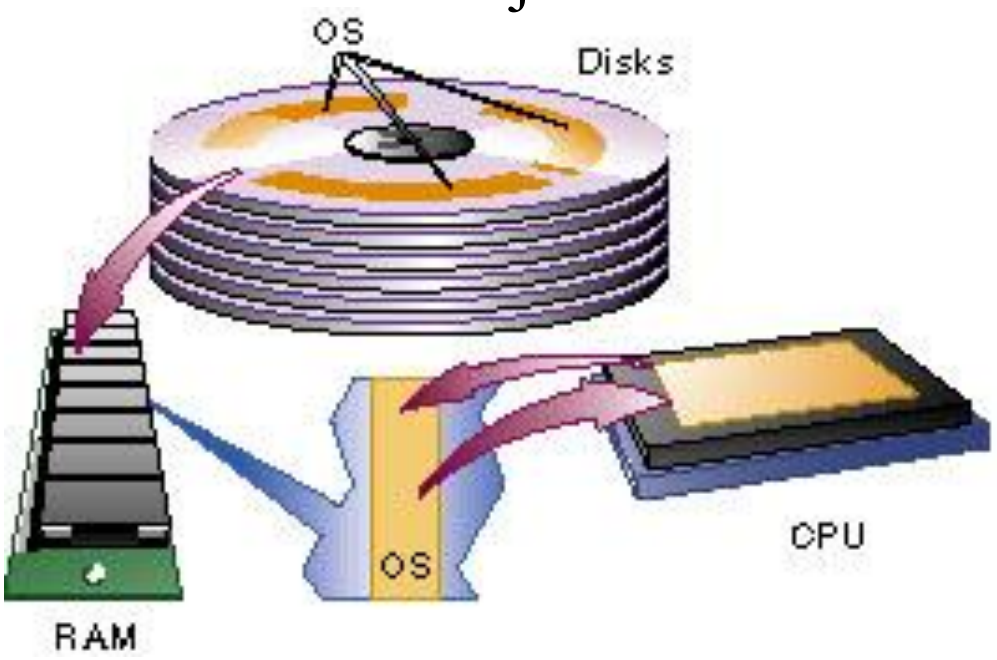
# S A D R Ž A J

# 5.1- Upravljanje memorijom

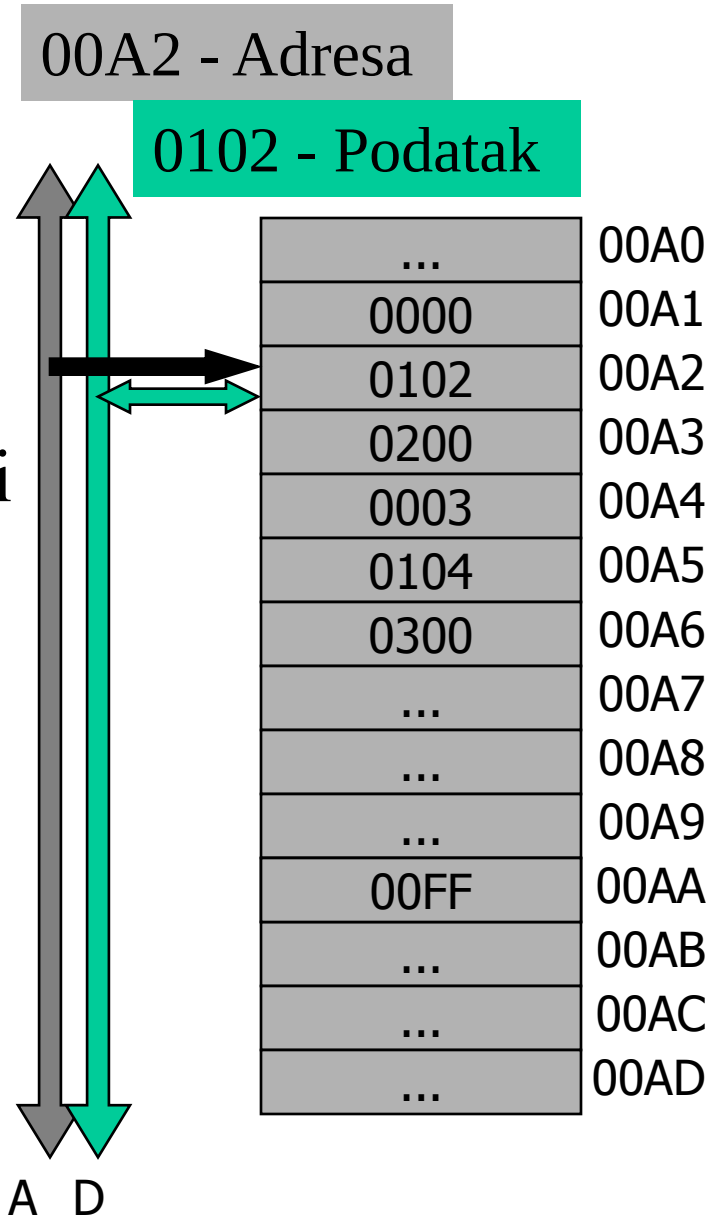
***Performanse celog sistema zavise od efikasnog upravljanja memorijom***

1. Fizička memorija (primarna) - OS i programi koji se izvršavaju
   - ✓ Deljenje podataka i koda u glavnoj memoriji
   - ✓ Brzina operativne memorije je znatno manja od brzine procesora
   - ✓ Razlika u brzini nadoknađuje se brzim (i skupim) keš memorijama
2. Virtuelna memorija (sekundarna) - čuvaju se podaci i programi koji se trenutno ne izvršavaju




# 5.1 - Upravljanje memorijom

- Fizička memorija računara: linearno uređen niz ćelija sa mogućnošću *direktnog pristupa (direct access)*
- Direktan pristup: svaka ćelija ima svoju *adresu (address)* preko koje se pristupa podacima koje ona sadrži
- **RAM** (*Random Access Memory*): memorija sa mogućnošću čitanja i upisa koja gubi sadržaj prestankom rada napajanja
- **ROM** (*Read Only Memory*): memorija koju možemo samo da očitavamo ali ona zadržava sadržaj po gubitku napajanja.



# 5.1- Upravljanje memorijom - zahtevi

**1. Relokacija**

- U multiprogramskom sistemu postoji više programa koji se izvršavaju
- Programi se ne izvršavaju uvek u istim memorijskom prostoru pa se zahteva da prevedu memorijske reference u stvarne fizičke adrese

**2. Zaštita**

- Svaki proces bi trebalo zaštititi od neželjenog uticaja drugih procesa

**3. Deljenje memorije**

- Svaki zaštitni mehanizam mora da ima fleksibilnost kako bi dozvolio da više procesa pristupa istom delu glavne memorije.

**4. Logička organizacija**,

- Organizacija programa i podataka po modulima – lakši rad

**5. Fizička organizacija**

- Računarska memorija je organizovana u najmanje dva nivoa: primarna i sekundarna memorija - glavni sistemski problem je organizacija toka informacija između glavne i sekundarne memorije.

# 5.1- Upravljanje memorijom - ciljevi

1. **Alokacija memorije** - dodela memorije procesima
2. **Razdvajanja fizičkog i logičkog adresnog prostora** programa i vezivanje adresa (prevođenje relativnih relokatibilnih adresa u fiksne)
3. **Logička organizacija memorije -** razdvajanje neizmenljivih segmenata (modula i procedura) od segmenata s promenljivim sadržajem. Razdvanjanjem segmenata programa postiže se:
   a) nezavisan rad na segmentima koji čine program
   b) zaštita segmenata sa malim premašenjem, u vidu listi za kontrolu pristupa segmentima (u višeprocesnom okruženju, procesi ne smeju menjati vrednosti drugih memorijskih lokacija bez dozvole)
   c) deljenje segmenata između većeg broja procesa (u slučaju da proces ima dozvolu da izmeni vrednost memorijskih lokacija koje koristi drugi proces, tj. postoje zajedničke lokacije)
4. **Relokacija -** obuhvata sažimanje tj. defragmentaciju radne memorije i *swap* (suspendovanje procesa njegovim smeštanjem na disk)
5. **Dinamičko punjenje memorije** programom i dinamičko vezivanje

# 5.1 - Dodeljivanje memorije

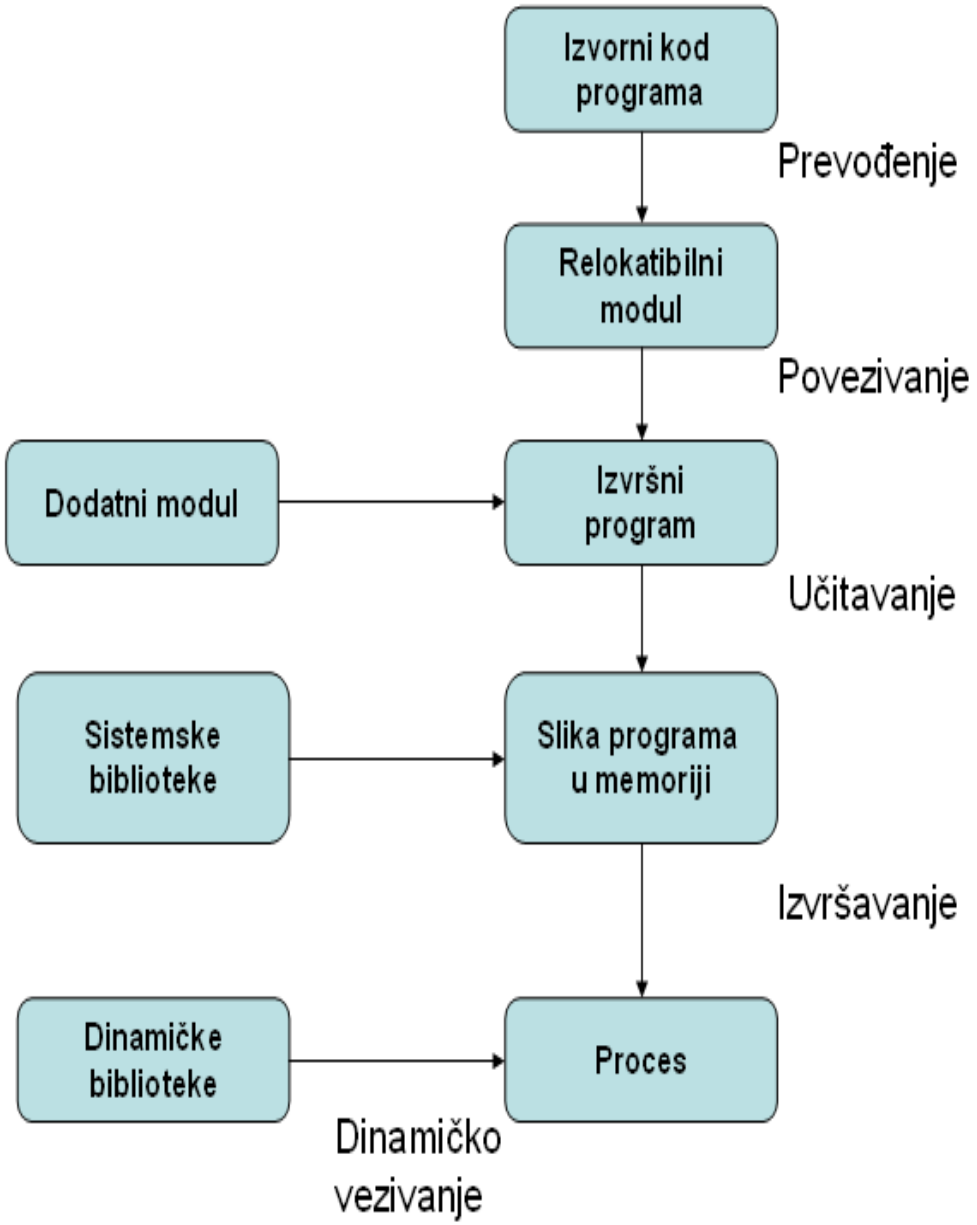
- Program se piše u izvornom kodu *(source code – source modules)* koji je nerazumljiv računaru, pa pre njegovog izvršavanja mora da se:

