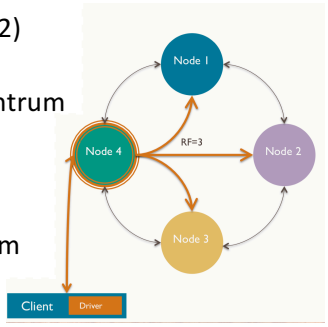
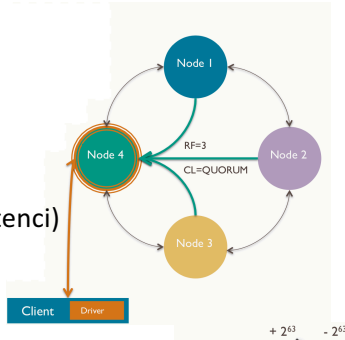
# 04. Databázový stroj Cassandra – metody čtení a zápisu

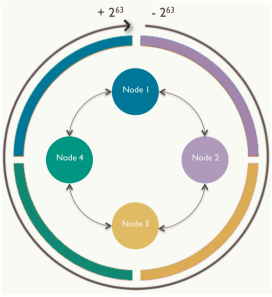
- Cassandra je distribuovaná
  - nemá single point of failure
  - automatická replikace (systém sám na základě kritérií od uživatele replikuje)
    - každý uzel je replika ← systém o jednom uzlu = 1 replika
      - first replica = ta, které odpovídá token z primárního rozsahu
      - replica = opravdu „repliky“ – odpovídají tokenu ze sekundárního rozsahu
    - **replikační faktor (RF)** – na kolik uzlů bude replikováno
    - **replikační strategie**
      - možno definovat způsoby replikace (aby např. brala v potaz geolokaci)
      - SimpleStrategy (jedno datacentrum, jeden rack)
        - jeden replikační faktor pro celý cluster (např. =2)
      - NetworkTopologyStrategy (pro více datacenter, racků)
        - samostatný replikační faktor pro každé datacentrum



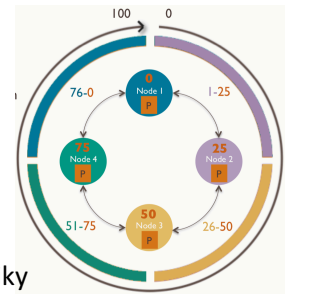

- **koordinátor**
  - uzel, který vyřídí klientův požadavek (vybrán Cassandra driverem)
    - každý uzel může být koordinátor pro danou operaci
    - každý klientův požadavek může být vyřízen jiným koordinátorem
  - stará se o zajištění **replikačního faktoru** (RF)
    - na kolik uzlů mají být data nakopírována?
    - každý zápis do každého uzlu je označen časovou značkou
    - RF se nastavuje na úrovni
      - keyspaců
      - data center
  - stará se o zajištění **consistency level** (CL)
    - kolik uzlů musí potvrdit čtení/zápis?
    - může být jiný pro každý dotaz
    - úrovně konzistence (vysvětleno níže, sry za nekonzistenci)
      - ANY
      - ONE
      - QUORUM *(RF/2)+1*
      - ALL

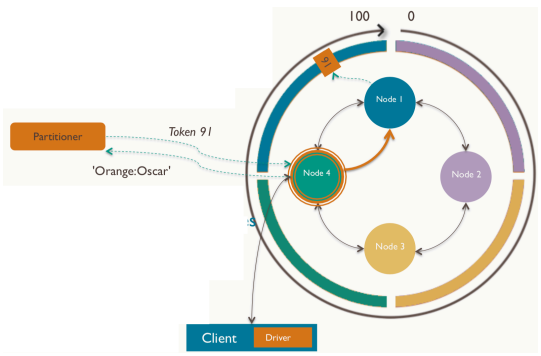



- **consistent hashing**
  - data v uzlech jsou identifikována unikátním **tokenem**
  - **partition**
    - umístění dat v uzlu (podobné řádku v tabulce)
  - **token**
    - integer generovaný hashovacím algoritmem
    - určuje umístění partitiony v clusteru
    - rozsah tokenu 2^128 (je to modulo) ➔ token ring




