

Virtuelna memorija

- Background
- Učitavanje stranica prema potrebi (*Demand Paging*)
- Kreiranje procesa (*Process Creation*)
- Zamena stranica (*Page Replacement*)
- Alokacija okvira po procesima (*Allocation of Frames*)
- Thrashing

Background

■ Osnovna zamisao:

- ☞ Omogućava da se izvrši proces koji je **samo delimično** u memoriji
- ☞ Deo procesa je u memoriji (onaj koji se trenutno koristi)
- ☞ Ostatak programa je na disku (swap ili exe datoteka)

■ Program > Fizičke Memorije

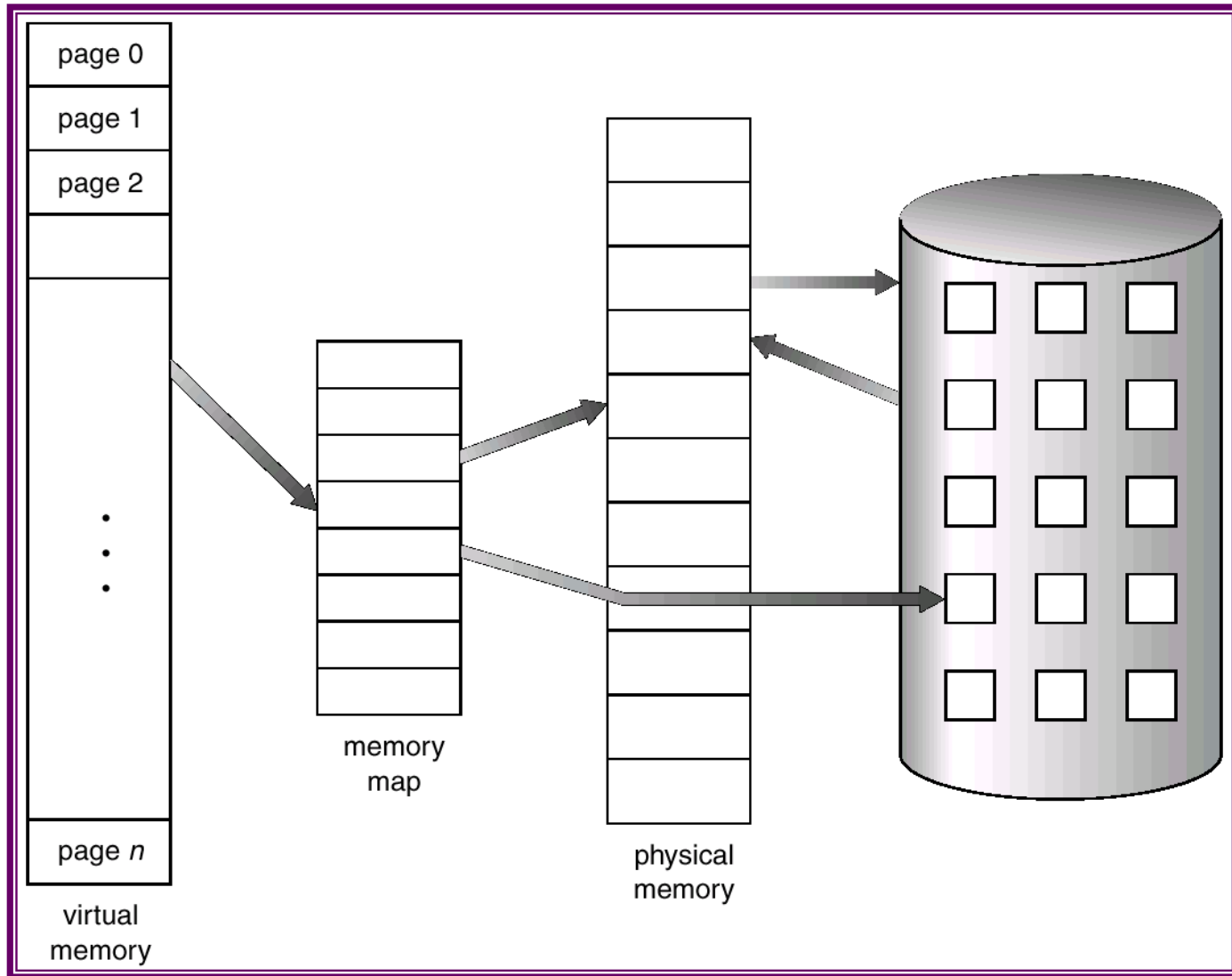
- ☞ VM nije jednostavna za implementaciju
- ☞ VM može smanjiti performanse

■ Virtuelna memorija:

■ odvajanje korisničke logičke memorije od fizičke memorije.

- ☞ Potrebno je da **samo deo** programa bude u memoriji da bi se izvršavao.
- ☞ Zbog toga **logički adresni prostor** može biti **mного veći** od **fizičkog adresnog prostora**.
- ☞ Omogućava da više procesa dele adresni prostor
- ☞ Omogućava mnogo efikasnije kreiranje procesa

Virtuelna memorija je veća od fizičke memorije



Realizacija virtuelne memorije

■ Virtuelna memorija se najčešće realizuje preko **sledećih tehnika:**

☞ Učitavanje stranica prema potrebi

☞ (***Demand paging***)

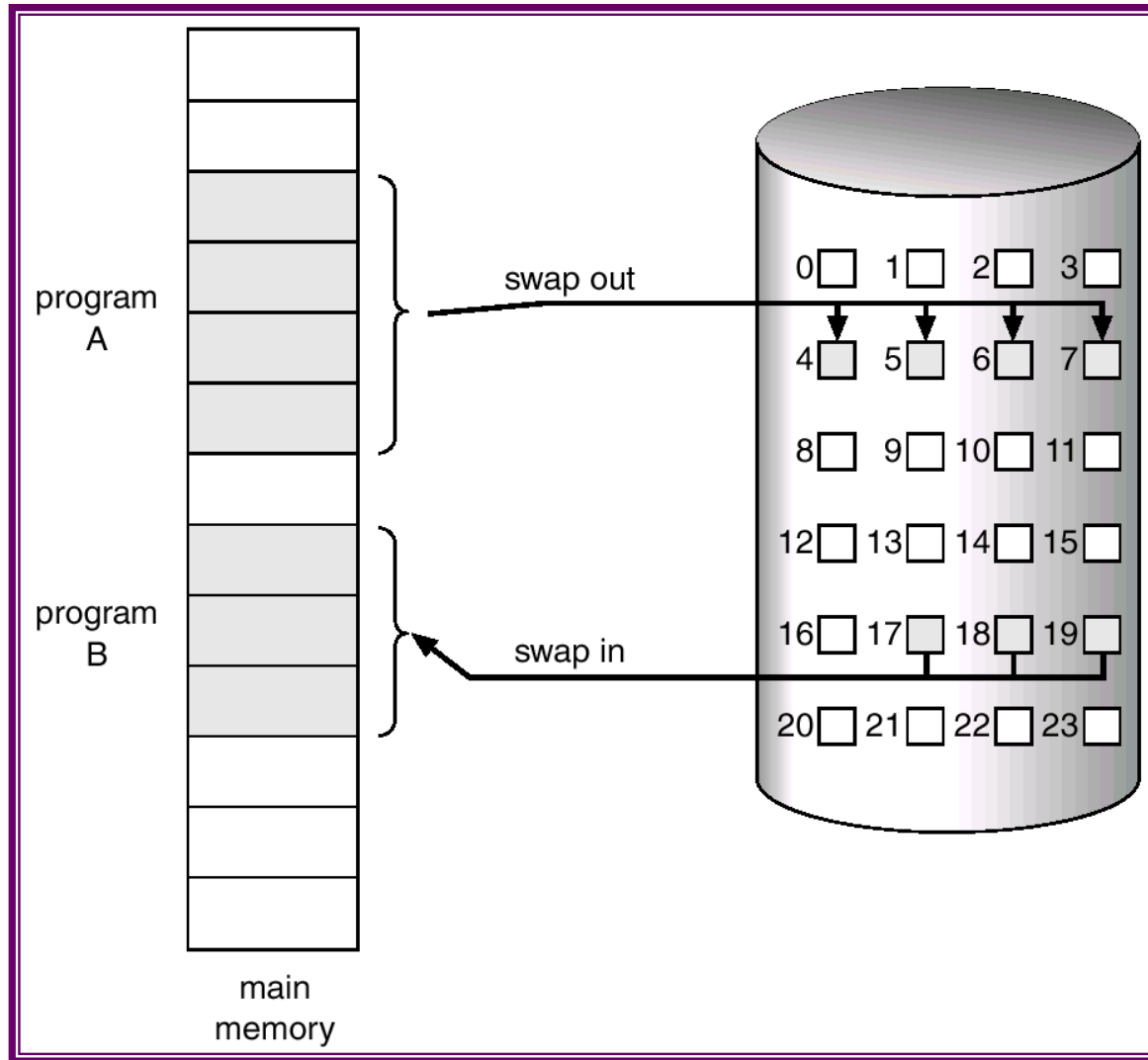
☞ Učitavanje segmenata prema potrebi

☞ (***Demand segmentation***)

Učitavanje stranica prema potrebi

- Učitava stranicu u memoriju samo kada je potrebna
- **lazy swapper** (lenji razmenjivač)
 - ☞ potrebno je manje I/O operacija
 - ☞ potrebno je manje memorije
 - ☞ brži odziv
 - ☞ više korisnika
- Postoje **dva velika** problema:
 - ☞ 1. Algoritam alokacije okvira
 - ☞ 2. Algoritam zamene stranica
- Stranica je potrebna \Rightarrow prebacuje se u memoriju
 - ☞ ako je nevažuća referenca \Rightarrow prekid
 - ☞ ako nije u memoriji \Rightarrow prebacuje se u memoriju \Rightarrow restart