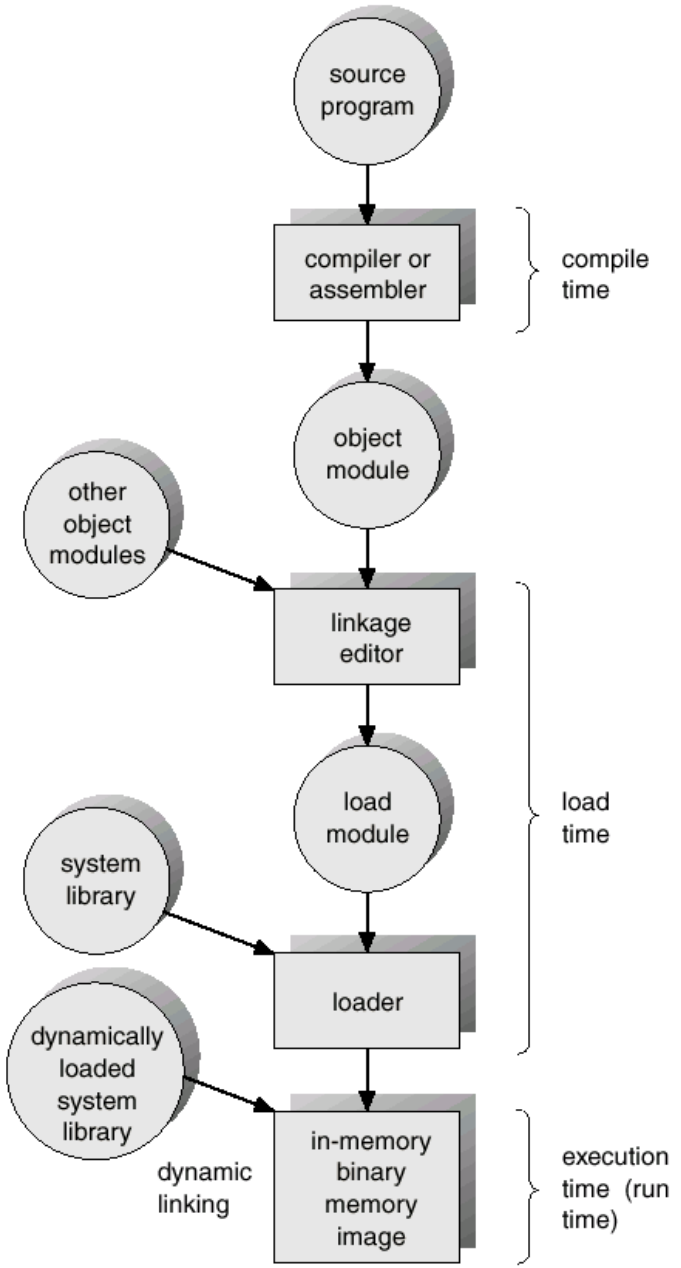
I. **Prevede** (*compilation)* - prevođenje izvornog koda radi **programski prevodilac** (*compiler)* i rezultat prevođenja je objektni kod (*object code – object modules)* koji još uvek nije spreman da se izvršava

II. **Povezivanje** (*linking)* - povezivanje objektnih modula sa različitim bibliotekama vrši **povezivač** (*linker)* i ono može biti:
   1. Statičko - svi objektni moduli su povezani u jednu datoteku
   2. Dinamičko - povezivanje pojedinih modula se vrši trenutno

III. **Punjenje** (*loading)* - nakon povezivanja program još uvek nije spreman za izvršenje jer je potrebno da se program smesti u memoriju i povežu logičke i fizičke adrese tj. da se logičke adrese prevedu u stvarne fizičke adrese na kojima će se on izvršavati u OM. To radi **punilac** (*loader*). Za svaki objektni modul postoji i tabela eksternih simbola (*external  symbol table)* koja se povezuje u vreme punjenja

# 5.1 - Dodeljivanje memorije



# 5.1 - Dodeljivanje memorije

- Pored korisničkih procesa u OM se smešta i rezidentni deo OS (jezgro)
- Da bi u višeprocesnom okruženju obezbedio stabilan i pouzdan rad sistema, neophodno je što efikasnije dodeliti različite delove memorije.
- Memorija se deli na najmanje **dve particije** od kojih je jedna (najčešće niži deo) namenjena **rezidentnom delu OS**(*kernel space*) a druga particija (viši delovi) – **korisničkim procesima** (*user space*).
- Obično se u najnižem delu memorije nalazi **tabela prekidnih rutina**.
- Glavni problem pri upravljanju memorijom jeste dodela slobodne memorije procesima koji su u ulaznom redu (alokacija memorije).

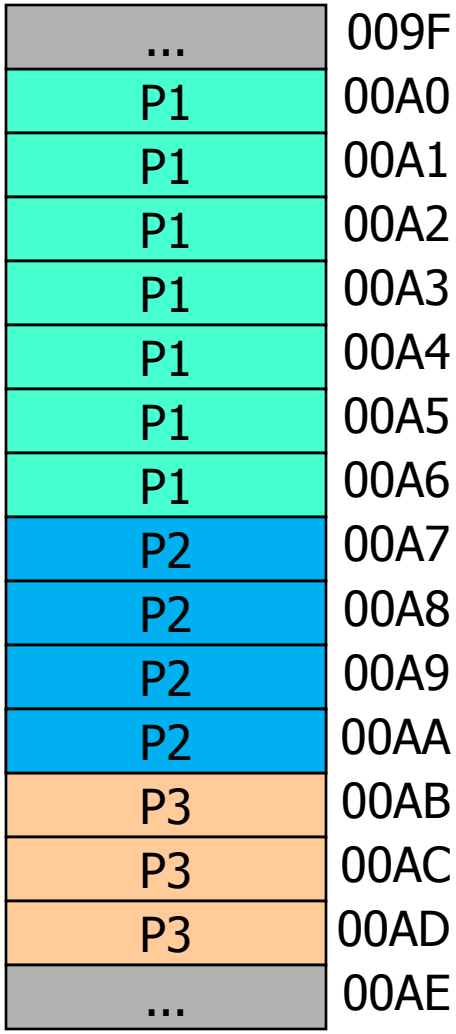
***Tehnike za dodelu memorije procesima grubo se mogu podeliti na***:

I. ***Kontinualna alokacija*** *(contiguous allocation)* - i logički i fizički adresni prostor procesa sastoje se od kontinualnih niza memorijskih reči, pri čemu mem.particije po veličini mogu biti jednake ili različite

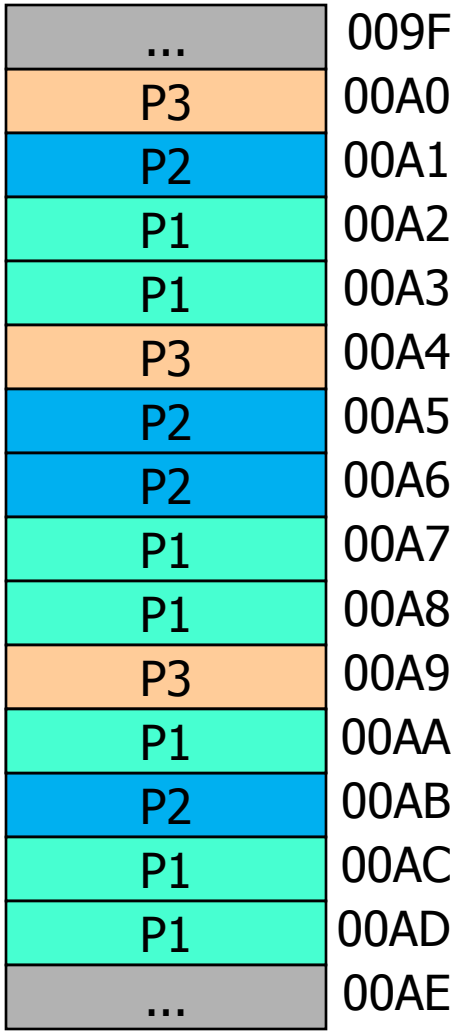
II. ***Diskontinualna alokacija*** *(discontiguous allocation)* – fizički adresni prostor procesa nije realizovan kao kontinualan niz memorijskih adresa; obuhvata straničenje, segmentiranja i straničenja sa segmentiranjem

# 5.1 - Dodeljivanje memorije

Kontinualna dodela




Diskontinualna dodela



# 5.1 – Vezivanje adresa

- Program se nalazi na disku kao binarna izvršna datoteka (*binary executable).*
- Program sa diska se mora učitati u memoriju, unutar adresnog prostora novostvorenog procesa da bi mogao da se pravilno izvrši
- Proces se u toku izvršavanja može više puta pomerati na relaciji disk – memorija. Još veći problem je kod multiprogramskih okruženja

***Kolekcija procesa na disku, koja čeka povratak u memoriju i nastavak izvršenja, naziva se ulazni red (input queue).***

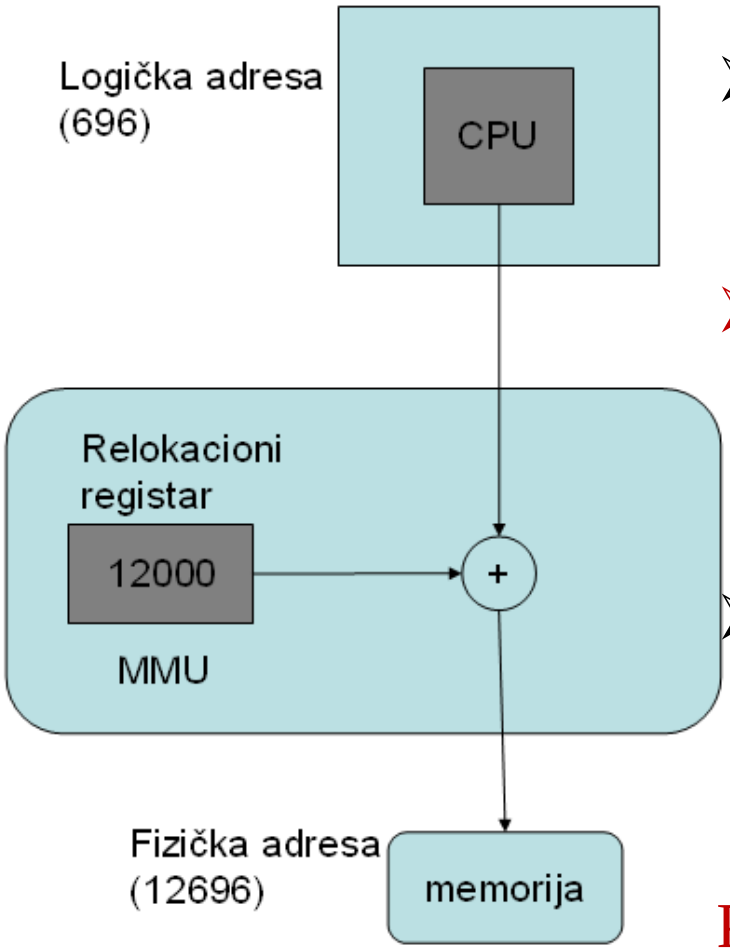
- Programer ne može unapred znati koji će procesi biti zastupljeni u memoriji prilikom izvršenja programa kao niti koje će memorijske lokacije biti slobodne.
- Dužnost operativnog sistema je da prevede relativne tj. relokatibilne adrese u fiksne prilikom učitavanja programa u memoriju.
- Prevodilac (*compiler*) prevodi izvorni kod programa i vezuje (*bind*)
- Simboličke adrese iz izvornog koda (relativne ili relokatibilne) punilac (*loader*) pretvara u apsolutne kod učitavanja programa u memoriju

