OPERATIVNI SISTEMI

VII - Virtuelna memorija

VII - Virtuelna memorija

S A D R Ž A J

Uvod

- virtuelna memorija je tehnika koja dozvoljava izvršavanje procesa čiji delovi mogu biti smešteni na diskovima
- virtuelna memorija
  - formira apstrakciju
    - logička memorija = operativna memorija + sekundarna memorija
  - omogućava izvršenje programa većih od same fizičke memorije
  - dozvoljava smeštaj znatno većeg broja procesa, odnosno delova procesa, u memoriju

7.1 - Učitavanje stranica prema potrebi

- virtuelna memorija se najčešće realizuje tehnikom **učitavanja stranica prema potrebi**
- DP (*demand paging*) sistem podseća na straničenje sa razmenjivanjem
- DP funkcioniše na sledeći način
  - memorija i prostor na disku koji se koristi za razmenjivanje su izdeljeni na stranice
  - u fizičku memoriju se ne prebacuje ceo proces
  - prebacuju se samo stranice koje se trenutno traže
    - najčešće se prebacuje samo ona koja je neophodna
    - DP može prebaciti i više stranica u memoriju na osnovu pretpostavke o stranicama koje će biti potrebne procesu
- ovaj sistem ima prednost u odnosu na *swap* isključivo celih procesa
  - izbegava se nepotrebno čitanje sa diska
  - smanjuje se potrebna količina fizičke memorije

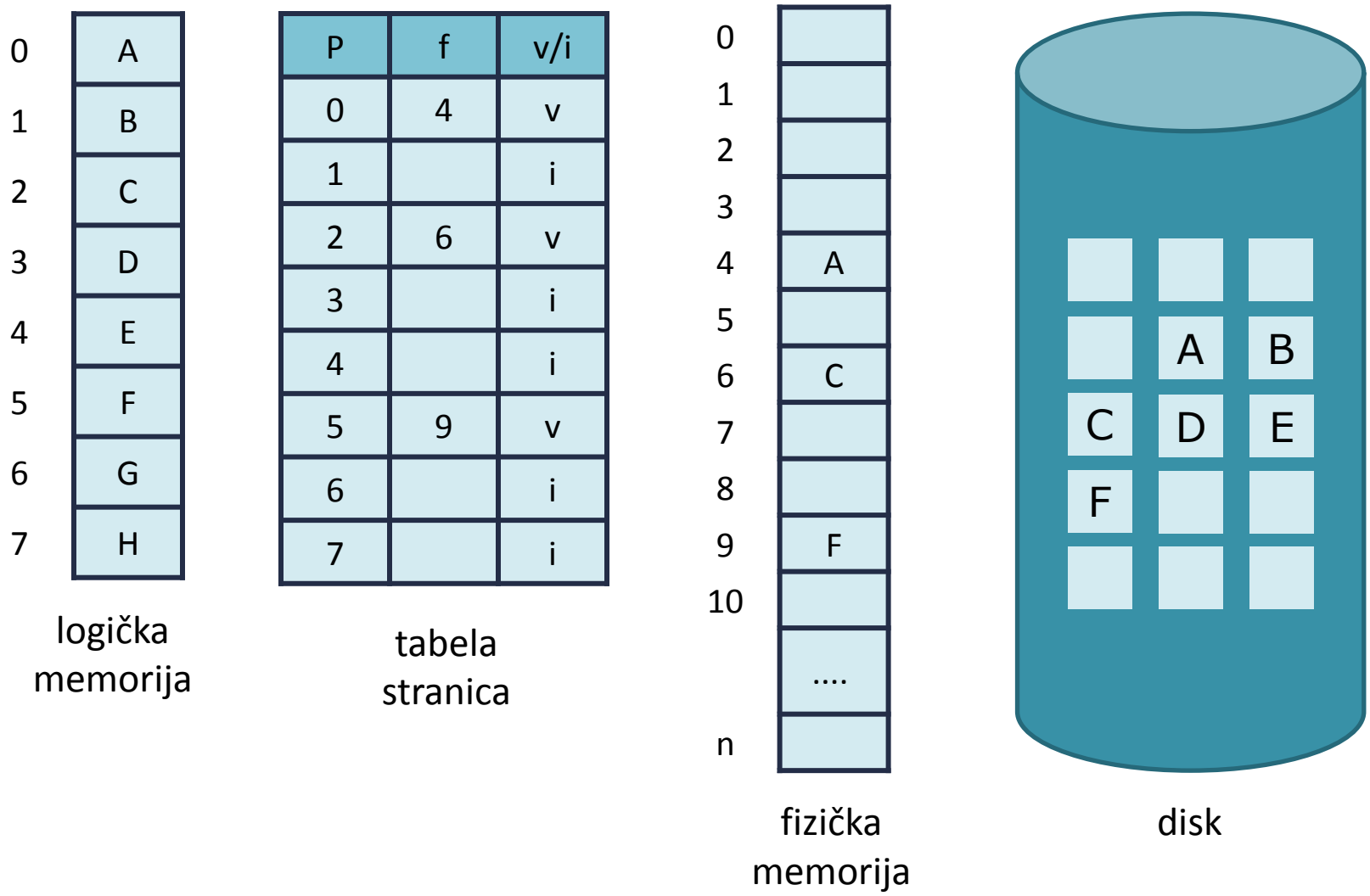
7.1 - Učitavanje stranica prema potrebi

Neophodan hardver

- unija hardvera potrebnog za straničenje i razmenjivanje
- sledeće dve komponente su apsolutno neophodne
- **tabela stranica**
  - eksplicitno se zahteva prisustvo **bita validnosti** u tabeli stranica
  - pomoću bita validnosti može se opisati **trenutni položaj** stranice
    - vrednost bita **v (*valid*)** ukazuje da se logička stranica nalazi u memoriji
  - vrednost bita **i (*invalid*)** ukazuje da se:
    - stranica ne nalazi u memoriji već u *swap* prostoru na disku
    - takođe može označavati da je to stranica koja ne pripada adresnom prostoru diska
- **sekundarna memorija**
  - služi za smeštaj svih stranica koje nisu prisutne u memoriji
  - uređaj koji se, po pravilu, koristi je disk (hard disk ili SSD)

7.1 - Učitavanje stranica prema potrebi

Sistem DP (*demand paging*)



7.1 - Učitavanje stranica prema potrebi

Prebacivanje stranica sa diska u fizičku memoriju

- posmatrajmo kako teče izvršavanje programa u sledeće dve situacije
  - proces se izvršava ili pristupa stranici koja je **u memoriji**
    - izvršavanje programa teče normalno
  - proces pokušava da pristupa stranici koja **nije u memoriji** (nego na disku)
    - proces pristupa logičkoj stranici čija je vrednost bita validnosti "i"
    - to izaziva prekidni signal **PF** (*page-fault trap)*
    - operativni sistem tada poziva **rutinu za opsluživanje PF**
    - rutina za opsluživanje PF prebacuje stranicu sa diska u memoriju