Prenos stranica u kontinualni prostor na disku



Valid-Invalid Bit

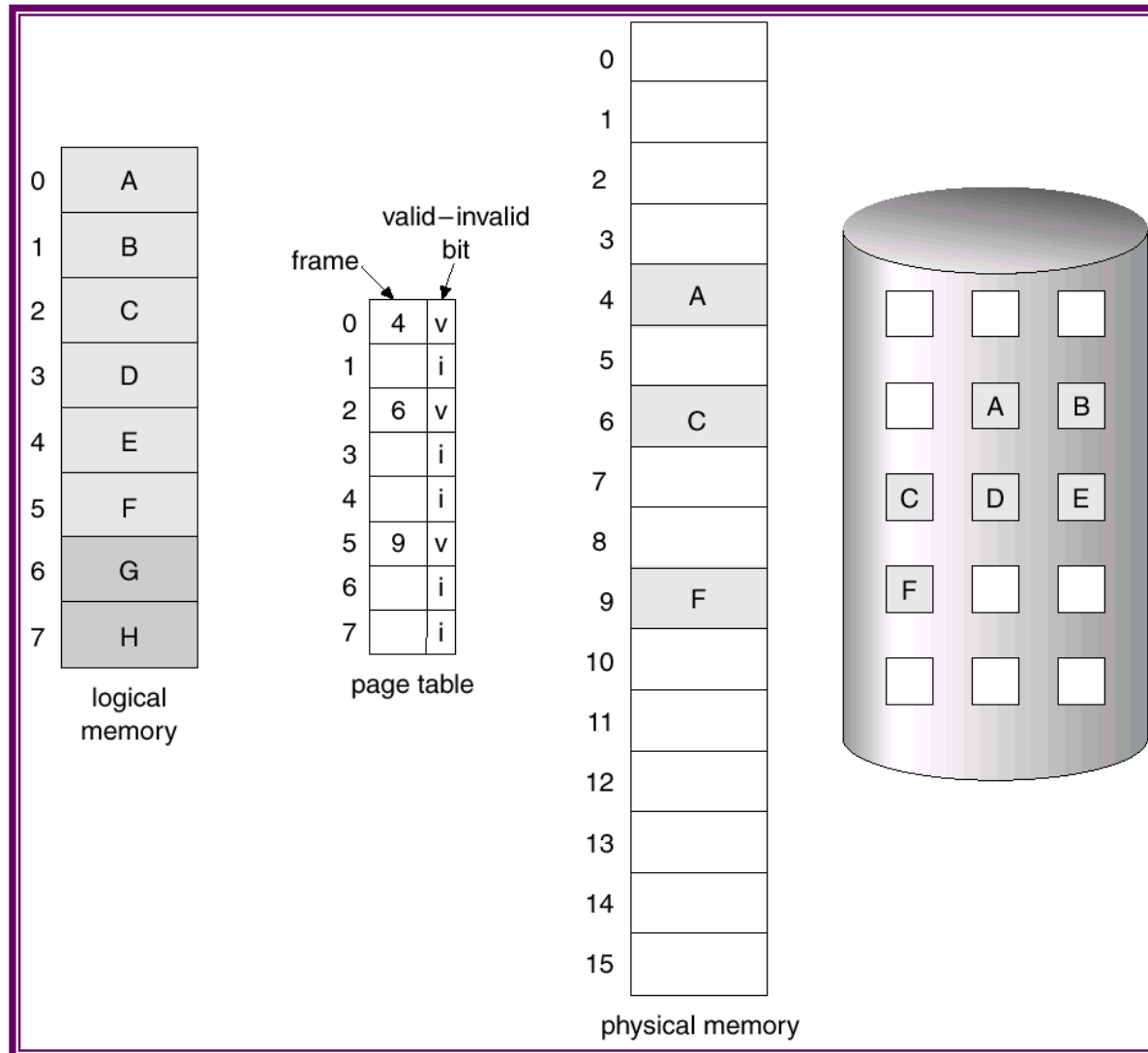
- Za svaki ulaz u tabelu stranica
 - ☞ asociran je **valid-invalid** bit is (1 ⇒ u memoriji, 0 ⇒ nije u memoriji)
- **Valid-invalid bit** je inicijalizovan na 0 za sve ulaze
- Primer tabele stranica:

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

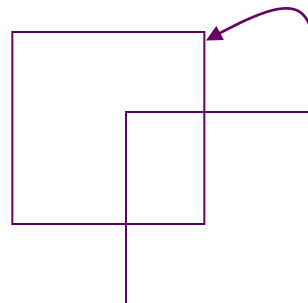
Ako je u toku prevođenja adresa, **valid-invalid bit** na ulazu u tabeli stranica =0 ⇒ greška u straničenju (*page fault*)

Tabela stranica kada neke stranice nisu u glavnoj memoriji



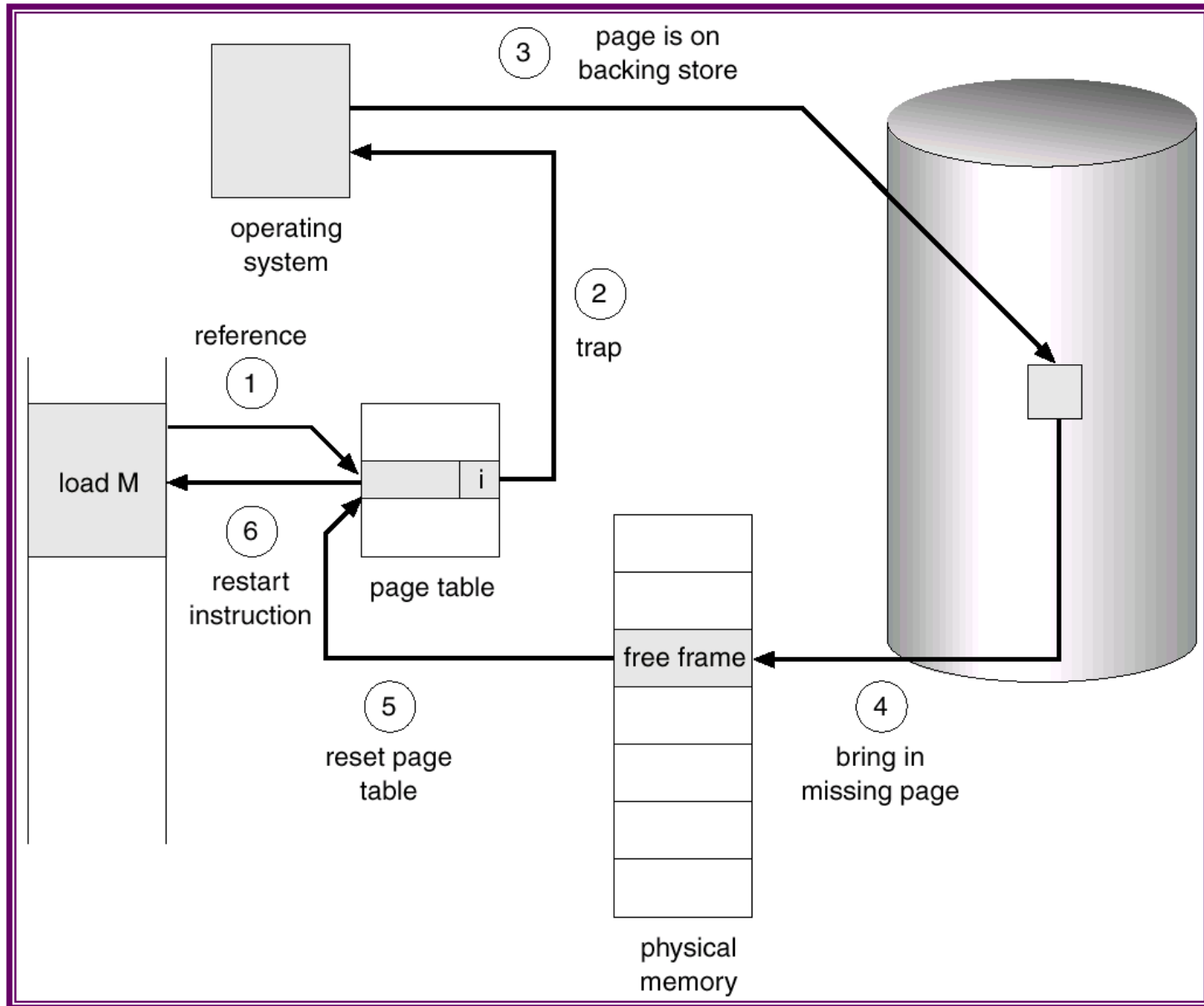
Greška u stranicenju (*Page Fault*)

- **PF rutina** ima **zadatak**,
 - ☞ da stranicu sa diska
 - ☞ prebaci u memoriju
- PF rutina **proverava** da li je referenca validna ili invalidna:
 - ☞ ako je referenca nevalidna \Rightarrow proces se prekida
 - ☞ ako je validna učitava se u memoriju
- **Uzima prazan okvir**
- **Zamenjuje stranicu za okvir**
- **Resetuje** tabele, **validation bit = 1**
- **Instrukcija se restartuje**: (carefully)
 - ☞ **pomeranje bloka**



- ☞ **automatsko inkrementiranje/dekrementiranje lokacije**

Koraci u manipulisanju sa PF-om



Šta se događa ako nema slobodnih okvira?

■ Zamena stranica:

- ☞ nađe se jedna stranicu u memoriji
- ☞ koja se ne koristi,
- ☞ upiše se na disk, a potom se oslobodi njena memorija

■ algoritam za najbolje performanse

- ☞ traži se algoritam
- ☞ koji daje rezultat
- ☞ sa najmanjim brojem grešaka u stranici.

