kroth – Aula 02*

# Professores

## VINÍCIUS KROTH

Professor Convidado

Desenvolvedor de aplicações SOA nas áreas de contabilidade, financeira e de comércio exterior por mais de 5 anos. Referência em assuntos como projeto e desenvolvimento de SOA e Microservices, Java (EE, Spring framework MVC/WebFlux, junit e Gradle), SQL e NoSQL Db's (PostgreSQL, MySQL, MongoDB, Redis e Elasticsearch), AWS Cloud computing, Stress/Chaos testing (Gatlin, Jmeter), ferramentas de Desenvolvimento (Jenkins, Docker e Terraform), Telemetria e Observabilidade (Kibana, Datadog).

## EDUARDO HENRIQUE PEREIRA DE ARRUDA

Professor PUCRS

Fundador e CEO da Doc.Space Documentos Digitais. Professor da Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS), onde atua desde 1994 em cursos de graduação, pós-graduação e extensão nas áreas de Ciência da Computação, Engenharia de Software e Sistemas de Informação. Possui graduação (1992) e mestrado (1995) em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS) e formações complementares em gestão de TI e Segurança da Informação. Cursa Doutorado no Programa de Pós-Graduação em Ciência da Computação (PPGCC) da PUC-RS. Dedicar parte de seu tempo a atividades de incentivo ao empreendedorismo inovador e investe em empresas que adotem modelos de negócio inovadores e escaláveis. Apoia projetos de empreendedorismo social, tendo sido coordenador do projeto Adoções, parceria entre o Poder Judiciário e o Ministério Público do RS com a PUC-RS que, por meio de aplicativo, realiza a aproximação entre candidatos a adoção e crianças e adolescentes em processo de adoção tardia.

# *Ementa da disciplina*

Introdução aos conceitos e características de Big Data como: volume, velocidade, variedade, validade, volatilidade e valência. Introdução aos conceitos de cluster, domínios, agregados, distribuição, tolerância a falhas e sharding. Estudo do Teorema CAP: consistência (Consistency), disponibilidade (Availability), tolerância de partição (Partition). Introdução a Bancos de dados sem esquema prévio, a Banco de dados baseado em documentos, a Banco de dados chave-valor, a Banco de dados colunar e a Banco de dados baseado em grafos.

**Pós Graduação PUCRS**  
**Desenvolvimento FullStack –**  
**Banco de dados NoSql**



## Sobre mim

- Vinícius Frantz Kroth
- Engenheiro de Software pela PUCRS
- Engenheiro de Software na inpowered.ai
- Medium :  
*<https://medium.com/@vinicius.kroth>*
- Github : *<https://github.com/ViniKroth>*

# Como irá funcionar este módulo?

- Exemplos práticos sobre **modelagem** e situações do cotidiano;
- **Teoria** sobre o funcionamento de cada banco de dados;
- **Prática** com os bancos de dados escolhidos;
- Visão sobre decisões **arquiteturais**;

# Sumário

- Introdução: Setup, conceitos base, explicação;
- Banco de dados orientados a documentos (**MongoDB**);
- Bancos de dados orientados a colunas (**Cassandra DB**);
- Bancos de dados chave-valor (**Redis**);
- Bancos de dados orientados a grafos (**Neo4J**);
- Meus 20 centavos sobre **modelagem** de banco de dados;

# Pré-Requisitos

- Instalar: <https://docs.docker.com/get-docker/>
- Instalar: <https://www.mongodb.com/products/compass> (ou similares)
- Instalar: <https://redis.io/docs/getting-started/>
- Acessar: <https://github.com/ViniKroth/material-noSQL>



# Revisando: Palavras-Chave

- **Consistência:** Dado deve ser visualizado de forma igual, por todos os requerentes;
- **Durabilidade:** Quanto tempo um dado deve estar disponível;
- **Disponibilidade:** A capacidade do dado estar disponível quando solicitado;
- **Escalabilidade:** Capacidade de um servidor aumentar sua capacidade computacional (memória, cpu, etc);
- **Fonte da verdade:** Base durável, contendo uma versão confiável dos dados;
- **Cluster:** Agrupamento de unidades computacionais;
- **Shard:** Partição/peça dos dados de um banco;
- **Throughput:** Capacidade de transmissão de dados;

