# Upravljanje memorijom

## Operativni sistemi 1

Institut za matematiku i informatiku
Prirodno-matematički fakultet, Kragujevac

doc. dr Miloš Ivanović

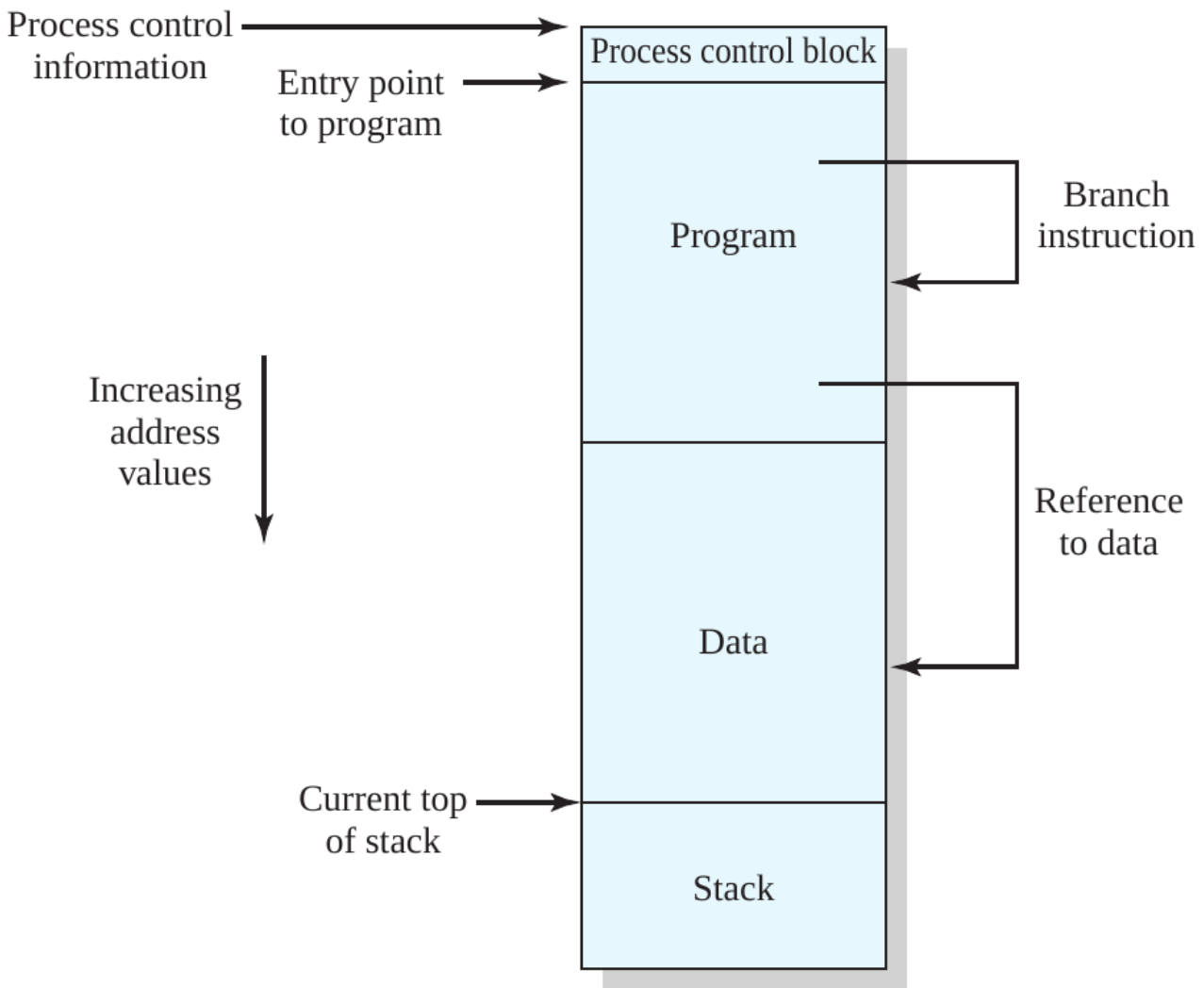
Decembar 2013. god.