■ Ista stranica može biti dovedena u memoriju nekoliko puta

Performanse DP tehnike

- **PF: $0 \leq p \leq 1.0$**

- ☞ ako je $p = 0 \Rightarrow$ nema grešaka u stranici

- ☞ ako je $p = 1 \Rightarrow$ svaka referenca ima grešku (trashing)

- **Effective Access Time (EAT)**

- **$EAT = (1 - p) \times \text{memory access} + p \times (\text{Page Fault time})$**

- **Page Fault time =** **page fault overhead**

- + swap page out /* if write*/**

- + swap page in**

- + restart overhead**

Page Fault

■ 1. Servisiranje PF prekida

- ☞ prekidni signal PF izaziva prelazak u operativni sistem
- ☞ čuvanje konteksta prekinutog procesa (registri procesora)
- ☞ određivanje da li je prekidni signal izazvan PF greškom

■ 2. Čitanje stranice

- ☞ disk I/O izazva čekanje, tj. blokadu PF rutine
- ☞ dok se čeka na disk CPU se daje nekom drugom
- ☞ prekidni signal koji znači da je disk I/O operacija završena
- ☞ određivanje da li je disk I/O prekidni signal

■ 3. Restartovanje procesa koji je izazvao PF

- ☞ korekcija tabele stranica
- ☞ obnova konteksta procesa

Primer DP

- Vreme pristupa memoriji MC = 1 mikrosekunda = 1000nsec
- Proces punjenja stranice u memoriju može uključiti dve I/O
 - ☞ stranica koja je zamenjena
 - ☞ ukoliko je promenjena dok je bila u memoriji
 - ☞ ponovo se upisuje na disk.
- ->1 ili 2 disk I/O operacije ->prosek =1.5 disk I/O
- Vreme razmene stranica = 10 msec = 10,000 microsekundi
- $EAT = \frac{(1 - p) \times 1 + p (15000)}{1 + 15000p}$ (microsec)

Kreiranje procesa

- **Virtuelna memorija može dosta pomoći**
- u toku
- **kreiranja procesa:**
 - **Copy-on-Write**
 - **Memorijski mapirane datoteke (*Memory-Mapped Files*)**

Copy-on-Write

- **Copy-on-Write (COW)**
 - ☞ omogućava da roditelji i dete proces
 - ☞ inicijalno
 - ☞ dele iste stranice u memoriji

- **Ako bilo koji proces**
 - ☞ **pokuša da modifikuje** deljenu stranicu,
 - ☞ tada će se prvo kreirati kopija za tu stranicu

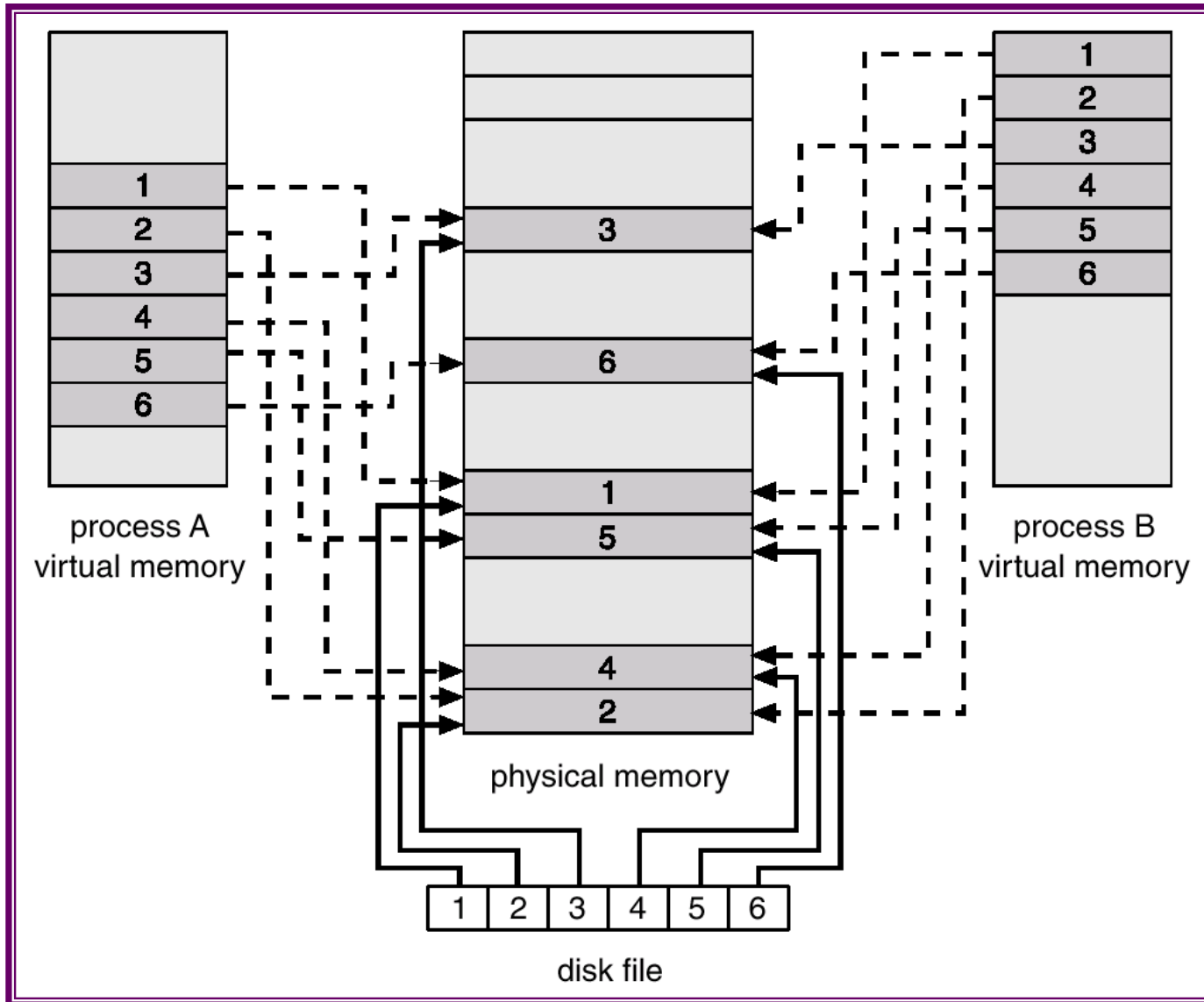
- **COW omogućava mnogo efikasnije kreiranje procesa**
 - ☞ tako što se samo modifikovane stranice kopiraju

- **Kod dodele novih stanica,**
 - ☞ poželjno je da one budu prazne, to se postiže tehnikom
 - ☞ koja puni nule u okvir koji se dodeljuje (zero-fill-on-demand)

Memorijski mapirane datoteke

- **Memorijski mapirana I/O datoteka** omogućava da se
 - ☞ I/O datotekama pristupa
 - ☞ preko memorijskih referenci
 - ☞ tako što se **deo virtuelnog memorijskog prostora**
 - ☞ **dodeli datotekama**
- **Inicijalni pristup datoteci se odvija preko DP tehnike**
 - ☞ koja će izazvati PF grešku
 - ☞ i kao rezultat te greške
 - ☞ **deo datoteke će se učitati u memoriju**
- **Sledeći čitanje/upis u/iz datoteke**
 - ☞ se tretira
 - ☞ kao običan pristup memoriji.
- **Pristup datoteci je uprošćen (brži)**
 - ☞ ako se I/O datoteci pristupa kroz memoriju
 - ☞ češće nego preko
 - ☞ read() write() sistemskih poziva
- **Deljenje datoteke:** omogućava da više procesa
 - ☞ mogu deliti istu datoteku
 - ☞ tako što se datoteka učita u memoriju

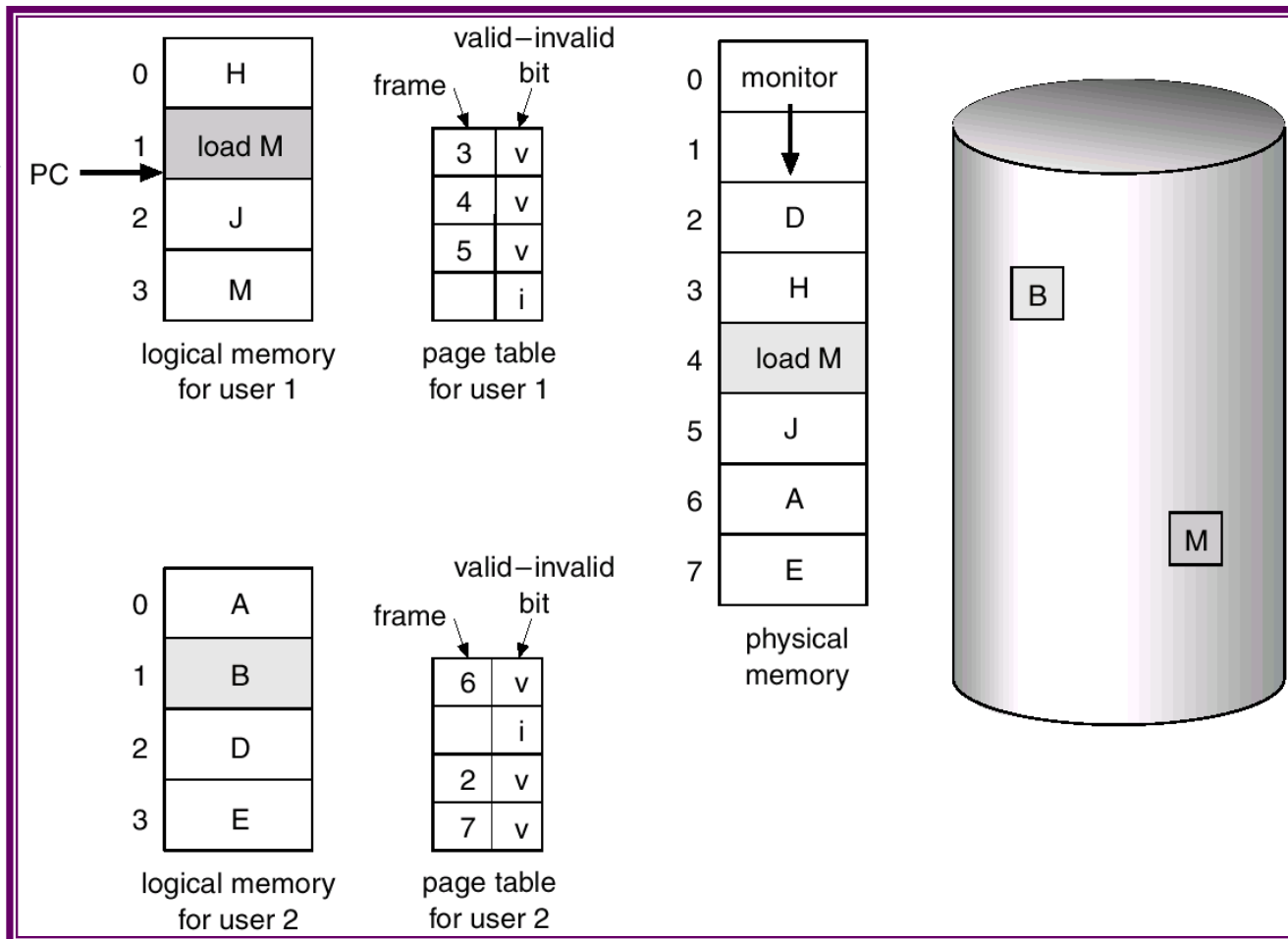
Memorijski mapirane datoteke



Zamena stranica

- **Kod alociranja memorije**
 - ☞ pojavljuje se veliki broj PF grešaka,
 - ☞ tada se koristi zamena stranica
- **Koristi se **dirty bit****
 - ☞ koji opisuje da je stranica modifikovana
 - ☞ na disk se upisuju samo modifikovane stranice
- **Zamena stranica kompletira razdvajanje između logičke i fizičke memorije:**
 - ☞ velika virtuelna memorija
 - ☞ se koristi samo kada je
 - ☞ fizička memorija mala

