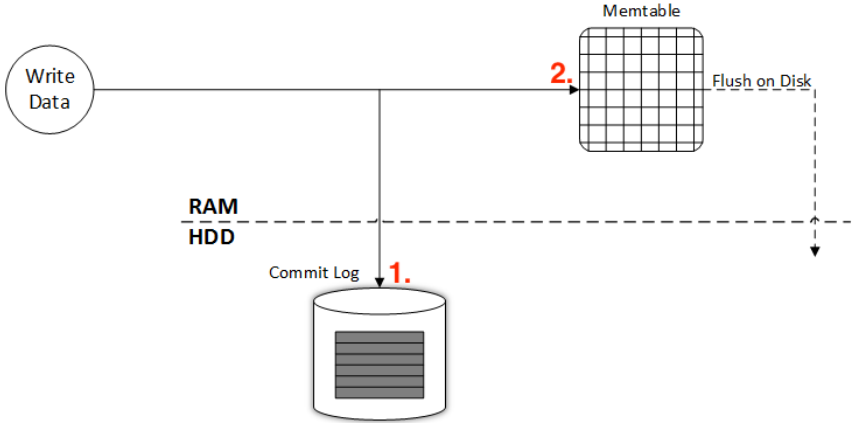
Napredni algoritmi i strukture podataka

Memorijske tabele (Memtable), Eksternalizacija podešenja, Put zapisa (Write path)

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Memorijska tabela - ideja

- Ideja iza Memorijske tabele (Memtable) je relaltivno jednostavna — zapisati podatke u memorju i čitati podatke iz memorije
- **AKO** se podaci nalaze u memoriji, sve operacije su relativno brže nego da su podaci **striktno** na disku
- Memorija je brza, memorija je super, memorija je kul, svi vole memoriju
- Memorija je aktivna dok je sistem aktivan
- **ALI** memorija nije sigurna :/
- Zato sistem komunicira sa WAL-om prvo, koji nam daje ove garancije, pa onda zapisuje u Memtable



(Cassandra memtable)

Memorijska tabela — struktura podataka

- Jednostavna struktura koju smo radili, i koja se dosta koristi za Memtable je *SkipList*
- RocksDB i LevelDB na primer direkno koristi SkipList
- Izvod iz RocksDB dokumentacije
  - *Skiplist-based memtable provides general good performance to both read and write, random access and sequential scan.* (RocksDB Memtable Docs)
- Što se nas tiče, mi se možemo držati ove strukture podataka — učimo i ugledamo se na najbolje :)
- I **plus**, implementirali ste je na vežbama :D

Memorijska tabela — zapis na disk

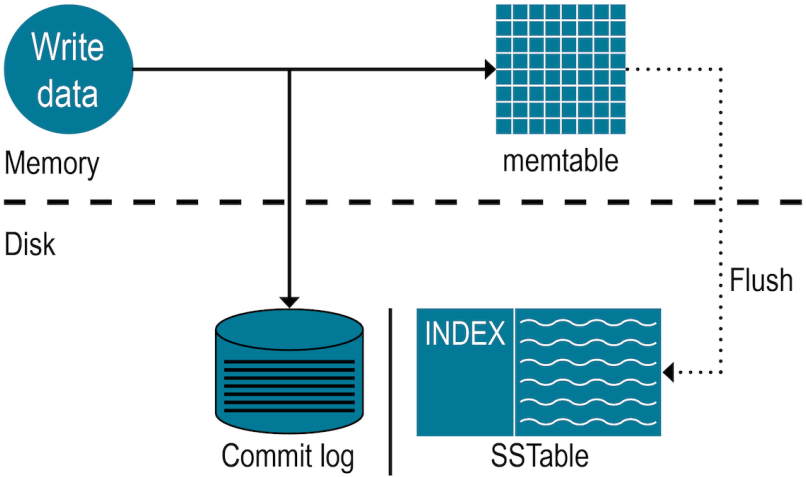
- Memtable se implementira kao struktura **fiksong kapaciteta**
- Setimo se segmenata i WAL-a, jedan segment može biti veličine kao i Memtable
- Memtable imaju granicu ili prag zapisa — **trashold** koji je varijabilnog karaktera
- Kada se Memtable struktura popuni, prekorači se granica, ona se perzistira na disk — operacija **Flush**
- Na primer podrazumevana veličina Memtable-a kod LevelDB-a je oko 4MB (koristi SkipList)
- Izvod iz njihove dokumentacije:
  - *When the log file reaches a certain size (around 4 MB), its content is transferred to a new SST file and a new log file and memtable are initiated, the previous memtable is discarded.* (LevelDB docs)
- Iz ovoga možemo da zaključimo i veličinu segmenta koju LevelDB koristi

Eksternalizacija podešenja

- Kada pravimo sistme koji se konfiguriše kroz eksterne fajlove, trebamo obezbediti podrazumevane vrednosti — **default**
- Ovo možemo da uradimo na dva mesta, da se osiguramo i zaštitimo od potencijalnih problema
  1. Obezbediti fajl sa default vrednostima — isti fajl za konfiguraciju samo već popunjen vrednostima
  2. **AKO** takav fajl ne postoji, obezbediti da kroz kod postoje default opcije koje program može da iskoristi
- Na ovaj način imamo redudanciju, i sistem nam je stabilniji
- Ovo nije obaveza, ali je generlano lepa praksa

Putanja zapisa — Algoritam

1. Korisnik je poslao zahtev — nekakvu operaciju (dodavanje, čitanje, izmena, brisanje — CRUD)
2. Podatak se prvo zapisuje u **WAL**
3. Kada WAL potvrdi zapis, podatak se zapisuje u **Memtable**
4. Koraci **(2)** i **(3)** se ponavljaju dokle god ima mesta u Memtable-u
5. Ako je kapacitet Memtable-a popunjen, Memtable sortira parove ključ-vrednost
6. Sortiraten vrednosti se zapisuje na disk formirajući **SSTable**
7. Možemo isprazniti **Memtable** ili napravitu nov, a prethodni uništiti ili rotirati



(Cassandra write path)

Zadaci

- Implementirati Memtable gde se kao strutura podataka koristi SkipList koju ste implementirali na vežbama
- Napraviti saradnju Write Ahead Log-a, tako da se podaci **prvo** zapišu tu, pa kada se odbije potvtda zapisa, da se podaci **onda** zapišu u Memtable
- Omogućiti da se veličina Memtable-a specifcira kroz YAML konfiguracini fajl
- Omogućiti da se veličina WAL segmenta specificira kroz YAML konfiguracini fajl
- Omogućiti da se **trashold** specificira kroz konfiguracini fajl
- Kada se **trashold** postigne, sortirati vrednosti po ključu i ispisati na ekran, a zatim obrisati podatke iz Memtable
- Memtable prihvata dodavanje; izmenu **AKO** je podatak sa tim ključem prisutan, ako nije uraditi dodavanje; brisanje je logičko postaviti vrednost **tombstone** parametra na true