SSD
baziran na flash memoriji

Profesor:
Dr Miloš Ivanović

Autor:
Dragutin Ostojić

Sadržaj:

Uvod

Nakon decenija dominacije hard diskova usled sve većih potreba za boljim performansama i njihove nemogućnosti da to isprate došlo je vreme za prelazak na sasvim nove tehnologije. SSD (Solid-state drive) ili na našem jeziku poluprovodnički (takođe i električni) uređaj za čuvanje podataka vremenom se ustalio kao rešenje. Njegova glavna razlika u odnosu na hdd i flopy tehnologije je to što nema disk niti bilo koju mehaničku komponentu. Danas se uglavnom baziraju na flash memoriji (preciznije NAND flash memoriji) koja se sastoji od tranzistora, ima sposobnost da i po prestanku napajanja čuva informacije, a izumeo je Fujio Masuoka 1980. godine.

Odstustvo mehaničkih komponenti i silicijumski sastav SDD čini znatno otpornijim na fizička oštećenja, nečujan je u radu, ima manju potrošnju energije, dok uz sve to ostvaruje mnogo bolje performanse. Međutim cena je dugo predstavljala problem, a specifičnosti rada sa flash memorijom kao i njen glavni nedostatak koji se ogleda u tome da jedinice za čuvanje podataka imaju ograničen broj I/O operacija pre otkazivanja stvorile su potrebu za pronalaženjem niza zanimljivih rešenja. Njima ćemo se baviti u ovom tekstu.

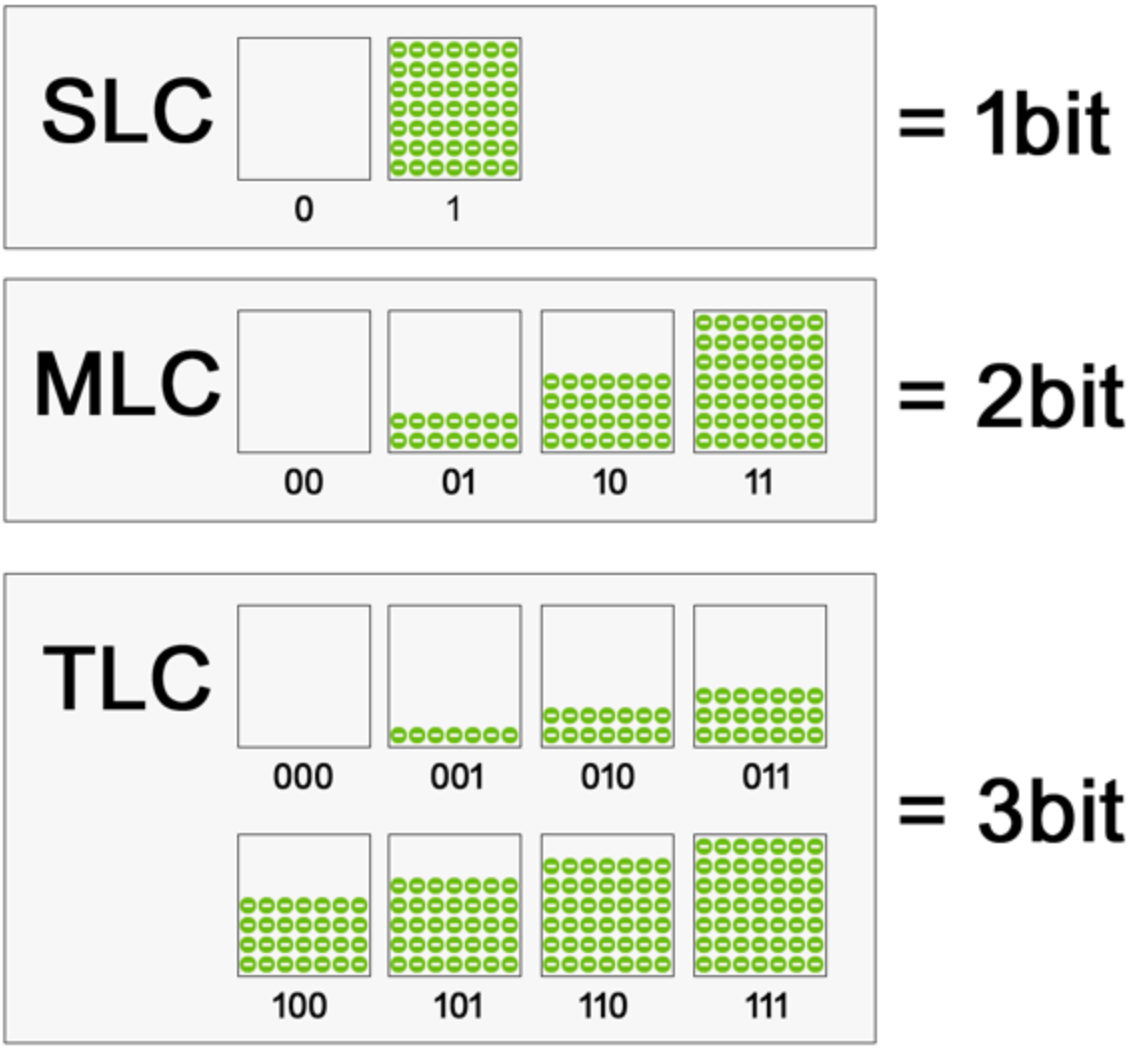
Slika 1: HDD i SSD

Čuvanje jednog bita

Fleš čipovi su dizajnirani da čuvaju jedan ili više bitova u jednom tranzistoru. Nivo naelektrisanja zarobljenog u tranzistoru određuje binarnu vrednost koja se čuva.
U upotrebi su tri vrste tranzistora:

- **single-level cell (SLC)** - samo jedan bit po tranzistoru 0 ili 1
- **multi-level cell (MLC)** - dva bita mogu biti upisana u jednom tranzistoru, što kao rezultat daje četiri vrednosti 00, 01, 10, 11, a one su ekvivalenti niskom, donekle niskom, donekle visokom i visokom nivou naelektrisanja.
- **triple-level cell (TLC)** - tri bita mogu biti zapisana po ćeliji na sličan način