# 5.1- Logički i fizički adresni prostor

- Adresa koju generiše procesorska instrukcija je logička, a adresa same memorijske jedinice je fizička.
- Logičke i fizičke adrese su potpuno iste u fazi prevođenja i u fazi učitavanja programa, ali se razlikuju u fazi izvršavanja (logičke adrese se u fazi izvršavanja nazivaju i virtuelnim adresama).
- Skup svih logičkih adresa naziva se logički ili virtuelni adresni prostor, a skup svih fizičkih adresa koje im odgovaraju - fizički adresni prostor.
- Mapiranje (preslikavanje) virtuelnog adresnog prostora u fizički obavlja hardver koji se naziva **MMU** (*Memory-Management Unit*)

1. **Vreme prevođenja**:generišu se relativne a ne apsolutne adrese osim ako nije unapred poznata adresa lokacije od koje će se program izvršavati
2. **Vreme punjenja**: generišu se apsolutne adrese programa (*linker* i *loader*) povezivanje sa drugim relokatibilnim modulima (biblioteke)
3. **Vreme izvođenja**: u ovom slučaju moguće je da program menja svoje adrese i tokom izvođenja, odnosno prebacuje se iz jednog u drugo memorijsko područje.

# 5.1- Logički i fizički adresni prostor

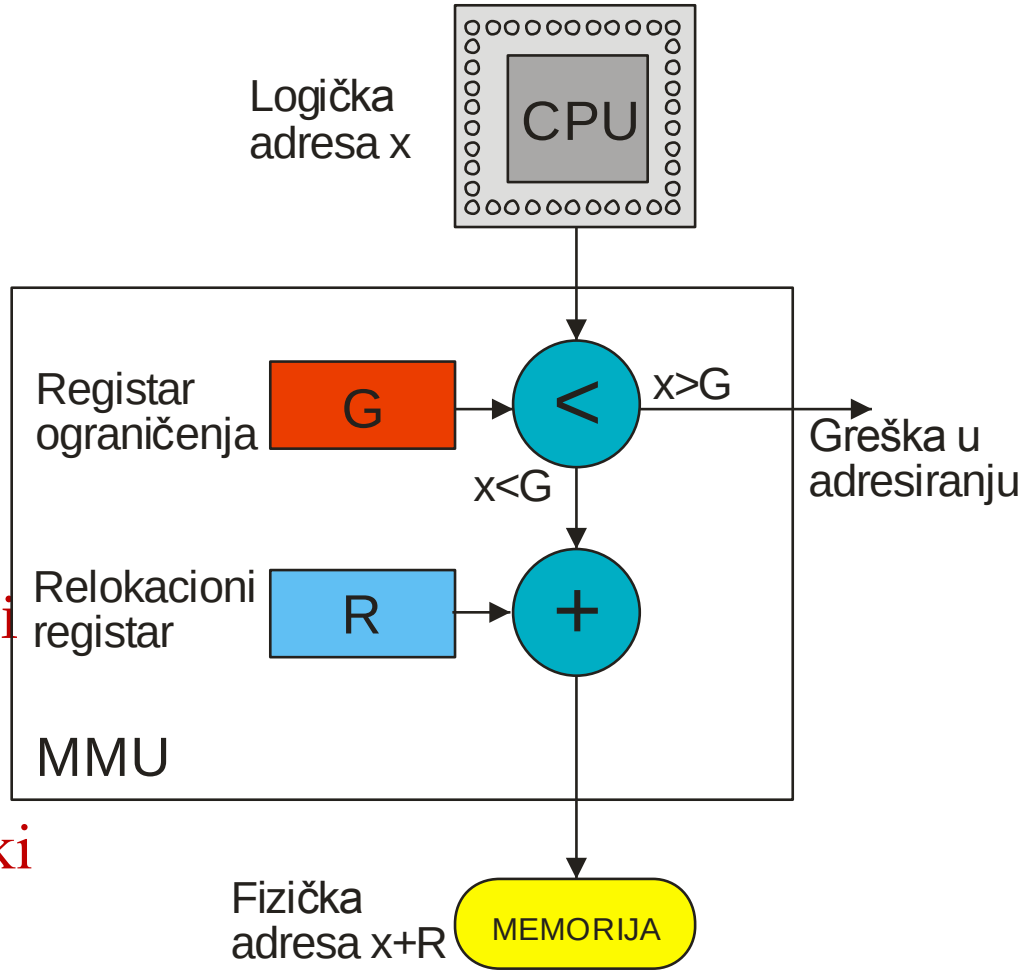



- **Relokacioni registar** definiše adresu fizičkog početka programa.
- Svaka logička adresa koju generiše program se sabira s vrednošću relokacionog registra i tako se dobija fizička adresa.
- Korisnički program uvek počinje od nulte adrese i ne treba voditi računa o svom fizičkom prostoru, osim o gornjoj granici programa (max).
- Logički adresni prostor koji se nalazi u opsegu [0,max] mapira se u opseg [R+0, R+max], gde je R vrednost relokacionog registra - fizička adresa početka programa.

Primer: *Prevodilac vezuje neku promenljivu za lokaciju na adresi 696 u odnosu na početak programskog modula, koju punilac pretvara u apsolutnu adresu **12696**.*

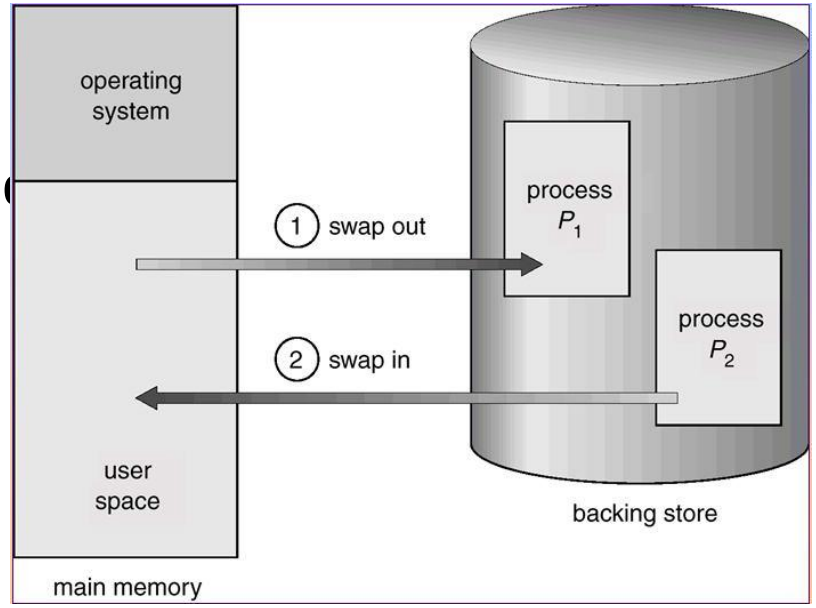
# 5.1 - Zaštita memorije

- Zaštita pristupa memorijskim lokacijama sprovodi se putem:
1. Relokacionog registra - sadrži najmanju adresu procesa
2. Registra ograničenja - sadrži najveći opseg logičkih adresa procesa
- Zahteva se podrška hardvera tj. CPU-a jer su oba registra registri koji pripadaju CPU
- Omogućuju da se veličina OM koja pripada OS menja dinamički prema potrebama
- Tranzijentni kod – može da se menja
- Rezidentni kod – fiksan deo OS



# 5.1 – Razmena (Swapping)

- Segment jednog procesa se privremeno sklanja na spoljnu memoriju da bi se oslobodila memorija za neki drugi proc
- Deo diska predviđen za ovu namenu naziva se ***swap space***
- Segment se vraća u glavnu memoriju kada to bude ponovo potrebno
- Dolazi do degradacije performansi




- Razmena (swap) koristi se u prioritetnim šemama za raspoređivanje
- Ova varijanta razmenjivanja naziva se ***roll out***, ***roll in***.

***Proces koji se razmenjuje mora biti potpuno oslobođen aktivnosti***.

- Tehnika razmene zahteva postojanje 3 komponente:

1. Prostor na disku (***swap space***) na koji će se smeštati uspavani procesi
2. Mehanizam ***swap-out*** koji prebacuje proces iz memorije na disk
3. Mehanizam ***swap-in*** koji vraća uspavani proces sa diska u memoriju

# 5.2-Programerske tehnike upravljanja memorijom

- Prilikom projektovanja programa koji zbog veličine ne mogu stati celi u memoriju,programeri koriste odgovarajuće tehnike da bi to omogućili

***1. Dinamičko učitavanje programa u memoriju***

- Suština dinamičkog punjenja (*dynamic loading*) odnosi se na smeštaj samo potrebnih delova programa u memoriju,pri čemu se odgovarajuće rutine učitavaju u memoriju samo kad ih program pozove
- Prednosti dinamičkog punjenja su u tome što rutine koje nisu trenutno potrebne ne zauzimaju mesto u OM, što je zgodno za velike programe.
- Takođe, dinamičko punjenje ne zahteva specijalnu podršku od OS
- OS pomaže tako što obezbeđuje biblioteku za dinamičko punjenje.

***2. Dinamičko povezivanje*** (*dynamic linking)*

- Dinamičko povezivanje je slično konceptu dinamičkog punjenja.
- Sistemske biblioteke povezuju se (link) i pune u OM prema zahtevima programa i to u samom toku izvršavanja programa
- Pri dinamičkom povezivanju naparavi se vezivna funkcija (*stub)*, kao deo koda za svaki poziv sistemske biblioteke-smanjuje se veličina koda
- Dinamičko povezivanje traži podršku na nivou OS

# 5.2 – Tehnika preklapanja

- Da bi se omogućilo izvršenje procesa koji je veći od same fizičke memorije, koristi se **tehnika preklapanja** (*overlay*).

***Preklapanje omogućava da se u memoriji čuvaju samo oni delovi programa koji su potrebni u tom trenutku.***

- Programer može podeliti sam izvorni kôd programa na dva dela.