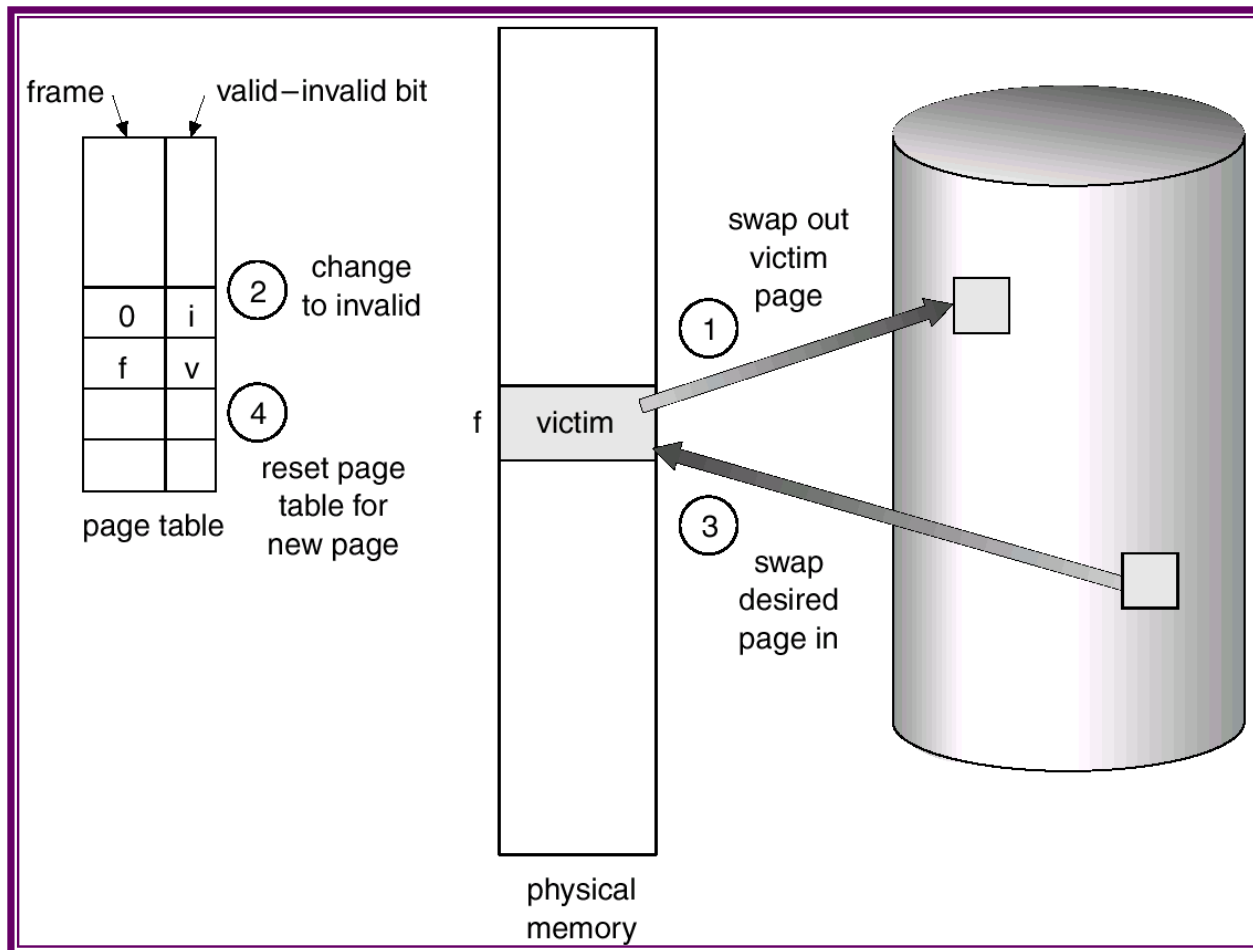
Zamena stranica



Opis zamene stranica

1. **Potrebno je pronaći lokaciju na disku gde je smeštena stranica**
2. **Traži se slobodan okvir:**
 - Ako ima slobodnih okvira uzima se jedan od njih
 - Ako **nema nijednog slobodnog**, pomoću algoritama za zamenjivanje stranica izabra se **žrtva: okvir koji će biti zamenjen**
3. **Željena stranica se učitava u oslobođeni okvir. Tabele stranica i okvira se ažuriraju**
4. **Proces se restartuje**
5. **Pažljivo sa:**
 - pomeranjem instrukcija
 - ili
 - automatskim inkrementiranjem/dekrementiranjem

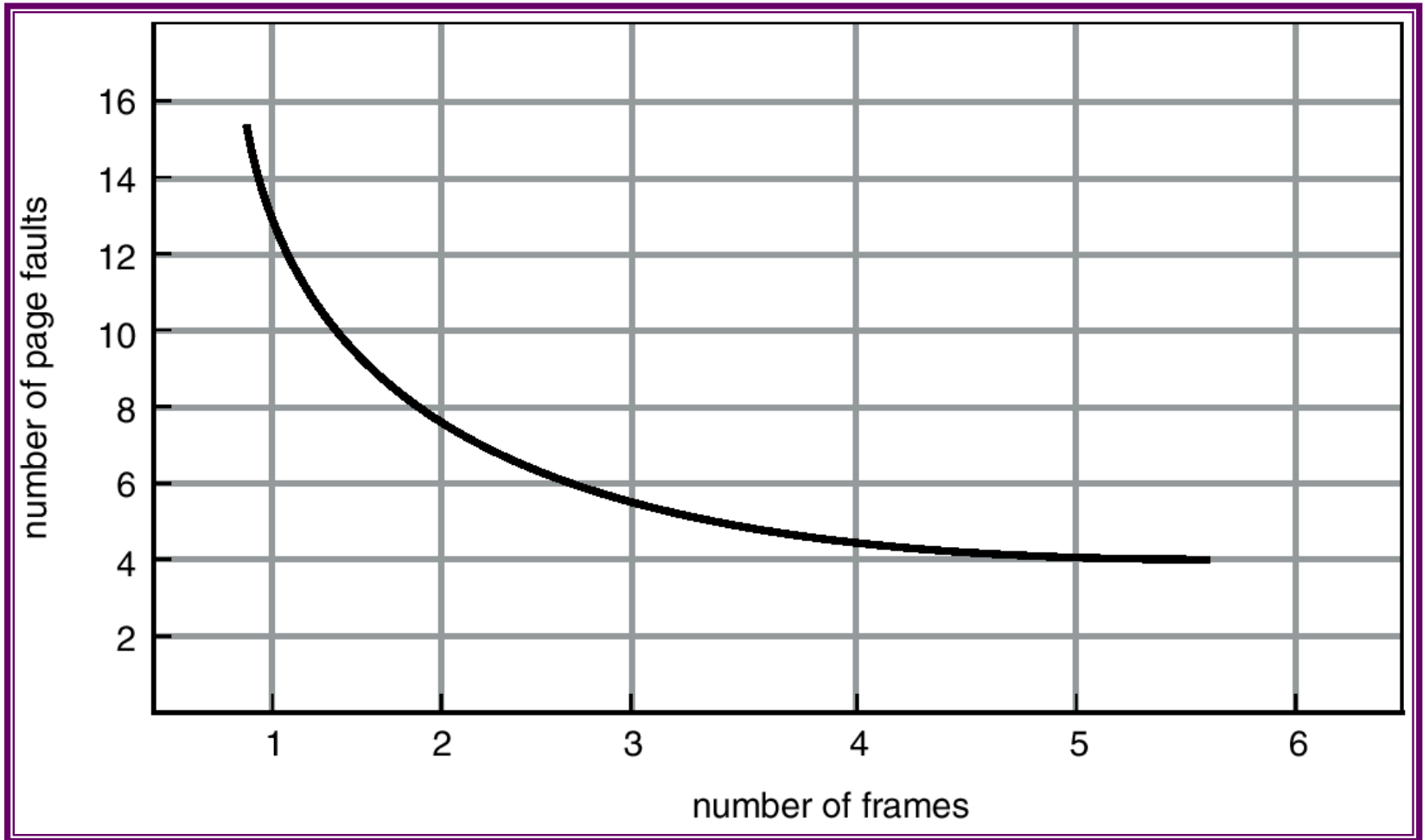
Zamena stranica



Algoritmi za zamenu stranica

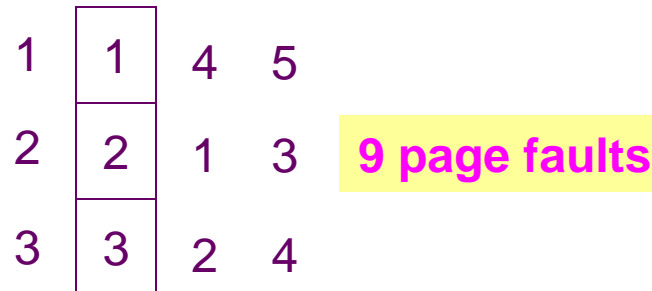
- **Potrebna je najmanja verovatnoća PF grešaka**
- **Prilikom evaulacije algoritama**
 - ☞ **definiše se ulazna sekvenca referenci**
 - ☞ **i**
 - ☞ **na datoj veličini fizičke memorije**
 - ☞ **analizira se broj PF grešaka**
- **U svim primerima, memorijske reference su**
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Graph of Page Faults Versus The Number of Frames

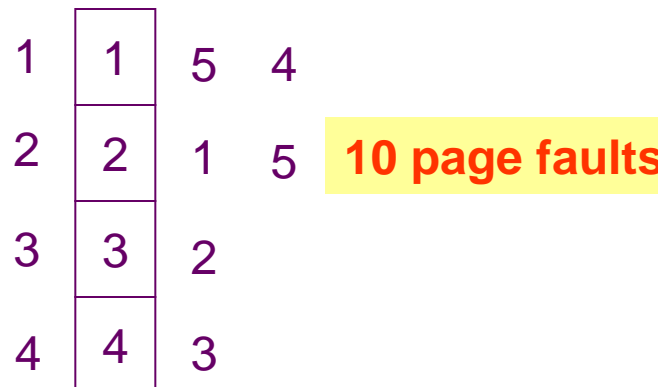


First-In-First-Out (FIFO) Algorithm

- **Najstarija stranica će biti zamenjena**
- **Niz memorijskih referenci: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- **3 okvira** (3 stranice po procesu istovremeno mogu biti u memoriji)