7.1 - Učitavanje stranica prema potrebi

Prebacivanje stranica sa diska u fizičku memoriju

- prebacivanje stranice sa diska u fizičku memoriju se obavlja u više koraka

1. referenca (load M) je prouzrokovala prekid PF
   - prilikom čitanja stranice u tabeli detektovan je *invalid bit*
2. OS poziva sistemsku rutinu za obradu PF
   - ukoliko referenca nije validna, proces se prekida jer sadrži pogrešnu instrukciju
   - ukoliko je referenca validna, PF započinje učitavanje stranice u memoriju
3. PF rutina pronalazi stranicu na disku u *swap* prostoru
4. PF rutina traži slobodan okvir u fizičkoj memoriji i prebacuje stranicu sa diska u okvir

7.1 - Učitavanje stranica prema potrebi

Prebacivanje stranica sa diska u fizičku memoriju

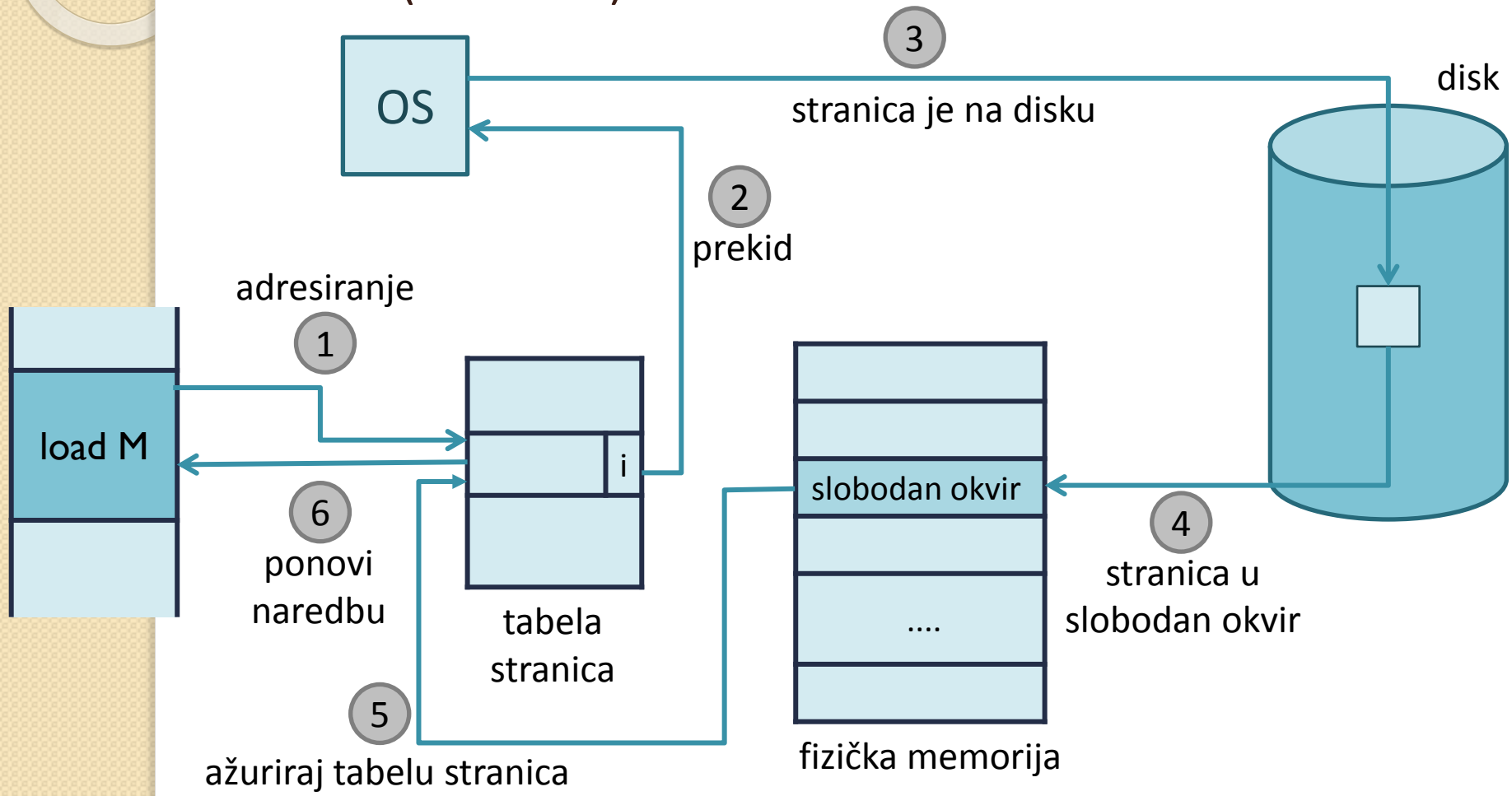
- prebacivanje stranice sa diska u fizičku memoriju se obavlja u više koraka (*nastavak*)

5. PF rutina ažurira tabelu stranica
   - na ulazu koji je napravio PF prekid, upisuje se adresa okvira i poništava *invalid bit* (postavlja se **v** bit)
6. prekinuta instrukcija koja je uzrokovala PF prekid se izvršava iz početka, s tim što sada ima sve što joj treba u memoriji
7. postupak se ponavlja za svaku stranicu procesa koja nije u memoriji, pri čemu svaki PF prekid učitava samo jednu stranicu sa diska

7.1 - Učitavanje stranica prema potrebi

Prebacivanje stranica sa diska u fizičku memoriju

- prebacivanje stranice sa diska u fizičku memoriju se obavlja u više koraka (*nastavak*)



7.1 - Učitavanje stranica prema potrebi

Performanse DP tehnike

- performanse DP tehnike zavise od
  - verovatnoće da se dogodi PF greška ($p \in [0,1]$)
  - trajanja memorijskog ciklusa *(t_{MA})*
  - trajanja obrade PF prekida *(t_{PF})*
- efektivno vreme pristupa memoriji ukoliko se koristi DP tehnika je
  - $t_{EA} = (1 - p)\, t_{MA} + p\, t_{PF}$
- od čega zavisi **trajanje obrade PF prekida** (t_{PF})?
- uopšteno, komponente obrade PF prekida su
  - servisiranje PF prekida
  - čitanje stranice
  - ponovno izvršenje procesa koji je izazvao PF

7.2 - Alternativne tehnike učitavanja stranica

Kopiranje na ciklus upisa *(CoW, Copy-on-Write)*

- sistemski poziv *fork* duplira adresni prostor procesa roditelja
- posle toga, potrebno je formirati i kopirati sve stranice koje pripadaju roditelju
- kako se može izbeći DP tehnika u ovom slučaju?
  - stranice proces roditelja se inicijalno ne kopiraju
  - nova memorija se inicijalno ne dodeljuje procesu detetu
  - procesi roditelj i dete inicijalno dele sve stranice, koje se označavaju kao **CoW**
  - ako bilo koji proces pokuša da modifikuje stranicu, najpre se kreira kopija stranice koja se dodeljuje tom procesu
  - nakon toga proces može da modifikuje svoju kopiju