# O čemu će biti reči?

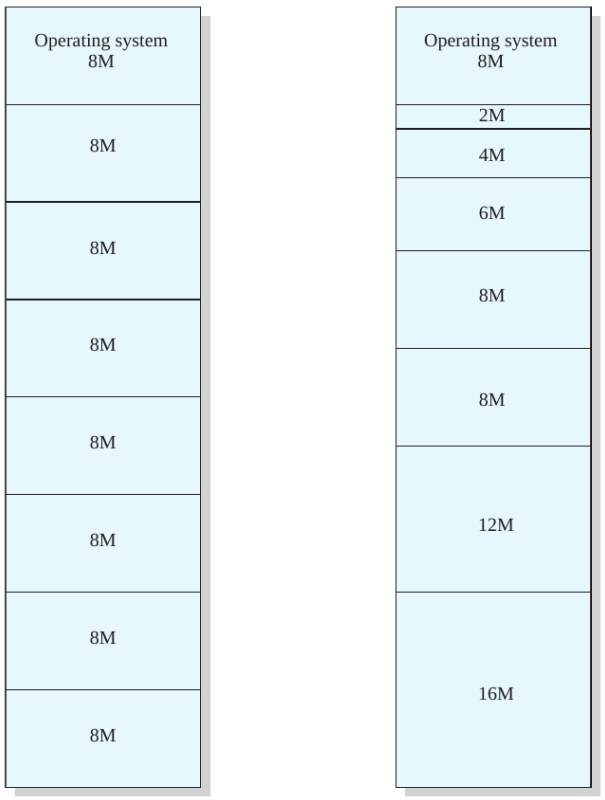
# Zahtevi pred sistemom za upravljanje memorijom

1. **Relokacija** - Od koje početne adrese će program biti učitan nakon povratka sa diska?
2. **Zaštita** - Sve memorijske adrese se moraju proveriti u vreme izvršavanja programa
3. **Deljenje** - Procesima treba obezbediti deljena memorijska područja preko kojih mogu da komuniciraju, bez ugrožavanja zaštite.
4. **Logička organizacija** - Omogućiti modularnu organizaciju programa, npr. preko statičkih i dinamičkih biblioteka. Postiže se *segmentacijom*.
5. **Fizička organizacija** - Primarna i sekundarna memorija. Upravljanje funkcijom razmene ne treba prepustiti programeru (*overlaying* tehnika), već OS-u.

