

Upravljanje memorijom

Nemanja Maček

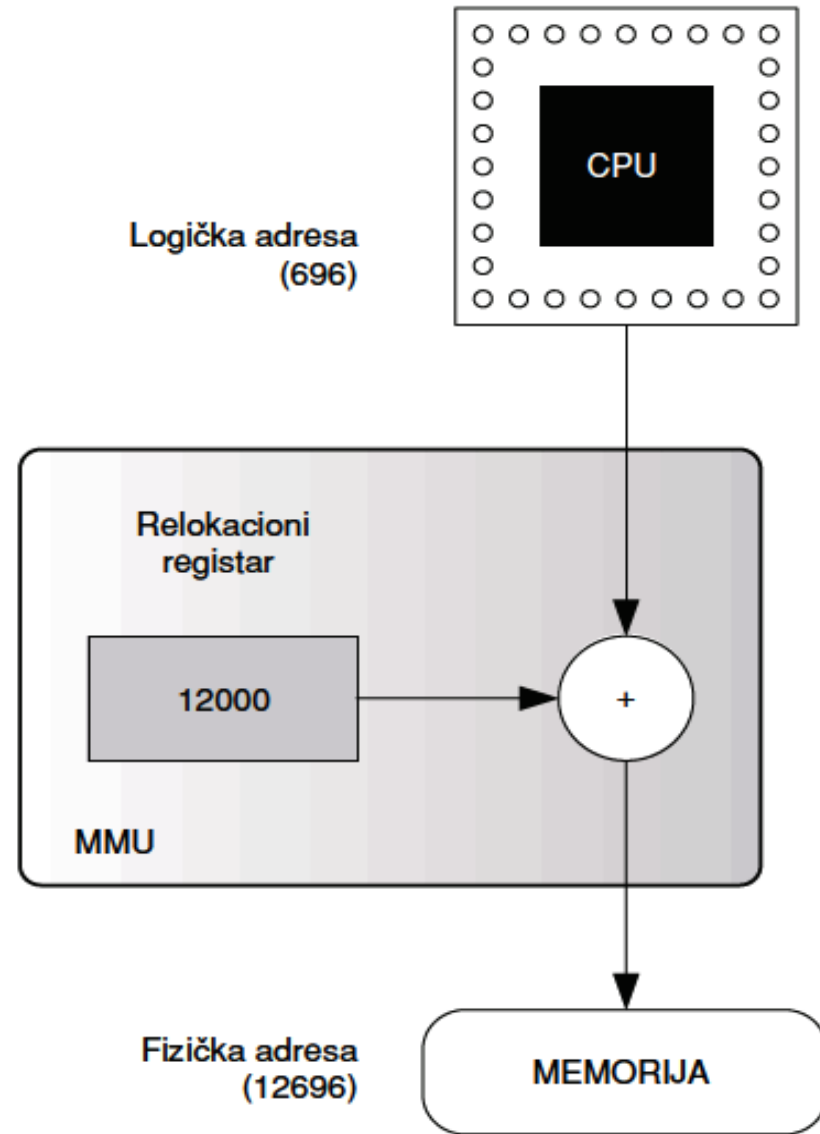
- Uvodne napomene
- Programerske tehnike upravljanja memorijom
- Kontinualna alokacija memorije
- Straničenje
- Segmentacija

- Memorija je niz memorijskih reči od kojih svaka ima jedinstvenu adresu.
- Prilikom izvršavanja procesa:
 - Procesor na osnovu vrednosti programskog brojača čita instrukcije iz memorije.
 - Instrukcije u toku izvršenja mogu zahtevati:
 - čitanje podataka sa drugih lokacija,
 - upis podataka na druge memorijske lokacije.
- Pitanja:
 - Da li proces bez memorije može nešto da uradi?
 - Da li može da se izvrši?
 - Da li ima logike izvršavati proces sa diska?
- Odgovori na ova pitanja kažu da je fizička memorija pored procesora fundamentalni deo računarskog sistema!

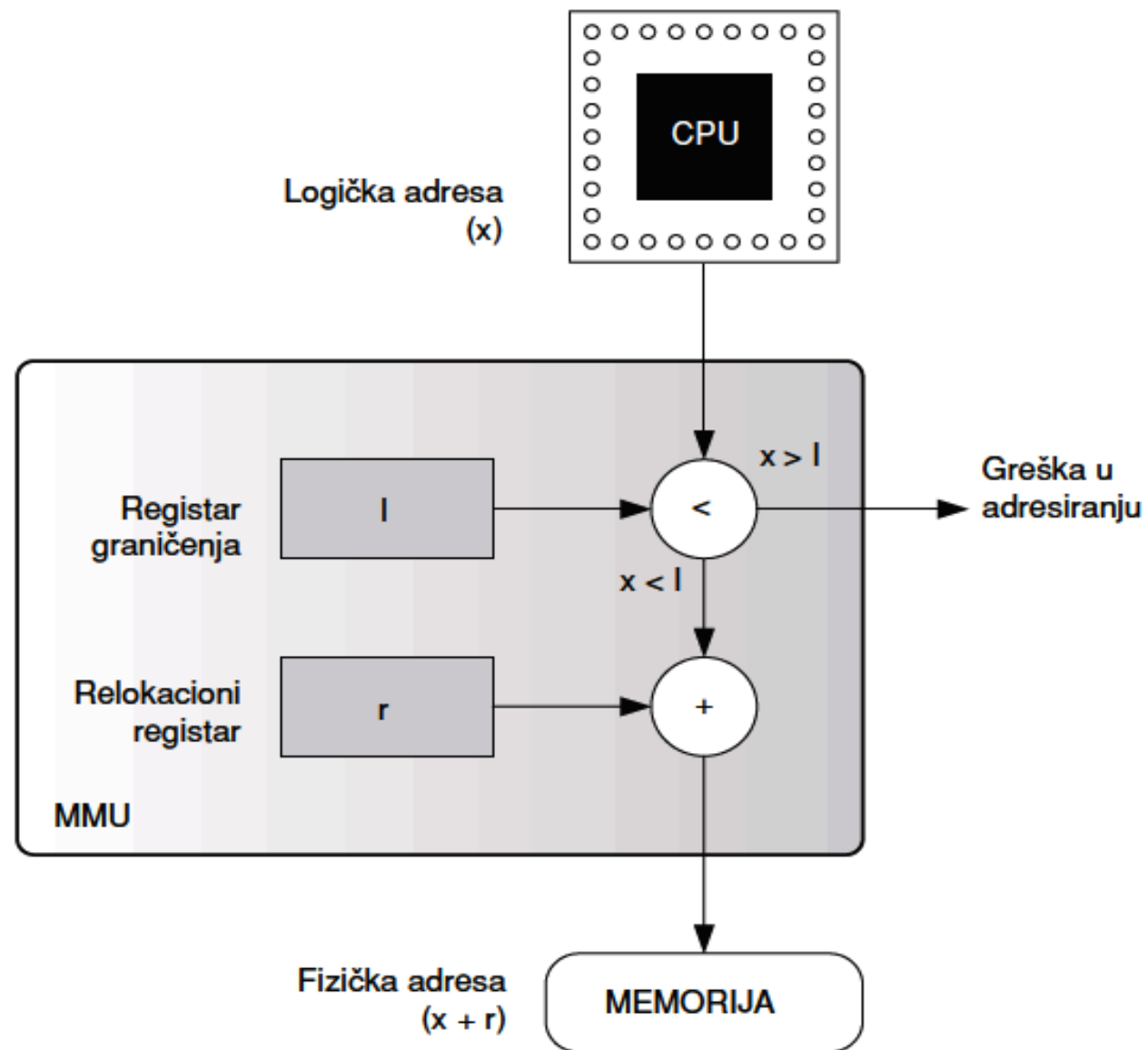
- Sloja za upravljanje memorijom (engl. *memory manager*):
 - vodi računa o korišćenju memorije,
 - dodeljuje memoriju procesima kad je zatraže,
 - oslobađa memoriju od procesa kad završe svoje aktivnosti,
 - vrši razmenu između memorije i diska (kada glavna memorija nije dovoljno velika da sadrži sve procese).
- *Memory manager* takođe:
 - razdvaja fizički i logički adresni prostor,
 - prevodi relativne (relokatibilne) adrese u fiksne (vezivanje adresa),
 - obavlja relokaciju (kompakciju, odnosno defragmentaciju operativne memorije), itd.