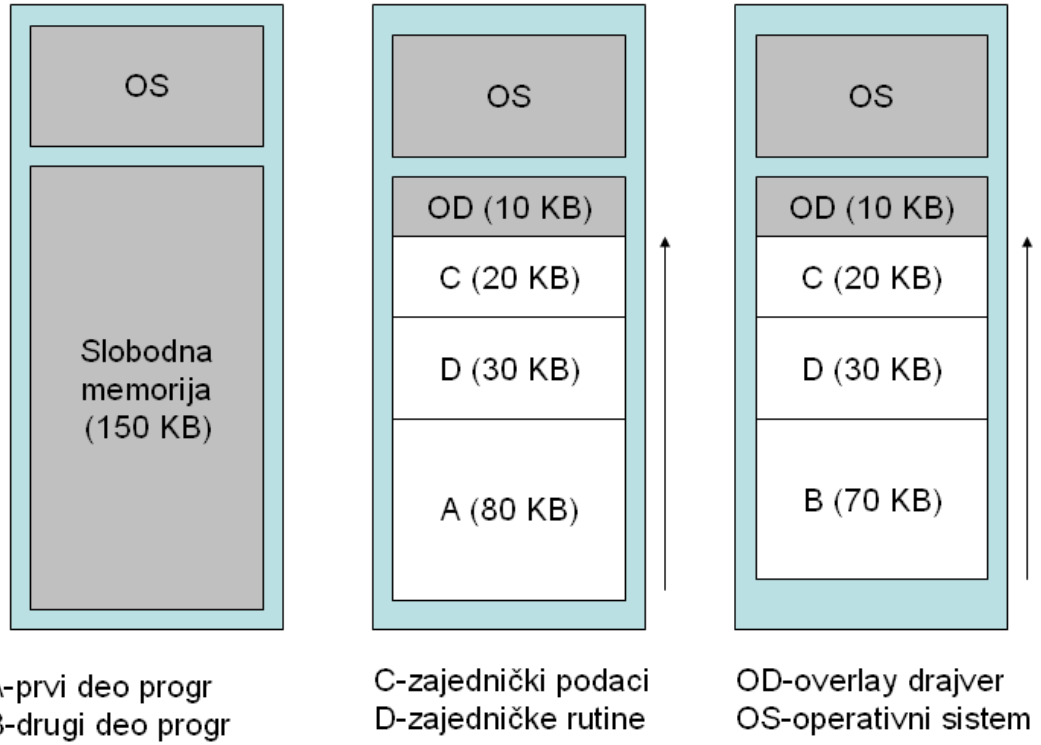
**Primer**:*Pretpostavimo da je veličina programskih komponenata sledeća: prvi deo (80 KB), drugi deo (70 KB), zajednički podaci (20 KB), zajedničke rutine (30 KB) i drajver koji upravlja tehnikom (10 KB)*

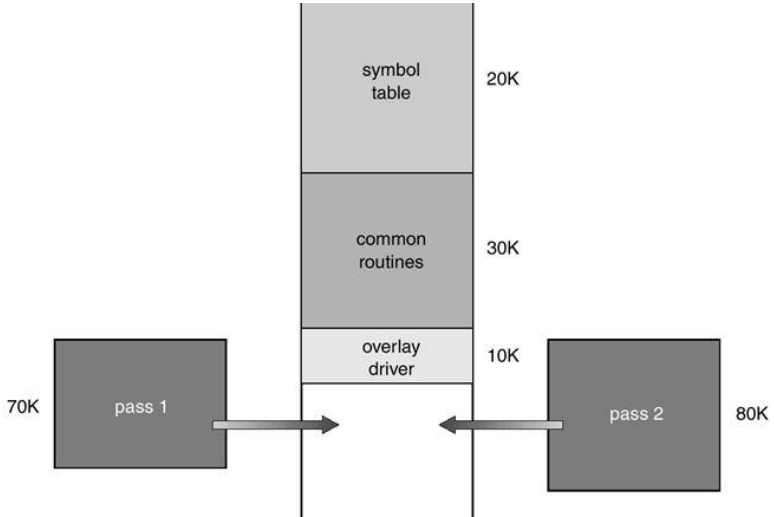
- Za izvršenje programa potrebno je najmanje 200 KB slobodne memorije (bez rezidentnih delova OS), što znači da program ne može da se izvšava na hipotetičkom sistemu od 150 KB radne memorije
- Zato se definišu 2 komponente preklapanja:

I. **Overlay A**, koji se sastoji od zajedničkih podataka i rutina, i koda za prvi deo programa, što iznosi **130 KB** (80+20+30+10 KB)
II. **Overlay B**, koji se sastoji od zajedničkih podataka i rutina, i koda za drugi deo programa, što iznosi **120 KB** (70+20+30+10 KB)

# 5.2 – Tehnika preklapanja

- Tehnika preklapanja usporava rad samog programa, jer se delovi programa učitavaju u memoriju u više iteracija,
- Overlay drajveri su uključeni u neke programske jezike (Borland Pascal, Clipper), tako da primena tehnike preklapanja ne zahteva posebnu podršku operativnog sistema.
- Konkretnu strukturu overlay delova definiše programer jer najbolje poznaje svoj program



# 5.3-Kontinualno dodeljivanje memorije

Ukoliko se koristi kontinualno dodeljivanje memorije, i logički i fizički adresni prostor procesa se sastoje od kontinualnog niza memorijskih adresa.

***Svaki proces dobija jedan kontinualni deo memorije***.

Metode kontinualnog dodeljivanja memorije su:

I. Multiprogramiranje sa fiksnim particijama

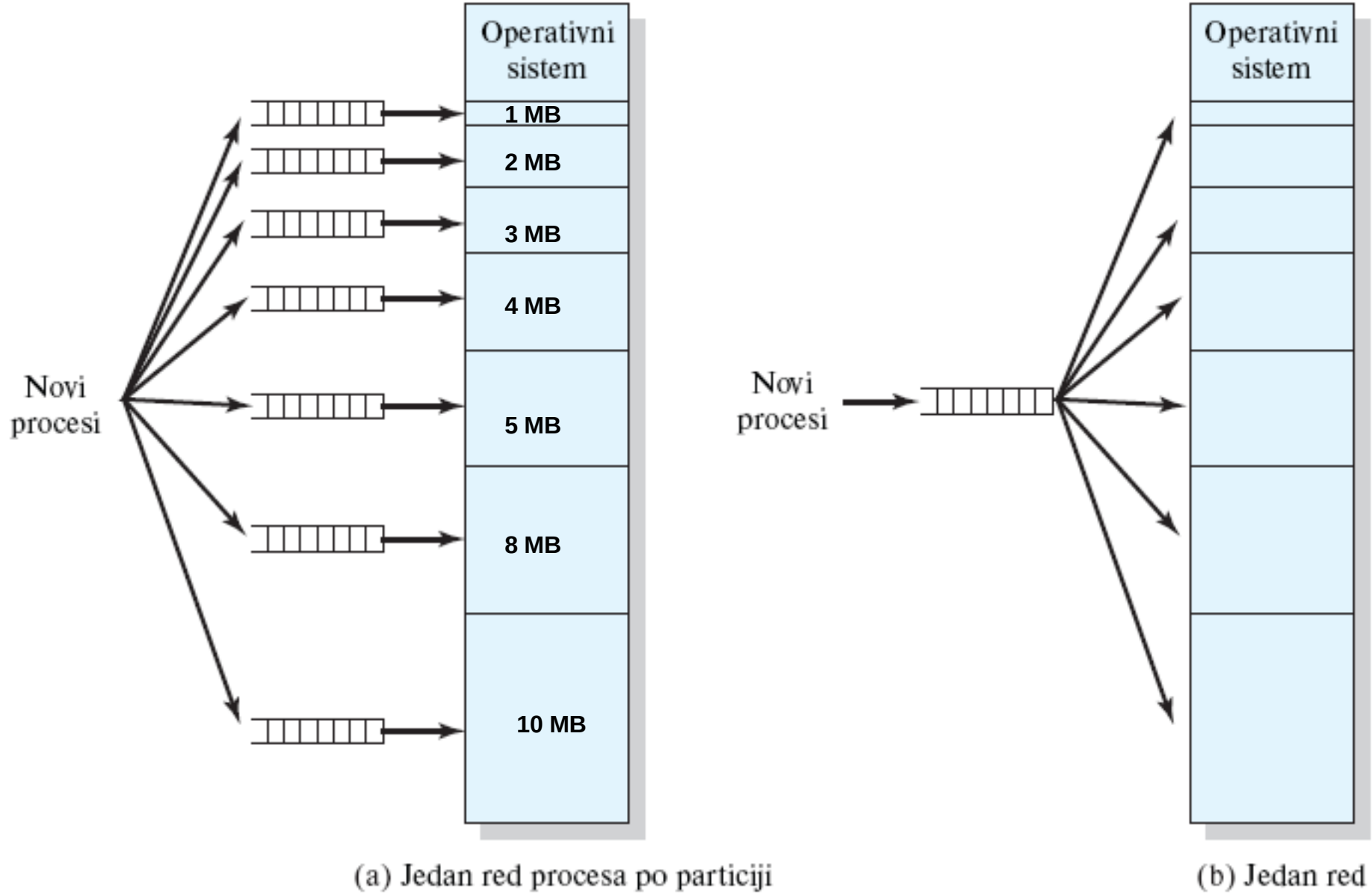
II. Multiprogramiranje sa particijama promenljive veličine

# 5.3Multiprogramiranje sa fiksnim particijama

- Jedan od najstarijih i najprostijih metoda dodeljivanja memorije jeste podela cele fizičke memorije na više delova fiksne veličine, pri čemu se u jednom delu može naći samo jedan proces.
- Stepen multiprogramiranja je jednak broju memorijskih particija. Ova metoda je korišćena u sistemu IBM OS/360.
- Svi procesi se stavljaju u red čekanja (*input queue)* koji može biti jedinstven za ceo sistem ili poseban za svaku particiju.
- Višestruki redovi čekanja obično se formiraju za opsege veličina Q=1 KB, Q=2 KB, Q=4 KB.
- Ukoliko za proces koji je došao na red nema dovoljno memorije, uzima se sledeći manji proces iz liste.
- Kada postoji više redova čekanja, veći broj malih procesa može čekati u redu za male particije, dok su velike particije neiskorišćene.
- Bolji je jedinstveni red čekanja, jer ako nema mesta u memoriji u redu čekanja za particiju koja odgovara veličini procesa, procesu se dodeljuje veća particija.