# Zahtevi adresiranja u okviru procesa



# Fiksna podela na particije



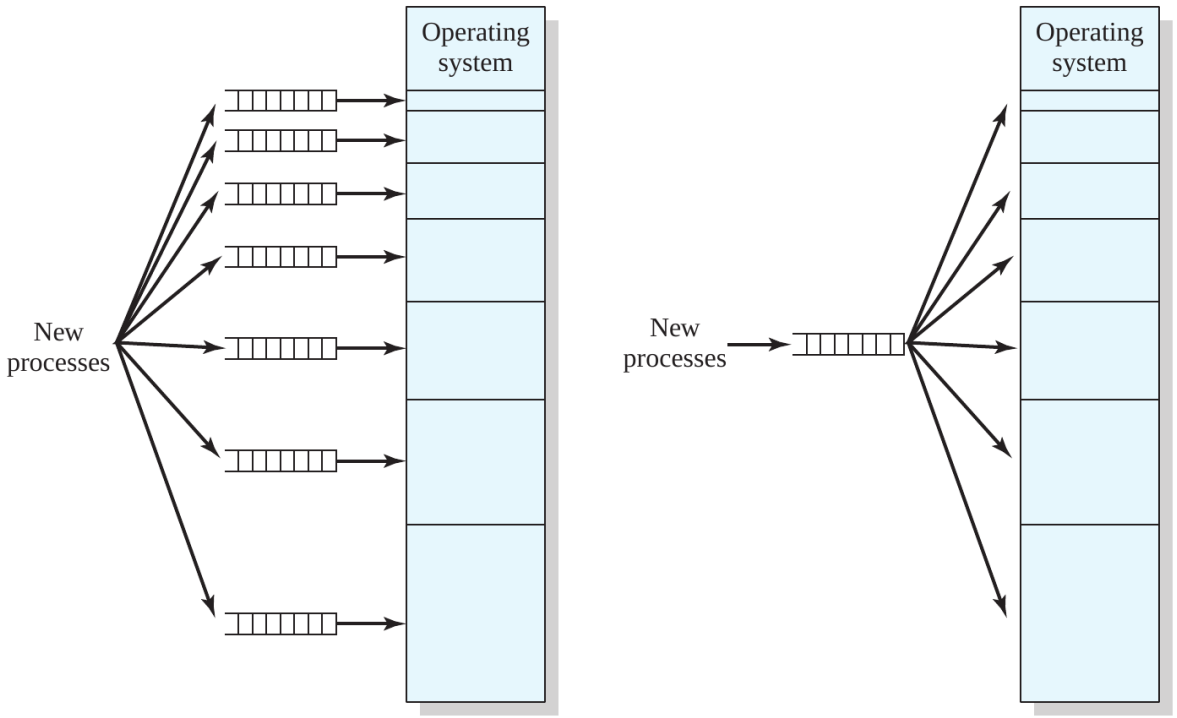

Slika: Particionisanje sistema sa 64MB RAM. *Levo*: Particije jednake dužine *Desno*: Particije različite dužine

## Problemi

- Program može biti prevelik za jednu particiju
- **Unutrašnja fragmentacija** - neefikasno iskorišćenje prostora
- Problemi mogu da se umanje, ali ne i reše uvođenjem nejednakih particija
- Broj particija, a prema tome i **najveći broj procesa u memoriji je fiksan** i ograničen
- **Da li se zahtevi procesa za memorijom znaju unapred?**