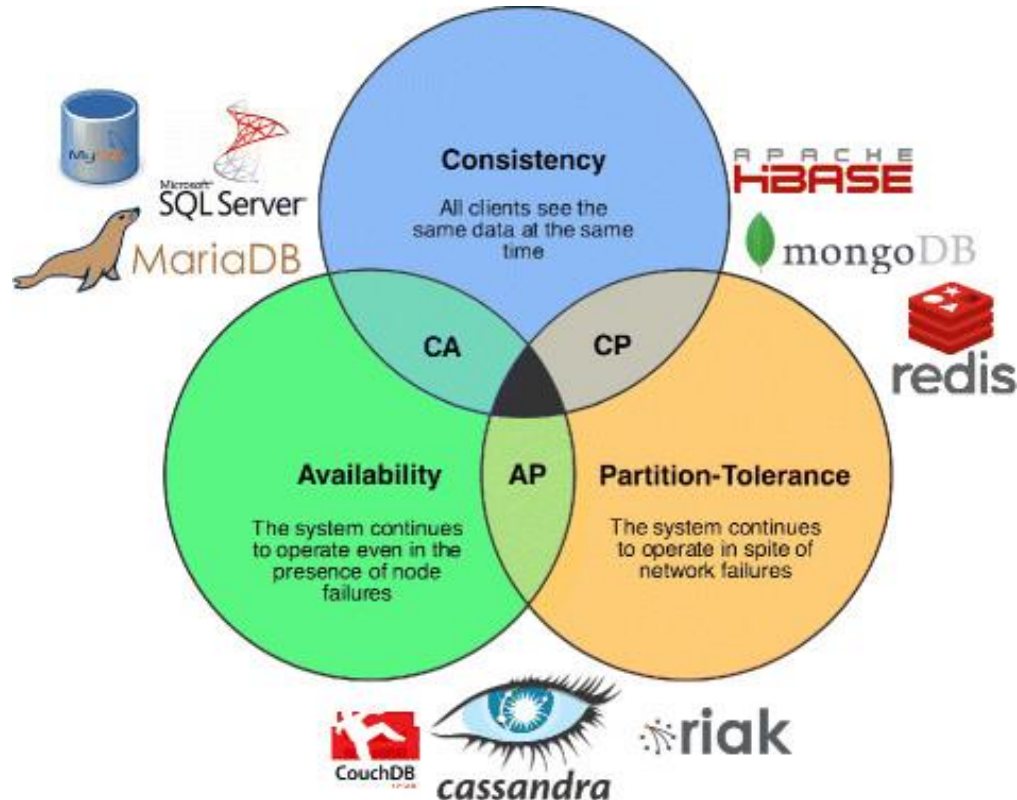
# Revisando: Bancos de Dados Relacionais

- Avançaram **significativamente** nos últimos anos;
- Resolvem **grande parte** dos problemas do dia-a-dia;
- SQL atende **diversos** requisitos;
- Forte **consistência** de dados;
- Suportam transações **ACID**, crucial para diversos cenários;

# O porquê do noSQL?


- Casos de uso cada vez **mais complexos**;
- **Clusters**, cada vez mais comuns (advento da cloud);
- Maiores **quantidades de dados**;
- Maiores **throughputs** de leitura/escrita necessários;
- **Flexibilidade** na estrutura dos dados;

# Revisando: Teorema CAP (CDT)



# Revisando: Famílias de banco de dados noSQL

**Key Value**



**Example:**  
Riak, Tokyo Cabinet, Redis  
server, Memcached,  
Scalaris

**Document-Based**



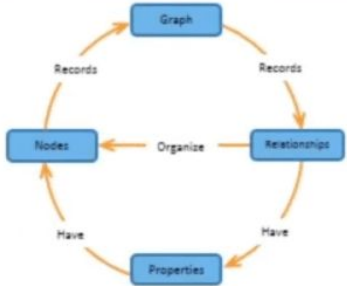
**Example:**  
MongoDB, CouchDB,  
OrientDB, RavenDB