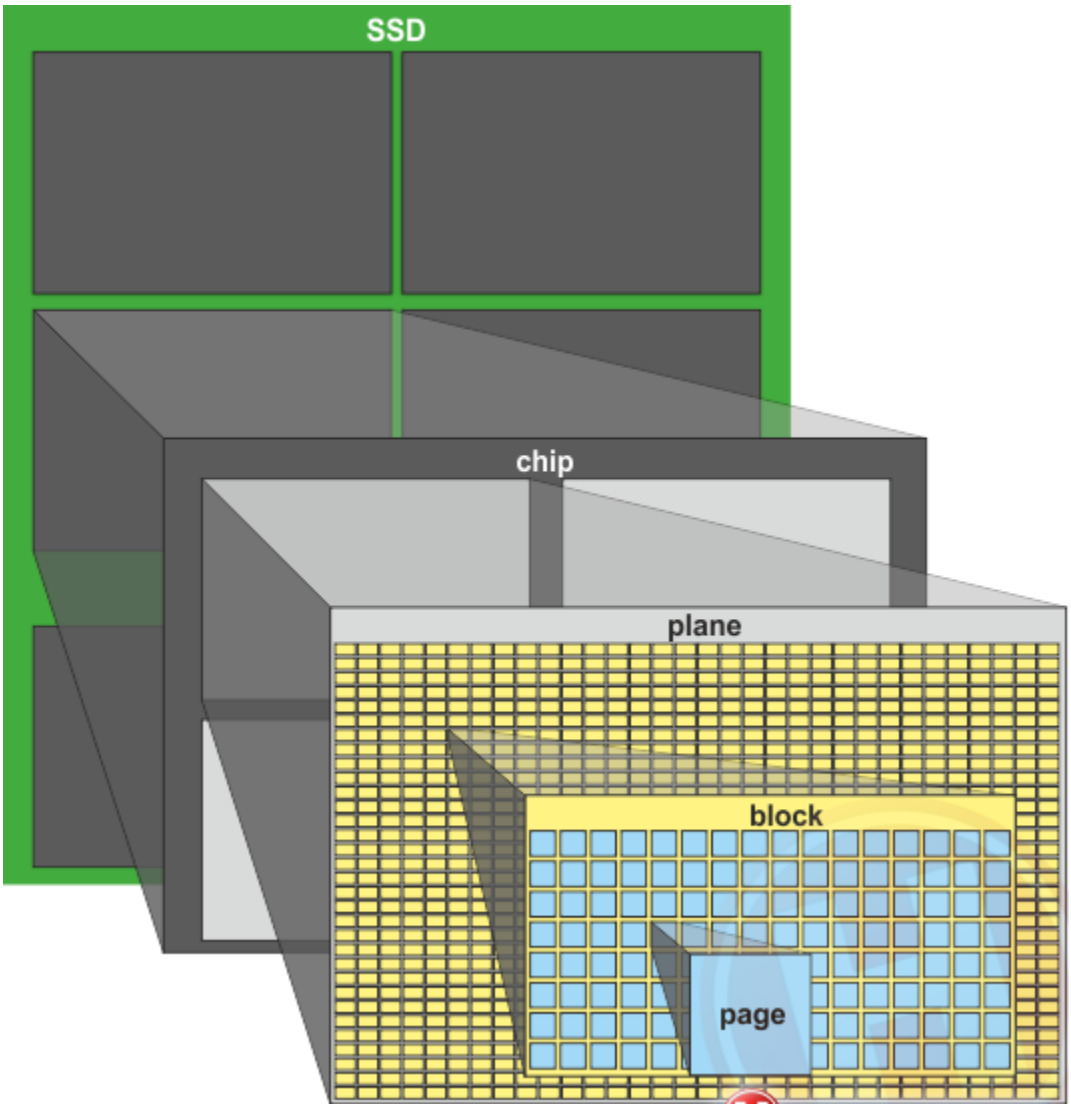
Generalno, SLC imaju najbolje performanse ali i najvišu cenu, a TLC najslabije performanse ali najpristupačniju cenu.



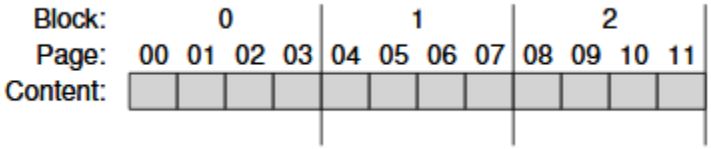
Slika 2: SLC, MLC i TLC

Od bitova do oblasti

Interno flash memorija je organizovana po oblastima koje su podeljene na blokove, a oni na stranice u kojima se nalaze ćelije koje čuvaju informacije. Tipična veličina stranice je 4K, a bloka 128K - 256K, dok se oblasti sastoje od velikog broja blokova.



Slika 3: Struktura SSD-a



Slika 4: Uprošćen prmer 3 bloka sa po 4 stranice

Osnovne operacije flash memorije

S obzirom na organizaciju fleša, postoje tri operacije niskog nivoa koje fleš čip podržava:

1) **Čitanje (stranice)**: Korisnik fleš čipa može čitati bilo koju stranicu (npr. 2KB ili 4KB), jednostavnim navođenjem komande za čitanje i odgovarajućeg broja stranice. Ova operacija se poprilično brzo izvršava, traje svega nekoliko desetina mikrosekundi, nezavisno sa kog dela uredjaja se čita i nezavisno od lokacije poslednjeg citanja (što nije slučaj kod hard diska). Mogućnost pristupa bilo kojoj lokaciji istom brzinom, čine ovaj uređaj uređajem sa slučajnim pristupom (**random access**).
2) **Brisanje (bloka)**: Pre upisivanja stranice, priroda uređaja zahteva da se prvo izbriše ceo blok unutar kog se stranica nalazi. Brisanje uništava sadržaj bloka (postavljanjem svakog bita na vrednost 1). Zato morate biti sigurni da je svaki važan podatak koji se nalazi u tom bloku, sačuvan na nekom drugom mestu pre brisanja. Komanda brisanja je prilično skupa, izvršenje traje nekoliko milisekundi. Nakon brisanja ceo blok se resetuje i svaka stranica je spremna za programiranje.
3) **Programiranje (stranice)**: Kada se blok obriše, komanda za programiranje se može koristiti za promenu odgovarajućih bitova u okviru stranice sa jedinica na nule. Na taj način se upisuje željeni sadržaj na fleš. Programiranje stranice je manje skupo u odnosu na brisanje bloka, ali je skuplje od čitanja stranice (na modernim fleš čipovima ovaj proces traje nekoliko stotina mikrosekundi).

Jedan od načina razmišljanja o fleš čipovima jeste da se svakoj stranici pridružuje stanje. Na početku stranica se nalazi u ***INVALID*** stanju. Brisanjem bloka unutar kojeg se stranica nalazi, stanje stranice se postavlja na ***ERASED***, kao i svih stranica unutar tog bloka, i čini ih programabilnim. Nakon što se stranica isprogramira, stanje postaje ***VALID***, što znači da je njen sadržaj upisan i da se može čitati.