- Program se nalazi na disku kao izvršna binarna datoteka.
- Program se sa diska učitava u memoriju unutar adresnog prostora novokreiranog procesa.
- Veći broj procesa deli operativnu memoriju računara.
 - Programer **ne može unapred odrediti fiksne** memorijske lokacije za smeštaj programa.
 - Zato koristi **simboličke adrese**.
- Kako se simboličke adrese pretvaraju u apsolutne?
 - U izvornom kodu programa imamo **simboličku promenljivu** `count`.
 - Kompajler prevodi ovu simboličku adresu u **relativnu** (relokatibilnu).
 - Promenljiva `count` se npr. vezuje na lokaciju na adresi 14 **u odnosu na početak modula**.
 - OS prevodi relativne adrese u **fiksne** prilikom učitavanja programa u memoriju.
 - Punilac pretvara relokatibilnu adresu u apsolutnu adresu (npr. 74014).
- **Vezivanje adresa** (engl. *binding*) je mapiranje iz jednog adresnog prostora u drugi.

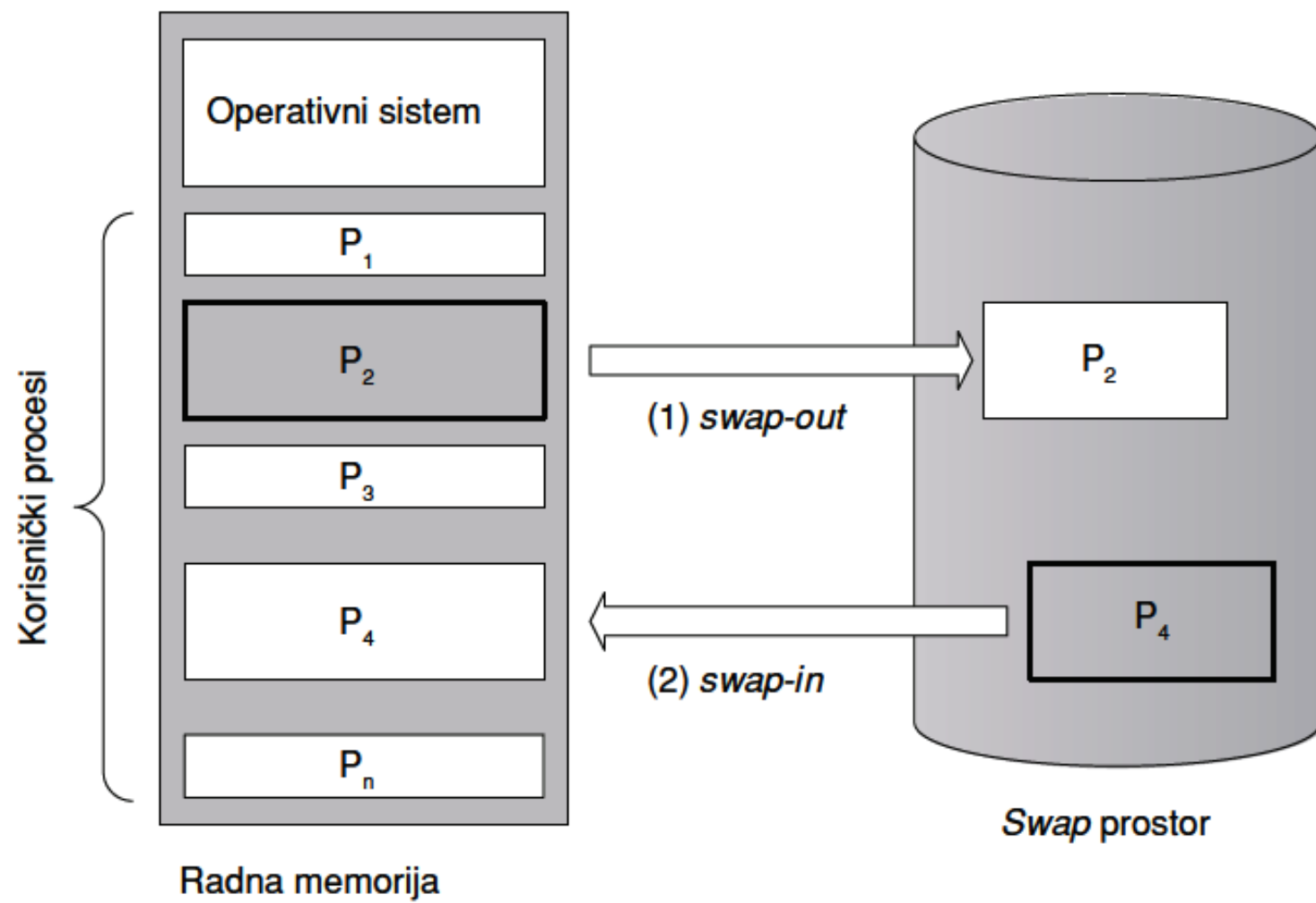
- **Logička adresa** je adresa koju generiše procesorka instrukcija.
 - U fazi izvršavanja programa naziva se **virtuelna adresa**.
- **Fizička adresa** je adresa same memorijske jedinice.
- **Logički (virtualni) adresni prostor** je skup svih logičkih adresa koje generiše program.
- **Fizički adresni prostor** je skup svih fizičkih adresa koje njima odgovaraju.
- Jedinica za upravljanje memorijom – **MMU** (engl. *Memory Management Unit*) je hardverski uređaj koji mapira virtuelni adresni prostor u fizički.
- Najprostija MMU šema: **relokacioni registar**.
 - Relokacioni registar definiše adresu **fizičkog početka programa**.
 - Fizička adresa = logička adresa koju generiše program + vrednost relokacionog registra.
 - Logički adresni prostor koji se nalazi u opsegu **[0, max]** se mapira u opseg **[R+0, R+max]**.
 - R je vrednost relokacionog registra (fizička adresa početka programa).
 - Vrednost max je gornja granica.



- Zaštita memorije obuhvata:
 - zaštitu OS od korisničkih procesa,
 - međusobnu zaštitu korisničkih procesa po pitanju pristupa memorijskim sekcijama.
- Dva registra:
 - **relokacioni registar** (sadrži najnižu adresu procesa),
 - **limit registar** (sadrži najveći opseg logičkih adresa procesa).
- MMU mapira svaku logičku adresu procesa dinamički:
 - Proverava da li je logička adresa manja od vrednosti limit registra.
 - Ako jeste, dodaje vrednost relokacionog registra.
- Relokacioni i limit registar su dva registra procesora.
 - Pune se onda kada proces dobija procesor na izvršenje.
 - Dispečer čita vrednosti ova dva registra iz konteksta procesa i postavlja ih prilikom aktiviranja procesa.



- Proces se mora prilikom izvršavanja nalaziti u operativnoj memoriji.
- Proces se može **privremeno prebaciti iz memorije na disk**, kako bi se oslobodila memorija.
 - Oslobodjena memorija puni se drugim procesom.
 - Posle izvesnog vremena, proces se može **vratiti sa diska u memoriju** kako bi nastavio izvršavanje.
- Koristi se u **prioritetnim šemama za raspoređivanje procesa**.
 - Procesi visokog prioriteta se čuvaju u memoriji.
 - Procesi niskog prioriteta se upisuju na disk i čekaju da se oslobodi memorija.
- Najveći deo vremena u razmeni otpada na transfer podataka između memorije i diska.
 - Trajanje jedne razmene zavisi od:
 - količine podataka za transfer,
 - tipa i karakteristika sekundarnih memorija i pratećeg hardvera.
 - To vreme je znatno veće u odnosu na performanse savremenih procesora!
 - Ne preporučuje se često korišćenje tehnike razmenjivanja!