**Column-Based**

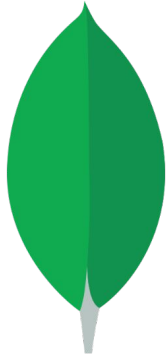


**Example:**  
BigTable, Cassandra,  
Hbase,  
Hypertable

**Graph-Based**



**Example:**  
Neo4J, InfoGrid, Infinite  
Graph, Flock DB



mongoDB<sup>®</sup>

**Mongodb**

# Porquê usar MongoDB?

- Esquema de dados **flexível (*schemaless*)**;
- Nativamente **escalável** (suporta ***sharding*/clusterização**);
- Oferece um bom ***throughput*** para leitura/escrita.
- Suporta operadores nativos para **agregação de dados**;
- Suporta transações **ACID** parcialmente;
- Documentação **extensiva**;
- Integração testada com as principais **linguagens**;
- Linguagem baseada em **JavaScript**;

# Principais conceitos

- A unidade básica é chamada de **Documento**;
- **Documentos** são armazenados em formato **BSON** (similar a **JSON**);
- Todo documento tem um **identificador único**;
- Os documentos podem ter formatos/campos variados.
- **Documentos** são agrupados em **Coleções** (equivalentes a tabelas SQL).
- Nativamente “**clusterizável**”;



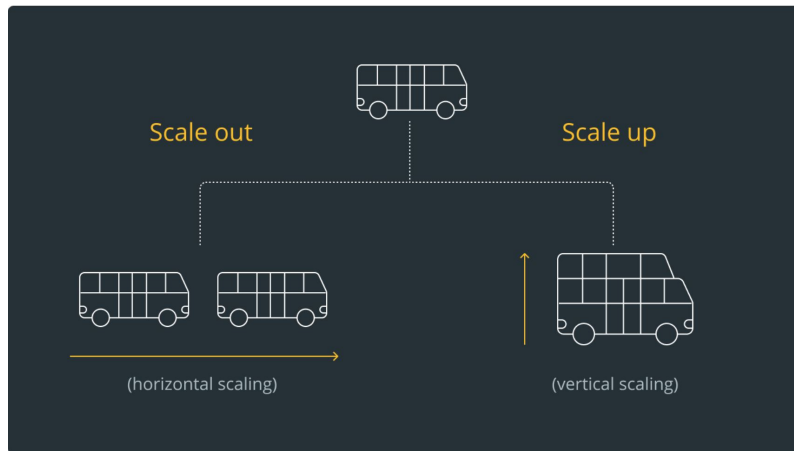
# Revisando: Schemaless

- Apesar de muitos bancos serem considerados *schemaless*, ou sem esquema. Isto na prática é um pouco diferente...

# Principais conceitos

- Cada documento só pode ter até **16 MB**. (Ao menos que se use **GridFS**);
- Todos os elementos suportam **indexamento**;
- Suporte de **transações ACID** dentro de um mesmo documento.
- Suporta **aninhamento** de documentos (documentos dentro de documentos);
- Devido à **falta** de transação entre documentos, dados

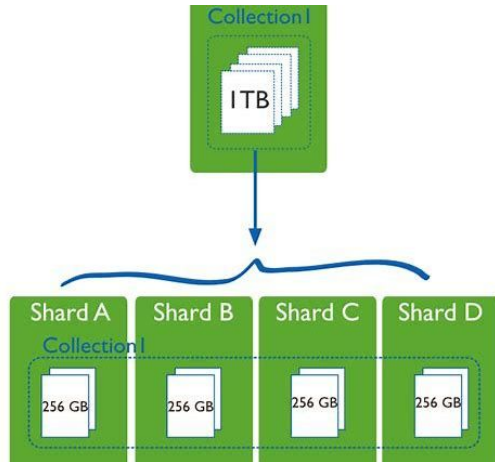
# Revisando: Escalando servidores



- Escalabilidade **vertical**:  
Quando aumentamos a capacidade de um servidor/nodo; + memória, + cpu ...
- Escalabilidade **horizontal**:  
Quando aumentamos o número de instâncias (nodos) dentro de *cluster*;