7.2 - Alternativne tehnike učitavanja stranica

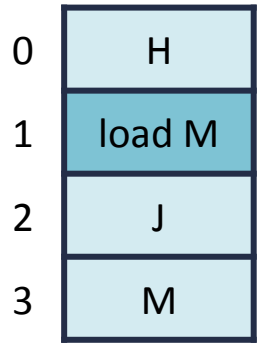
Memorijski mapirane datoteke

- koncept virtuelne memorije dozvoljava korisnicima da datotekama pristupaju preko **memorijskih referenci**
- to se postiže pomoću **memorijski mapiranih datoteka**
  - deo virtuelnog memorijskog prostora dodeli se datotekama
  - slika stranice na disku predstavlja deo datoteke a ne stranicu u *swap* prostoru
  - inicijalni pristup datoteci se odvija preko DP tehnike koja će izazvati PF prekid i kao rezultat te greške pročitaće se deo datoteke u memoriju
  - sledeći delovi datoteke dobijaju se na isti način
- više procesa može na ovaj način deliti datoteke u memoriji
  - datoteka se učita u memoriju pomoću DP tehnike
  - svaki proces podesi svoj adresni prostor tako da ukazuje na datoteku
  - upis koji uradi bilo koji proces, automatski je vidljiv svim ostalim procesima

7.3 - Zamena stranica

- proces može da se izvršava i ako se samo deo procesa nalazi u memoriji
- posledica toga je
  - povećanje broja aktivnih procesa u memoriji
  - povećanje stepena multiprogramiranja
- pojave koje prate sisteme sa virtuelnom memorijom su
  - iznenadna pojavljivanja velikog broja PF prekidnih signala
  - potpuno zauzeće cele fizičke memorije
- **problem zauzeća memorije** se može rešiti kao što je prikazano na sledećem primeru
  - imamo dva procesa sa po 4 logičke stranice i memoriju sa 8 okvira koja je trenutno popunjena
  - proces p1 izvršava instrukciju *load M* koja ima referencu za stranicu 3
  - stranica 3 se ne nalazi u fizičkoj memoriji, nego na disku, tako da nastupa PF prekidni signal
  - kako nema slobodnih okvira, jedan mora da se žrtvuje i da se oslobodi za stranicu M

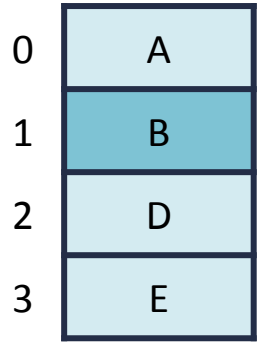
7.3 - Zamena stranica



logički adresni prostor (korisnik 1)

P	f	v/i
0	3	v
1	4	v
2	5	v
3		i

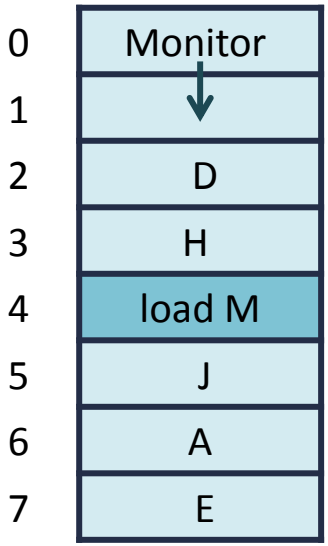
tabela stranica (korisnik 1)



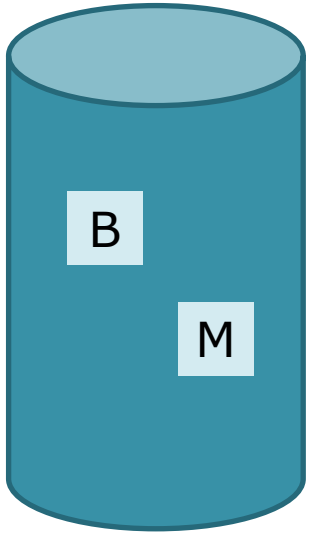
logički adresni prostor (korisnik 2)

P	f	v/i
0	6	v
1		i
2	2	v
3	7	v

tabela stranica (korisnik 2)



fizički adresni prostor



disk

7.3 - Zamena stranica

- zamena stranica funkcioniše na sledeći način

1. najpre se bira okvir u koji će biti smeštena željena stranica
   - ako ima slobodnih okvira, uzima se jedan od njih
   - ako nema slobodnog, pomoću algoritama za zamenjivanje stranica bira se **žrtva**
   - žrtva = okvir u kome se nalazi stranica koja će biti zamenjena
2. žrtvovana stranica se upisuje na disk
   - upis se ne radi ako sadržaj okvira nije modifikovan u odnosu na postojeću sliku na disku
     - ovaj korak zahteva da se u tabelu stranica uvede **bit čistoće** (*dirty flag)*
     - znatno povećava performanse
3. žrtvovani okvir se u tabeli stranica obeležava kao **nevalidan**
4. željena stranica se učitava u oslobođeni okvir
5. tabela stranica se ažurira i nova stranica se obeležava kao **validna** bitom validnosti

7.3 - Zamena stranica

- u slučaju pojave PF prekidnog signala smanjuju se performanse procesa
- broj PF prekida mora se održati na što manjem nivou
- potrebno je naći algoritam koji će za datu sekvencu referenci da postigne **najmanju verovatnoću pojave PF grešaka**
- algoritmi za izbor žrtve
  - FIFO
  - optimalni
  - LRU
  - druga šansa
  - NRU
  - frekvencijski algoritmi (LFU, MFU)

7.3 - Zamena stranica

Algoritam FIFO

- FIFO (*First In First Out*) - prvi ušao, prvi izlazi
- formira se FIFO red na čiji se kraj stavljaju stranice prilikom učitavanja u memoriju
- prilikom zamene, memoriju napušta stranica koja se nalazi na početku reda
- primer

Niz referenci

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Okviri

0	7	7	7	2	2	2	4	4	4	0	0	0					7	7	7
1		0	0	0	3	3	3	2	2	2	1	1					1	0	0
2			1	1	1	0	0	0	3	3	3	2					2	2	1

7.3 - Zamena stranica

Algoritam FIFO

- objašnjenje primera
  - prve tri reference (7, 0 i 1) napraviće tri uzastopne PF greške i popuniti sva tri okvira
  - četvrta referenca (2) usloviće PF grešku koja mora da izazove zamenu stranice
  - kao žrtva, bira se okvir 0 u kome se nalazi najstarija stranica (stranica 7)
  - peta referenca (0) ne izaziva PF jer se stranica 0 nalazi u okviru 1 u memoriji
  - šesta referenca (3) izaziva novi PF i novu zamenu
  - kao žrtva bira se okvir 1 koji sadrži stranicu 0 koja je najstarija
  - ....
  - data sekvencu je imala 15 PF prekida