Dinamičko punjenje memorije programom

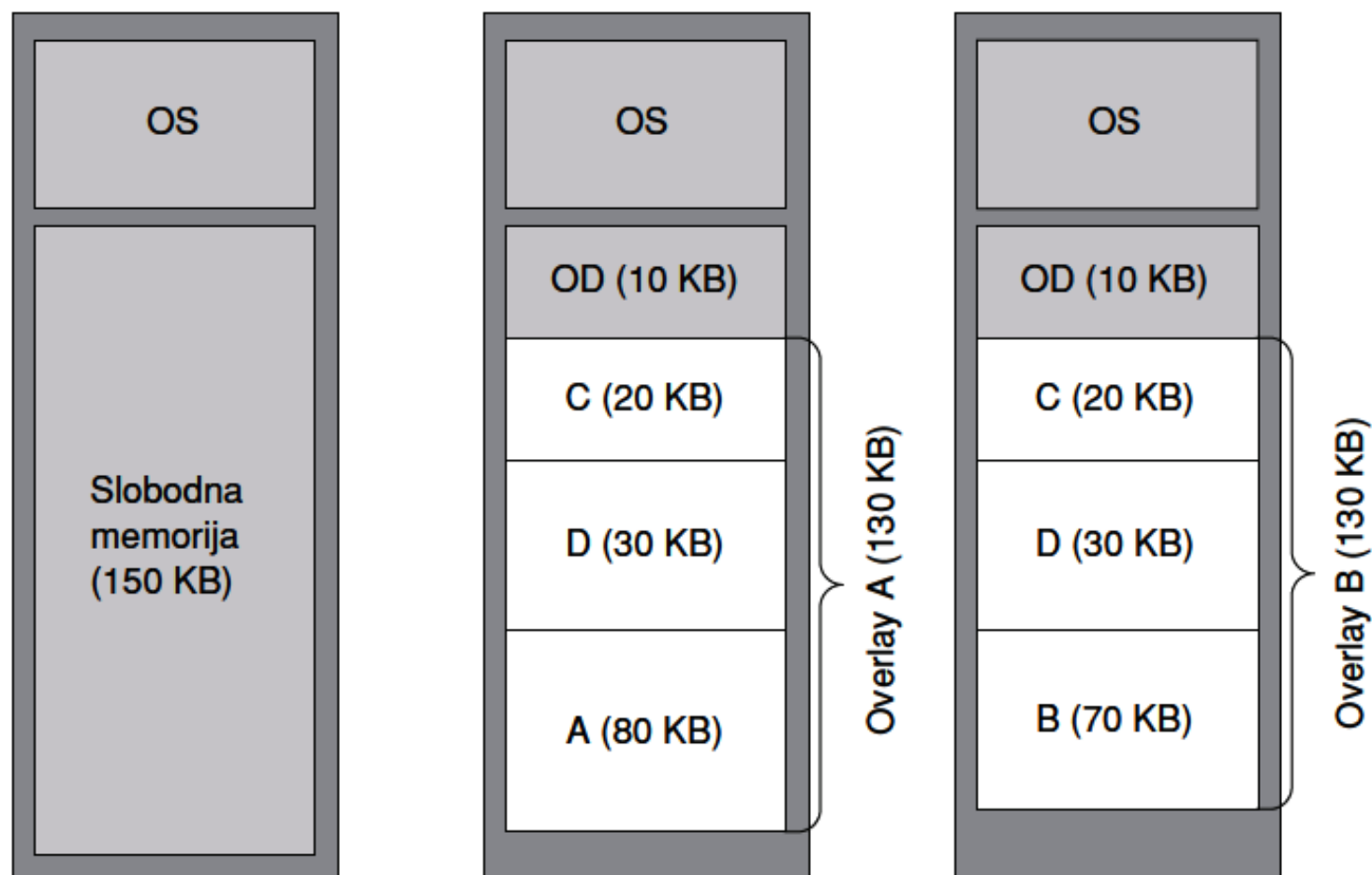
- Suština **dinamičkog punjenja** (engl. *dynamic loading*):
 - U memoriju se smeštaju samo neophodni delovi programa.
 - Odgovarajuće rutine se učitavaju u memoriju samo kada ih program pozove.
- Da bi to bilo moguće, sve rutine programa se čuvaju na disku u **relokatibilnom formatu**.
- Kada se rutina pozove iz programa:
 - proverava se da li je ona već u memoriji,
 - ako nije, poziva se punilac da je učitava u memoriju.
- Prednosti:
 - Rutine koje nisu trenutno potrebne ne zauzimaju mesto u memoriji.
 - Zgodno je za velike programe.
 - Ne zahteva specijalnu podršku od operativnog sistema.
 - Programer mora sam da projektuje svoje programe da koriste principe dinamičkog punjenja.
 - OS može pomoći programeru tako što obezbeđuje biblioteku za dinamičko punjenje.

Tehnika preklapanja (engl. *overlay*)

- Primer: program čine dve relativno nezavisne komponente koje se izvršavaju jedna za drugom.
- Programer može da podeliti izvorni kod programa na dva dela.
- Veličine programskih komponenti:
 - prvi deo (80KB),
 - drugi deo (70KB),
 - zajednički podaci (20KB),
 - zajedničke rutine (30KB).
- Za izvršavanje programa potrebno je **najmanje 200KB slobodne memorije**.
 - To znači da program ne može da se izvršava na hipotetičkom sistemu sa 150KB memorije.
- Pošto su komponente nezavisne:
 - Prilikom izvršavanja prvog dela programa, kod drugog dela ne mora biti učitani u memoriju!
 - Drugi deo programa se može izvršavati ukoliko prvi deo nije prisutan u memoriji.

Tehnika preklapanja (engl. *overlay*)

- Možemo da definišemo dve *overlay* komponente:
 - **Overlay A:** zajednički podaci i rutine + **kod za prvi deo programa**
 - Ukupno 130KB.
 - **Overlay B:** zajednički podaci i rutine + **kod za drugi deo programa**
 - Ukupno: 120KB.
- Programu se mora dodati *overlay* drajver koji upravlja *overlay* tehnikom.
 - Pretpostavimo da on zauzima 10KB memorije.
- Kako se program izvršava?
 - Prva faza: *overlay* drajver + *overlay* A.
 - Zauzima 140KB i može da se izvrši na sistemu sa 150KB.
 - Kada *overlay* A završi svoje, drajver će učitati u memoriju *overlay* B preko njega.
 - Druga faza: *overlay* drajver i *overlay* B.
 - Zauzima 130KB i može da se izvrši na sistemu sa 150KB memorije.



A – prvi deo programa
B – drugi deo programa

C – zajednički podaci
D – zajedničke rutine

OD – overlay drajver
OS – operativni sistem

- **Kontinualna alokacija memorije.**
 - Logički i fizički adresni prostor procesa sastoje se od kontinualnog niza memorijskih adresa.
 - Prosto rečeno, svaki proces dobija jedan kontinualni deo memorije.
 - Metode:
 - multiprogramiranje sa fiksnim particijama,
 - multiprogramiranje sa particijama promenljive veličine.
- **Diskontinualna alokacija memorije.**
 - Fizički adresni prostor procesa nije realizovan kao kontinualan niz memorijskih adresa.
 - Metode:
 - straničenje,
 - segmentacija,
 - straničenje sa segmentacijom.