Primer promene stanja nakon više operacija brisanja i programiranja unutar bloka koji se sastoji od 4 stranice:

		`iiii`	*Initial: pages in block are invalid* (`i`)
Erase()	→	`EEEE`	*State of pages in block set to erased* (`E`)
Program(0)	→	`VEEE`	*Program page 0; state set to valid* (`V`)
Program(0)	→	**error**	*Cannot re-program page after programming*
Program(1)	→	`VVEE`	*Program page 1*
Erase()	→	`EEEE`	*Contents erased; all pages programmable*

Slika 5: Operacije nad blokom

Performanse i pouzdanost fleš memorije

Pošto nas interesuju uređaji za čuvanje podataka izgrađeni od fleš čipova, dobro je da znamo njihove osnovne karakteristike preformansi. *Slika 6* prikazuje kašnjenje prilikom izvršavanja osnovnih operacija (čitanja, programiranja i brisanja), poredeći SLC, MLC i TLC fleš čipove, koji skladište 1, 2 i 3 bita informacija po ćeliji, respektivno.

Na osnovu tabele možemo videti, da je kašnjenje prilikom čitanja prilično malo i potrebno mu je svega desetak mikrosekundi da se izvrši. Kašnjenje programiranja je veće i varira od 200 mikrosekundi kod SLC-a, ali raste sa povećanjem broja bitiva u ćeliji. Konačno, brisanja su prilično skupa i potrebno im je nekoliko milisekundi da se izvrše. Suočavanje sa ovim troškovima predstavlja glavni zadatak u izradi modernih fleš memorija.

	SLC	MLC	TLC
Bits per cell	1	2	3
P/E Cycles	100,000	3,000	1,000
Read Time	25 μs	50 μs	~75 μs
Program Time	200-300 μs	600-900 μs	~900-1350 μs
Erase Time	1.5-2 ms	3 ms	4.5 ms

Slika 6: Poređenje performansi flash čipova

Što se tiče pouzdanosti, za razliku od mehaničkih diskova koji mogu propasti iz različitih razloga (uključujuci i dolazak do fizičkog kontakta glave za pisanje i površine za snimanje), fleš čipovi su izrađeni od čistog silicijuma, a samim tim ima i manje razloga za brigu. Primarni problem je habanje prilikom brisanja i programiranja bloka (**wear out**), gde dolazi do nakupljanja viška naelektrisanja, vremenom taj višak naelektrisanja se uvećava i postaje sve teže napraviti razliku između 0 i 1. Kada je nemoguće napraviti razliku blok postaje neupotrebljiv.

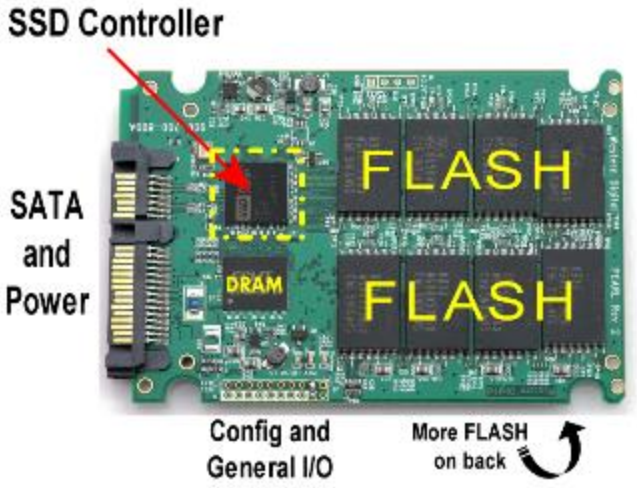
Prosečan vek trajanja bloka trenutno nije poznat. Proizvodjači fleševa zasnovanih na MLC tehnologiji ocenjuju da je njihov životni vek 10000 pisanja/brisanja. Pošto čipovi zasnovani na SLC tehnologiji čuvaju jedan bit po tranzistoru, njihov životni vek se procenjuje na oko 100 000 pisanja/brisanja. Međutim naknadna istraživanja su pokazala da je životni vek zapravo dosta veći od očekivanog.

Još jedan problem koji se javlja kod fleš čipova su smetnje (**disturbance**). Prilikom pristupa određenoj stranici fleša, moguće je da neki bitovi pređu u susedne stranice. Taj prelazak bitova može nastati i prilikom čitanja (read disturbs), ali i prilikom programiranja (program disturbs).

Od flash-a do flash-baziranog SSD-a

S obzirom na naše razumevanje fleš čipova, susrećemo se sa sledećim zadatkom: kako pretvoriti osnovni skup fleš čipova u nešto što izgleda kao tipičan uređaj za skladištenje podataka. Standardni interfejs za skladistenje podataka je zasnovan na blokovima, gde su blokovi (sektori) veličine 512 bajtova i iz njih se mogu čitati ili u njih upisivati podaci, sa zadate adrese. Zadatak SSD-a zasnovanog na fleš tehnologiji je da obezbedi standardni blok interfejs povrh fleš čipova unutar njega.

SSD se sastoji od određenog broja fleš čipova za trajno skladištenje podataka. Takođe se sastoji od određene količine memorije koja nije trajna (SRAM) ta memorija je korisna kod keširanja i baferovanja podataka kao i za mapiranje tabela. SSD sadrži kontrolnu logiku kojom upravlja operacijama uredjaja.



Slika 7: Arhitektura SSD-a

Jedna od osnovnih funkcija ove kontrolne logike je da ispuni korisnikove zahteve za čitanje i pisanje, pretvarajući ih u osnovne operacije fleša, po potrebi. Sloj za prevođenje (**flash translation layer - FTL**), pruža upravo tu funkcionalnost. FLT prihvata zahteve za pisanje i čitanje i pretvara ih u operacije nižeg nivoa (čitanje, brisanje, programiranje) koje se izvršavaju na osnovnim fizićkim blokovima i stranama od kojih je sam fleš uređaj sačinjen. FLT bi trebalo da izvrsi ovaj zadatak ostvarujući odlične preformanse i visoku pouzdanost.

Odlične performanse mogu biti ostvarene kombinacijom tehnika. Jedan od načina je upotreba više fleš čipova paralelno. Još jedan od načina da se povećaju performanse je smanjenje amplifikacije pisanja (**write amplification**), koja se definiše kao ukupna količina podataka (u bajtovima), zadata od strane FLT-a, koja je zapisana u fleš memoriju, podeljena sa količinom podataka (u bajtovima) koju je korisnik želeo da zapiše. Loše konstruisan sloj za prevođenje (FLT) dovodi do visoke amplifikacije pisanja i niskih performansi.

Visoka pouzdanost će biti ostvarena kroz kombinaciju nekoliko različitih pristupa. Glavni problem predstavlja habanje (**wear out**). Ako se u jedan blok previše puta upisuju i iz njega brišu podaci, postaće neupotrebljiv. Zbog toga bi jedna od uloga FLT-a bila da obezbedi da se svi blokovi uređaja ravnomerno koriste.

Još jedan problem pouzdanosti jesu smetnje prilikom programiranja (**program disturbance**). Kako bi minimizovali takve smetnje FTL uglavnom programira stranice unutar izbrisanog bloka redom od najniže ka najvišoj stranici. Ovakav pristup minimizuje smetnje i široko se koristi.