Niz referenci

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Okviri

0	7	7	7	2	2	2	4	4	4	0	0	0					7	7	7
1		0	0	0	3	3	3	2	2	2	1	1					1	0	0
2			1	1	1	0	0	0	3	3	3	2					2	2	1

7.3 - Zamena stranica

Algoritam FIFO

- iako je algoritam FIFO jednostavan za upotrebu on nije i optimalan
  - kao kriterijum algoritam koristi samo vreme punjenja okvira stranicom, ali ne i broj referenci upućen ka pojedinim stranicama
  - može se desiti da se stranice koje se najčešće koriste izbacuju umesto onih koje se retko koriste
- drugi nedostatak ovog algoritma se održava kroz **Belady**-jevu anomaliju
  - isti niz referenci je primenom FIFO algoritma bio raspoređivan u više ciklusa, gde je pri svakom novom ciklusu bio povećan broj raspoloživih memorijskih okvira
  - umesto očekivanog smanjenja broja PF grešaka sa povećanjem broja okvira, desilo se da je broj PF grešaka bio veći u slučaju memorije sa 4 okvira nego u memoriji sa 3 okvira

7.3 - Zamena stranica

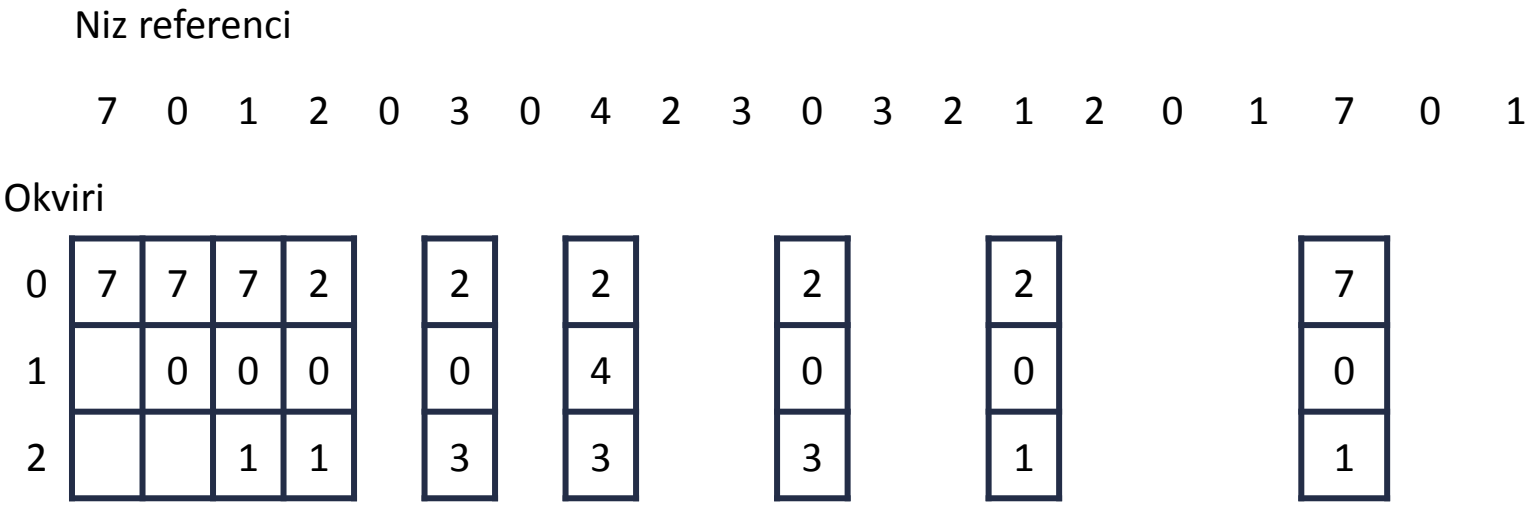
Optimalni algoritam (OPT, MIN)

- FIFO svoj kriterijum formira na istoriji referenci koja je uvek poznata
- OPT algoritam svoj kriterijum formira **na budućnosti**
  - OPT mora unapred da poznaje sve procese, što **nije moguće**
  - ovo naročito važi ako se radi o interaktivnim sistemima
- OPT funkcioniše na sledeći način
  - OPT žrtvuje stranicu koja se neće koristiti najduže vremena
  - time se maksimalno odlaže pojavljivanje sledeće PF greške
  - primer
    - stranica A će biti potrebna posle 100 instrukcija
    - stranica B će biti potrebna posle 1000 instrukcija
    - bolje je žrtvovati stranicu B jer se izazivanje sledeće PF greške duže odlaže
- OPT proizvodi minimum PF grešaka ali **ne može da se implementira u praksi**

7.3 - Zamena stranica

Optimalni algoritam (OPT, MIN)

- primer OPT algoritma nad istim nizom referenci koji je prikazan u primeru FIFO algoritma
  - prva razlika u odnosu na FIFO se pojavljuje kod prve reference 3
  - tada se ne izbacuje 0, koja je najstarija u odnosu na 2 i 1, zbog toga što će se u narednim koracima ona više od njih koristiti
  - umesto 0 se izbacuje 1 (iz memorijskog okvira 2)
  - za razliku od 15 PF registrovanih u FIFO algoritmu ovde ih ima 9



7.3 - Zamena stranica

Algoritam LRU

- LRU algoritam (*least recently used)* žrtvuje stranicu koja najduže nije korišćena
- LRU svakoj stranici dodeljuje vreme poslednjeg korišćenja
- vreme poslednjeg korišćenja se ažurira sa sa svakim pristupom stranici
- primer

Niz referenci

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Okviri

0	7	7	7	2		2		4	4	4	0		1		1		1		
1		0	0	0		0		0	0	3	3		3		0		0		
2			1	1		3		3	2	2	2		2		2		7		

- za datu sekvencu LRU generiše 12 PF prekida, dok za istu FIFO generiše 15 a OPT 9

7.3 - Zamena stranica

Algoritam LRU

- algoritam se može realizovati na sledeća dva načina
- **pomoću brojača**
  - procesor ima brojač memorijskih instrukcija
  - posle svake memorijske instrukcije vrednost brojača se povećava za jedan
  - svaki okvir ima svoj interni brojač
  - svaki put kada se pristupi okviru sadržaj brojača instrukcija se kopira u interni brojač okvira
  - kada dođe do PF prekida, algoritam bira **stranicu čiji je broj najmanji,** što znači da ona najduže nije korišćena
- **pomoću steka**
  - formira se stek koji opisuje redosled pristupanja stranicama
  - nakon PF prekida, žrtvuje se stranica sa vrha steka
  - na njeno mesto se stavlja stranica kojoj se pristupa
  - ova realizacija je veoma spora jer se pri svakom pristupu memoriji stek mora ažurirati

7.3 - Zamena stranica

Druga šansa (satni algoritam)