Multiprogramiranje sa fiksnim particijama

- Podela cele fizičke memorije na više delova **fiksne veličine**.
 - U jednom delu se može naći samo jedan proces.
 - Stepenn multiprogramiranja je jednak broju memorijskih particija.
- Novi proces se smešta **u najmanju moguću particiju**.
- Problem: **interna fragmenatcija**.
 - Delovi memorije koji su veći od veličine procesa potpuno neiskorišćeni.
- Svi procesi se stavljaju u **red čekanja** (engl. *input queue*) koji može biti:
 - Jedinstven za ceo sistem.
 - Problem: veliki neiskorišćen prostor (mali procesi u velikim particijama).
 - Poseban za svaku particiju.
 - Problem: veći broj malih procesa može čekati u redu za male particije.
 - Velike particije ostaju **neiskorišćene**.
 - U tom slučaju ima dovoljno memorije, ali se ne koristi.

Multiprogramiranje sa particijama promenljive veličine

- Memorija se deli dinamički.
- **Šupljina** (*hole*) je slobodan kontinualni deo memorije.
 - Šupljine raznih veličina su razbacane po memoriji!
 - Kada proces naiđe u sistem traži se šupljina dovoljno velika da zadovolji proces.
 - Sav prostor koji proces ne zauzme od cele šupljine, predstavlja novu šupljinu u koju se može smestiti drugi proces.
- Operativni sistem vodi evidenciju o:
 - particijama koje su dodeljene procesima,
 - slobodnim šupljinama.
- Metode vođenja evidencije:
 - bit mape,
 - povezane liste,
 - sistem udruženih parova.

- Memorija se deli na delove **iste veličine**.
- Svakom delu dodeljujemo po jedan bit: 1 označava da je taj deo zauzet, a 0 da je slobodan.
- Niza nula i jedinica predstavlja **bit mapu** (engl. *bitmap*) **memorije**.
- Pitanje: koje veličine treba da budu ovi delovi memorije?
 - Ako su delovi manji, bit mapa je veća!
 - Veličina delova 32 bita → za bit mapu koristimo 1/33 ukupne memorije.
 - Veličina delova 16 bita → za bit mapu koristimo 1/17 od ukupne memorije
 - Problem: pretraživanje slobodne memorije (niz nula) je sporo.
 - Ako su delovi veći, memorija je loše iskorišćena.
 - Primer: delovi veličine 4KB.
 - Ako proces zauzme 1KB od nekog dela, 3KB ostaje neiskorišćeno.
 - 3KB ne može biti alocirano od strane drugog procesa jer je formalno zauzeto.

- **Povezane liste** (engl. *linked lists*) gradimo od slogova sledeće strukture:
 - **tip memorije** (P znači da se radi o procesu, a H da se radi o slobodnoj memoriji),
 - **početna adresa** dela memorije koju opisuje dati slog,
 - **dužina** opisane memorije,
 - **pokazivač** na sledeći slog.
- Upravljanje memorijom se odvija na sledeći način:
 - Prilikom **zauzimanja memorije** u povezanoj listi se traži rupa (slog tipa H) dovoljne veličine.
 - Ako se nađe rupa odgovarajuće veličine, umesto H se upisuje P.
 - Ako je rupa veća ubacuje se i novi čvor tipa H.
 - Kada proces završi sa radom, **oslobodja se zauzeta memorija**.

Sistem udruženih parova (buddy system)

- Koristi se po jedna lista za blokove slobodne memorije veličine 2^n bajtova (1, 2, 4, ...)
- Primer:
 - Imamo 1MB memorije.
 - Treba nam 21 lista ($2^0=1$, $2^2=2$... $2^{20}=1\text{MB}$).
- Na početku rada, cela memorija je prazna.
 - U listi za rupe veličine 1MB imamo jedan slog.
 - Ostale liste su prazne.
- Proces A veličine 70KB nailazi u sistem.
 - Može se smestiti u particiju veličine najmanje 128KB (mora biti stepen od 2).
 - Lista koja sadrži particije te veličine je **prazna**.
 - Particija veličine 1MB deli se na dva dela od 512KB (ovi delovi se zovu **drugovi** – engl. *buddy*).
 - Zatim se prvi deo deli na dva dela od 256KB, pa još jednom, na dva dela od 128KB.
 - U prvu particiju od 128KB **smešta se proces!**

- Nakon smeštanja procesa lista od 128Kb sadrži dva čvora:
 - jedan je zauzet – P,
 - drugi je slobodan – H.
- Liste od 256KB i 512KB sadrže po jedan slobodan čvor:

P (128KB) {proces A:70KB, neiskorišćeno:58KB}

H (128KB)

H (256KB)

H (512KB)

- Nedostatak ovog sistema je **unutrašnja fragmentacija**.
 - Proces veličine 70KB zauzima čitavu particiju od 128KB.
 - 58KB memorije ostaje neiskorišćeno (unutrašnja fragmentacija).

- Proces B veličine 35KB nailazi u sistem.
 - Može se smestiti u particiju veličine 64KB.
 - Lista za takve particije je prazna.
 - Slobodna particija od 128KB deli na dva dela veličine 64KB.
 - Proces B se smešta u jednu particiju, dok druga ostaje prazna.
 - Sada imamo:
 - po jedan čvor u listama za particije veličine 128KB, 256KB i 512KB,
 - dva čvora u listi za particije veličine 64KB.

P (128KB) {proces A:70KB, neiskorišćeno:58KB}

P (64KB) {proces B:35KB, neiskorišćeno:29KB}

H (64KB)

H (256KB)

H (512KB)

- Proces C veličine 80KB nailazi u sistem.
 - Može se smestiti u particiju veličine 128KB.
 - Lista za takve particije je prazna.
 - Slobodna particija od 256KB deli na dva dela veličine 128KB.
 - Proces C se smešta u jednu particiju, dok druga ostaje prazna.
 - Sada imamo:

P (128KB) {proces A:70KB, neiskorišćeno:58KB}

P (64KB) {proces B:35KB, neiskorišćeno:29KB}

H (64KB)

P (128KB) {proces C:80KB, neiskorišćeno:48KB}

H (128KB)

H (512KB)