FTL organizacija: loš pristup, direktno mapiranje

Najjednostavnija organizacija FTL-a se naziva direktno mapiranje (**direct mapped**). U ovom pristupu, čitanje logičke stranice N je direktno čitanje fizičke stranice N. Međutim pisanje u logičku stranicu N je komplikovanije, jer bi u tom slučaju FTL prvo pročitao ceo blok u kome se stranica N nalazi, zatim obrisao čitav blok, a nakon toga upisao kako stare tako i novu stranicu N.

Ovakav pristup je jako loš zbog nepotrebnog brisanja i pisanja koji su jako skupi procesi. Ovaj vid konstrukcije FTL-a je čak sporiji i od tipičnih hard diskova.

Stuacija se još gora kada se sagleda pouzdanost uređaja koji koristi ovakav pristup. Ako se npr. metapodaci fajlsistema često menjaju sto je obično slučaj ili jednostavno se to dešava sa korisničkim podacima, jedan isti blok se neprestano brise i ponovo upisuje što brzo dovodi do njegovog otkazivanja. Ovaj način je dakle jako loš i po pitanju performansi i po pitanju pouzdanosti, jer daje previše kontrole korisniku kada je u pitanju brisanje i upisivanje stranica, koje ako se ne rasporedi ravnomerno dovodi do brzog otkazivanja popularnih lokacija. Ovaj prisup se u praksi ne koristi.

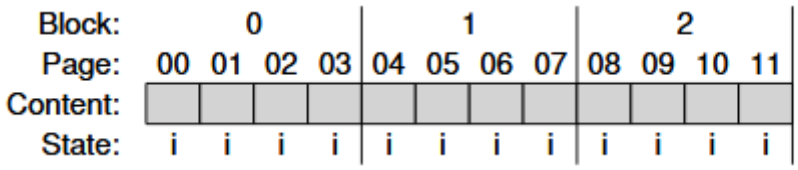
Log-struktuirani FTL

Iz prethodno navedenih razloga vićina današnjih FTL-ova su log-struktuirani, a to je ideja za koju ćemo dalje videti da je jako korisna kod svih vrsta uređaja za skladištenje podataka. Umesto upisivanja logičke stranice N na fizičko mesto N na disku, ona se ovim pristupom upisuje kao sledeća stranica na bloku na kome je upisana prethodna stranica (ili se briše prvi slobodan blok ako je trenutni popunjen), a stvarne lokacije logičkih stranica se pamte u tabeli (koja se privremeno čuva u radnoj memoriji uređaja, a povremeno snima u specifičnoj formi i na samom uređaju).

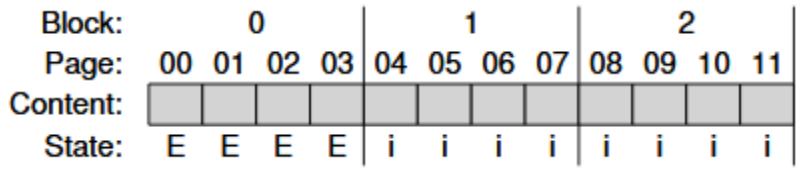
Na primeru će biti redom izvršeni sledeći zahtevi:

- Write(100) with contents a1
- Write(101) with contents a2
- Write(2000) with contents b1
- Write(2001) with contents b2

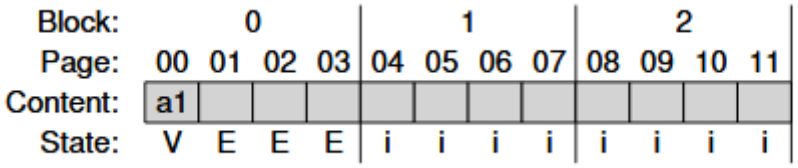
Slika 8: Grupa zahteva



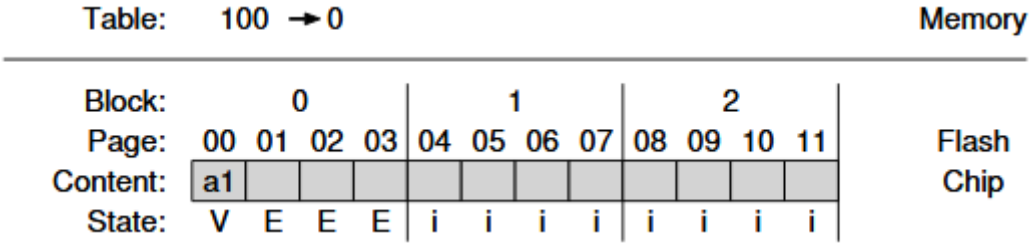
Slika 9: Početno stanje



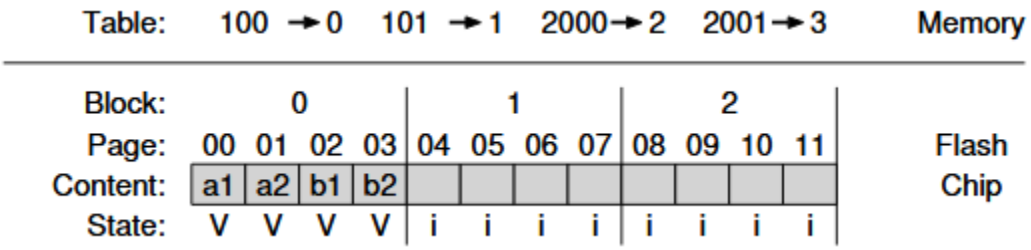
Slika 10: Brisanje prvog slobodnog bloka