- **partitioner**
  - služba na každém uzlu, která hashuje klíče do tokenů
  - **node token** je vždy nejvyšší hodnota uložitelná na daném uzlu
    - pomocí ní se daný uzel identifikuje
    - tomuto se říká **přímární tokenový rozsah**
  - **sekundární tokenový rozsah**
    - „druhá vrstva“ tokenů v rámci daného uzlu, která uchovává repliky
  - proces zápisu dat do clusteru (v tomto příkladu je Node 4 koordinátorem a cílovým uzlem (určeným pomocí tokenu z Partitioneru) je Node 1
  - existují různé (tři) partitionery založené na různých (třech) hashovacích funkcích (Murmur3, MD5, lexikální pořadí bytů)



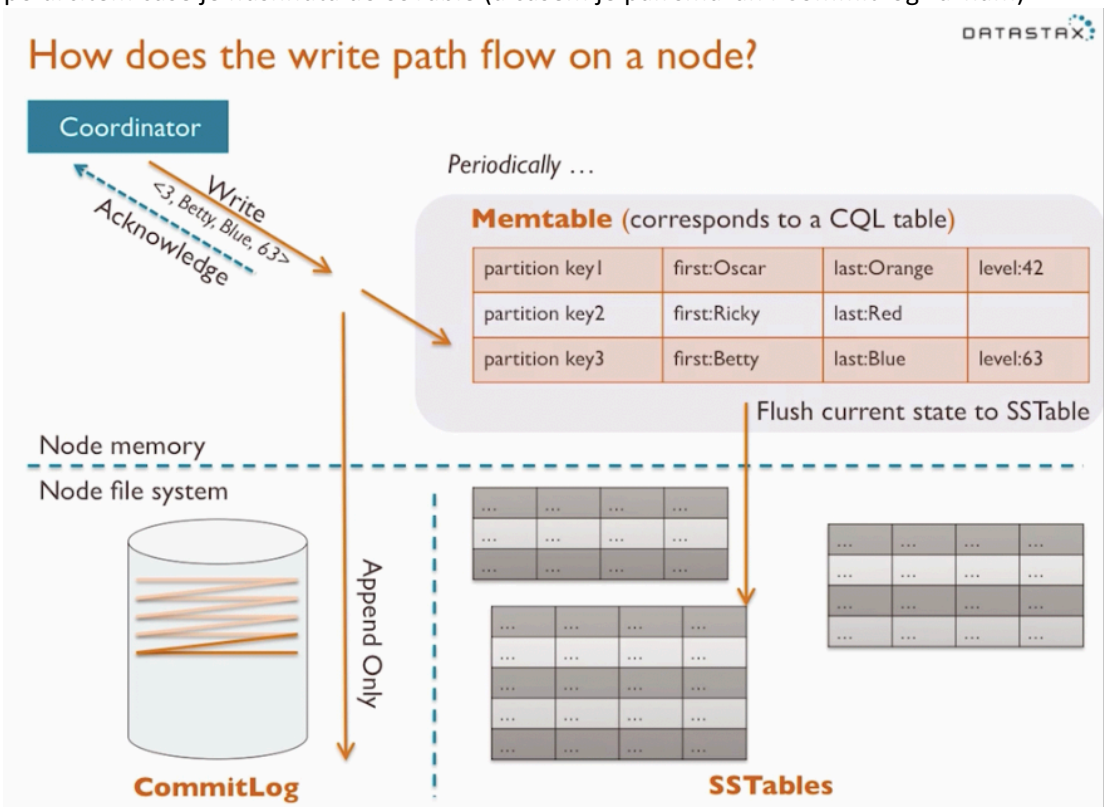




- **replikace**
  - cílový keyspace zápisu udává
    - replikační faktor
    - replikační strategii