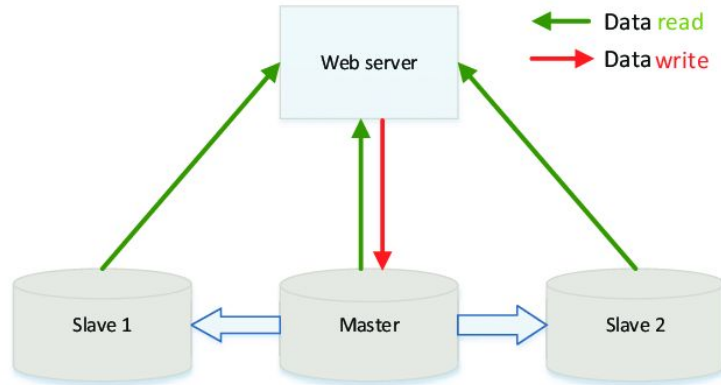
# Revisando: Tipos de escalabilidade horizontal

- **Sharding**, ou fragmentação, é a técnica de dividir o servidor de banco de dados, em partes, as quais serão totalmente responsáveis pela leitura/escrita de seus dados;



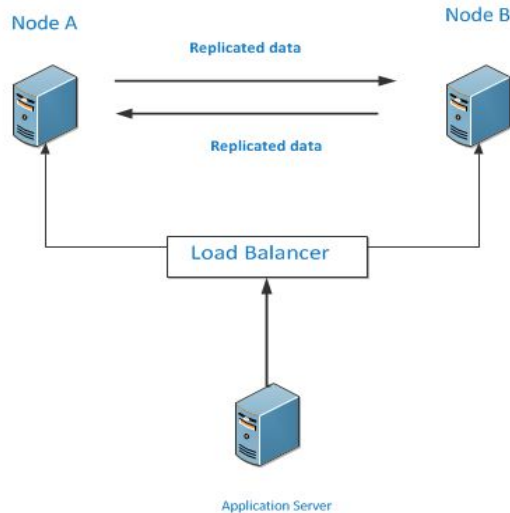
# Revisando: Tipos de escalabilidade horizontal

- **Replicação Mestre/Escravo** ou controlador/trabalhador: onde o nodo principal fica responsável pela escrita e replicação dos dados, e os nodos secundários, ficam responsáveis por processar as leituras;