Slika 11: Upivinje stranice a1 na prvo slobodno mesto bloka 0



Slika 12: Pamćenje u log tabeli fizičke adrese logičke stranice 100



Slika 13: Upisivanje ostalih stranica

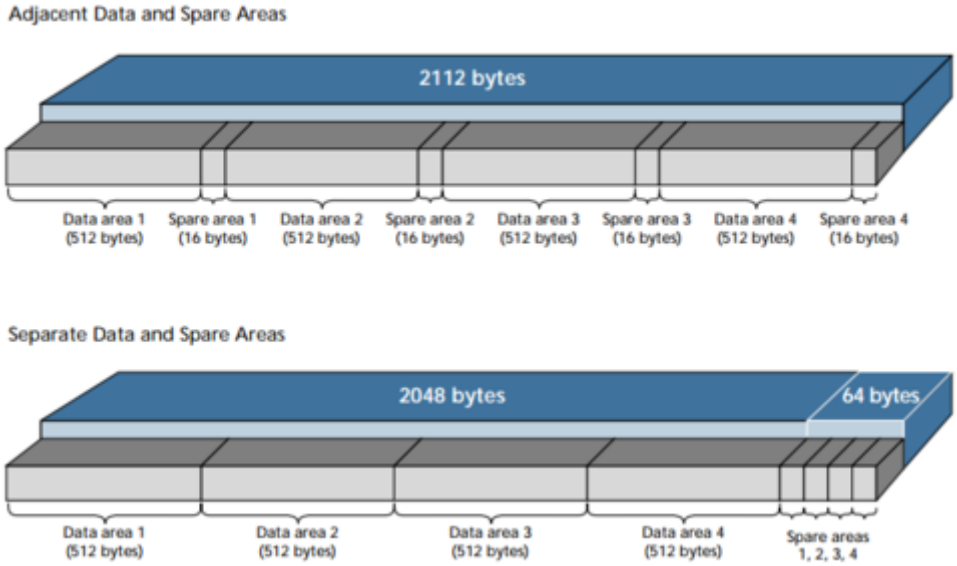
Log-baziran način svojom prirodom poboljšava performanse (brisanje se dešava mnogo ređe, a troškovi čitanja, ažuriranja i upisivanja su drastično smanjeni u poređenju sa direktnim mapiranjem. FTL sada može ravnomerno da rasporedi upisivanje na svim blokovima što se naziva **wear leveling** i što znatno produžava životni vek uređaja.

Na žalost osnovi način log-struktuiranja ima i svoje mane. Prva je to što prepisivanje logičkih blokova (usled izmena na njima) prouzrokuje takozvani otpad (**garbage**) koji predstavalja stare verzije podataka i zauzima prostor. Zbog toga uređaj mora periodično da obavlja sakupljanje otpada (**garbage collection**) da bi pronašao takve blokove, obrisao ih i oslobodio za ponovno upisivanje. Prekomerno sakupljanje otpada povećava amplifikaciju pisanja i degradira performanse. Drugi veliki trošak predstavljaju tablice mapiranja koja se nalaze u radnoj memoriji uređaja. Što su veći uređaji to zahtevaju više radne memorije.

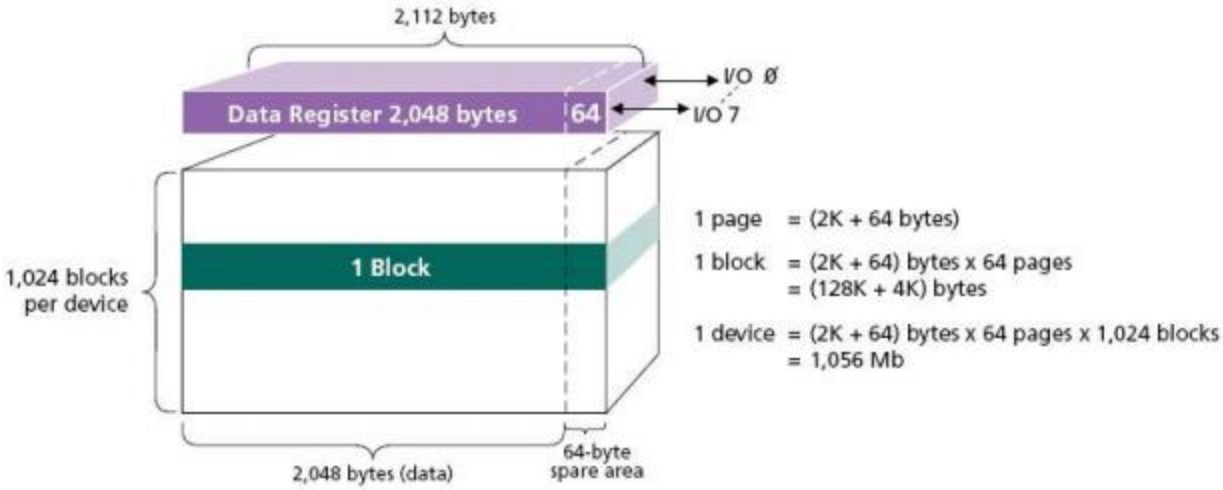
FTL Mapiranje: pouzdanost tabela sa podacima

Sigurno ste se zapitali šta se dešava ako nestane struja i tabela sa podacima u radnoj memoriji SSD-a bude obrisana. Da budemo jasni, ona nikako ne sme biti trajno izgubljena, jer bi u tom slučaju podaci na disku postali neupotrebljivi, tako da moramo obezbediti neki sistem obnavljanja.

Najjednostavnije što možemo uraditi je čuvanje informacija o mapiranju sa svakom stranom u nečemu što se zove **OOB (out-of-band)** oblast ili **spare area**. Kada uređaj izgubi napajanje i bude restartovan on mora da rekonstruiše tabelu mapiranja skeniranjem svih OOB oblasti. Glavni problem ovog pristupa je dužina trajanja skeniranja velikih SSD-ova. Da bi se prevazišao napredniji uređaji koriste dosta komplikovanije logging i checkpointing tehnike.



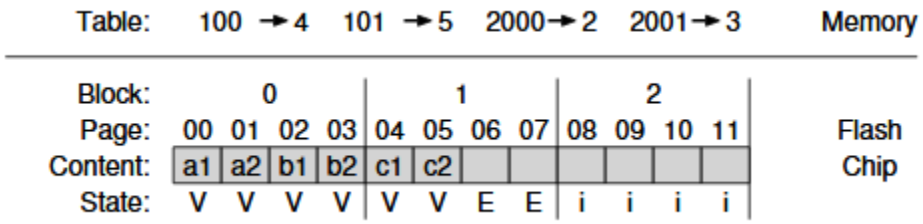
Slika 14: Dva načina smeštanja OOB oblasti



Slika 15: Struktura flash-a sa OOB oblastima

Sakupljanje otpada

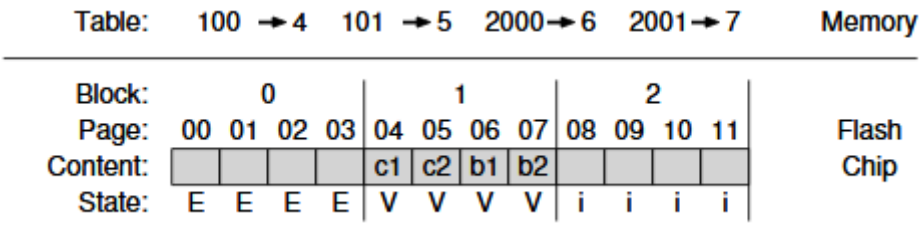
Svaki log-strukturirani način ima problem da stvara otpad koje se povremeno mora sakupiti. Nadovežimo se na primer sa slike 13 i obradimo zahtev za promenom logičkih stranica 100 i 101 vrednostima c1 i c2:



Slika 16: Upisivanje novih vrednost iza logičke stranice 100 i 101

Primetimo da sada postoji otpad na fizičkim stranicama 00 i 01. U nekom momentu da bi FTL oslobodio prostor za nova upisivanja započeće proceduru za sakupljanje otpada. Za to su mu potrebne informacije o tome da li su stranice koje posmtra otpad ili ne. Do te informacije može doći ako za svaku fizičku stranicu pamti na koju logičku stranicu se ona odnosi (OOB), a potom iz tabele mapiranja saznaje da li je ona aktuelna ili zastarela verzija.