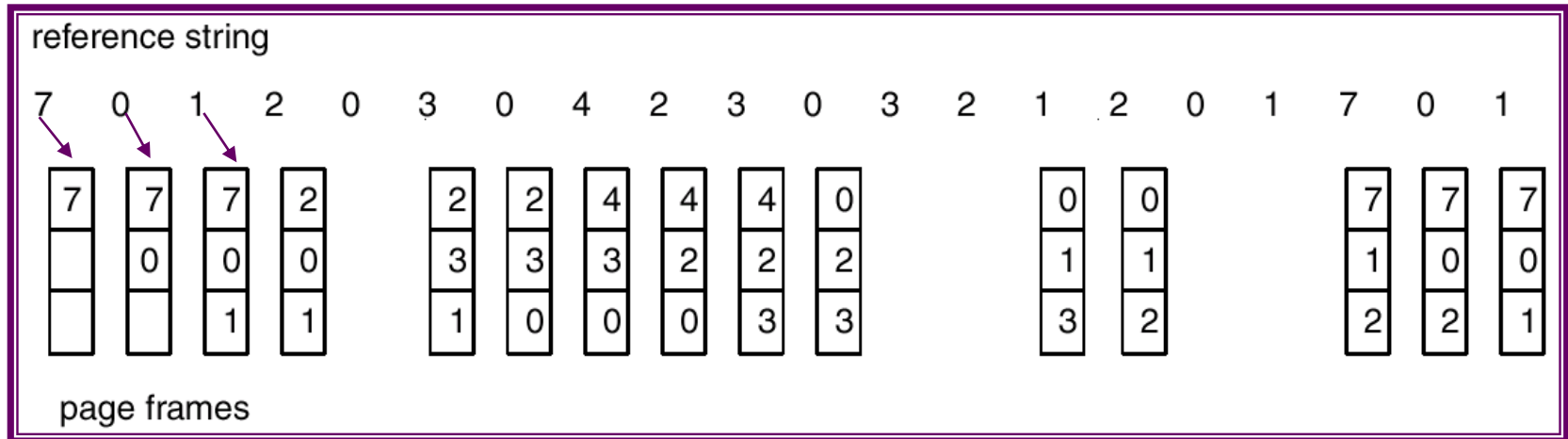
- **4 okvira**



- **FIFO zamena – Belady anomalija**

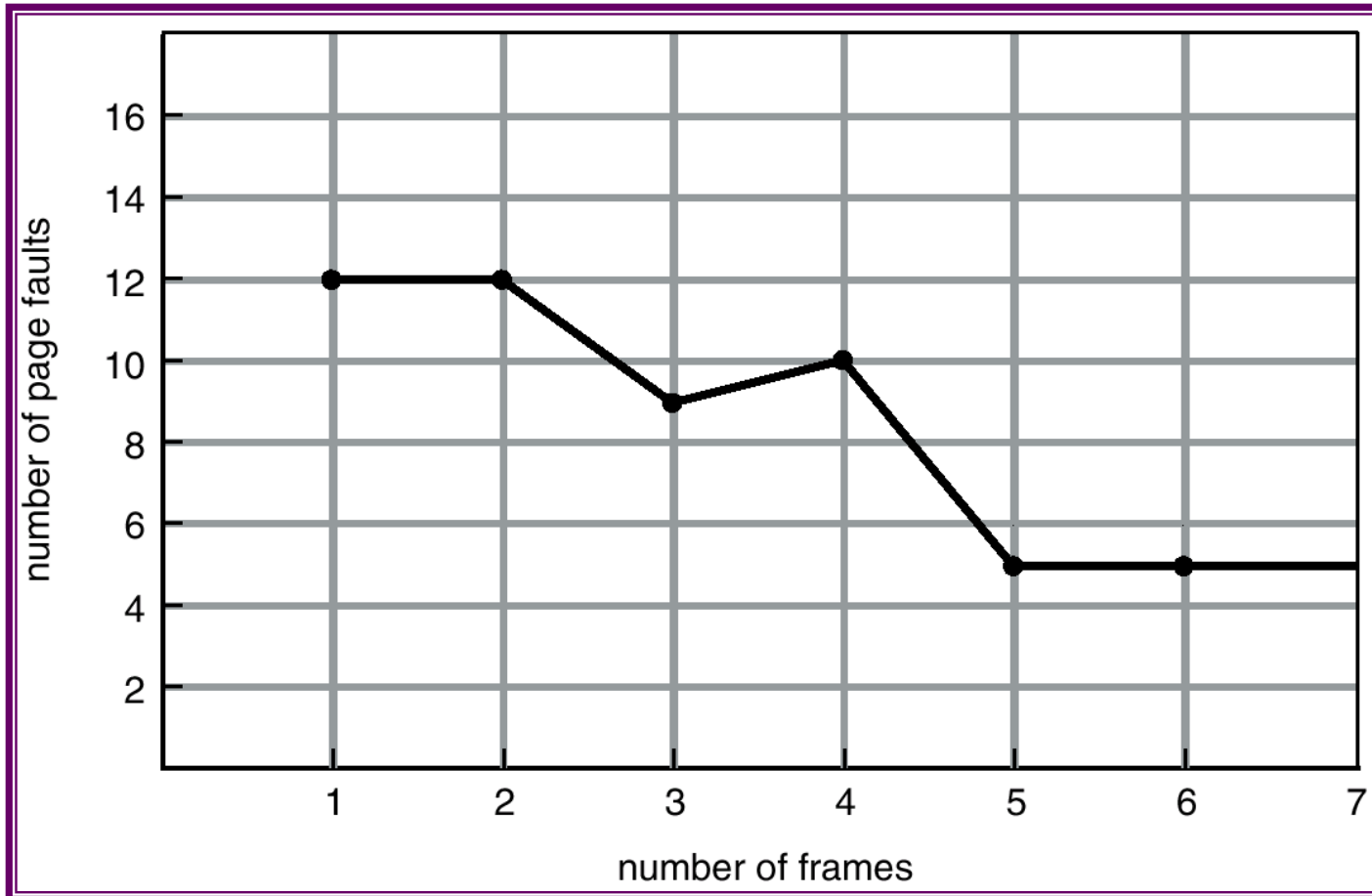
☞ **više okvira ⇒ manje PF**

FIFO algoritam za zamenu stranica



PF=15

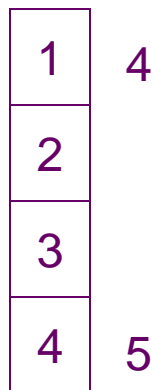
Belady anomalija



Optimalni algoritam

- Zamenjuje stranice
- koje se neće koristiti
- najduže vremena od svih.
- primer sa 4 okvira

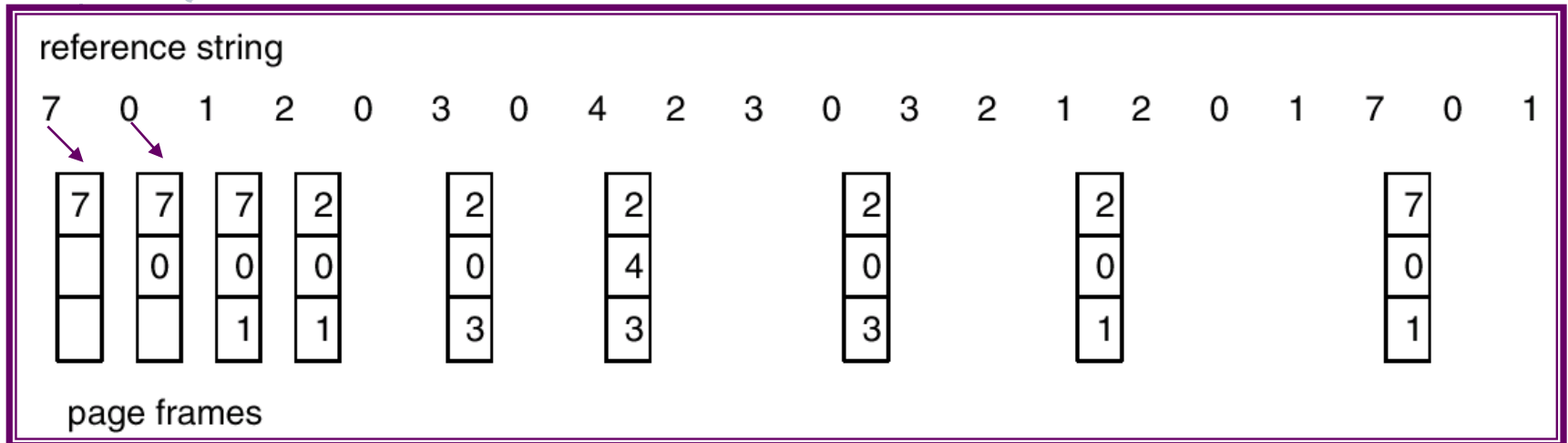
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

- OPT je najbolji mogući algoritam, ali na žalost ne može se implementirati.
- Ne postoji način da operativni sistem dođe do potrebne informacije o tome, kada će koja stranica biti potrebna.