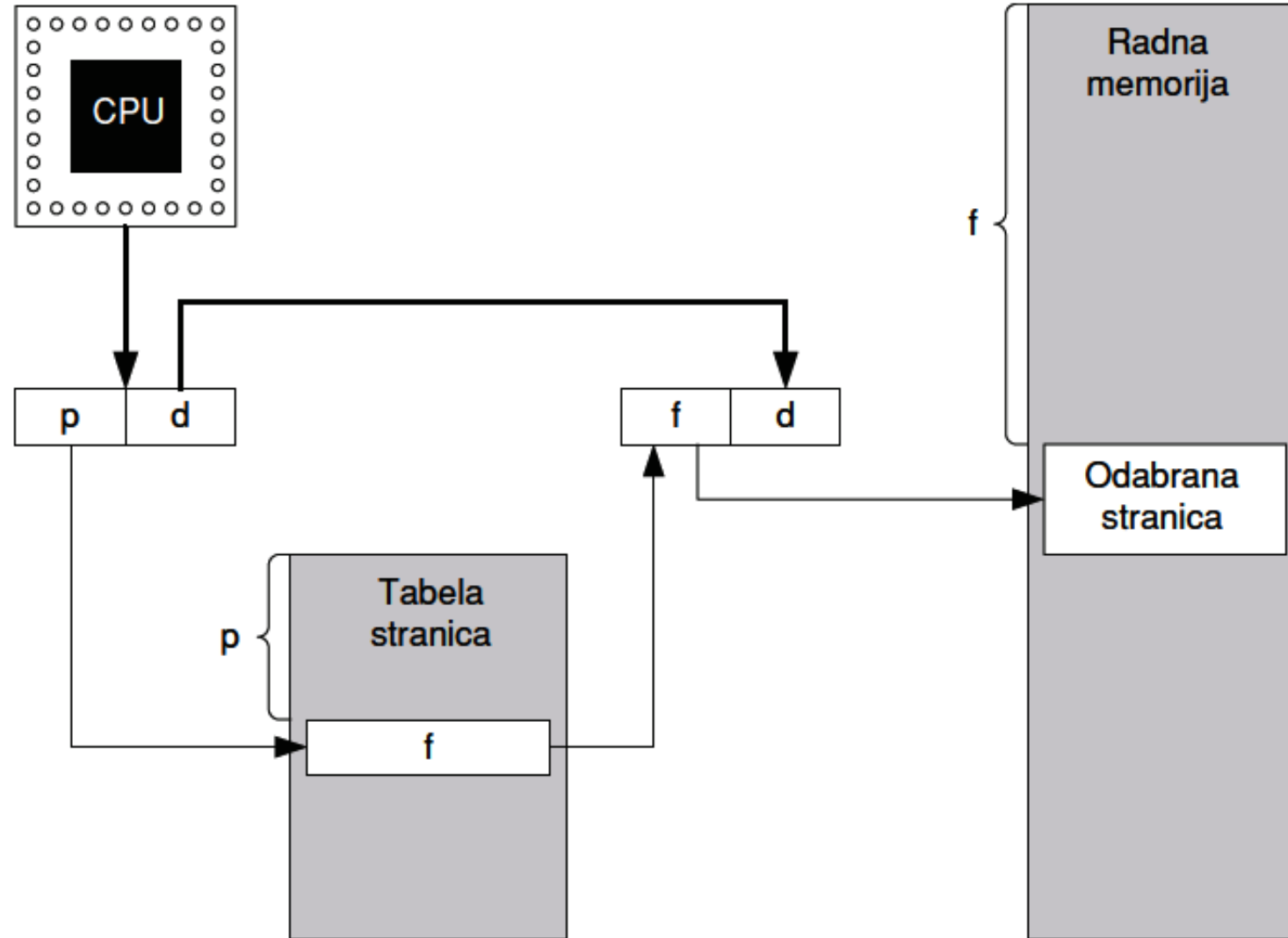
- Stanje memorije nakon dodele memorije procesima A, B i C.
 - Tri procesa ukupne veličine $70+35+80=185\text{KB}$
 - Zauzimaju $128+64+128=320\text{KB}$ memorije
 - 135KB ostaje neiskorišćeno kao posledica unutrašnje fragmentacije.
- **Susedne rupe iste veličine** predstavljaju drugove.
 - Susedni procesi to nisu.
- Kada neki proces završi rad, particija u kojoj se proces nalazio postaje rupa.
 - Rupa se spaja sa susedom (svojim drugom ukoliko on postoji).
 - Time se formira duplo veća rupa.
 - Važno: veličina novoformirane rupe mora da bude stepen broja dva!
 - Ukoliko to nije slučaj, spajanje se ne vrši.

Algoritmi za izbor slobodne particije

- Po pravilu, postoji više šupljina koje se mogu iskoristiti za smeštaj procesa, a u tom kontekstu postoji i više algoritama za alokaciju šupljina.
- **Prvo podudaranje** (engl. *first-fit*).
 - Pretraživanje se odvija ili od početka liste.
 - Pretraga staje kada se nađe prva odgovarajuća šupljina bez obzira što ima i boljih rešenja.
 - Vreme pretraživanja je minimalno, jer se ne pretražuje cela lista.
- **Najbolje podudaranje** (engl. *best-fit*).
 - Procesu se **dodeljuje najmanja šupljina** koja je dovoljno velika za smeštaj procesa.
 - Od postojeće šupljine nakon smeštaja procesa **ostaje najmanja moguća šupljina**.
 - Pretražuje se cela lista (**intenzivna pretraga**) osim ako nije uređena po veličini.
- **Najgore podudaranje** (engl. *worst-fit*).
 - Procesu se **dodeljuje najveća slobodna šupljina**.
 - Od postojeće šupljine nakon smeštaja procesa **ostaje najveća moguća šupljina**.
 - Pretražuje se cela lista (**intenzivna pretraga**) osim ako nije uređena po veličini.

- **Unutrašnja (interna) fragmentacija.**
 - Procesu je dodeljena memorijska particija veća od memorije koju zahteva proces.
 - Preostala memorija je neupotrebljiva.
- **Spoljašnja (eksterna) fragmentacija.**
 - Ukupan memorijski prostor može da zadovolji zahtev procesa ali **nije kontinualan**.
- Eksterna fragmentacija se može smanjiti **kompakcijom memorije**.
 - Sadržaj memorije se ispremešta.
 - Dobijaju se veće šupljine, odnosno kontinualan prazan prostor.
 - Problem:
 - Sistem mora da prekida procese i da ih privremeno prebacuje na disk.
 - To je vremenski zahtevno.
 - Kompakcija svakako unosi degradaciju performansi sistema.

- Straničenje je metoda sa **hardverskom podrškom na nivou procesora**.
- Fizička memorija se deli na blokove fiksne veličine: **okvire** (engl. *page frames*).
- Logički adresni prostor se deli na blokove istih veličina: **stranice** (engl. *pages*).
 - Veličine stranica su po pravilu stepen broja 2.
- Svakoj logičkoj stranici odgovara jedna fizička.
- Korespodencija između njih se čuva u **tabeli stranica** (engl. *page table*).
 - Kontinualni logički prostor procesa može da se razbaca po fizičkoj memoriji.
- Svaka logička adresa koju generiše procesor se deli na dva dela:
- **Broj stranice** (engl. *page number*) – p .
 - Koristi se kao indeks u tabeli stranica koja sadrži baznu adresu okvira.
 - Bazna adresa predstavlja viši deo adrese.
- **Pomeraj unutar stranice** (engl. *page offset*) – d .
 - Definiše položaj u odnosu na samu stranicu.



- Primer: imamo memoriju veličine 32B.
 - Definišimo 8 okvira veličine 4B.
 - Korisnički proces zauzima 4 logičke stranice.
 - Stranica 0 – logičke adrese 0-3.
 - Stranica 1 – logičke adrese 4-7.
 - Stranica 2 – logičke adrese 8-11.
 - Stranica 3 – logičke adrese 12-15.
 - Kako se interpretiraju logičke adrese (tabela stranica data na sledećem slajdu)?
 - Na primer, kako ćemo interpretirati logičku adresu 5?

Stranica 0
Stranica 1
Stranica 2
Stranica 3

Logički adresni
prostor

p	f
0	1
1	4
2	3
3	7

Tabela stranica

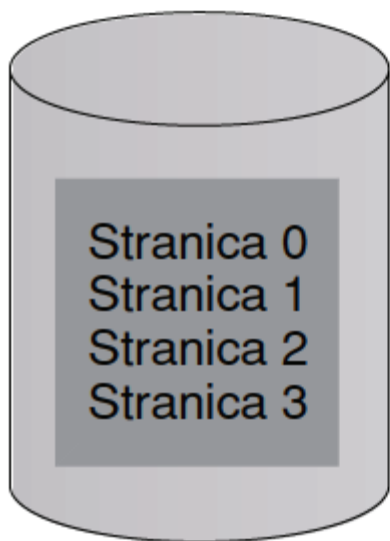
Broj
okvira

0	
1	Stranica 0
2	
3	Stranica 2
4	Stranica 1
5	
6	
7	Stranica 3

Fizički adresni
prostor

- Logička adresa 5 se mapira na sledeći način:
 - Adresa 5 je logička stranica 1 sa pomerajem 1.
 - Na osnovu tabele stranica to je okvir 4 sa pomerajem 1.
 - Dakle, adresa je $16+1=17$.

- Proces koji ulazi u stanje izvršavanja mora da dobije potrebnu memoriju.
- Za dati proces preračunava **koliko mu stranica memorije treba**.
 - Svaka stanica mora da se mapira u okvir.
 - Ako proces zatheva n stranica, tada se alocira n okvira.
 - Okviri se pune procesom.
 - Mapiranje stranica-okvir upisuje u tabelu stranica.
 - Alokacija se vrši na osnovu liste slobodnih okvira.
- Kao posledica, raspored zauzetih i slobodnih stranica je slučajan.