# Revisando: Tipos de escalabilidade horizontal

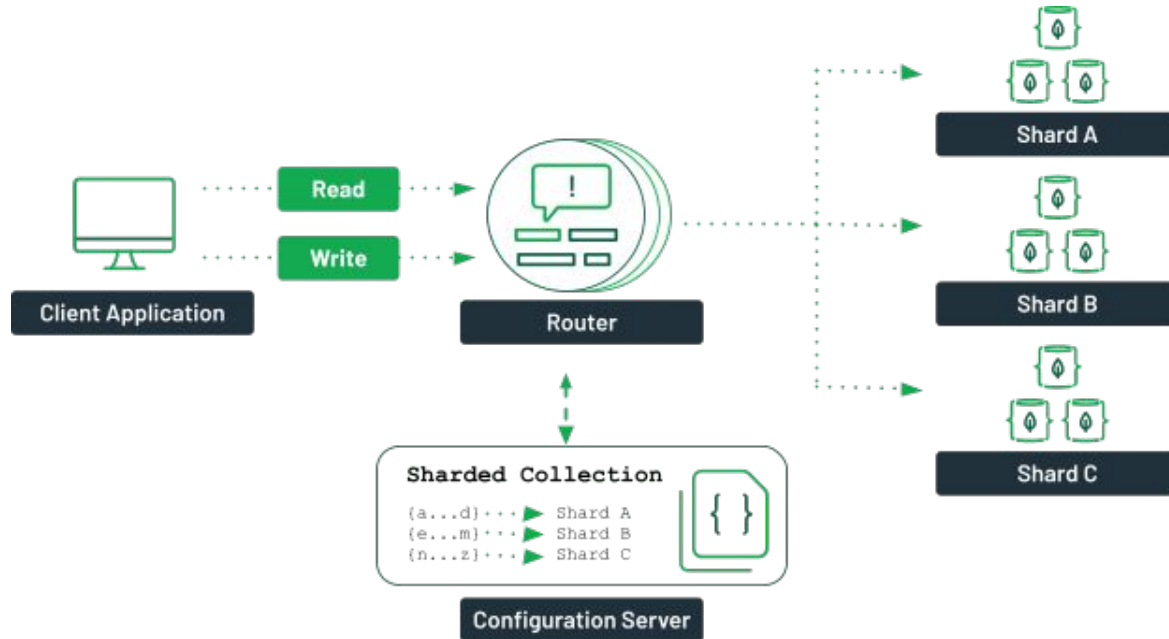
- **Peer-to-peer** ou ponto-a-ponto: onde todos os nodos dividem a responsabilidade de gravar/ler todos os dados, além de replicar seus dados com os demais nodos;



# Revisando: Tipos de escalabilidade horizontal

- **Sharding:**
  - **Prós:** Divide a carga/responsabilidade de escrita e leitura
  - **Contras:** Se um **shard** cai, aquela fração de dados se torna indisponível;
- **Replicação Mestre/Escravo:**
  - **Prós:** Melhora a capacidade de leitura; Aumenta a disponibilidade dos dados;
  - **Contras:** Pode gerar problemas de consistência ou disponibilidade; Não melhora a capacidade de escrita; Cria um *SPOF*;
- **Peer-to-peer :**
  - **Prós:** Melhora a capacidade de leitura e escrita; Aumenta a disponibilidade dos dados;
  - **Contras:** Pode gerar problemas de consistência ou disponibilidade; Custo elevado;

# Arquitetura MongoDB: Como escalar?





# Casos de uso

- Análise de **dados** em **tempo real**;
- Controle/Gerenciamento de **informações** de **usuários**;
- Armazenamento de **dados** de **sessão**;
- **IOT** (Internet das coisas);
- Catálogo de **produtos**;