U našem primeru pristupićemo čišćenju bloka 0 i na prethodno opisani način doći do zaključka da su informacije na stranama 02 i 03 validni podaci, a 00 i 01 su kandidati za brisanje. Kada obavimo proces sakupljanja otpada dobićemo sledeće stanje flash-a:



Slika 17: Stanje nakon sakupljanja otpada iz bloka 0

Kao što vidimo sakupljanje otpada može biti skupa operacija jer zahteva brisanje i upisivanje. Idealni kandidat za sakupljanje otpada bio bi blok koji sadrži samo otpad, jer bi tada mogli samo da ga obrišemo.

Da bi smanjili troškove sakupljanja otpada mnogi SSD-ovi podležu tehnici dodavanja dodatne količine flash memorije preko naznačenog kapaciteta uređaja (**overprovision**). To im omogućava da ređe vrše skupe operacije čišenja, tj. da to rade samo kada je uređaj pod malim opterećenjem.

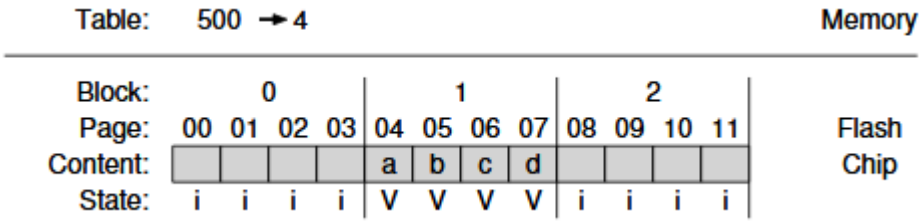
Veličina tabele mapiranja

Drugi trošak log-struktuiranja je potencijalno jako velika tabela mapiranja, sa jednim upisom za svaku 4KB stranicu. SSD od 1TB na primer, zahteva 1GB memorije potrebne samo za mapiranje. Tako da je šema mapiranja po stranici prilično nepraktična.

Blok-bazirano mapiranje

Jedan način da se redukuju troškovi mapiranja je pamćenje samo pokazivača na blokove umesto pamćenja pokazivača na stranice. Tako se znatno redukuje količina podataka potrebnih za mapiranje. Na žalost korišćenje blok-baziranog mapiranja u log-baziranom FTL-u nije baš najbolje po pitanju performansi. Najveći problemi se javljaju kod čestih i malih upisa (upis je manji od veličine bloka). U tom slučaju FTL mora da pročita veliku količinu podataka iz bloka i upiše je u drugi blok zajedno sa tim malim upisom. To kopiranje podataka znatno povećava amplifikaciju pisanja i smanjuje performanse.

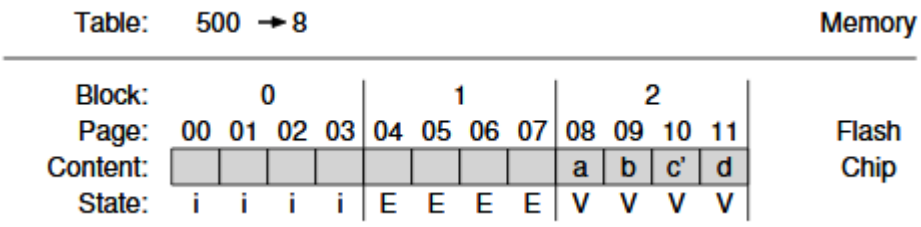
Predstavimo to na sledećem primeru 2000→4, 2001→5, 2002→6, 2003→7 :



Slika 18: Primer blok-baziranog adresiranja

Primetimo da su logičke adrese svih stranica prevedene u istu fizičku adresu početka bloka, a možemo ih prepoznati po ostatku koji daju pri deljenu sa adresom bloka. (adresa logičkog bloka se dobija kada se adresa logičke stranice podeli sa brojem stranica po bloku, u našem primeru u bloku ima 4 stranice, što znači da logička stranica 2002 pripada logičkom bloku 500).

Lako možemo izračunati adresu stranice koju treba pročitati i izvršiti operaciju čitanja u ovom modelu, ali šta se dešava kada treba npr. u logičku stranicu 2002 upisati vrednost c'?



Slika 19: Promena vrednosti stranice u blok-baziranom modelu

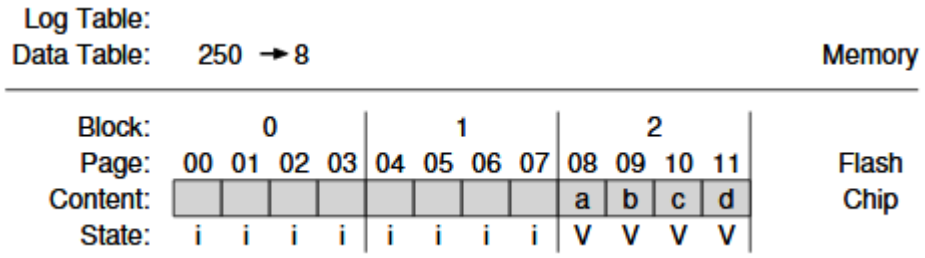
Kao što vidimo potrebno je pročitati ceo blok 1, obrisati sledeći slobodan blok i upisati u njega sve pročitane informacije sa izvšenom izmenom. Nakon toga se može i obrisati blok 1.

Možemo primetiti da ovaj pristup znatno smanjuje potrebe za radnom memorijom, ali znatno smanjuje performanse kada su upisi manji od veličine bloka, a sobzirom da je veličina bloka 256KB ili više, možemo očekivati da će se to prilično često dešavati. Dakle potrebno nam je bolje rešenje.