Nov proces na disku

Pre dodele stranica novom procesu

f
10
13
14
12
8

Lista slobodnih okvira

Broj
okvira

8	
9	
10	
11	
12	
13	
14	
15	

Fizički adresni
prostor

Nakon dodele stranica novom procesu

p	f
0	10
1	13
2	14
3	12

Tabela stranica
novog procesa

f
8

Lista slobodnih okvira

8	
9	
10	Stranica 0
11	
12	Stranica 3
13	Stranica 1
14	Stranica 2
15	

Fizički adresni
prostor

- Šta je važno napomenuti u vezi straničenja?
 - Korisnik svoj deo memorije doživljava kao **kontinualni prostor**.
 - Mapiranje logičkog i fizičkog prostora je zadatak OS-a i **transparentno je za korisnika**.
 - **Eksterne fragmentacije nema**, jer se svaki okvir može iskoristiti za smeštaj dela bilo kog procesa.
 - **Interna fragmentacija postoji**.
- Koliko se memorije može adresirati?
 - Primer:
 - Tabela stranica je 32 bitna.
 - Veličina stranice je 4KB.
 - Može se adresirati $2^{32} * 4 * 2^{10} = 16\text{TB}$ fizičkog adresnog prostora.

- Zaštita memorije postiže se uvođenjem novih bitova koji imaju specijalno značenje.
- **Bit validnosti v** ukazuje da je vrednost u tabeli stranica važeća, ispravna i da se okvir može se koristiti za mapiranje.
 - To se ponekad koristi da se nekom programu zabrani pristup određenim logičkim adresama.
 - Primer (slika na sledećem slajdu):
 - Svaka referenca programa koja pripada logičkim stranicama 6 i 7 završice se kao ilegalan zahtev.
 - Razlog: te stranice su u tabeli proglašene kao nevažeće (engl. *invalid bit*).
- **Bit za zaštitu od upisa rw** (odnosno ro) određuje da li je stranica sa punim pristupom ili je zaštićena (može se samo čitati).
 - Poželjno je zaštititi stanice koje sadrže kod programa (sprečava se nehotično prepisivanje).

Stranica 0
Stranica 1
Stranica 2
Stranica 3
Stranica 4
Stranica 5

Logički adresni
prostor

p	f	v / i
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

Tabela stranica

p – logička stranica

f – okvir

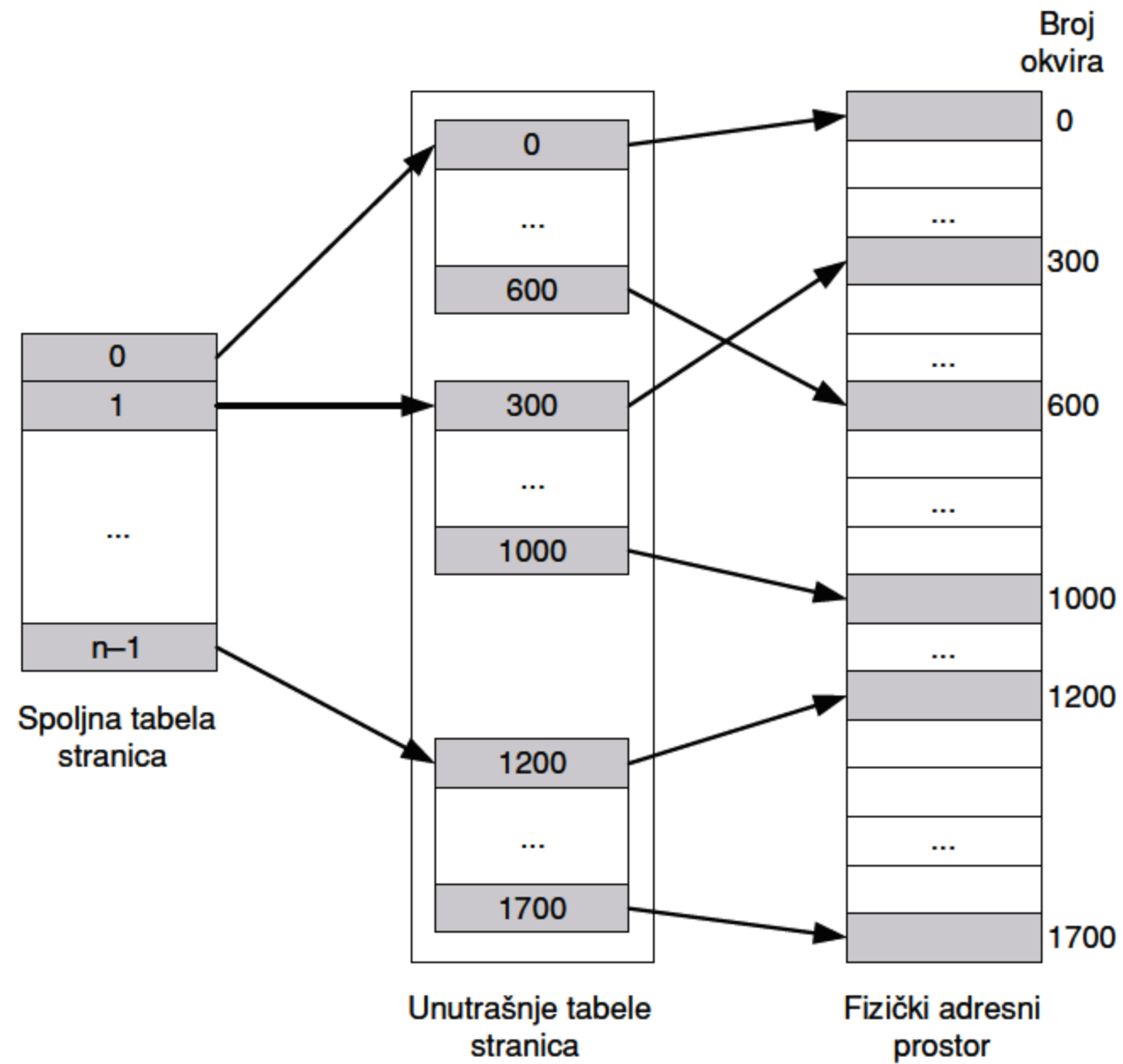
v / i – bit validnosti

Broj
okvira

0	
1	
2	Stranica 0
3	Stranica 1
4	Stranica 2
5	
6	
7	Stranica 3
8	Stranica 4
9	Stranica 5
	...
n	

Fizički adresni
prostor

- Logički adresni prostor savremenih računara je 2^{32} ili 2^{64} .
- U takvim situacijama, tabele stranica su jako velike!
- Primer:
 - 32 bitni system,
 - stranice veličine 4KB (2^{12}),
 - tabela stranica se sastoji od milion ulaza ($2^{32} / 2^{12} = 2^{20}$).
 - Ako se svaki ulaz sastoji od 4 bajta tada tabela stanica obuhvata 4MB.
- Međutim, ne treba svakom procesu milion ulaza (ceo logički adresni prostor).
 - Tabela stranica se može podeliti na više manjih delova.
 - Najjednostavniji primer je **dvonivovsko straničenje**.
 - **Spoljna tabela stranica** koja ukazuje na stranice u kojima su smeštene **prave tabele stranica**.



- Jedna od prednosti straničenja je mogućnost **deljenja zajedničkog koda** između većeg broja procesa.
- Primer:
 - Sistem na kome više korisnika istovremeno izvršava isti tekst editor.
 - Kada ne bi bilo deljenja svaki korisnički proces, pored sekcija podataka morao bi da učitava za sebe posebnu kopiju koda tekst editora u memoriju.
 - Ako se kod editora ne modifikuje u toku izvršavanja, deljenje stranica se može iskoristiti:
 - Jednu kopiju koda u memoriji koristiće svi procesi.
 - Njihove tabele stranica ukazivaće na isti memorijski prostor u kome se nalazi tekst editor.