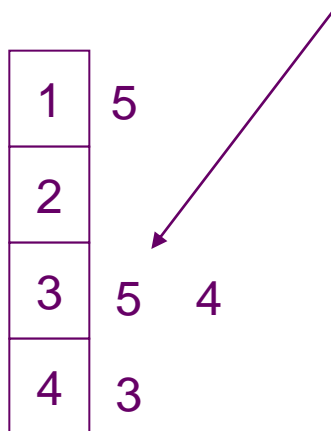
Optimalni algoritam za zamenu stranica



PF=9

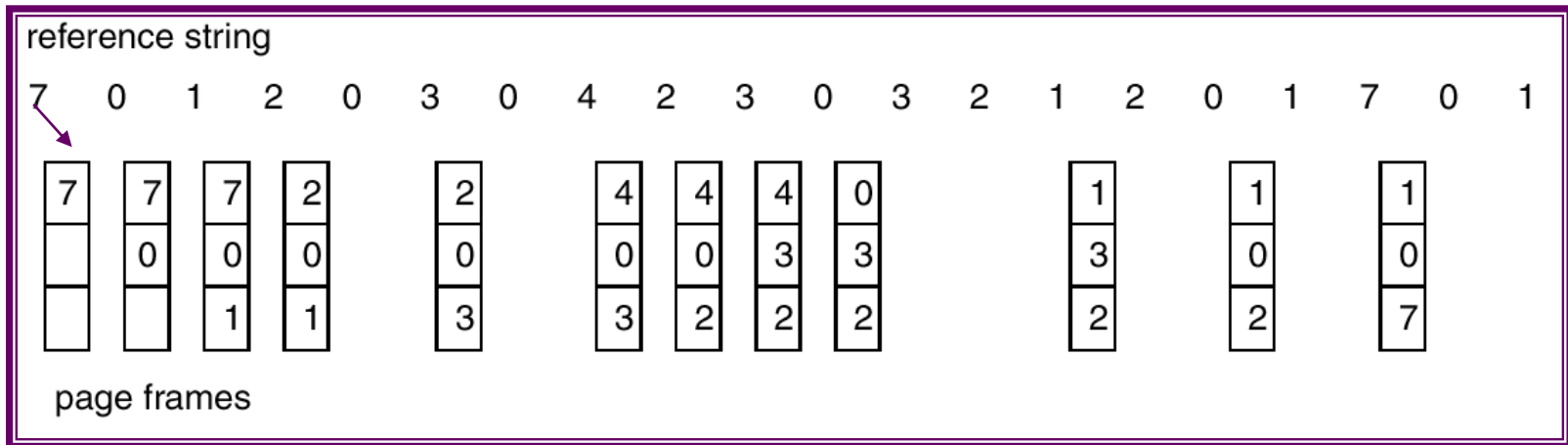
Least Recently Used (LRU) Algorithm

- Zamenjuje se stranica za koju je **proteklo najviše vremena od kada se nije koristila**
- Niz memorijskih referenci: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**



- Realizacija **pomoću brojača (timer)**
 - ☞ Svaka stranica ima svoj interni registar;
 - 📄 svaki put kada se pristupi stranici,
 - 📄 sadržaj **casovnika** kopira se u interni registar
 - ☞ Kada dođe do PF prekida,
 - 📄 algoritam pregleda sve interne registre
 - 📄 i bira stranicu **čiji je broj najmanji**.
 - 📄 (što znači da najduže nije korišćena)

LRU algoritam za zamenu stranica

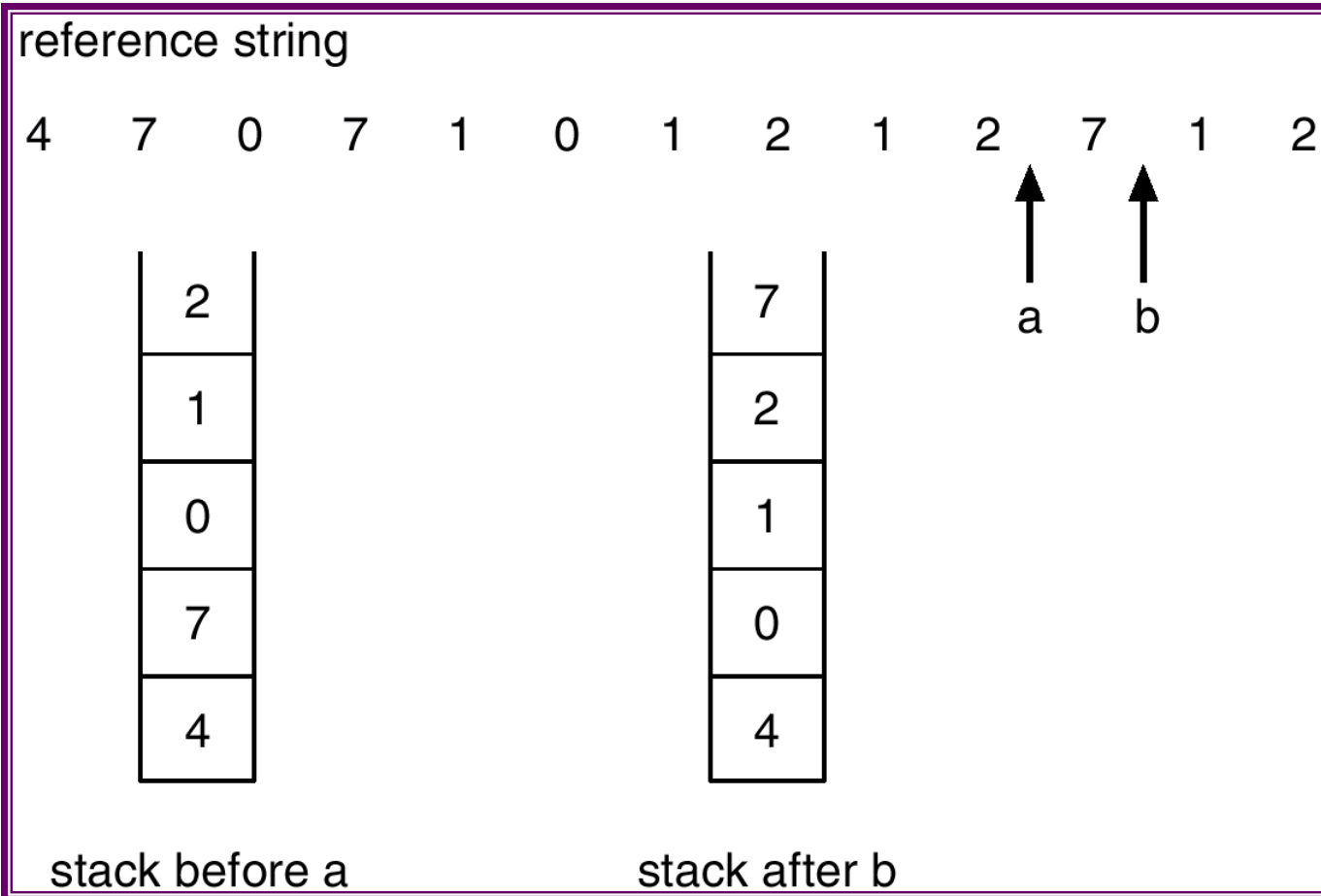


PF=12

LRU Algoritam (2)

- **Realizacija pomoću steka:**
- **formira se stek koji opisuje redosled pristupanja stranicama**
- **(head list & tail list):**
 - ☞ **Nakon PF prekida:**
 - 📄 **bira 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 ažurira.**

Algoritam LRU – realizacija pomoću steka



Aproksimativni LRU algoritmi

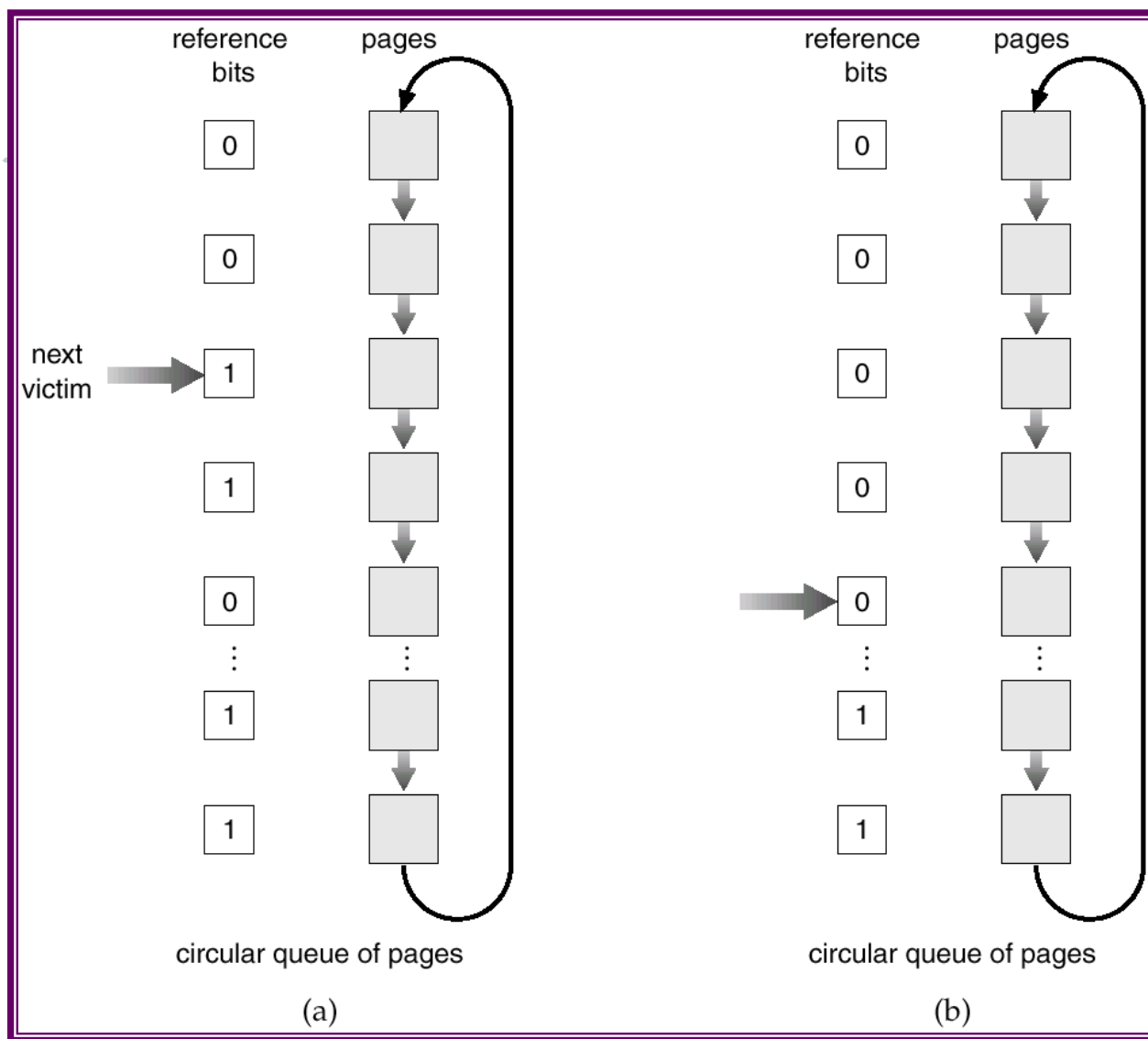
■ Reference bit

- ☞ Za svaki ulaz u tabeli stranica postoji **referentni bit**, inicijalizovan na 0
- ☞ **Kada se stranica referencira, bit se setuje na 1**
- ☞ Nakon određenog vremena R bit se briše pomoću tajmerskih prekida
- ☞ Nema informacije o redosledu korišćenja okvira