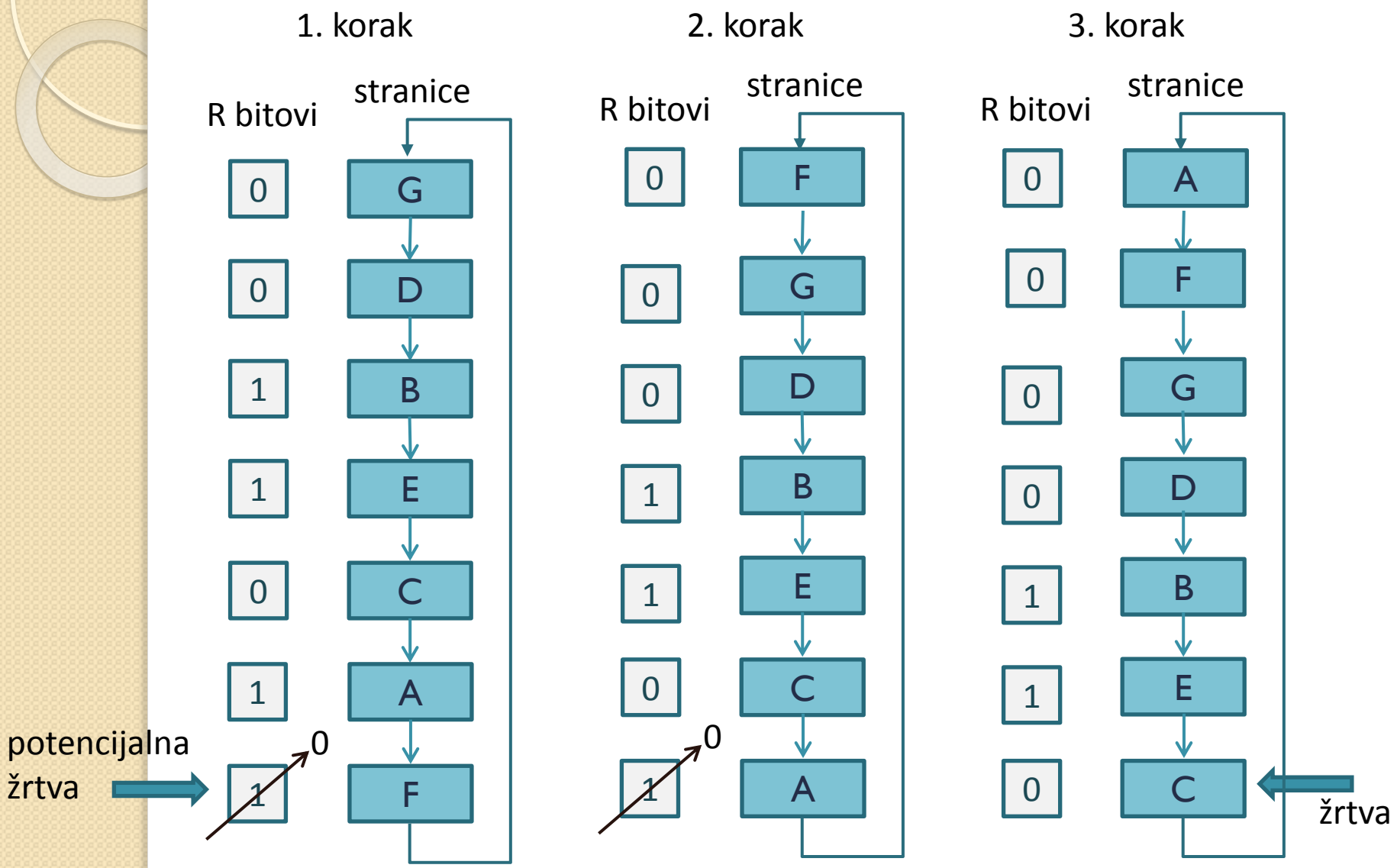
- druga šansa je varijanta FIFO algoritma koja u obzir uzima i korišćenost stranica
- formira se FIFO red na čiji se kraj stavljaju stranice prilikom učitavanja u memoriju
- koristi se vrednost **R** *(reference)* bitova
  - R bitovi se postavljaju inicijalno na 0
  - brisanje R bita nakon određenog vremena obavlja se pomoću tajmerskih prekida
  - R bit se postavlja na 1 ako smo od poslednjeg brisanja bar jednom pristupili stranici

7.3 - Zamena stranica

Druga šansa (satni algoritam)

- potencijalna žrtva je stranica na kraju reda čekanja
  - ako je R=0, stranica se dugo nalazi u memoriji, nismo joj pristupili
    - stranica očigledno nije potrebna, tako da se može žrtvovati
  - ako je R=1, stranica je dugo u memoriji ali je i nedavno korišćena
    - R bit te stranice biće resetovan posle provere
    - sama stranica biće prebačena na početak reda opsluživanja
    - pretraga se nastavlja, sledeća potencijalna žrtva je opeta stranica na kraju reda
    - ako je stranica korišćena (R=1) dobija drugu šansu, a ako nije, žrtvovaće se
    - pretraga se nastavlja dok se ne pronađe žrtva (R=0)