Editor 1
Editor 2
Editor 3
Podaci – 1

Proces P_1

0	3
1	4
2	6
3	1

P_1 – tabela stranica

Editor 1
Editor 2
Editor 3
Podaci – 2

Proces P_2

0	3
1	4
2	6
3	7

P_2 – tabela stranica

Editor 1
Editor 2
Editor 3
Podaci – 3

Proces P_3

0	3
1	4
2	6
3	2

P_3 – tabela stranica

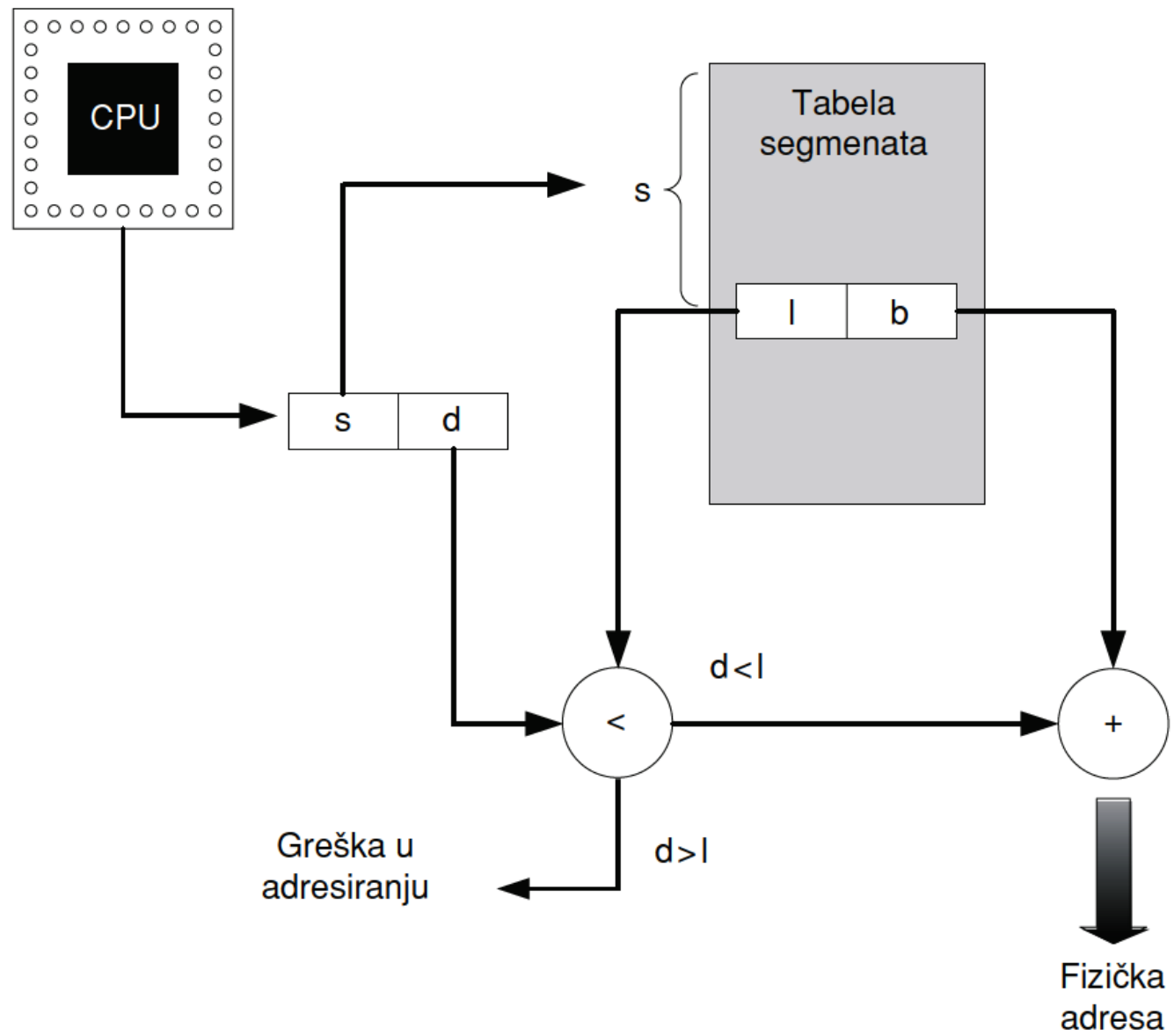
Broj
okvira

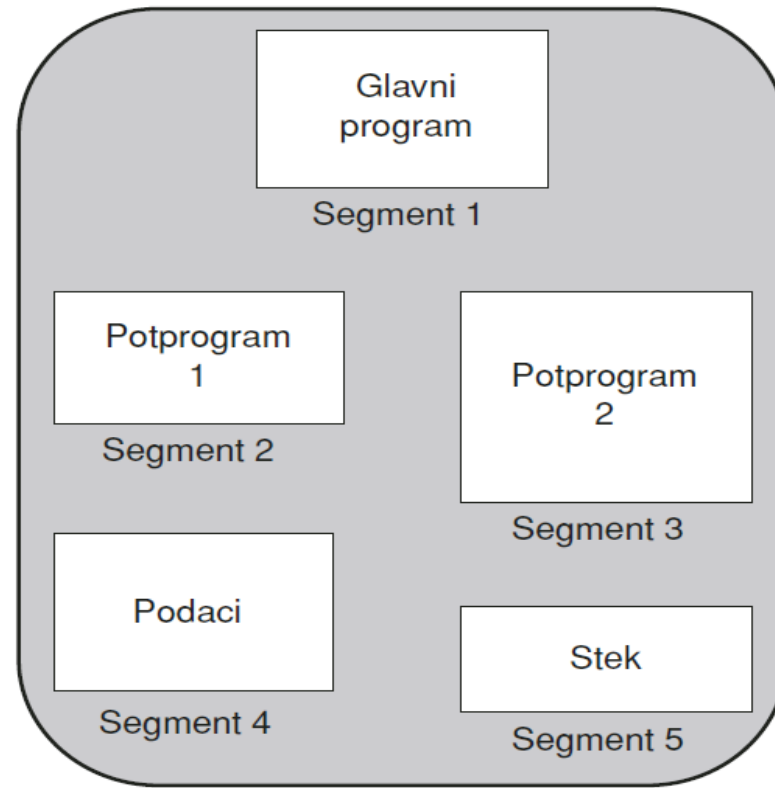
0	
1	Podaci – 1
2	Podaci – 3
3	Editor 1
4	Editor 2
5	
6	Editor 3
7	Podaci – 3
8	
9	
	...
n	

Fizički adresni
prostor

- Logički adresni prostor se sastoji kolekcije segmenata.
- Svaki segment ima jedinstveno ime i dužinu.
- Logička adresa se se sastoji od dva dela:
 - **imena segmenta** (umesto imena obično se zadaje broj koji predstavlja ID segmenta),
 - **pomeraja unutar segmenta**.
- Kod segmenatacije je prisutna **eksterna fragmentacija**.
 - Slobodna memorija se ne može iskoristiti za smeštaj segmenta ukoliko ne postoji dovoljno velika šupljina.
 - Problem se može umanjiti **kompakcijom memorije**.

- U metodi segmentacije korisničke adrese su dvodimezione.
- Mapiranje u fizičke adrese se obavlja preko **tabele segmenata**.
- Svaki ulaz u tabeli segmenata sadrži dva parametra:
 - **Baznu adresu segmenta** (početna fizičku adresu segmenta u memoriji)
 - **Limit segmeta** (dužina segmenta).
- Kako se logička adresa mapira u fizičku?
 - **Broj segmenta** (s) se koristi kao **ulaz u tabelu** segmenta.
 - Iz tabele se čitaju **bazna adresa** segmenta (b) i **limit** (l).
 - Ako je:
 - Pomeraj veći od limita ($d > l$):
 - Imamo grešku u adresiranju.
 - Pomeraj manji od limita ($d < l$):
 - Fizička adresa = bazna adresa + pomeraj.