***Dva procesa ne mogu biti smeštena u jednoj particiji***.

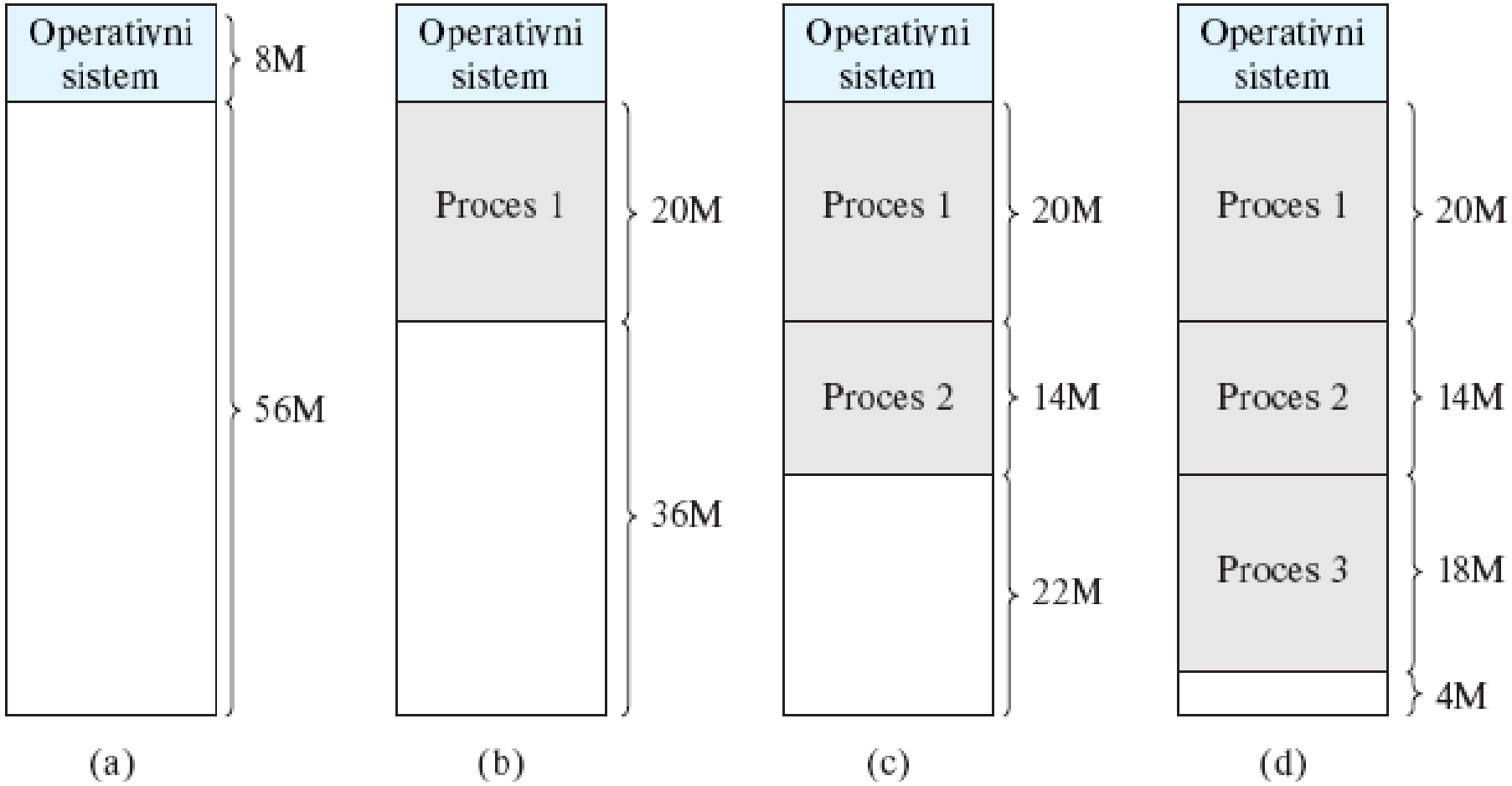
# 5.3Multiprogramiranje sa fiksnim particijama



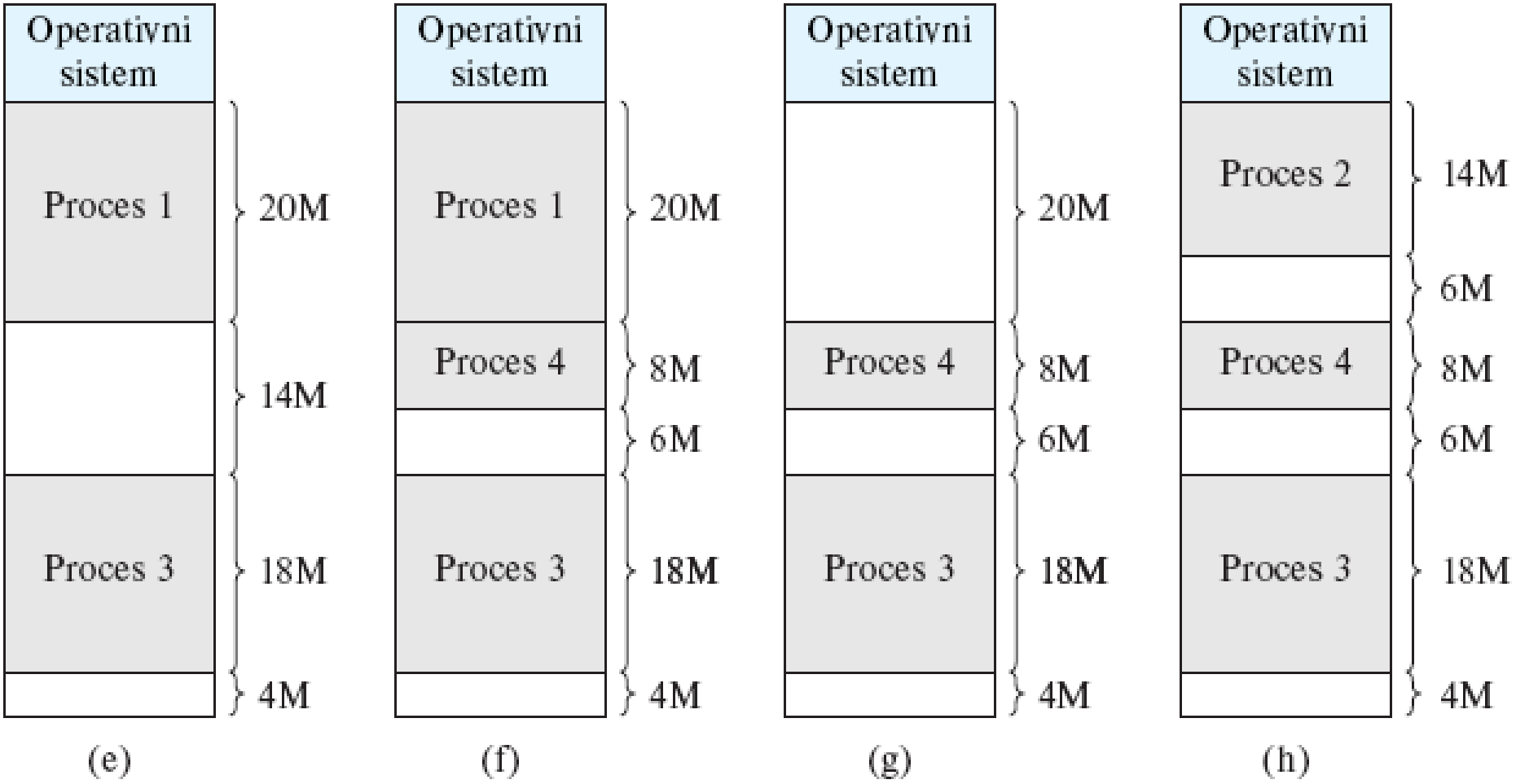

(a) Jedan red procesa po particiji

(b) Jedan red

# 5.3 Problemi koji nastaju kod multiprogramiranja sa fiksnim particijama



# 5.3 Problemi koji nastaju kod multiprogramiranja sa fiksnim particijama

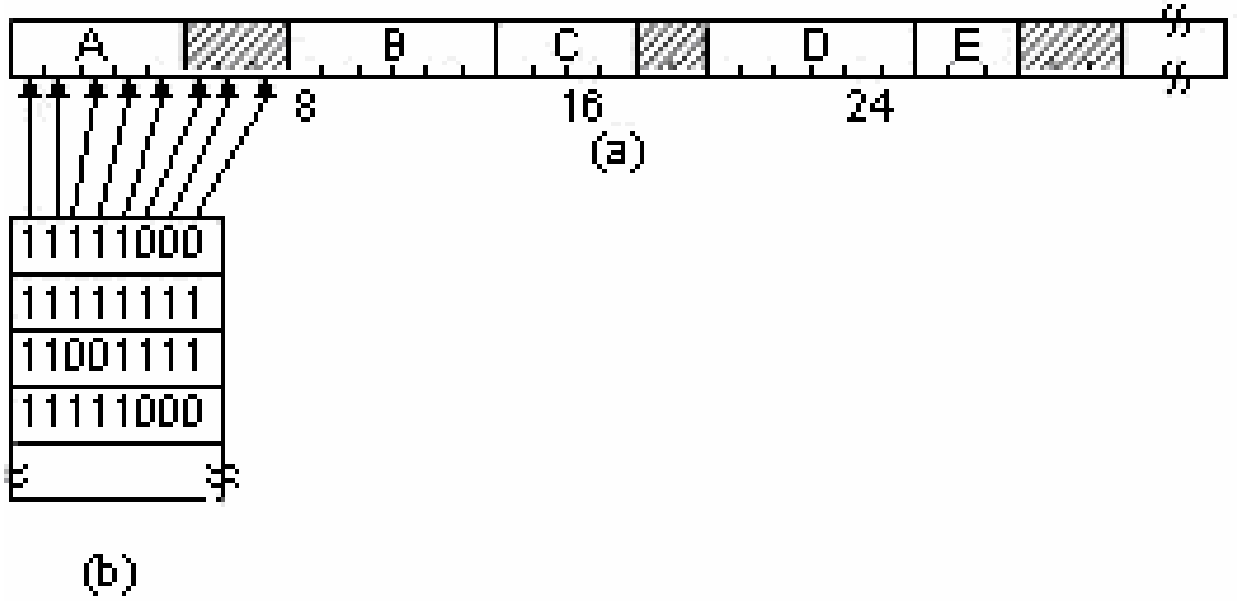


# 5.3 Particije promenljive veličine

- Umesto fiksnih particija, memorija se deli dinamički, a svaka šupljina (*hole)*, tj. slobodan kontinualni deo memorije, može se iskoristiti za smeštanje procesa – pod uslovom da je dovoljno velika.
- Šupljine dinamički nastaju i nestaju, zajedno sa procesima, mogu biti bilo gde u memoriji i imati bilo koju veličinu, što odgovara procesima na interaktivnim sistemima.
- Kada proces naiđe u sistem, traži se šupljina dovoljno velika za proces.
- Sav prostor koji proces ne zauzme od cele šupljine, predstavlja novu šupljinu u koju se može smestiti novi proces.
- Alokacija memorije je dinamička – memorija se sastoji od procesa i šupljina, a OS dinamički vodi evidenciju o zauzetosti memorije na jedan od sledećih načina:
  - **Bit mape** (*bit maps)*
  - **Povezane liste** (*linked lists)*
  - **Sistem udruženih parova** – drugova (*buddy system)*

# 5.3-Bit mape (Bit maps)

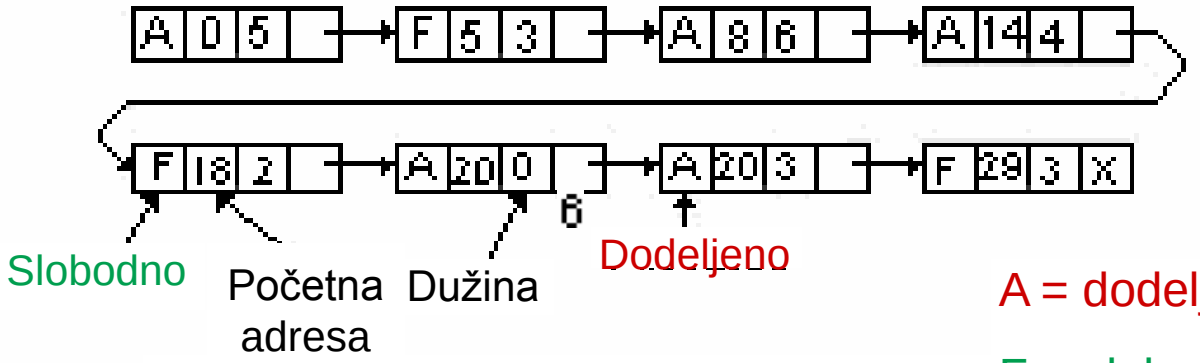
- Bitmapa je niz nula i jedinica
- Nula označava prazan blok a jedinica zauzet blok
- Blok je fiksne veličine, ali dovoljno mali da ne dolazi do interne fragmentacije
- Broj uzastopnih nula prikazuje veličinu slobodnog bloka
- Glavni problem nastaje kada je potrebno učitati *k* jedinica u memoriju jer je potrebno izvršiti pretraživanje i naći *k* uzastopnih 0.
- Ovo je spora operacija i zato se bit-mapa retko koristi