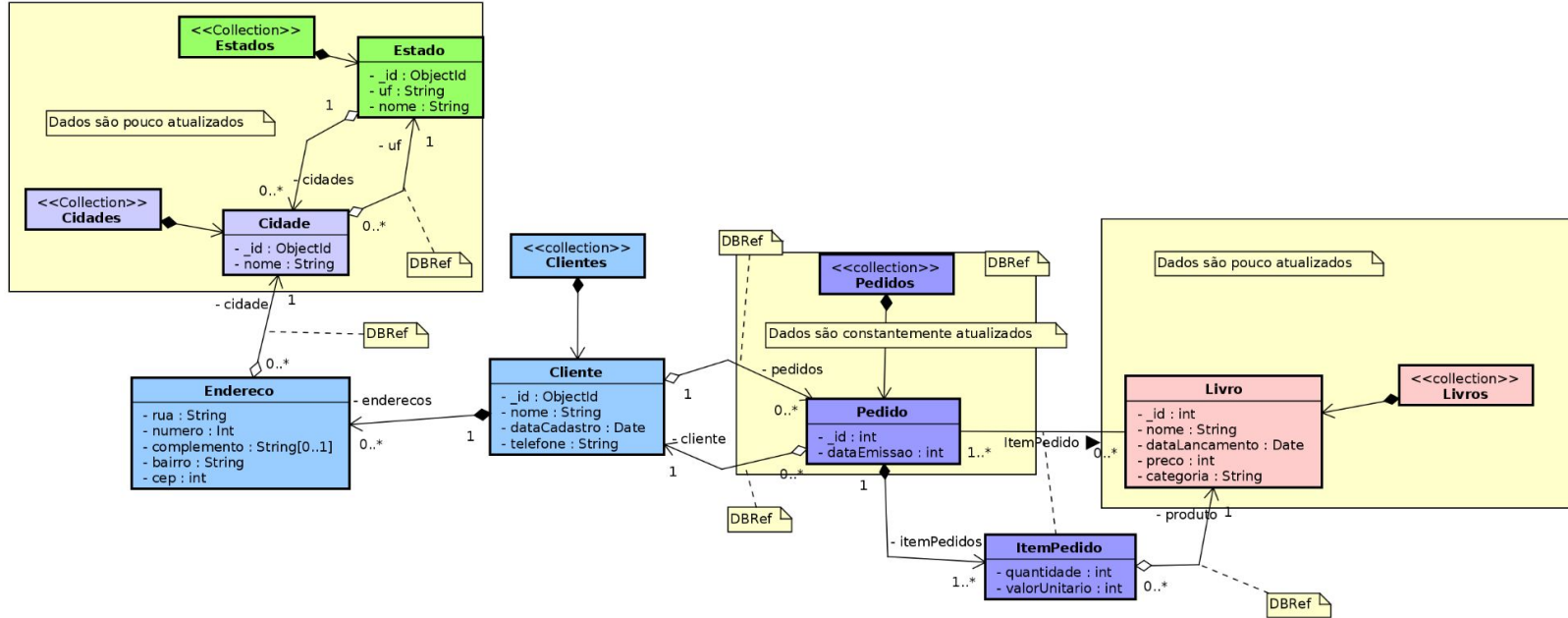
# Tradeoffs

- Não suporta **transações ACID** entre documento/coleções;
- Não suporta **JOINS** nativos entre coleções;
- Limite de **100mb** de utilização de memória em alguns estágios de **agregação**;
- Armazenamento de cada documento tem um **limite físico**;
- Utilização de **memória** para **armazenamento** é grande (mais **custos**);

# Exemplo prático

- Modelagem de dados;
- Introdução à linguagem;
- Buscas simples;
- Buscas complexas;
- Interagindo entre coleções;
- Agregações;

# Exemplo prático





*cassandra*

**Cassandra DB**

# Porquê usar Cassandra DB?

- Performance excelente na **escrita** de **dados**;
- Banco **cloud-native**, provendo nativamente replicação em diversas zonas;
- É possível **tunar/customizar** virtualmente todos os aspectos (**consistência, disponibilidade, throughput** de leitura/escrita);
- A arquitetura do banco é **masterless**, evitando pontos únicos de falha. (**SPOF'S**).
- Altamente **disponível** e **tolerante à falhas**.
- Facilmente e nativamente **horizontalmente escalável**.

# Principais conceitos

- Desenhado para suportar **escritas** em larga escala;
- Utiliza **CQL** como linguagem de pesquisa (similar a **SQL**);
- Trabalha nativamente com **consistência eventual**;
- Trabalha com o conceito de **chave-primária** (*pk*);
- Organiza os dados em **tabelas**;
- **Arquitetura em anel** (*masterless*);
- Divide os dados **nativamente** pelos nodos;

# Revisando: Problemas de Consistência

- **Tipos de conflito:**
  - **Escrita-Escrita:** Quando dois ou mais usuários tentam alterar o mesmo ponto, simultaneamente;
  - **Leitura-Escrita:** Quando um usuário obtém informações desatualizadas, devido a uma atualização concorrente (ex: saldos);
- **Janelas de inconsistência:** Basicamente a quantidade de tempo, em que um dado pode estar inconsistente (ex: tempo de replicação);
- **Evitando inconsistências:**
  - **Transações atômicas:** Atualizam todos os pontos dentro de uma transação em um único momento;
  - **Abordagem pessimista:** Impede que sejam feitas alterações no sistema através de *locks* mutuamente exclusivos;
  - **Abordagem otimista:** Deixa com que as escritas sejam feitas de forma concorrente, e ou resolve-as de forma automática, ou marca possíveis inconsistências;