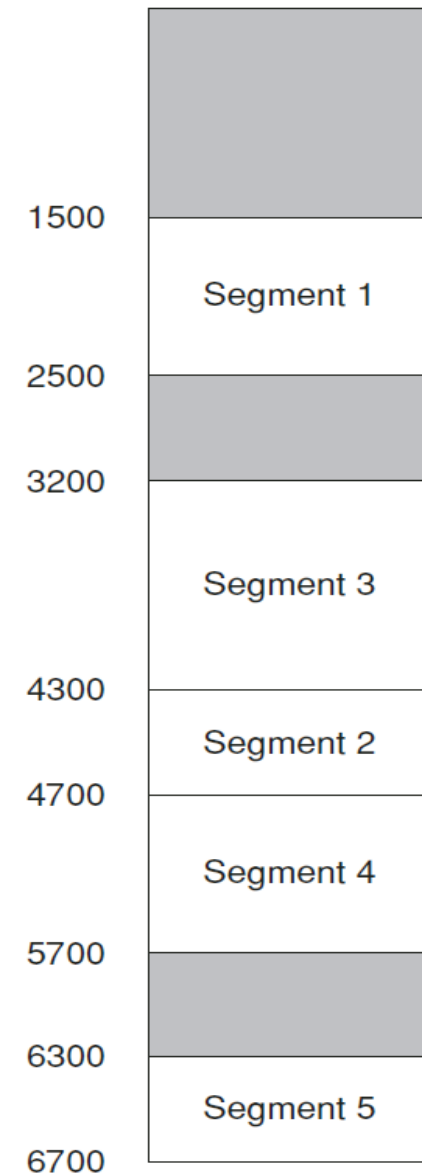




Logički adresni prostor

Segment	Bazna adresa	Ograničenje
1	1500	1000
2	4300	400
3	3200	1100
4	4700	1000
5	6300	400

Tabela segmenata



Fizički adresni prostor

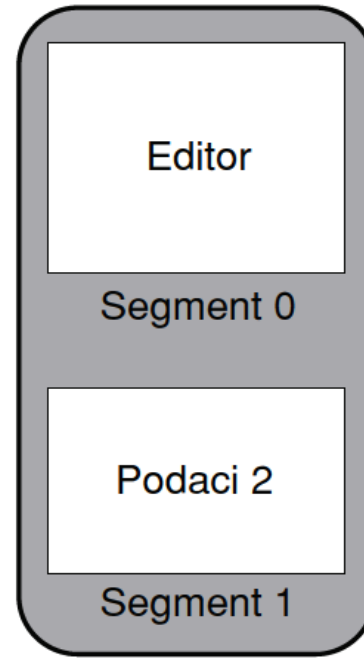
- Tabela segmenata može sadržati i neke oblike zaštite segmenata.
 - Najčešće koristi zaštita od izmene sadržaja.
 - Segmenti sa kodom mogu biti definisani kao **nepromenljivi** (engl. *read only*).
- Segmenti koji se ne menjaju se mogu **deliti**.
 - Na taj način se obavlja efikasna ušteda memorije.
 - Primer:
 - Dva procesa koji koriste isti editor teksta.
 - Svaki proces ima svoju tabelu segmenata.
 - Segment koji sadrži kod editora biće zajednički.



Logički adresni prostor procesa P_1

S	Bazna adresa	Ograničenje
1	3200	1100
2	4300	400

Tabela segmenata procesa P_1



Logički adresni prostor procesa P_2

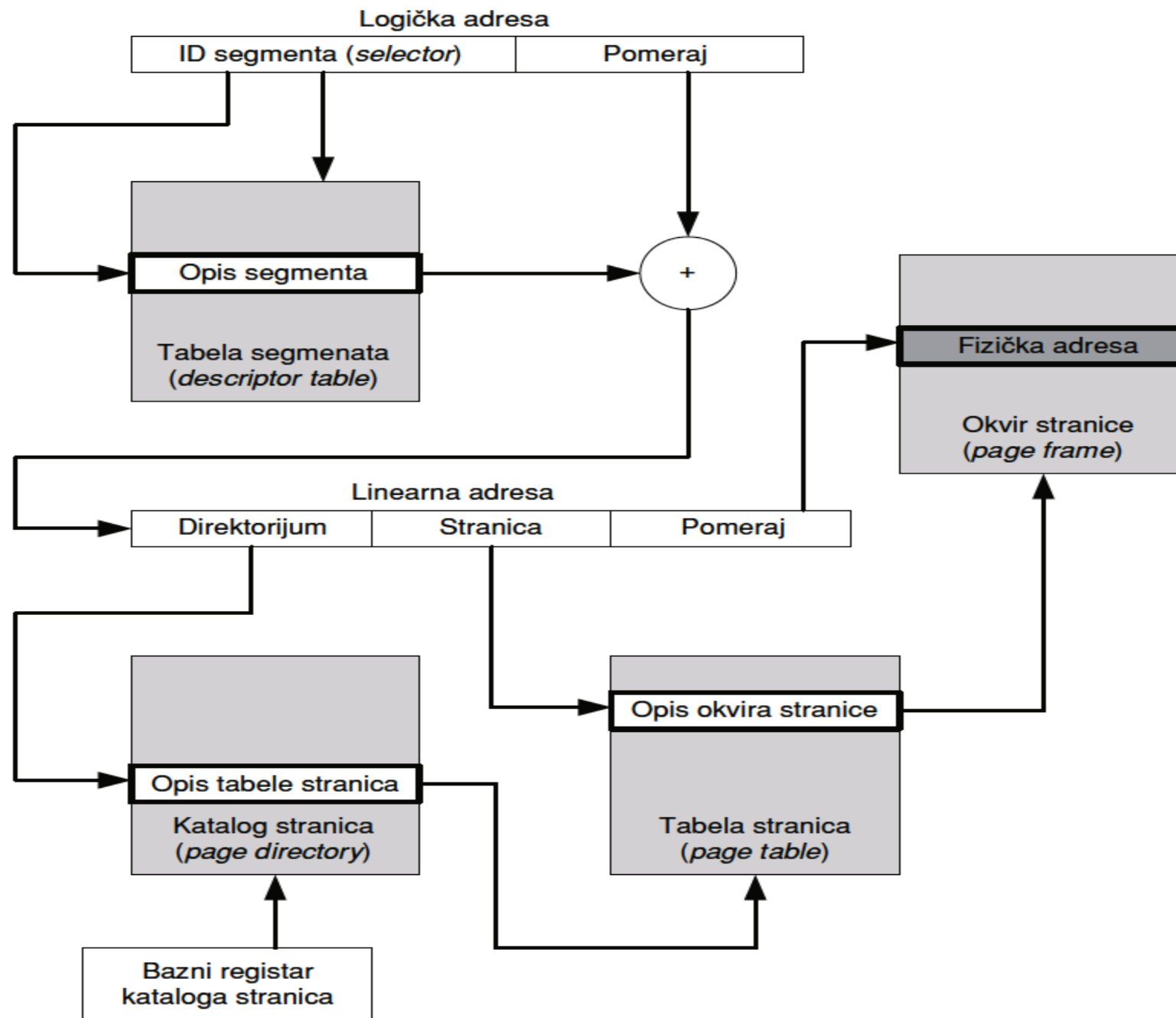
S	Bazna adresa	Ograničenje
1	3200	1100
2	1500	2500

Tabela segmenata procesa P_2



Segmentacija sa straničenjem

- Na procesorima ugrađenu podršku i za segmentaciju i za straničenje, mogu se kombinovati metode diskontinualne alokacije.
- Straničenje poništava eksternu fragmentaciju segmenata!
- Primer: **segmentacija sa dvonivovskom straničenjem.**



1. B. Đorđević, D. Pleskonjić, N. Maček (2005): Operativni sistemi: teorija, praksa i rešeni zadaci. Mikro knjiga, Beograd.
2. R. Popović, I. Branović, M. Šarac (2011): Operativni sistemi. Univerzitet Singidunum, Beograd.

Pitanja su dobrodošla.