# Algoritam smeštanja

## Dodeljivanje najmanje odgovarajuće partcije

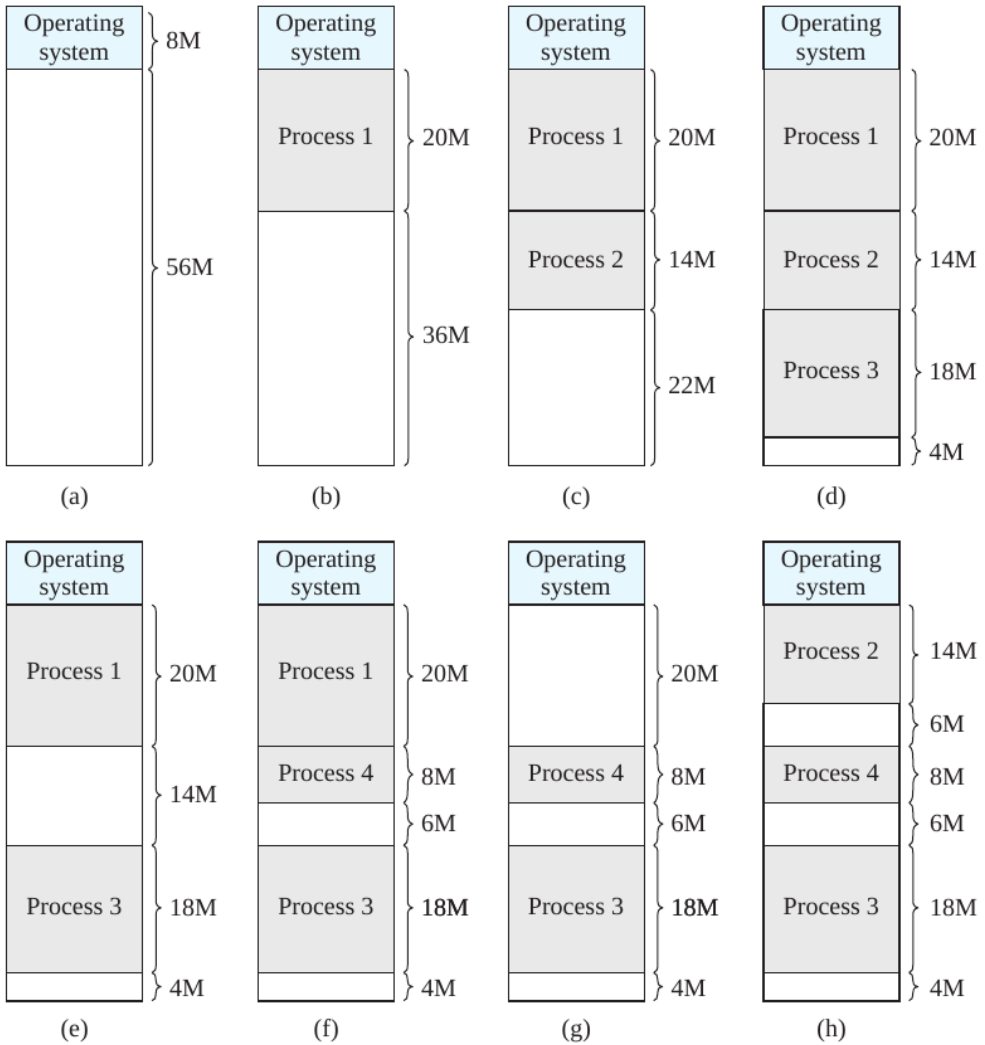



**Pitanje**

Da li je efikasnije imati red čekanja ispred svake particije ili samo jedan zajednički?

# Dinamičko particionisanje memorije

## Dodela tačno onoliko