■ Druga šansa (second chance)

- ☞ **FIFO baziran**
- ☞ **Koristi R bit**
- ☞ Kada treba izbaciti stranicu, **uzima se zadnja iz reda čekanja, i pogleda R bit:**
- ☞ Ako stranica ima referentni bit **R = 1**, tada:
 - 📄 se R setuje na 0
 - 📄 a stranica se ostavlja u memoriji
 - 📄 zatim se gleda sledeća stranica na kraju reda

Druga šansa – realizacija pomoću kružnog reda čekanja



Klase stranica

- Na osnovu vrednosti R i M bitova okvire delimo u 4 klase:
 - ☞ **clasa 0. (0,0) stranica nije ni korišćena ni modifikovana**
 - ☞ **clasa 1. (0,1) stranica nije korišćena ali je modifikovana**
 - ☞ **clasa 2. (1,0) stranica je korišćena ali nije modifikovana**
 - ☞ **clasa 3. (1,1) stranica je i korišćena i modifikovana**
- **Stranica za zamenu je svaka stranica u najnižoj klasi**
- **Svaka klasa sa više stranica je FIFO bazirana**

Brojački algoritmi

- Čuva informaciju **koliko je puta**
- **svaka stranica bila referencirana**

- **LFU** Algoritam:
 - **zamenjuje stranicu koja ima najmanji broj referenci**

- **MFU** Algoritam:
 - ☞ baziran na činjenici
 - ☞ da je stranica **sa najmanjim brojem referenci**
 - ☞ **verovatno upravo učitana**
 - ☞ i
 - ☞ **upravo treba da se sačuva**

Alokacija okvira

- Svaki proces zahteva minimalan broj stranica.
 - ☞ Minimalni broj okvira po procesu zavisi od procesorske arhitekture
- Primer:
- IBM 370 – 6 pages to handle SS MOVE instruction:
 - ☞ instruction is 6 bytes, might span 2 pages.
 - ☞ 2 pages to handle from.
 - ☞ 2 pages to handle to.
- Dve najvažnije šeme za alokaciju.
 - ☞ fiksna alokacija
 - ☞ prioritetna alokacija

Fiksna alokacija

- **Jednaka alokacija:**
- npr., ako ima 100 okvira i 5 procesa, svaki dobija 20 stranica.
- **Proporcionalna alokacija:**
- **Svakom procesu pripašće sledeći broj okvira:**

– s_i = size of process p_i

– $S = \sum s_i$

– m = total number of frames

– a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Prioritetna alokacija

- Koristi šemu za proporcionalnu alokaciju
 - ☞ upotrebljavajući
- **prioritet** umesto veličine.
- Ako proces P_i generiše PF, postoje dva načina za zamenu
 - ☞ 1. bira za zamenu
 - ☞ jedan od njegovih okvira.
 - ☞ 2. bira za zamenu okvir
 - ☞ čiji proces ima najmanji prioritet.

Globalna i lokalna zamena stranica

■ **Globalna zamena:**

- ☞ proces bira bilo koji okvir
- ☞ iz cele fizičke memorije;
- ☞ jedan proces može uzeti okvir od drugog.

■ **Lokalna zamena:**

- ☞ svaki proces bira
- ☞ samo okvire koji su mu dodeljeni

Thrashing

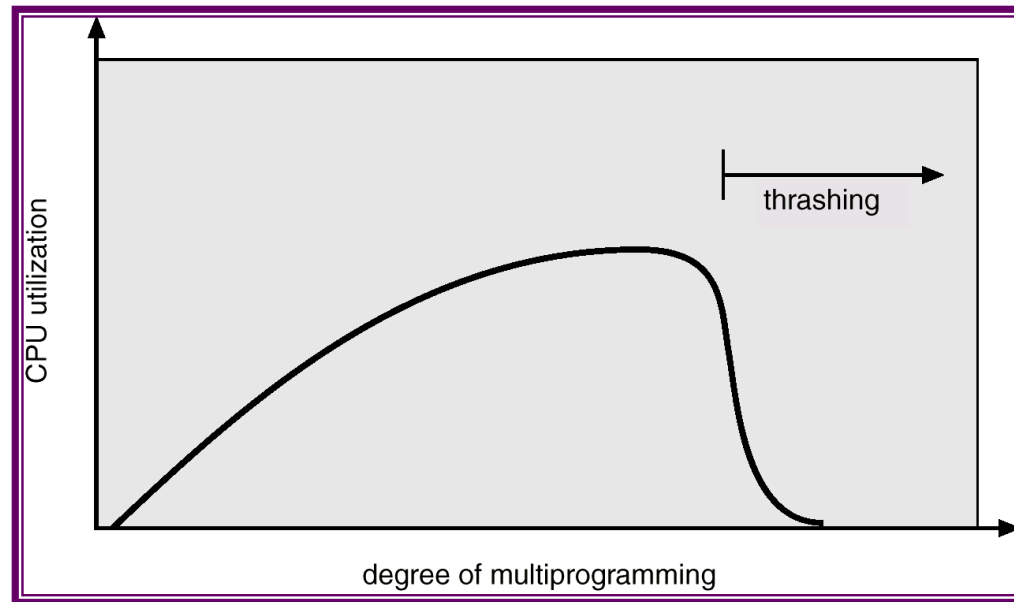
- Ako proces nema “dovoljno” stranica,
- pojava PF grešaka je veoma česta

- Razmatrajmo sledeći slučaj:
 - ☞ ako je **stepen iskorišćenja CPU-a** previše mali
 - ☞ **operativni sistem**
 - 📄 **povećava stepen multiprogramiranja**
 - ☞ tako što aktivira nove procese.

- **Thrashing** ≡ pojava čestog razmenjivanja stranica

- **Thrashing** ≡ proces je zauzet zamenom stranica
 - ☞ Proces troši više vremena za straničenje nego za izvršavanje

Thrashing



■ Kako se izbegava thrashing efekat?

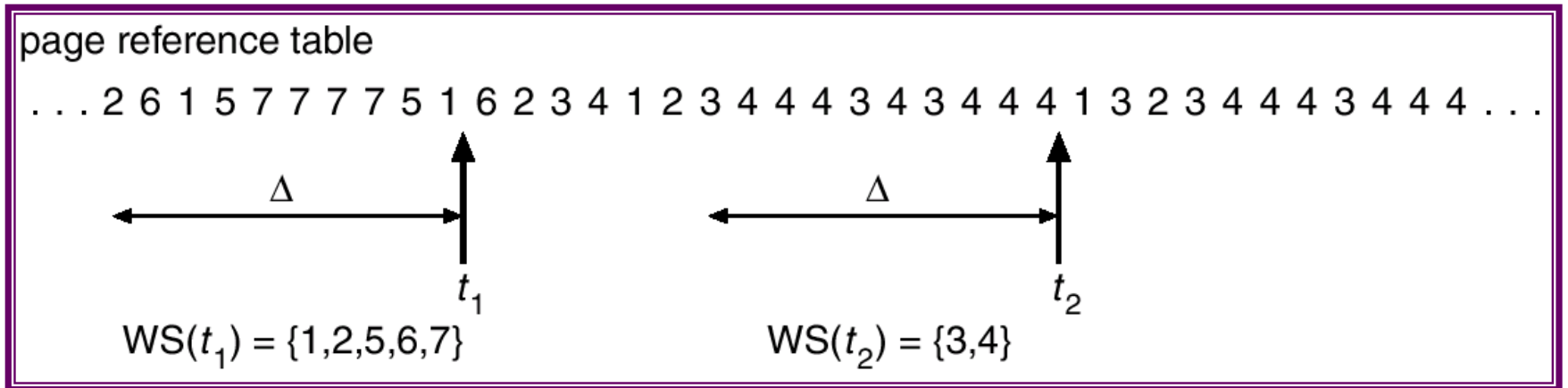
Model lokalnosti

- ☞ Lokalnost je skup stranica koje proces koristi zajedno u jednom intervalu vremena
- ☞ Proces može menjati svoje lokalnosti
- ☞ Lokalnosti se mogu preklapati.

■ Zašto nastupa thrashing efekat?

Σ suma lokalnosti > ukupne fizičke memorije

Radni model (*Working-Set Model*)



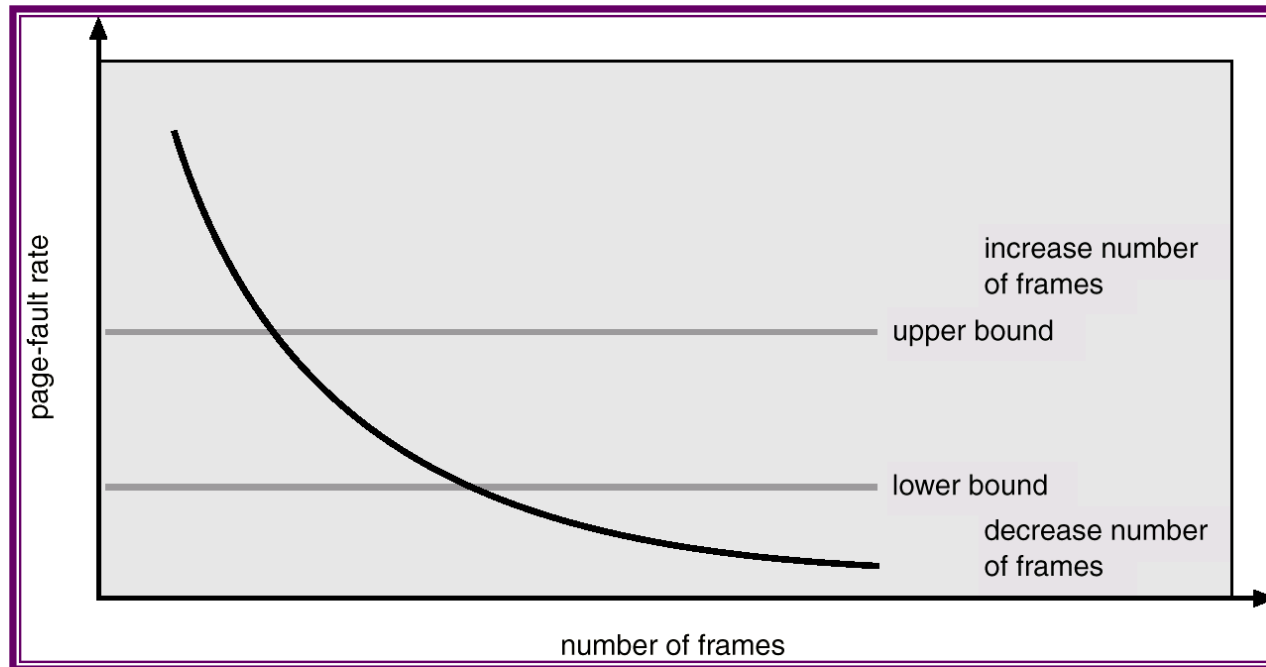
Radni model (*Working-Set Model*)