# 5.3 Povezane liste (linked lists)

- Svaki prazan blok ima veličinu i pokazivač na sledeći prazan blok
- Praćenjem pokazivača mogu se pretražiti svi prazni blokovi


a) Deo memorije




b) Odgovarajuća ulančana lista

A = dodeljena memorija

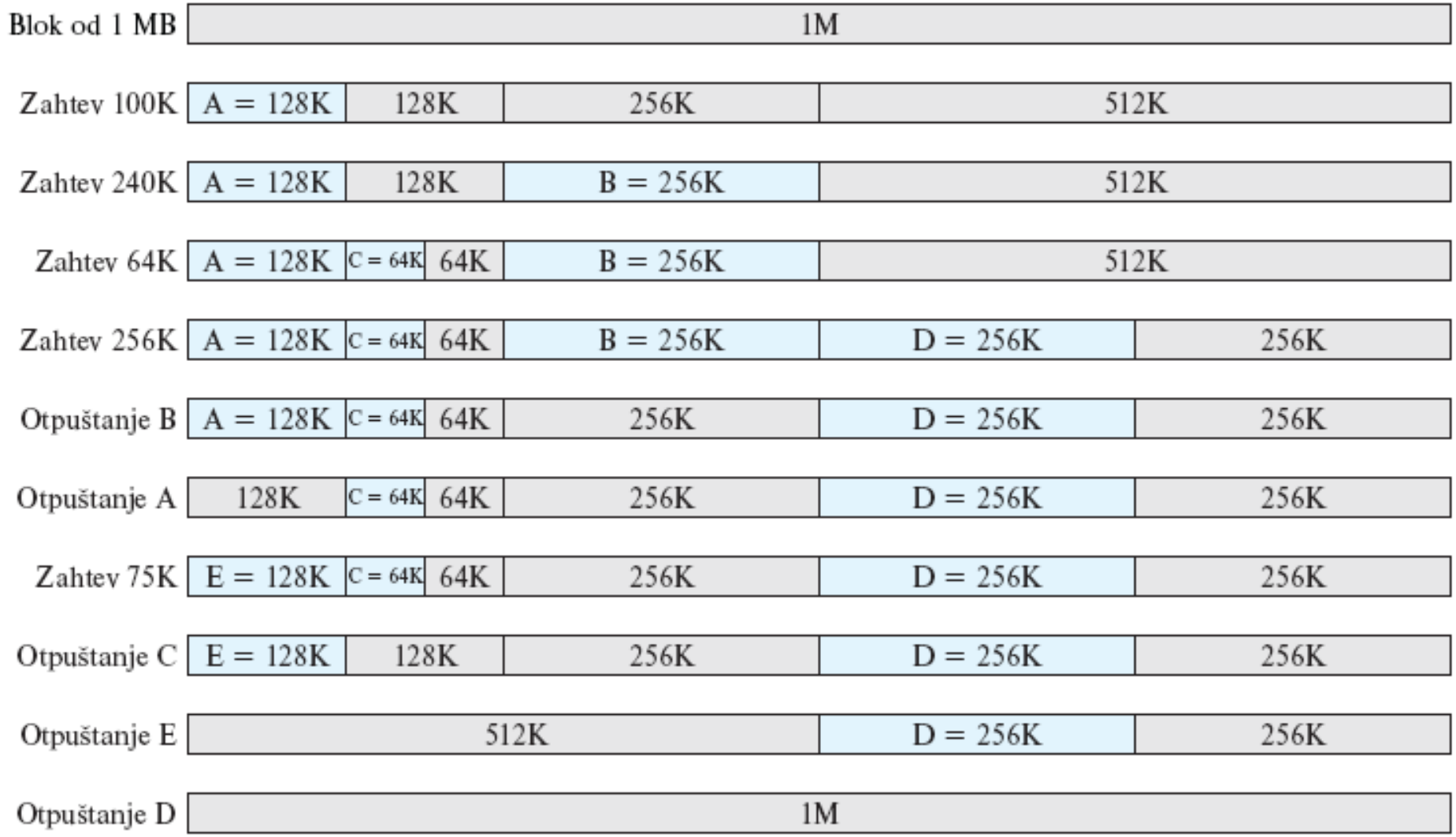
F = slobodna memorija

- I fiksno i dinamičko deljenje na particije ima svoje nedostatke.
- Fiksno deljenje ograničava broj aktivnih procesa i može neefikasno da koristi prostor ako je slaba podudarnost između veličina raspoloživih particija i procesa.
- Dinamičko deljenje na particije je složenije za održavanje i obuhvata dodatno opterećenje zbog sažimanja.

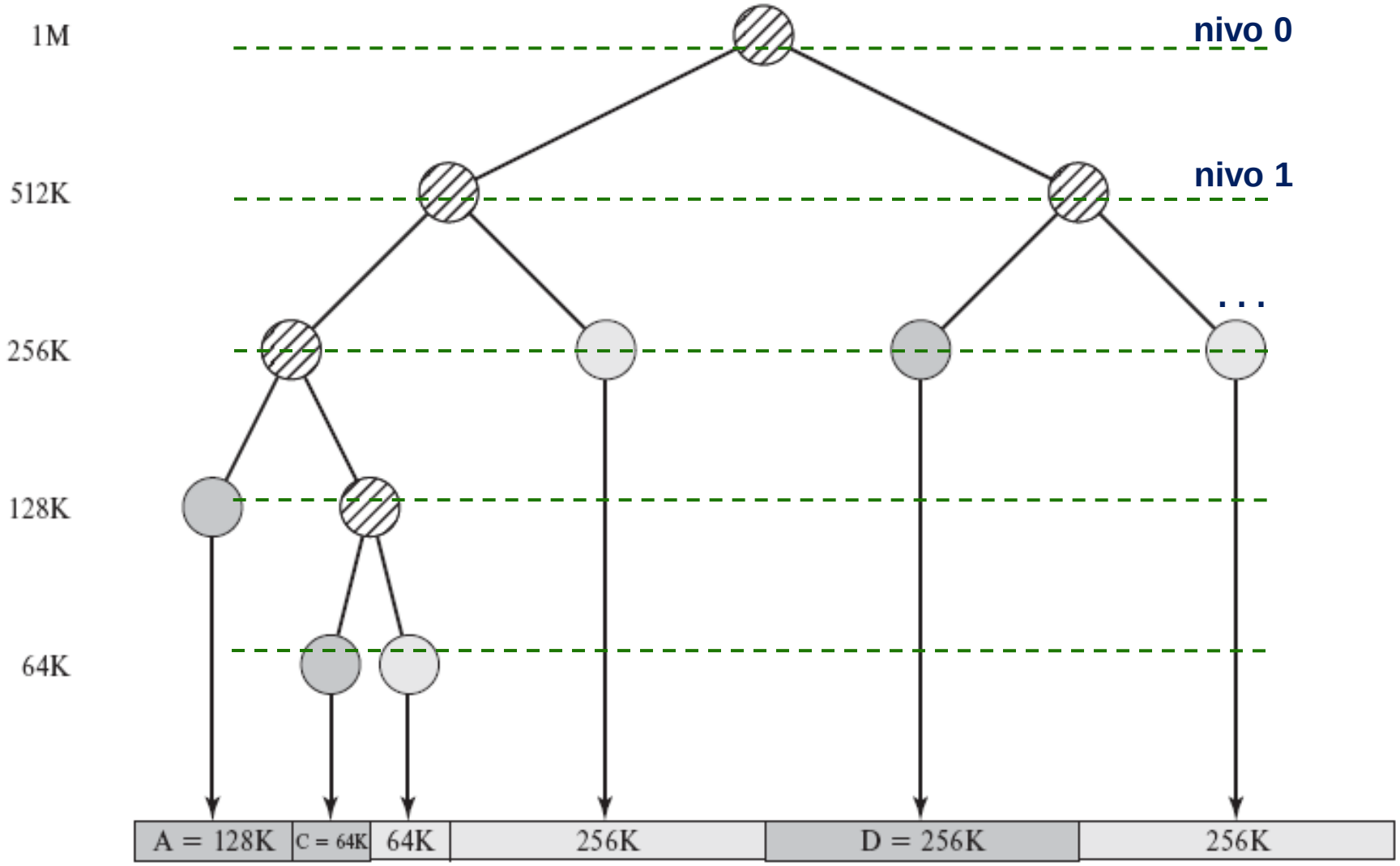
# 5.3 Sistem udruženih parova-drugova (Buddy system)

- Kompromisno rešenje
- Ceo memorijski prostor se tretira kao jedinstven blok veličine $2^n$.
- Na početku cela memorija je prazna, a kada se procesu dodeli prazan prostor u memoriji on mora da bude stepen 2
- Procesi se smeštaju u prvu šupljinu koja to omogućava
- Ako je zahtev za delom memorije takav da je $2^{n-1} <$ **zahtev** $<= 2^n$, dodeljuje se ceo blok.
- U suprotnom, blok se deli na dva identična - partnerska dela (***buddies***).
- Algoritam se nastavlja sve dok se ne generiše najmanji blok koji je veći ili jednak zahtevu.
- Dve susedne šupljine koje imaju istu veličinu se udružuju u veću

# 5.3 Sistem udruženih parova-drugova (Buddy system)



# 5.3 Sistem udruženih parova-drugova (Buddy system)



# 5.3-Algoritmi za izbor prazne particije

- Prvi koji zadovoljava (*first-fit*): Procesu se dodeljuje prvi segment koji zadovoljava postavljene memorijske zahteve. Obično pretraživanje počinje od početka liste slobodnih segmenata ili se nastavlja od mesta gde je prethodno ispitivanje zaustavljeno.
- Najbolje poklapanje (*best-fit*): Procesu se dodeljuje onaj segment koji najbolje odgovara njegovim memorijskim zahtevima. Iako se, na prvi pogled, ovim pristupom najbolje iskorišćava slobodna memorija, rezultat je stvaranje malih segmenata, šupljina, koji su posledica razlike u veličinama segmenta i programa a koje su neupotrebljive. Dobra strana: mala fragmentacija a loša strana: često sažimanje memorije
- Najlošije poklapanje (*worst-fit*): Procesu se dodeljuje najveći slobodni segment. Ovaj algoritam ima za cilj stvaranje što većih šupljina, suprotno prethodnom algoritmu.