prostora koliko proces traži



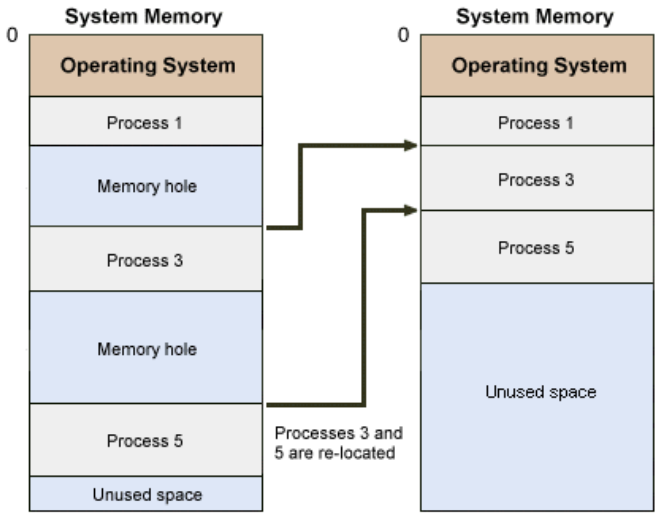
# Dinamičko particionisanje memorije
## Problemi

### Problem

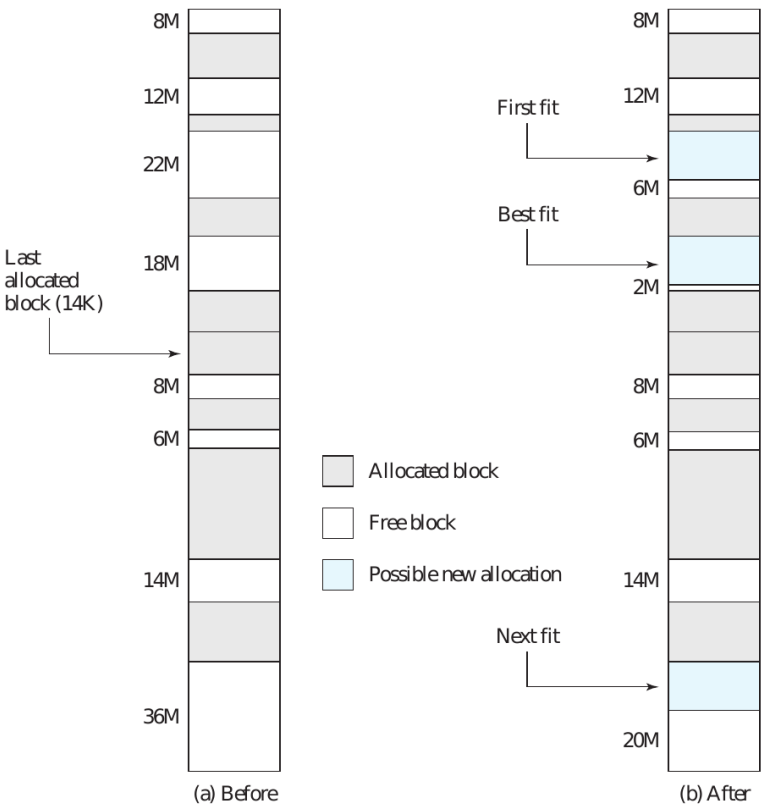
**Spoljašnja fragmentacija** - Memorija koja ostaje nedodeljena je sve više i više fragmentirana