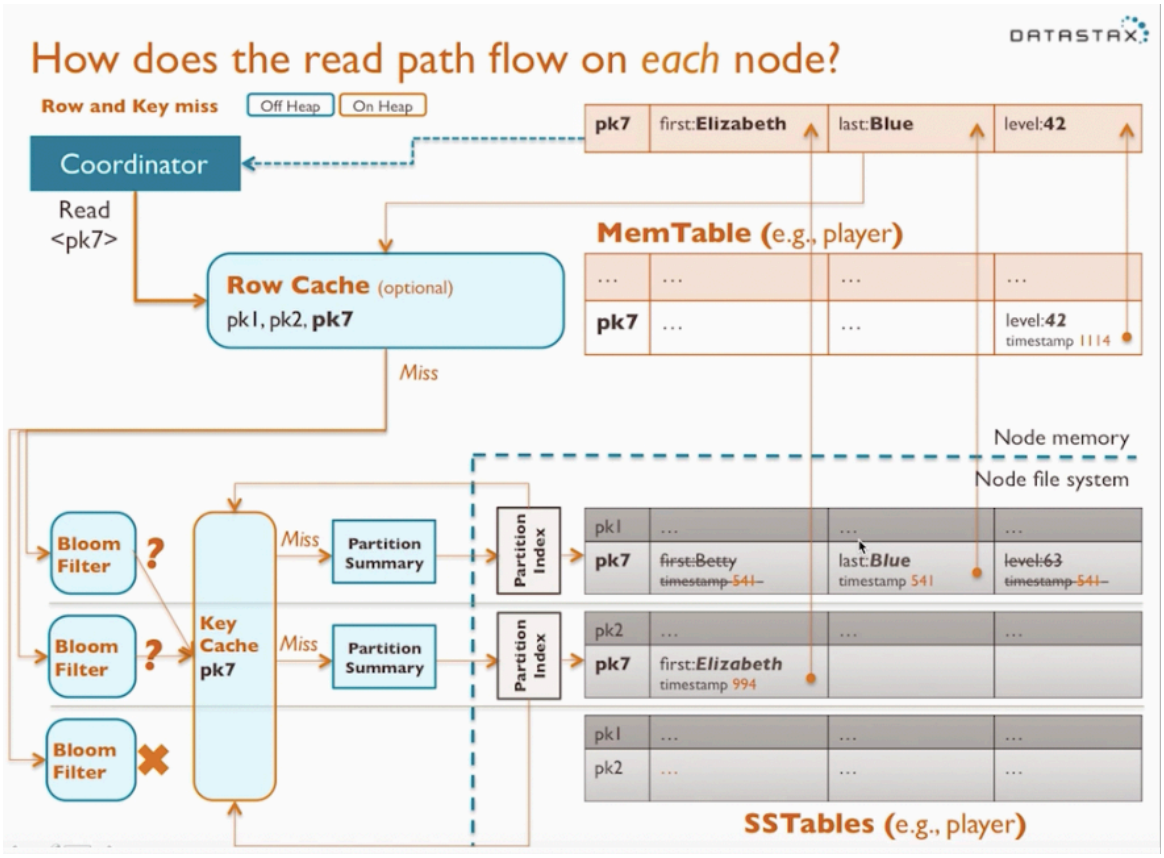
    - **hinted handoff**
        - v případě, že se snažíme zapsat data na nějaký uzel, který je zrovna nedostupný (neodpovídá/ví se o poruše) uloží koordinátor informaci o této transakci do system.hints tabulky
        - jakmile je cílový uzel online, dojde k přehrání této transakce
- rozdíl **konzistence** x **replikační faktor**
    - replikační faktor udává v podstatě dlouhodobou úroveň „zálohování“ (něco jako RAID) celých uzlů; toto „zálohování“ je odstíněno od uživatele; na úrovni keyspace, nikoliv jednotlivých dotazů
    - consistency level je naopak v rukou uživatele, který úroveň stanovuje pro každý požadavek (defaultně ONE); určuje na kolika uzlech má být informace uložena nebo naopak z kolika uzlů při čtení má být ověřena (distribuovaný systém ➔ problémy s konzistencí)
- **konzistence**
    - **consistency level**
        - říká, na kolik uzlů musí být zaslán dotaz, aby výsledek mohl být vrácen klientovi
        - požadavek na **zápis**
            - kolik uzlů musí potvrdit, že zapsali danou informaci
        - požadavek na **čtení**
            - kolik uzlů musí zaslat svoji kopii dat (pro zjištění konzistence)
        - **dostupné consistency levels**
            - **ANY**
                - pouze zápis
                - zapíše na libovolný uzel, případně do handoffu
                - nejrychlejší, nejmenší konzistence
            - **ALL**
                - čtení i zápis
                - potřeba potvrzení od všech uzlů – selže i při výpadku jediného uzlu
                - vysoká konzistence, malá dostupnost
            - **ONE** (nebo jakékoliv jiné číslo)
                - potvrzení od nejbližšího (nejbližších) uzlů vzhledem ke koordinátorovi
                - vysoká dostupnost a malá konzistence
            - **QUORUM**
                - potvrzení od (RF/2)+1 uzlů
                - vyvážená dostupnost a konzistence
        - defaultně je každý požadavek vyřizován s consistency level = ONE
    - **immediate consistency**
        - jistota vrácení aktuálních (konzistentních dat)
        - = level ALL
        - dlouhá latence (výsledky ze všech uzlů musejí být porovnány)
    - **eventual consistency**
        - v případě čtení MŮŽE vrátit konzistentní data, ale nemusí
        - = level ONE
        - zeptá se prvního uzlu, na který narazí
- Snitch = protokol, který informuje jednotlivé uzly o topologii clusteru