7.3 - Zamena stranica



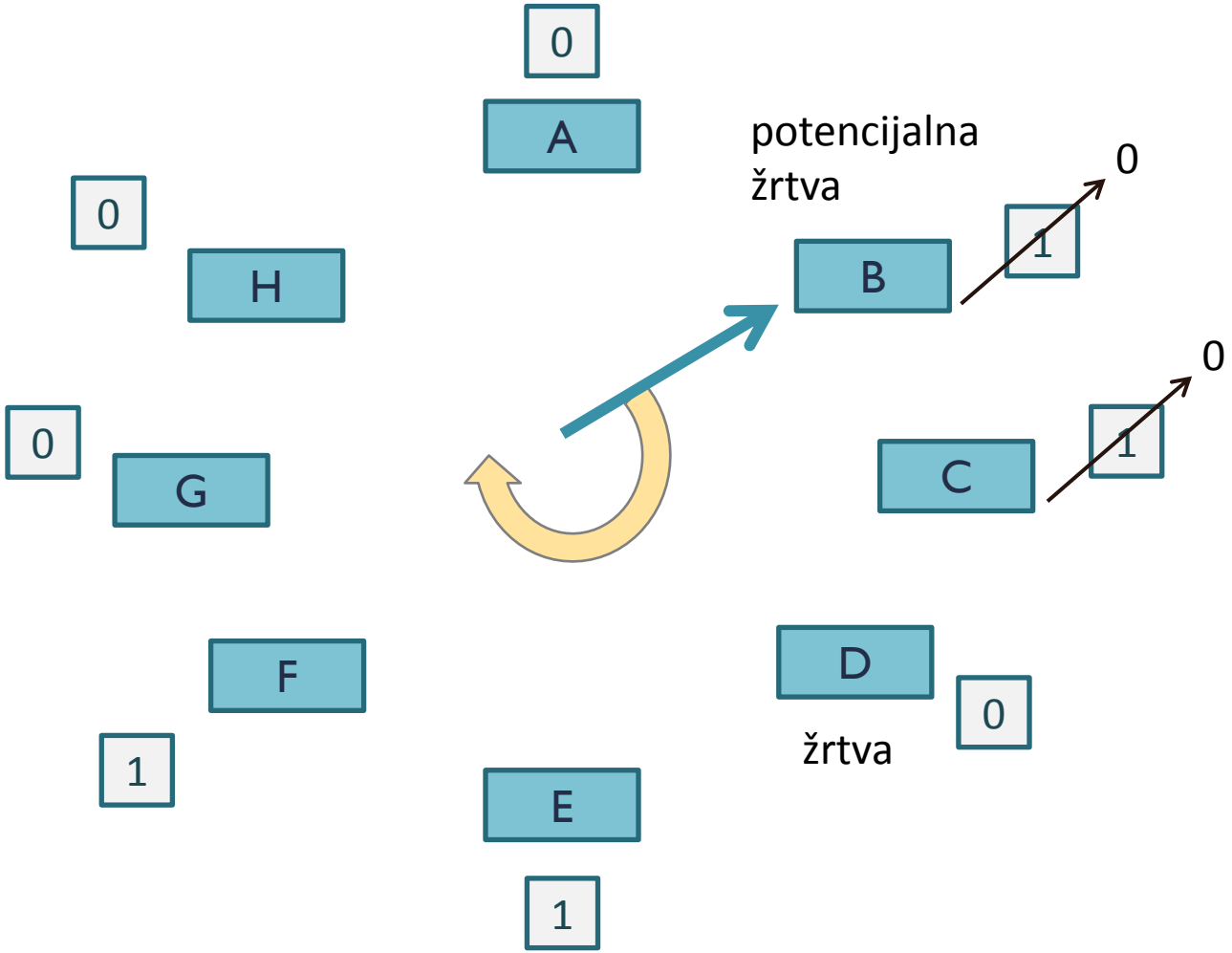
7.3 - Zamena stranica

Druga šansa (satni algoritam)

- jedna od mogućih realizacija ovog algoritma koristi kružni red čekanja (odatle i naziv **satni algoritam**)
- sve popunjene stranice se stavljaju u kružnu listu
- pokazivač je postavljen na stranicu koja je najranije učitana (FIFO princip), koja time postaje potencijalna žrtva
  - R=0
    - stranica se može izbaciti i na njeno mesto dolazi nova stranica
  - R=1
    - R bit te stranice biće postavljen na 0
    - pokazivač se pomera napred na sledeću stranicu u krugu
    - time je prethodna stranica postala poslednja u krugu i dobila drugu šansu
    - pretraga se nastavlja na isti način dok se ne pronađe stranica sa R=0, čime ona postaje žrtva i izbacuje se

7.3 - Zamena stranica

Druga šansa (satni algoritam)



7.3 - Zamena stranica

Algoritam NRU

- algoritam NRU (*not recently used*) zahteva da se svakom okviru pridruže dva bita
  - **R** (*reference bit*) - označava da li smo pristupili tom okviru
  - **M** (*modified*) - označava da li smo menjali sadržaj tog okvira
- na početku rada procesa svi R i M bitovi dobijaju vrednost 0
- na osnovu vrednosti R i M bitova okvire delimo u sledeće klase
  - **R=0, M=0** - stranica nije skoro ni korišćena ni modifikovana, idealna žrtva
  - **R=0, M=1** - stranica nije skoro korišćena, ali je modifikovana
    - nije pogodna za zamenu, jer mora da se upiše na disk, što će izazvati dva U/I ciklusa
  - **R=1, M=0** stranica je skoro korišćena, ali nije modifikovana
    - najverovatnije će se koristiti ponovo
  - **R=1, M=1** stranica je skoro korišćena i modifikovana
    - verovatno će se ponovo koristiti
    - nije pogodna za zamenu, jer mora da se upiše na disk, što će izazvati dva U/I ciklusa
- slično satnom algoritmu pretražuju se klasa (0,0), (0,1)...

7.3 - Zamena stranica

Algoritmi zasnovani na učestalosti korišćenja stranice

- prethodni algoritmi nisu uzimali u obzir koliko puta je stranica bila referencirana
- dva algoritma koji kriterijum formiraju na osnovu frekvencije korišćenja stranice su
- **LFU** (*least frequently used)*
  - LFU algoritam zamenjuje stranicu koja ima najmanji broj referenci u vremenu
  - svaka stranica koja se učita mora se određeno vreme štititi od samog algoritma
    - kada se pojavi njena frekvencija je jednaka 1, što je minimum
    - takva stranica bi bila potencijalna žrtva i ne bi dobila šansu da se takmiči sa drugim stranicama
  - algoritam je loš za neke stranice koje se intezivno koriste u kratkom vermenskom intervalu, nakom čega se uopšte ne koriste
    - takve stranice ostaju dugo u memoriji

7.3 - Zamena stranica

Algoritmi zasnovani na učestalosti korišćenja stranice

- **MFU** (*most frequently used)*
  - MFU algoritam zamenjuje stranicu koja ima najveći broj referenci
  - algoritam smatra da su te stranice odavno učitane u memoriju i da procesima više nisu potrebne, dok su stranice sa malim brojem referencu relativno nove i potrebne
- algoritmi zasnovani na učestalosti korišćenja stranice imaju svoje prednosti i mane ali se relativno teško realizuju

7.4 Raspodela okvira po procesima i efekat zasićenja

- u prethodnom tekstu je opisana osnovna strategija dodeljivanja fizičke memorije
  - najjednostavniji je DP, gde se stranica učitava u memoriju kada god se pojavi potreba, jer postoji slobodna memorija
  - ponekad međutim moramo koristiti i različite algoritme, u slučajevima kada nema slobodnih okvira, koji na razne načine omogućavaju da se odabere stranica koja će se izbaciti iz memorije
- postoje međutim i modifikacije ove osnovne strategije

7.4 Raspodela okvira po procesima i efekat zasićenja

Raspodela okvira po procesima

- kako rasporediti *m* okvira na *n* procesa?
  - minimalni broj okvira po procesu zavisi od konkretne procesorske arhitekture
  - najmanji je dva (instrukcija od više bajtova može da bude na granici između dve stranice)
  - maksimalni broj okvira koji proces može da dobije je ograničen veličinom fizičke memorije
- **metoda jednake raspodele**
  - svakom procesu se dodeljuje jednaka količina okvira m/n
- **metoda proporcionalne raspodele**
  - m - ukupan broj okvira fizičke memorije
  - S_i - veličina procesa p_i
  - n - broj procesa
  - S - ukupna virtuelna memorija: $S = \sum_{n}^{i=1} S_i$
  - svakom procesu pripašće sledeći broj okvira: $a_i = \frac{S_i}{\sum_{n}^{i=1} S_i} m$