### Rešenje - sažimanje

Delimično rešenje problema spoljašnje fragmentacije je **sažimanje**- (**compaction**). Osnovni uslov za funkcionisanje ovog mehanizma je mogućnost *relokacije procesa*.



# Algoritam smeštanja




## Algoritmi

1. **Najbolja odgovarajuća** - Najmanja particija koji odgovara zahtevu
2. **Prva odgovarajuća** - Prva slobodna koja odgovara zahtevu
3. **Sledeća odgovarajuća** - Prva odgovarajuća počevši od prethodne dodele

## Pitanje

Koji algoritam ima najlošije performanse?

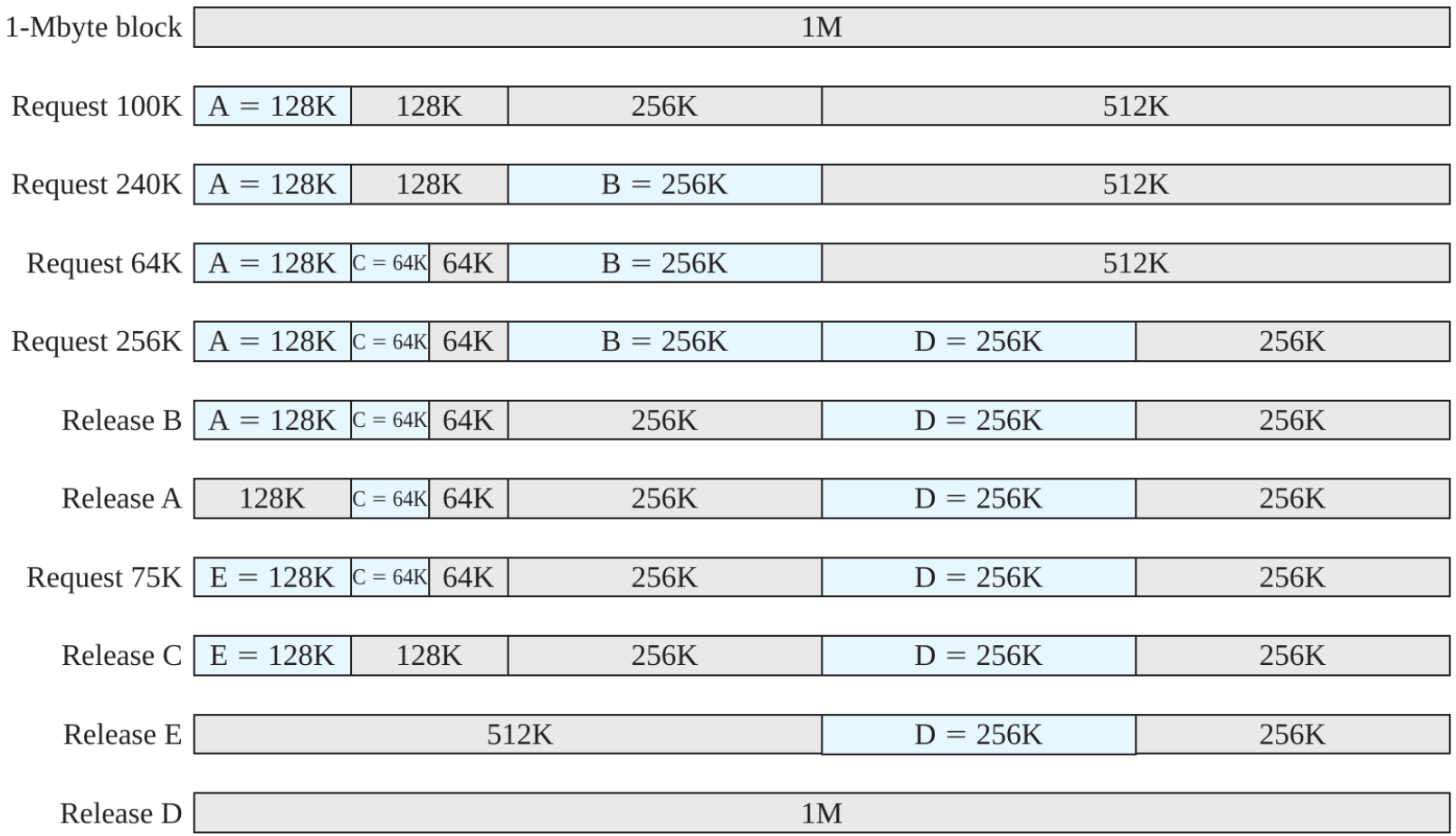
# Partnerski sistem dodele memorije

- Na raspolaganju su blokovi veličine $2^K$, gde je $L \leq K \leq U$ ($2^U$ je obično ukupna veličina memorije).
- Kada se postavi zahtev $s$ takav da je $2^{U-1} \leq s \leq 2^U$ dodeljuje se ceo blok.
- U suprotnom, blok se deli na dva partnera veličine $2^{U-1}$. Ako je $2^{U-2} \leq s \leq 2^{U-1}$, dodeljuje se jedan od blokova.
- Postupak se ponavlja sve dok se ne nađe odgovarajući najmanj blok $\geq s$ i dodeli zahtevu.
- Održava se lista rupa svake veličine $2^i$. Ako dva partnera $i$-te klase ostanu nedodeljena, spajaju se ponovo u jedan blok klase $(i + 1)$.

```
void get_hole(int i)
{
    if (i == (U + 1))
        <greska>;
    if (<i_list prazna>)
    {
        get_hole(i + 1);
        <podeli rupu na partnere>;
        <dodaj partnere na i_list>;
    }
    <uzmi prvu rupu na i_list>;
}
```