# Principais conceitos

- **Keyspace:** Similar ao “**database**” dos bancos relacionais;
- **Table:** O objeto do banco de dados, similar à tabelas SQL;
- **Primary Key:** Identificador **único** de uma tabela, pode ser composto;
- **Partition Key:** Chave derivada de elementos da **PK**, que direciona os dados para sua **partição** correspondente;
- **Clustering Key:** Chave derivada, também da **PK**, que ordena os dados dentro de uma partição

# Arquitetura Cassandra: *Tombstones*

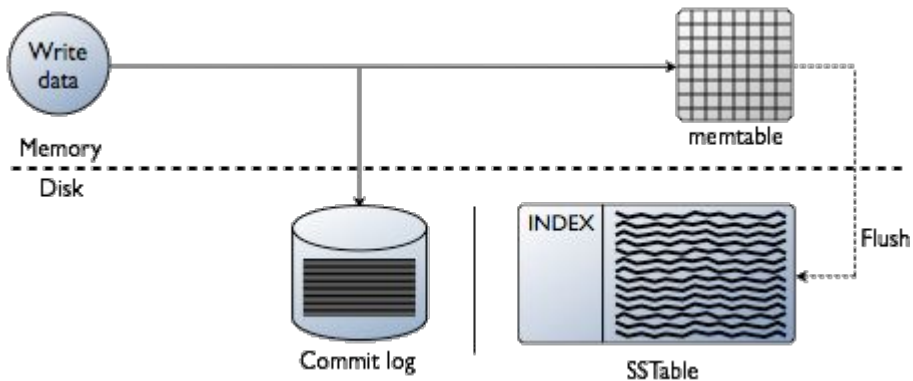


- Os dados **não são** inicialmente deletados, apenas marcados como removidos;
- Afeta a **performance** das queries, quanto mais **tombstones**, mais **filtering** é feito;
- Dependendo da frequência de deleções, é mais interessante realizar **soft deletes**;
- Os dados são **eventualmente** removidos (similar a um algoritmo de GC);