7.4 Raspodela okvira po procesima i efekat zasićenja

Globalna i lokalna zamena stranica

- **globalna zamena**
  - jedan proces može žrtvovati tuđu stranicu (odnosno poslati je u *swap* prostor)
  - broj okvira dodeljenih procesu može se menjati u vremenu
  - na broj PF prekida utiču i drugi procesi
- **lokalna zamena**
  - jedan proces može žrtvovati isključivo svoje stranice
  - broj okvira dodeljenih procesu se ne menja u vremenu
  - u slučaju lokalne razmene proces sam diktira svoje PF prekide
    - drugi procesi nemaju uticaj na broj PF prekida koje taj proces izaziva
  - mana lokalne zamene: **blokiran proces** ne može ustupiti svoje okvire drugim procesima
    - to znači da aktivan proces mora žrtvovati sopstvene stranice
    - zbog toga globalna razmena daje bolje rezultate

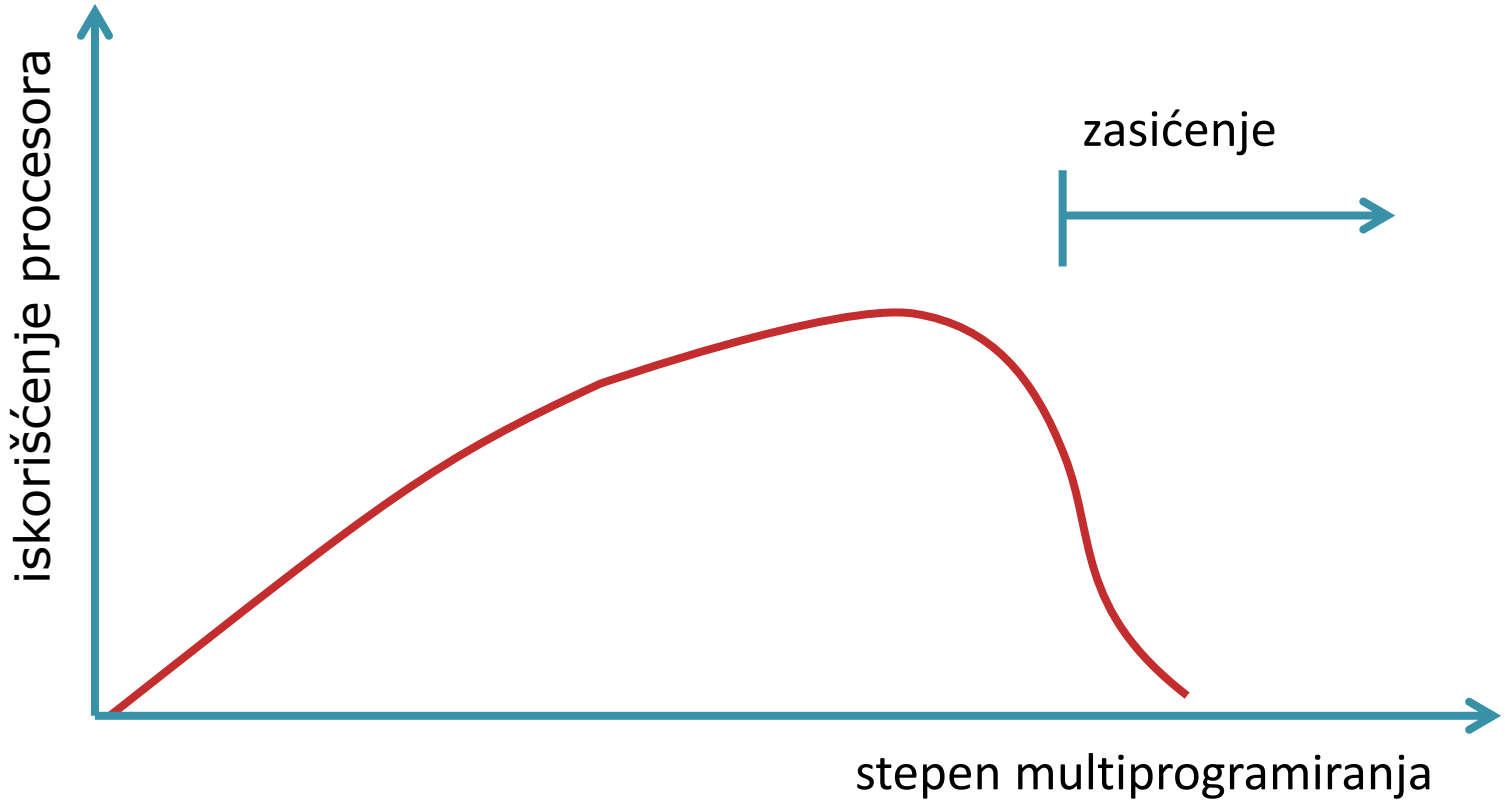
7.4 Raspodela okvira po procesima i efekat zasićenja

Efekat zasićenja

- proces zahteva operativnu memoriju kao resurs
- ukoliko **broj raspoloživih okvira za proces padne ispod minimalne vrednosti**
  - resurs (memorija) nije slobodan
  - proces ne može da se izvršava
  - proces se mora dovesti u stanje WAIT
- ukoliko je **broj okvira raspoloživih za proces veoma mali**
  - resurs (memorija) je slobodan
  - proces može da se izvršava
  - ali je pojava PF grešaka česta
  - proces veoma često zamenjuje svoje stranice
    - to prouzrokuje veliki broj U/I operacija sa diskom
- pojava čestog razmenjivanja stranica koja nastaje kao posledica visokog stepena multiprogramiranja naziva se **efekat zasićenja** (*thrashing)*
  - u takvim situacijama javljaju se ozbiljni problemi u vidu degradacije performansi

7.4 Raspodela okvira po procesima i efekat zasićenja

Efekat zasićenja



7.4 Raspodela okvira po procesima i efekat zasićenja

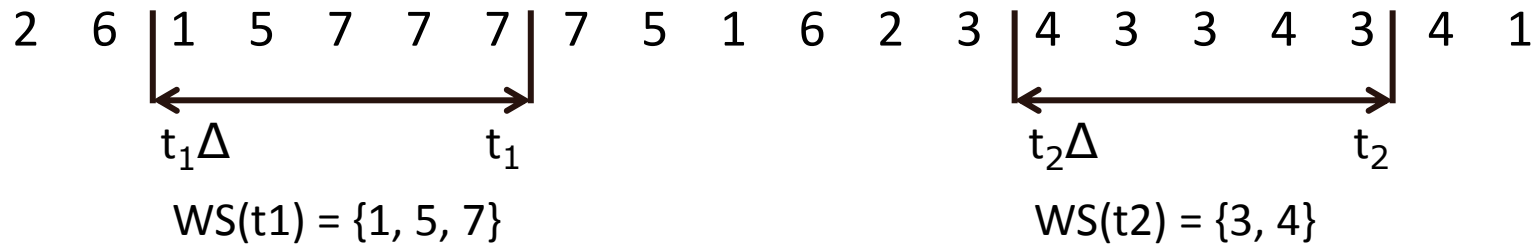
Efekat zasićenja

- uvešćemo model lokalnosti u svaki proces, koji će nam pomoći da objasnimo efekat zasićenja
- **lokalnost** je skup stranica koje proces koristi zajedno u jednom intervalu vremena
- proces može menjati svoje lokalnosti a one se mogu preklapati
- **primer**
  - svaki poziv potprograma napraviće novu lokalnost
  - kad se izađe iz tog potprograma lokalnost se menja
- zasićenje nastupa kada je **suma lokalnosti za sve aktivne procese** veća od fizičke memorije