## Jak Cassandra čte/zapisuje rychle?

- myšlenka
    - na každém nodu se uchovává
        - CommitLog
            - klasický log všech změn uchovávaný minimálně od posledního flushe do SSTable
            - slouží k zrekonstruování vnitřní Memtable (která je v RAM)
            - append-only
            - uložena na pevném disku (vhodné např. na SSD nebo samostatné HDD – rychlost)
        - Memtables
            - CQL tabulková datová struktura uložená v paměti
            - slouží k rychlému zápisu nově příchozích dat
            - pravidelně je uložena natrvalo do SSTable (Memtable snapshots)

        - po určité době se mažou uložené tabulky, které byly již odloženy do SSTables
    - SSTables
        - natrvalo uložené snapshoty Memtables
    - Compaction
- příchozí zápis
    - změna je provedena v Memtable a zároveň uložena do CommitLogu (pro případ výpadku paměti)
    - po určitém čase je flushnuta do SSTable (a časem je pak smazán i CommitLog záznam)
    - 

- čtení
    - nejprve se mrkneme do rychlé Memtable
    - pokud není v Memtable
        - musíme hledat v SSTable (např. na HDD = pomalá)
        - používá se tzv. Bloom filter
            - probabilistická funkce
                - oznámí s pravděpodobností, že prvek v partitioně **NENÍ**
                - oznámí, že prvek v partitioně **ASI JE**
                - funguje na základě hashe (udržuje si tabulku zapsaných hashů a nově příchozí data zahashuje a porovná)
                - hezký příklad
                    - https://llimllib.github.io/bloomfilter-tutorial/
        - v případě nalezení se data nakopírují do Memtable pro brzké využití

- 
How does the read path flow on each node?
DATASTAX
Row and Key miss
Off Heap
On Heap
Coordinator
Read
<pk7>
Row Cache (optional)
pk1, pk2, pk7
Miss
MemTable (e.g., player)
pk7
first:Elizabeth
last:Blue
level:42
level:42
timestamp 1114
Node memory
Node file system
Bloom Filter
Bloom Filter
Bloom Filter
Key Cache
pk7
Miss
Miss
Partition Summary
Partition Summary
Partition Index
Partition Index
pk1
pk7
first:Betty
timestamp 541
last:Blue
timestamp 541
level:63
timestamp 541
pk2
pk7
first:Elizabeth
timestamp 994
pk1
pk2
SSTables (e.g., player)