***Simulacije pokazuju da su algoritmi first-fit i best-fit nešto bolji po performansama i iskorišćenosti memorije od worst-fit***

# 5.3-Fragmentacija memorije

Prilikom dodele memorije javljaju se dve vrste fragmentacije:

1. Unutrašnja (interna) fragmentacija – kada se dodeli memorijska particija veća od memorije koju zahteva proces
2. Spoljašnja (eksterna) fragmentacija – memorijski prostor nije kontinualan pa ponekad ne može da zadovolji zahteve procesa za veličinom memorije iako ona postoji samo što je diskontinualna

***Unutrašnju fragmentaciju nije moguće rešiti za razliku od spoljašnje koja se rešava defragmentacijom memorijskog prostora***

- Defragmentacija nije uvek moguća – primer ako je relokacija programa statička u fazi prevođenja i učitavanja tako da se ona izvodi samo ako je relokacija programa dinamička
- Defragmentacija degradira performanse sistema jer zahteva da sistem prekine izvršavanja tekućih procesa, da ih privremeno prebacuje na disk i da im odredi nove adrese izvršavanja nakon vraćanja u OM

# 5.4–Diskontinualno dodeljivanje memorije

Koriste se sledeće metode:

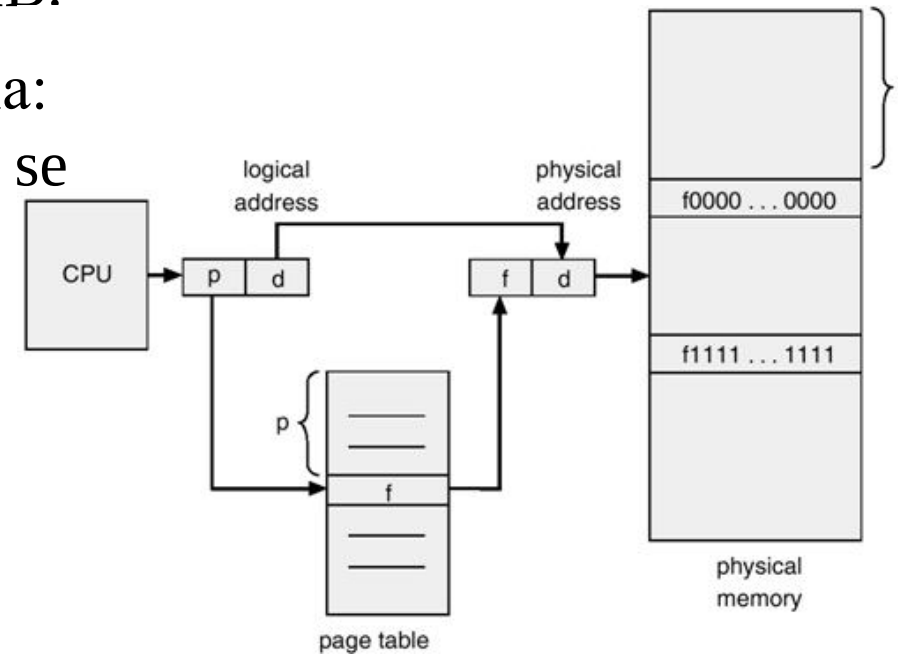
1. **straničenje (*paging)***
2. **segmentacija (*segmentation)***
3. **straničenje sa segmentacijom**
4. **segmentacija sa straničenjem**

# 5.4 - Straničenje

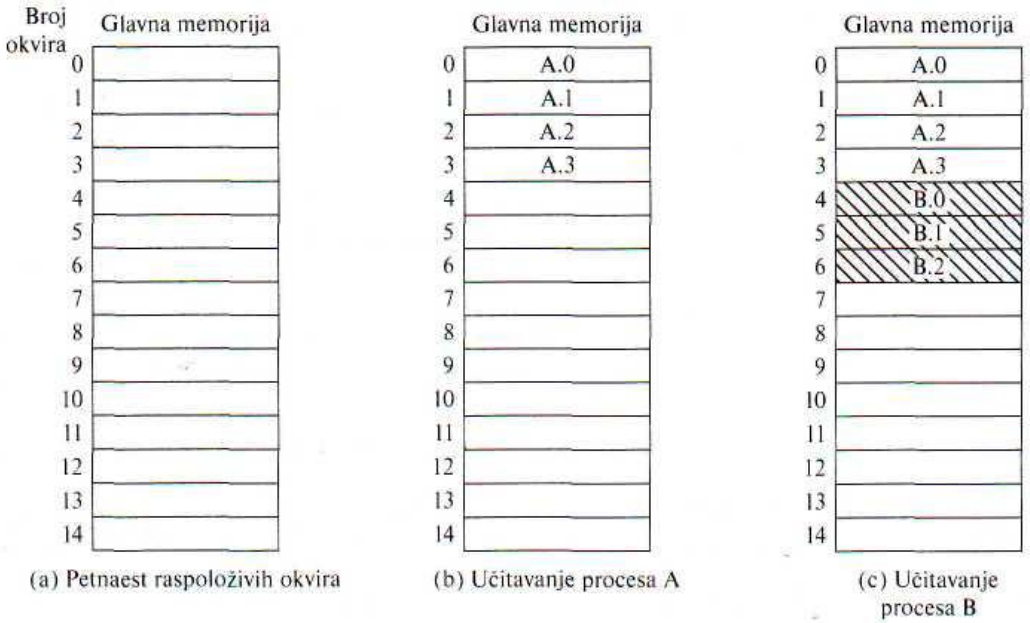
- Straničenje je metoda sa hardverskom podrškom na nivou procesora koja se koristi u svim OS i na svim računarskim arhitekturama.
- Fizička memorija, tj. fizički adresni prostor, izdeli se na blokove fiksne veličine, koji se nazivaju **fizičke stranice ili okviri (*page frames*).**
- Logički adresni prostor takođe se izdeli na blokove istih veličina koji se nazivaju **logičke stranice (*pages*).**
- Veličine stranica su po pravilu stepen broja 2, najčešće u opsegu od 512B do 8KB, a nekada i veće, 16 MB.

Svaka logička adresa deli se na dva dela:

1. Broj stranice (*page number*) – **p** koji se koristi kao indeks u tabeli stranica koja sadrži baznu adresu
2. Pomeraj unutar stranice (*offset*) – **d** definiše položaj u samoj stranici i u kombinaciji sa baznom adresom definiše stvarnu fizičku adresu



# 5.4 - Straničenje




(a) Petnaest raspoloživih okvira

(b) Učitavanje procesa A

(c) Učitavanje procesa B

(d) Učitavanje procesa C

(e) Zamena procesa B

(f) Učitavanje procesa D

# 5.4 - Straničenje

- Metoda straničenja funkcioniše na sledeći način: svakoj logičkoj stranici odgovara jedna fizička, a korespodencija između njih se čuva u tabeli stranica (*page table).*
- Bilo koja stranica može da se smesti u bilo koji okvir fizičke memorije.
- To omogućava da se kontinualni logički prostor procesa razbaca svuda po memoriji.
- Omogućava da se u memoriji istovremeno mogu izvršavati znatno veći broj programa nego što je fizička veličina memorije
- Svako straničenje predstavlja dinamičku relokaciju, a tabela stranica predstavlja relokacioni registar za svaki okvir fizičke memorije.
- Smanjenje veličine stranica dovodi do povećanja njihove tabele, jer je svaka stranica opisana jednim zapisom u tabeli. Kao posledica, povećava se kašnjenje prilikom mapiranja ili pretraživanja.
- Ako je veličina logičkog adresnog prostora $\mathbf{2^m}$, a veličina stranice $\mathbf{2^n}$, tada viši deo adrese dužine **m-n** definiše broj stranice, dok najnižih **n** bitova adrese predstavljaju pomeraj unutar stranice.

# 5.4 - Straničenje

**Primer:**

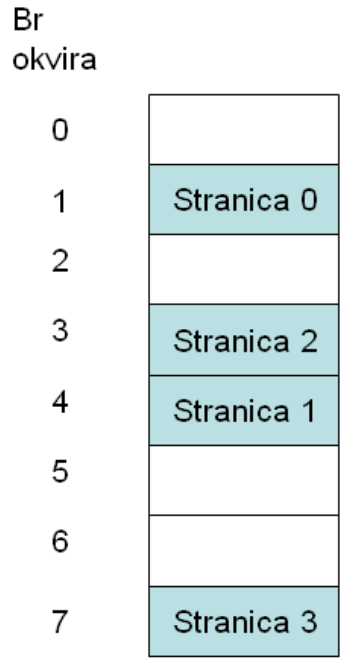
*Zamislimo da imamo memoriju veličine* 32 B ($2^6$). *Definišemo* 8 ($2^2$) *okvira veličine* 4 B, *što znači da je* $m=6$, *a* $n=2$. *Uzmimo korisnički proces koji zauzima* 4 *logičke stranice, sa logičkim adresama*: *stranica* 0 (0-3), *stranica* 1 (4-7), *stranica* 2 (8-11), *stranica* 3 (12-15)

- Logička adresa 0 ima broj logičke stranice 0.
- Pomoću broja logičke stranice ulazimo u tabelu stranica i u njoj nalazimo da je ona smeštena u okvir broj 1, i tako redom:
  - ✓ logička stranica 1 u okvir 4,
  - ✓ logička stranica 2 u okvir 3
  - ✓ logička stranica 3 u okvir 7.