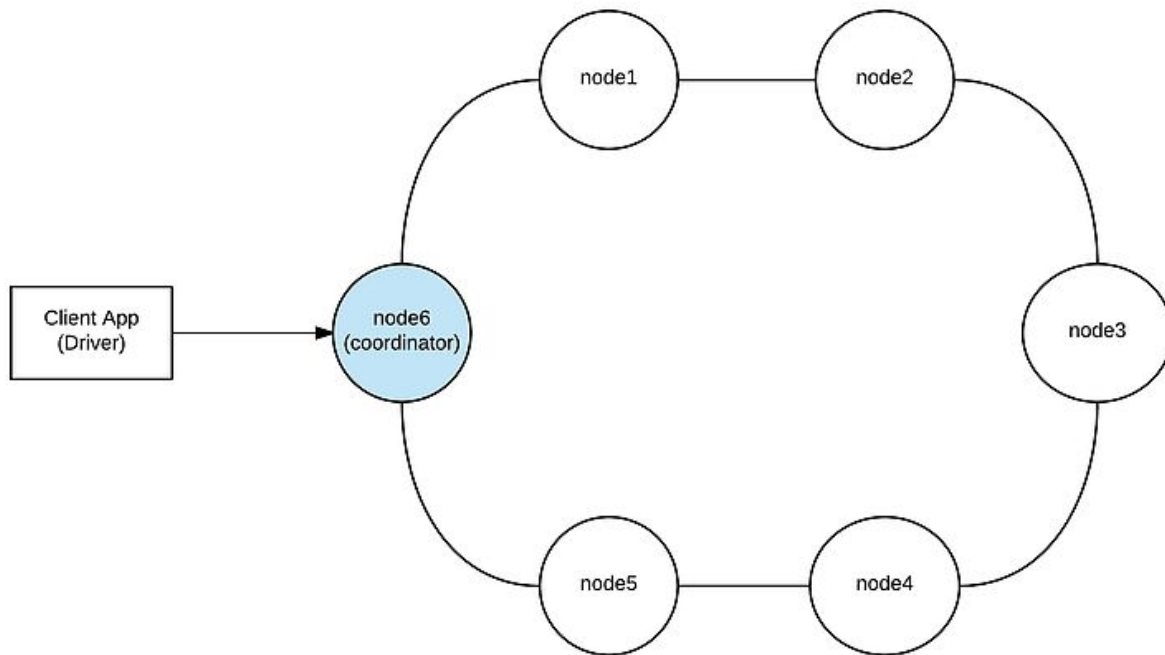
# Estratégias de compactação de dados

- **STCS:** Estratégia **padrão**. Excelente para cenários de muita **escrita**;
- **LCS:** Estratégia recomendada para cenários de muita **leitura**, e com escritas sendo majoritariamente **atualizações** (ex: saldos, agregados, etc);
- **DTCS:** Apropriada para quando trabalhamos com dados **“time-series”** (ex: logs, dados meteorológicos);

# Arquitetura Cassandra : Modelo de escrita



# Arquitetura Cassandra: Escalando



# Modelando dados

- Consultas **antes** de tabelas;
- Uma **consulta** -> uma **tabela** (**Duplicação** de dados é **comum**, e **esperado**);
- Pensar bem nas **chaves primárias**, pois consultas por outros termos, poderão causar **problemas** de **performance** (*filtering*);
- O objetivo é reduzir o número de **partições lidas** em cada consulta, então pensar nas **partition keys**;

## Casos de uso

- E-commerce e **gestão** de **inventário**;
- Eventos ***time-series***;
- Serviços de **pagamento/financeiros**;
- Distribuição e armazenamento de **conteúdo**;

NETFLIX

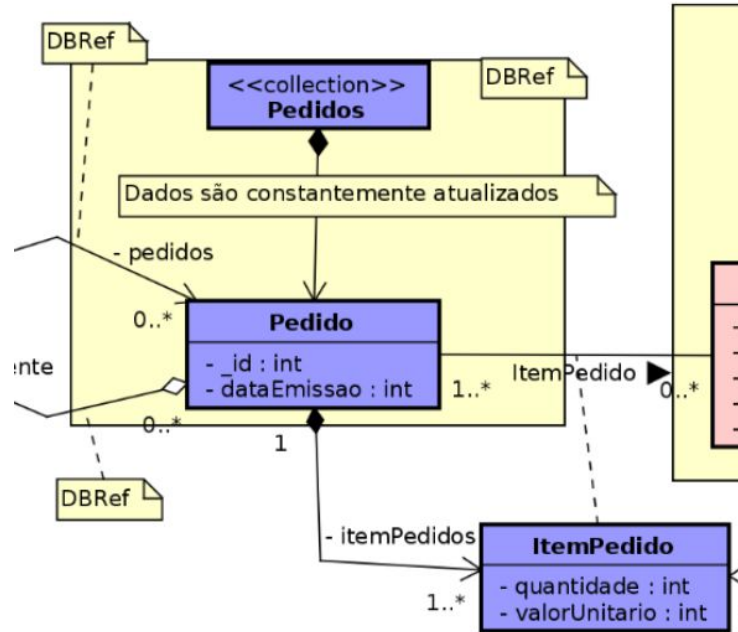


# Tradeoffs

- Não suporta transações **ACID**;
- Duplicação de dados pode se tornar um problema, rapidamente;
- De maneira geral, as **leituras** são mais **lentas** que as **escritas**;
- Não conta com **subqueries, agregações ou joins** nativos;
- **Consistência** não é o ponto forte deste banco, mesmo podendo ser tunado;
- Não serve **todos** os casos;



# Exemplo prático



# Exemplo prático

- Introdução à linguagem;
- Isolando pedidos (usando UDT);
- Remodelando pedidos;
- *Keys* em uso: buscas no Cassandra;
- Dividindo em tabelas;

**PUCRS** online  **uol**edtech.