7.4 Raspodela okvira po procesima i efekat zasićenja

Efekat zasićenja

- uvodimo novi model koga ćemo nazvati **radni model** (*working-set model)*
  - **prozor radnog skupa** Δ (*working-set window*) je vreme u kome se izvrši određeni broj instrukcija
  - sve stranice referencirane u tom periodu predstavljaju **radni skup** (*working-set)* za Δ
- primer: data su dva radna skupa sa po 5 memorijskih referenci
  - WS(t1) obuhvata stranice 1,5 i 7
  - WS(t2) obuhvata stranice 3 i 4



- potrebno je odrediti veličinu vremenskog prostora
  - ako je prozor isuviše mali, ne opisuje dovoljno lokalnost procesa
  - ako je veliki, tada se preklapa više lokalnosti

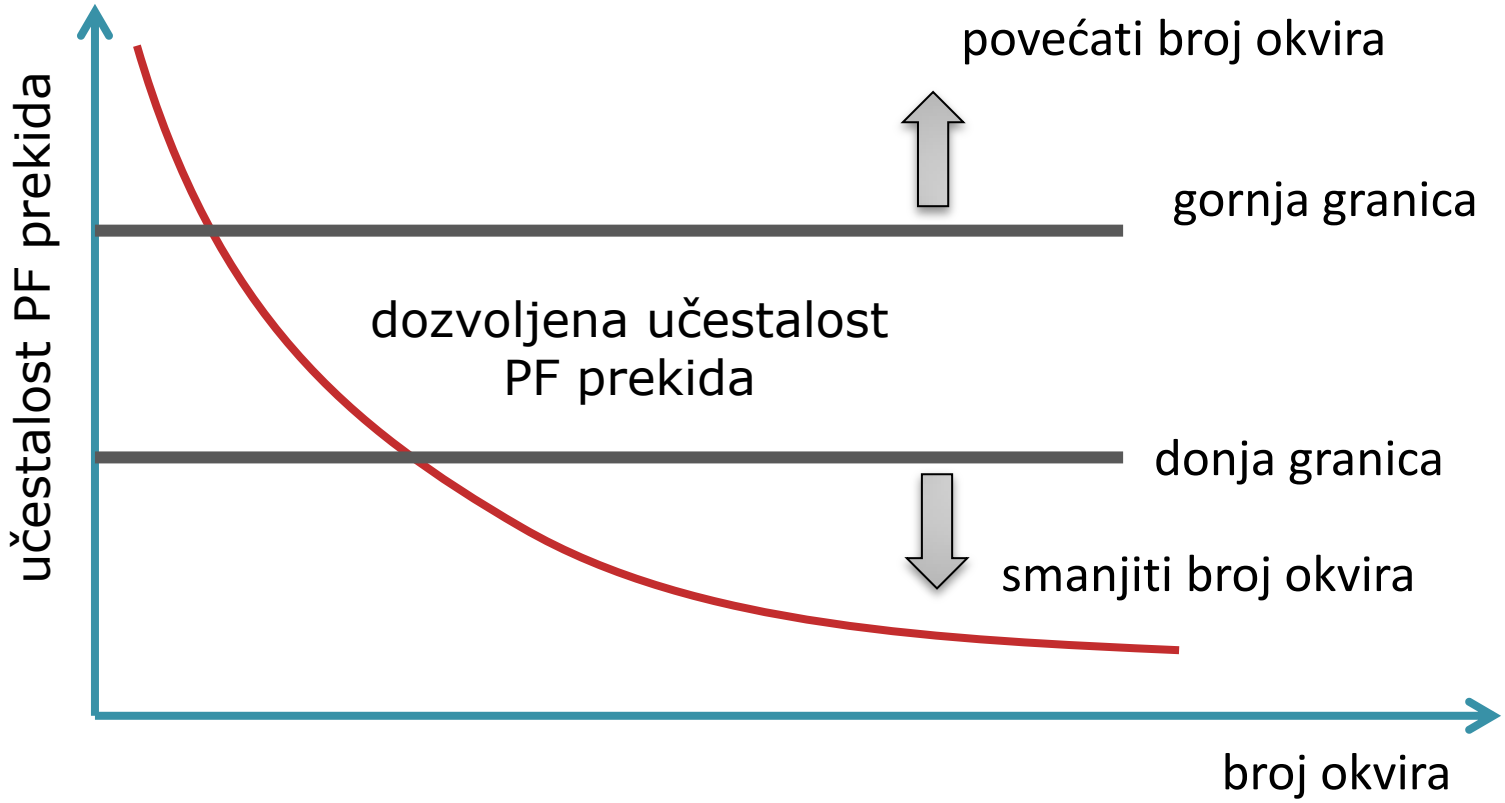
7.4 Raspodela okvira po procesima i efekat zasićenja

Efekat zasićenja

- u celoj ovoj priči najznačanija je veličina radnog skupa
- **veličina radnog skupa** WSS_i procesa P_i definiše se kao ukupan broj stranica koje proces traži u vremenskom prozoru Δ
- ukupna veličina radnog prostora u sistemu D je: $D = \sum_{i=1}^{n} WSS_i$
- ako je D > m (sistemska memorija), nastupiće efekat zasićenja
  - OS prati veličinu D
  - kada D dođe do veličine sistemske memorije
    - novi procesi se neće uvoditi
    - pojedini procesi moraju da se suspenduju na disk, kako bi se smanjila veličina radnog skupa
- efekat zasićenja se može sprečiti limitiranjem broja PF prekida koje jedan proces izaziva u okvirima dozvoljenog opsega
  - broj PF prekida je veliki u slučaju zasićenja

7.4 Raspodela okvira po procesima i efekat zasićenja

Efekat zasićenja



7.4 Raspodela okvira po procesima i efekat zasićenja

Efekat zasićenja

- **primena algoritma**
  - najpre se postavljanjem donje i gornje granice definiše **opseg dozvoljenog broja PF prekida** za jedan proces
  - zatim se prati broj, odnosno frekvencija izazivanja prekida
  - ako je broj PF grešaka procesa **manji od donje granice**
    - procesu je dodeljeno više okvira
    - poneki se može oduzeti i dodeliti drugim procesima
  - ako je broj PF grešaka **veći od gornje granice**
    - procesu nije dodeljen dovoljan broj okvira
    - proces može da dobije još okvira
  - ako se broj PF grešaka svakog procesa održava u dozvoljenom opsegu (između granica)
    - efekat zasićenja se neće dogoditi