Hibridno mapiranje

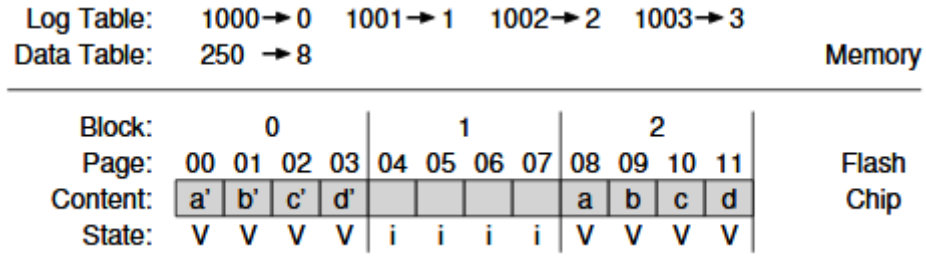
Da bi uključili fleksibilno pisanje ali i uštedeli na troškovima mapiranja, mnogi današnji FTL-ovi koriste tehniku hibridnog mapiranja. Ovim putem FTL zadržava nekoliko blokova obrisanih i upućuje sve upise njima. Oni se nazivaju log blokovi. Zbog toga što FTL želi da bude u mogućnosti da upisuje bilo koju stranicu na bilo koju lokaciju u log bloku bez kopiranja celog bloka koji zahteva čist blok-bazirani model mapiranja, ovaj pristup koristi mapiranje po strani za log blokove.
FTL tako logički ima dve tipa tabela za mapiranje u memoriji: malu količinu mapiranja po stranicama koju nazivamo log tabelom i veću količinu blok-baziranog mapiranja koju nazivamo tabela sa podacima. Kada tražimo neku logičku stranicu FTL prvo konsultuje log tabelu, a ako je tu ne nađe FTL pretražuje tabelu sa podacima. Ključna stvar hibridnog mapiranja je održavanje male log tabele. Da bi to postigao FTL periodično ispituje log tabelu tražeći grupu onih stranica koje mogu biti zapisane u tabeli sa podacima kao blok. Ta promena može biti ostvarena jednom od tri osnovne tehnike bazirane na sadžaju bloka.

Uzmimo kao primer flash memoriju u sledećem stanju:

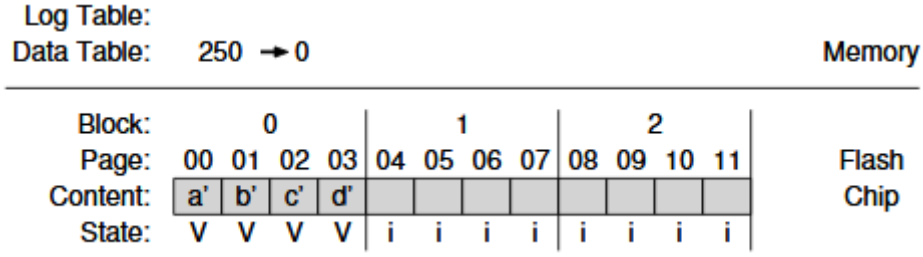


Slika 20: Početno stanje hibridno mapiranog flash-a

- Switch merge



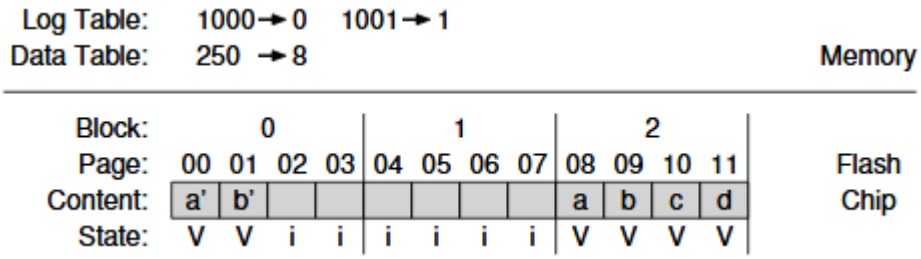
Slika 21: Ponovno dodavanje logičkih stranica 1000-1003 respektivno



Slika 22: Switch merge

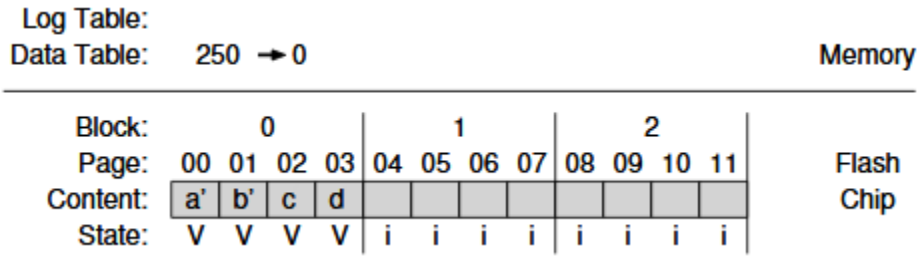
- Partial merge

Switch merge je najbolji scenario sa kojim FTL može da se susretne prilikom čišćenja, na žalost često se ova tehnika ne može primeniti. Primer je sledeća situacija:



Slika 23: Partial merge

U ovom slučaju potrebno je obaviti više posla. Prvo treba procitati blok 2 i aktuelne stranice c i d prepisati na pozicije 02 i 03. Rezultat je sličan, ali ovaj metod povećava amplifikaciju pisanja.



Slika 24: Partial merge

- Full merge

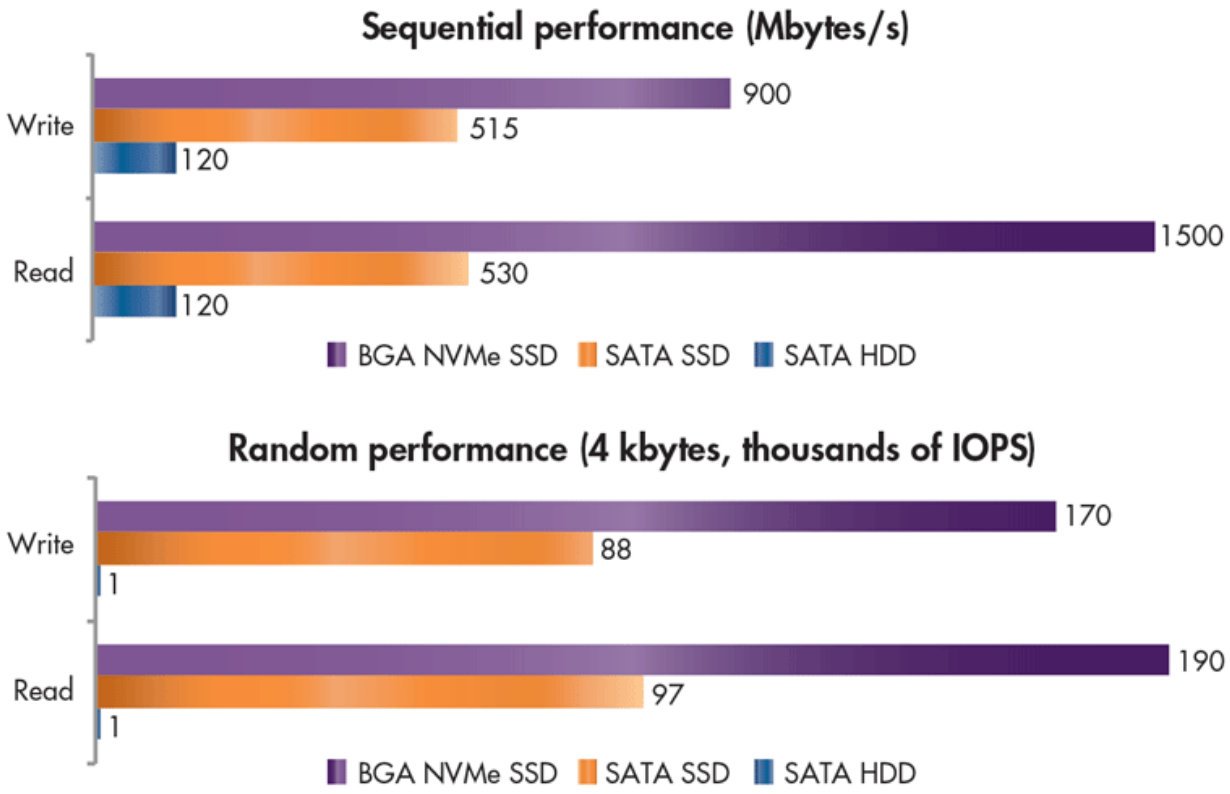
Poslednji i najmanje poželjan slučaj je situacija u kojoj imamo u log bloku npr. stranice 0, 4, 8, 12. Da bi ovakav log blok pretvorili u blok sa podacima FTL prvo mora da pronađe gde su zapisane stranice 1, 2 i 3 potom da ih objedini sa stranicom 0 i tek onda sačuva kao blok sa podacima. Isti postupak je potrebno izvršiti i za stranice 4, 8 i 12, a tek onda je moguće osloboditi početni log blok za nove upise. Česti slučajevi ovakvog čišenja (što se može očekivati) mogu ugroziti performanse.