- $\Delta \equiv$ prozor radnog skupa \equiv fiksni broj referenciranih stranica
- Primer: 10,000 instrukcija (memorijskih referenci)
- WSS_i (veličina radnog skupa procesa P_i) =
 - ☞ ukupan broj stranica
 - ☞ koje proces traži u vremenskom prozoru Δ
- $D = \sum WSS_i \equiv$ ukupna veličina radnog prostora
- Ako je $D > m \Rightarrow$ Thrashing
- Kada D dođe do veličine sistemske memorije,
- pojedini procesi treba da se suspenduju na disk

Keeping Track of the Working Set

- Aprokcimaci preko **Tajmera + referentni bit**
- Primer: $\Delta = 10,000$
 - ☞ **Tajmer** pravi prekide posle svakih 5000 vremeskih jedinica.
 - ☞ U memoriji se čuvaju 2 bita za svaku stranicu
 - ☞ **Uvek kada tajmer napravi prekid**
 - 📄 kopira i setuje vrednost svih referentnih bitova na 0.
 - ☞ Ako je jedan od bitova u memoriji = 1 \Rightarrow stranica je u radnom skupu.
- Zašto ovo nije kompletno ispravno?
- Poboljšanje =
 - ☞ 10 bita
 - ☞ i
 - ☞ prekid svakih 1000 vremenskih jedinica

Broj PF grešaka u funkciji broja okvira



■ Uspostavlja se “prihvatljiva” PF učestanost.

- ☞ Ako je aktuelna učestanost mala, proces gubi okvir (ima ih mnogo)
- ☞ Ako je aktuelna učestanost velika, proces dobija okvir (dodeljeno mu je malo okvira)

Dopunska razmatranja

■ Prepaging

☞ **DP=veliki broj PF grešaka pri startovanju programa**

☞ **Prepaging** je pokušaj da se spreči ovo inicijalno straničenje

☞ Primer:

 **obnoviće ceo radni skup prethodno suspendovanog procesa**

■ Veličina stranice

☞ **Fragmentacija** (manja stranica=>manja fragmentacija)

☞ **veličina tabele stranica**

☞ **I/O overhead** (zahteva veće stranice)

☞ **Lokalnost** (bolja rezolucija, bolja lokalnost, smanjen PF)

☞ **Tipične stranice:512, 1K, 2K, 4K**

Dopunska razmatranja (2)

- **TLB efikasnost:**
- **Količina dostupne memorije u TLB**
- **TLB efikasnost = (TLB Size) X (Page Size)**
- **Idealno,**
 - ☞ **radni skup svakog procesa se memoriše u TLB.**
- **U protivnom,**
 - ☞ **postoji visok stepen PF grešaka.**

Povećanje veličine TLB-a

■ Povećanje veličine stranice.

- ☞ Ovo može dovesti do **povećane fragmentacije**
- ☞ mada ne zahtevaju sve aplikacije velike stranice

■ Obezbediti stranice različitih veličina

- ☞ Ovo omogućava aplikacijama
- ☞ da traže veće stranice
- ☞ i da ih koriste
- ☞ bez povećanja fragmentacije

Dopunska razmatranja (3)

■ Struktura programa

☞ `int A[][] = new int[1024][1024];`

☞ Each row is stored in one page

☞ Program 1

```
for (j = 0; j < A.length; j++)  
  for (i = 0; i < A.length; i++)  
    A[i,j] = 0;
```

1024 x 1024 page faults

☞ Program 2

```
for (i = 0; i < A.length; i++)  
  for (j = 0; j < A.length; j++)  
    A[i,j] = 0;
```

1024 page faults

Dopunska razmatranja (3)

LA 2M
=1024x1024x2

PA 2M
=1024x1024x2

PA 2K =1024x2

■ Struktura programa

☞ `int A[][] = new
int[1024][1024];`

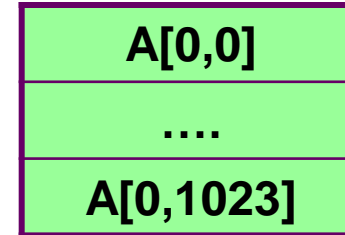
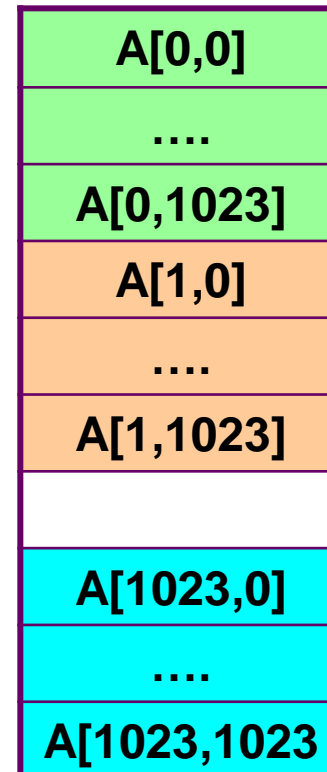
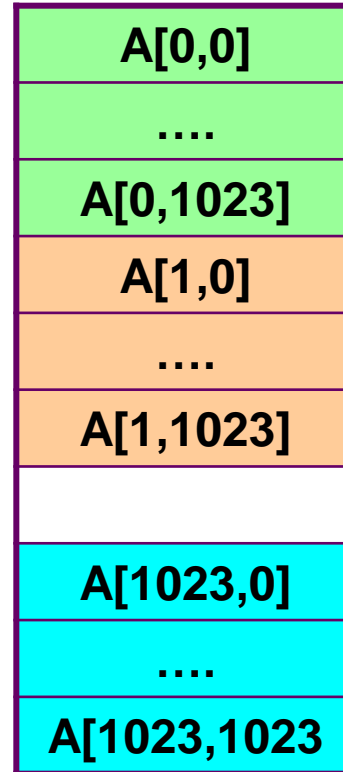
☞ Each row is stored in one page

☞ Program 1

☞ `for (j = 0; j < A.length; j++)
for (i = 0; i < A.length; i++)`

☞ `A[i,j] = 0;`

☞ 1024 x 1024 page faults



PF =1024

PF =1024x1024

Dopunska razmatranja (3)

LA 2M
=1024x1024x2

PA 2M
=1024x1024x2

PA 2K =1024x2

■ Struktura programa

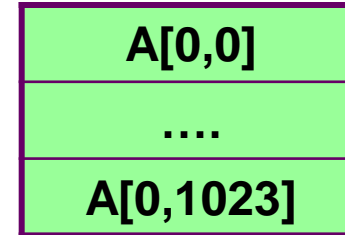
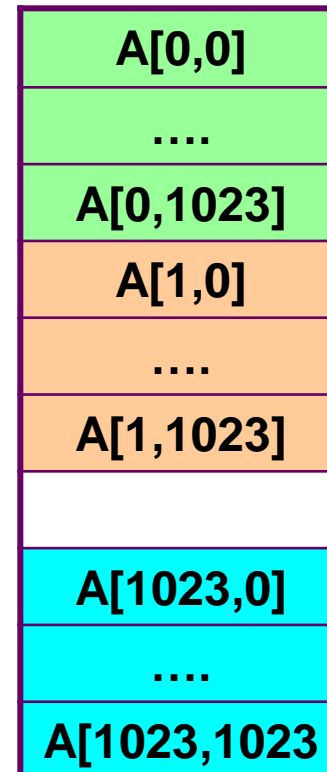
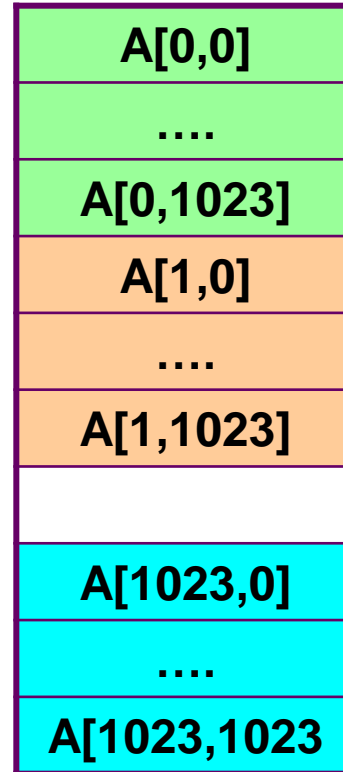
☞ `int A[][] = new
int[1024][1024];`

☞ **Each row is stored in one
page**

☞ **Program 2**

☞ `for (i = 0; i < A.length; i++)
for (j = 0; j < A.length; j++)
A[i,j] = 0;`

1024 page faults



PF =1024

PF =1024

Dopunska razmatranja (4)

■ Dok proces obavlja I/O transfer

- ☞ Može biti suspendovan
- ☞ Stranica za I/O transfer može biti dodeljena drugom procesu
- ☞ Postoje dva rešenja za ovaj problem
 - 📄 I/O se nikad ne izvršavaju korisničkoj memoriji nego u kernelnim baferima
 - 📄 Mehanizam za zaključavanje stranice

■ I/O zaključavanje:

■ Stranice mogu biti zaključane u memoriji

■ I/O zaključavanje

- ☞ Stranice koje se koriste za kopiranje datoteke sa uređaja
- ☞ trebaju biti zaključane od buduće selekcije
- ☞ u algoritma za zamenu stranica.

Razlog zbog čega okviri koji se koriste za I/O trebaju biti u memoriji