Logički adresni prostor

| p | f |
|---|---|
| 0 | 1 |
| 1 | 4 |
| 2 | 3 |
| 3 | 7 |

Tabela stranica

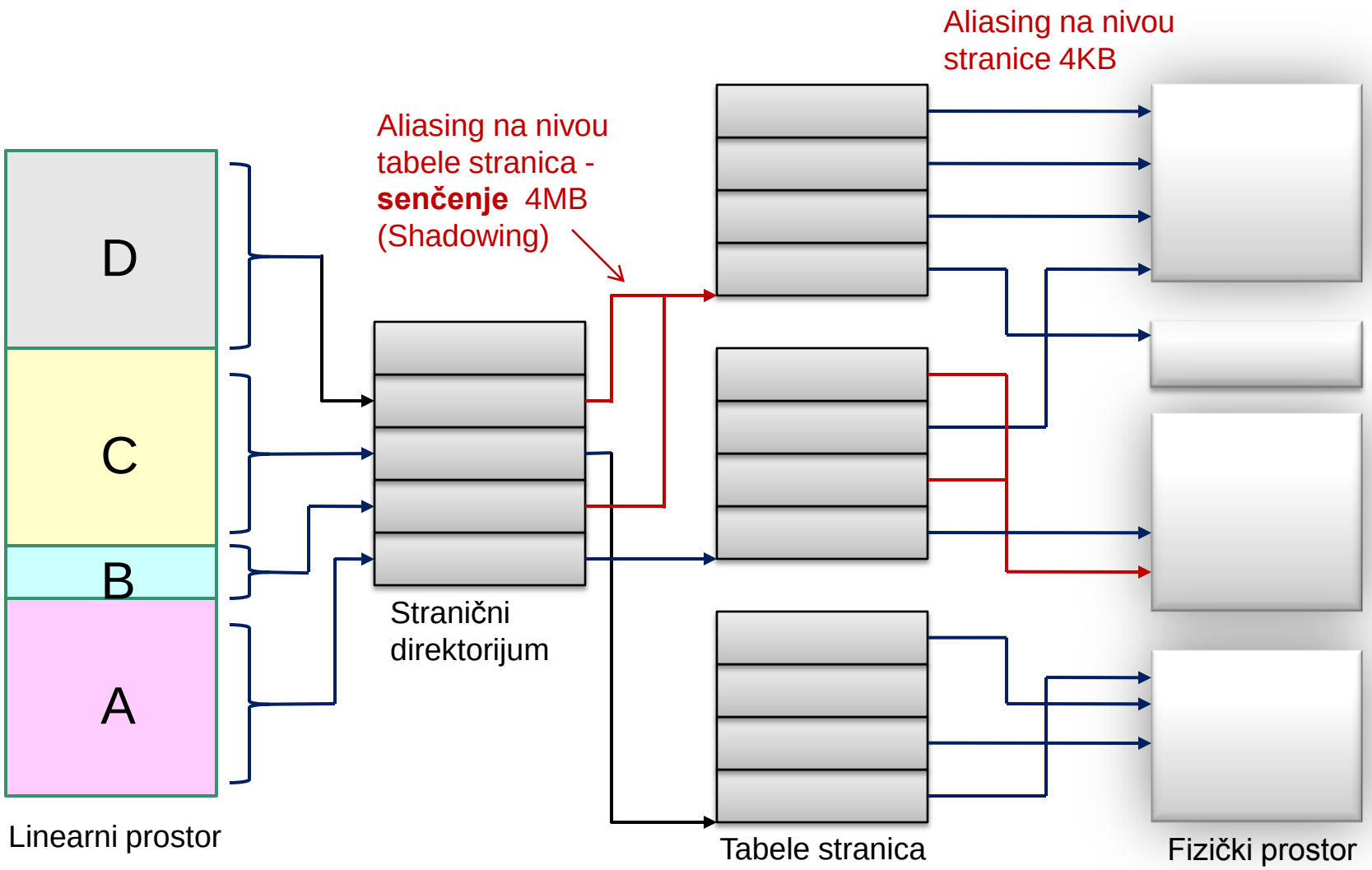


Fizički adresni prostor

# 5.4 - Straničenje

- Glavna motivacija za upotrebu manjih stranica je ušteda u memoriji izbegavanjem interne fragmentacije (prosečna interna fragmentacija iznosi 1,5 veličina stranice).
- Ovo se može zanemariti kod računara sa stotinama megabajta memorije i veličinom stranice od 4 KB.
- Veličina tabele stranica je obrnuto proporcionalna veličini stranice.
- Zbog efikasnijeg pretraživanja tabela, stranično adresiranje vrši se hijerarhijski, u dva nivoa, upotrebom direktorijuma tabela stranica i samih tabela stranica.
- Cena za ovo je dvostruki pristup memoriji radi izračunavanja adrese, ali i to se rešava adekvatnim keširanjem.
- Dobra stvar je da direktorijum staje u jednu stranicu.
- Isto važi i za svaku od tabela stranica, što znatno pojednostavljuje manipulaciju.

# 5.4 - Straničenje



# 5.5 - Segmentiranje

- Prethodno razmatrani slučajevi odnose se na korisničke procese, tj programe, koji zahtevaju kontinualan memorijski prostor.
- Međutim, sama struktura programa nije kontinualna, jer programi se logički dele na više nezavisnih celina.
- Sam kôd se sastoji od više celina: tabele, polja, stek, promenljive, itd.
- Svaki od ovih logičkih segmenata dobija ime pomoću koga se referencira i može nezavisno da se učita u memoriji.
- Po pravilu, programer definiše logičke segmente u izvornom programu, a prevodilac na osnovu toga pravi memorijske segmente.

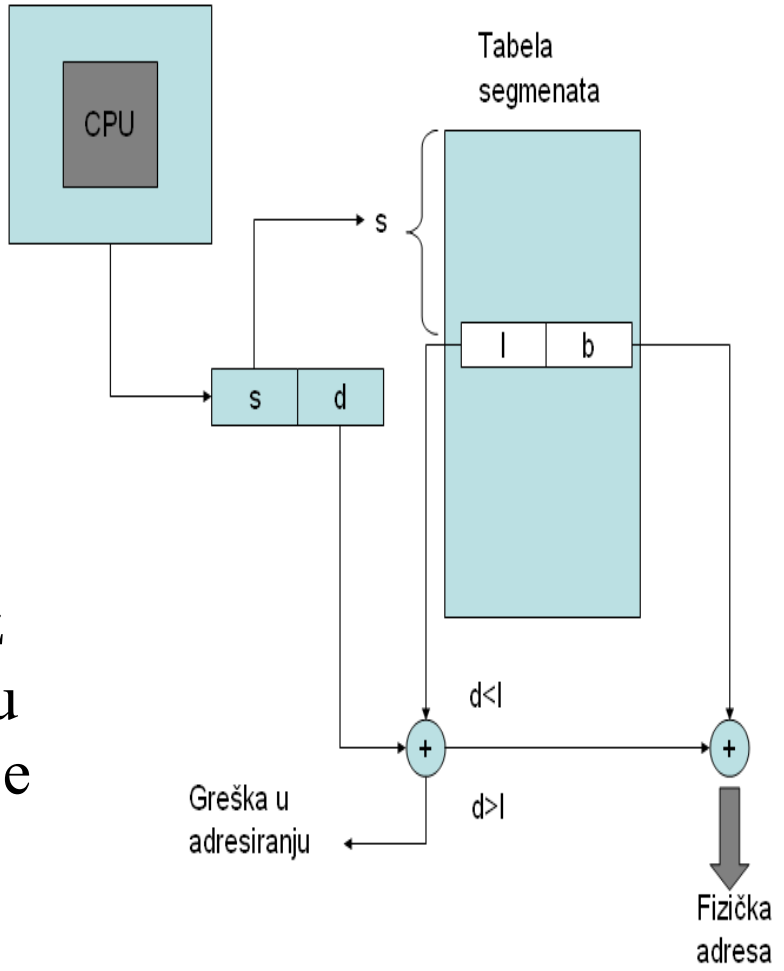
***Segmentacija je metoda upravljanja memorijom koja podržava logički korisnički pogled na memoriju.***

- Logički adresni prostor sastoji se od kolekcije segmenata, a svaki segment ima jedinstveno ime i dužinu.
- Logička adresa se sastoji od:
  1. imena segmenta (obično se zadaje broj kao identifikator segmenta)
  2. pomeraja unutar segmenta

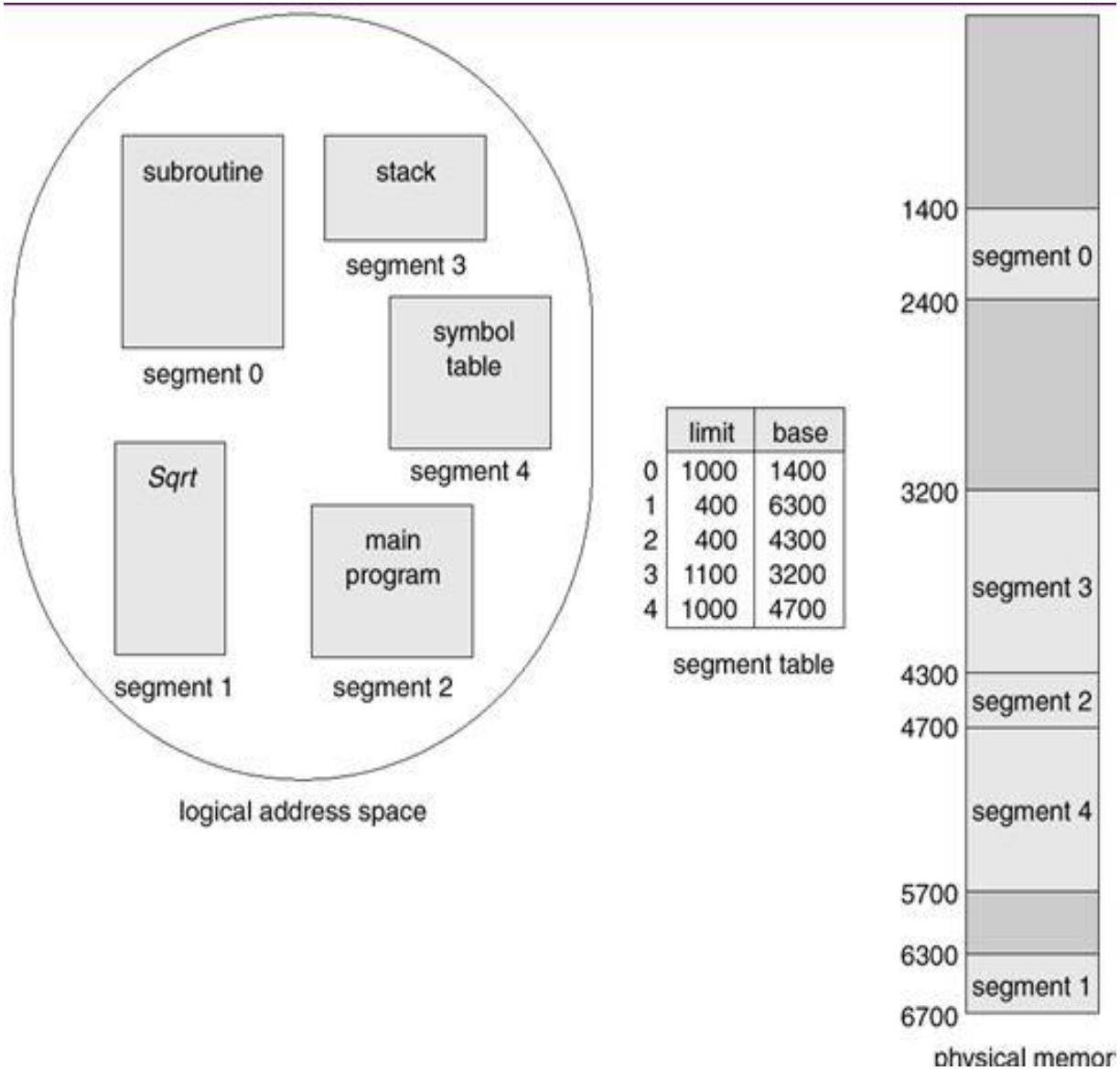
# 5.5 - Segmentiranje

***Hardverska podrška***

- U metodi segmentacije, korisničke adrese su dvodimenzionalne, ali se moraju translirati u jednodimenzionalne fizičke adrese.
- Ovo mapiranje se obavlja preko tabele segmenata a ima podršku i u hardveru CPU
- Svaki ulaz u tabeli segmenata opisuje jedan segment, a sadrži:
  1. baznu adresu segmenta, koja definiše početnu fizičku adresu segmenta u memoriji
  2. ograničenje segmenta, koje definiše dužinu segmenta
- Broj segmenta **s**, koristi se kao ulaz u tabelu segmenata, iz koje se čitaju bazna adresa segmenta i ograničenje
- Pomeraj **d** svakako mora biti manji od ograničenja segmenta



# 5.5 - Segmentiranje



# 5.5 – Segmentacija sa straničenjem

- Najpopularnije serije procesora Intel (80x86 i Pentium) i Motorola (68000) imaju ugrađenu podršku i za segmentaciju i za straničenje, tako da omogućavaju primenu kombinovanih metoda diskontinualne alokacije pa se procesi mogu deliti na fizički diskontinualne logičke celine.
- Pri tome, straničenje poništava eksternu fragmentaciju segmenata.
- Procesor Intel 80386 koristi segmentacije sa straničenjem.
- Logička adresa se sastoji od identifikatora segmenta (*selector) i* pomeraja u okviru segmenta *(offest).*
- Iz tabele segmenata čita se adresa logičke stranice u katalogu stranica.

Straničenje je realizovano u dva nivoa:

1. spoljna tabela se zove katalog stranica (*page directory)*
2. unutrašnja tabela - tabela stranica

# Hvala na pažnji !!!

Pitanja

? ? ?