Wear leveling

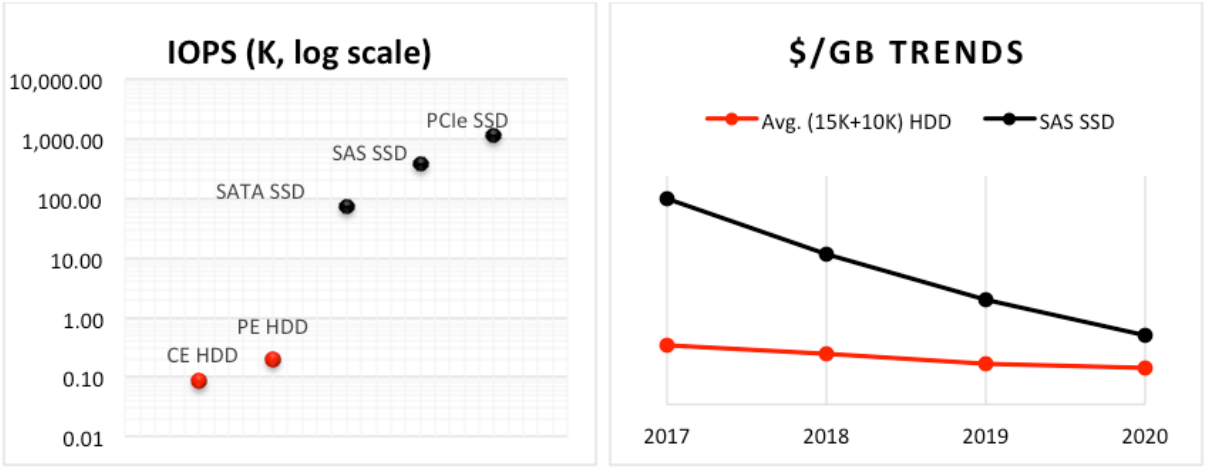
Konačno, vezano za pozadinske aktivnosti svaki moderan SSD mora obezbediti wear leveling, koji predstavlja tehniku za ravnomerno trošenje ćelija na celom uređaju. Osnovna ideja je jednostavna, treba upisivanje ravnomerno rasporediti po čitavom disku, da populane lokacije ne bi prebrzo otkazale. Osnovi log-bazirani pristup sa sakupljanjem otpada sam po sebi ovaj problem jako dobro tretira, sa izuzetkom koji se odnosi na validne stranice koje se ne dugo vremena ne menjaju, što uzrokuje da se one sporije troše od ostatka uređaja. Da bi se postigla revnoteđa FTL povremeno pronalazi ovakve stranice i prebacuje ih na druge lokacije, a njihov prostor oslobađa za nove upise. Postoji veliki broj algoritama koji se bave ovim problemom.

Performanse i cena

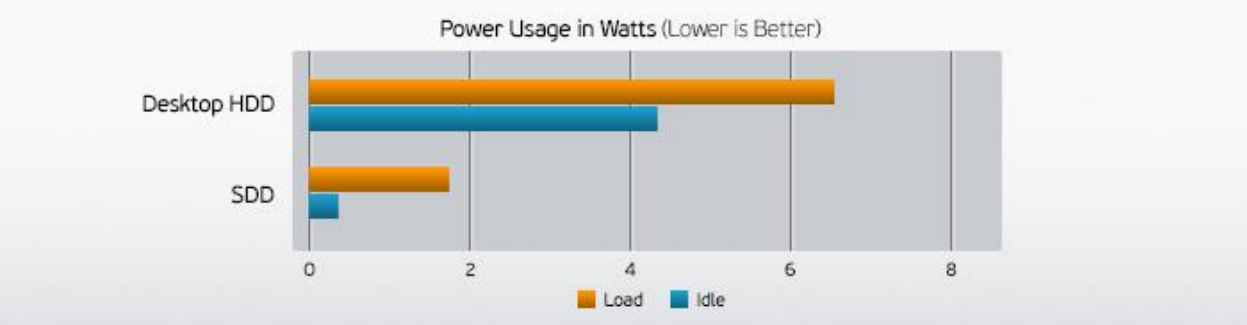
Primenom svih pomenutih tehnika ostaje nam da sagledamo rezultat i da ga uporedimo prema performansama, potrošnji i ceni u odnosu na hard disk.



Slika 25: Poređenje performansi



Slika 26: Prognoza kretanja cena



Slika 27: Poređenje potrošnje

Zaključak

Zbog svih navedenih prednosti i tehnika kojima su relativno uspešno prevaziđeni mnogi nedostaci SSD-ovi se već sada nameću kao pravi naslednici decenijama aktuelnih hard diskova. Jedino što ih još uvek sprečava da steknu potpunu dominaciju je i dalje veća cena.

Fajl sistemi su se vremenom prilagodili, ali se i dalje prilagođavaju SSD-ovima, tako što ih ne tretiraju na isti način kao HDD-ove zbog njihovih specifičnosti (npr. uvođenje TRIM komande koja obaveštava FTL koje stranice više neće biti u upotrebi).

Ova tehnologija je tek skoro dostigla upotrebnu vrednost i možemo očekivati da ima još puno prostora za napredak.