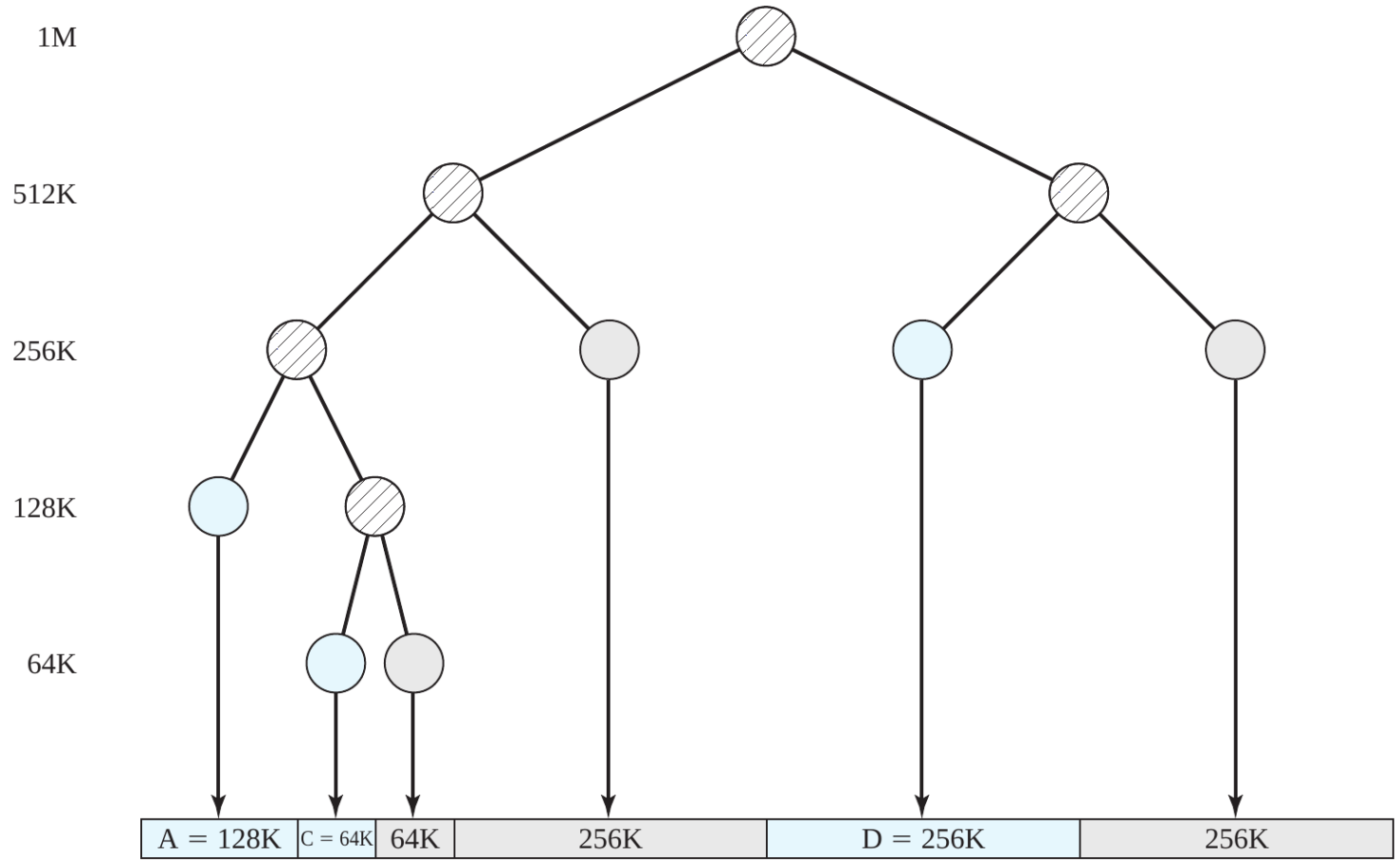
# Partnerski sistem dodele memorije

## Primer



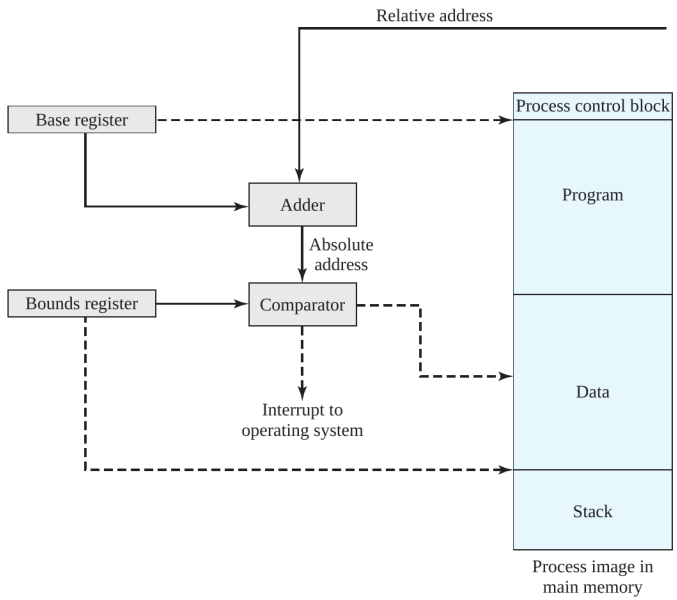
# Partnerski sistem dodele memorije

## Organizacija u formi binarnog stabla



# Relokacija

- Ukoliko se ograniči da proces posle zamene (*swap*) može da se učita samo u istu particiju, algoritam je jednostavan. Samo prvi put treba dodati početnu adresu particije svim adresama u kodu.
- **Logička adresa** je referenca na memorijsku lokaciju nezavisna od trenutne situacije u memoriji. Mora da se prevede u fizičku adresu.
- **Relativna adresa** je poseban primer logičke adrese koja se izražava relativno u odnosu na neku poznatu tačku.
- **Fizička adresa (apsolutna adresa)** je stvarna lokacija u RAM-u.



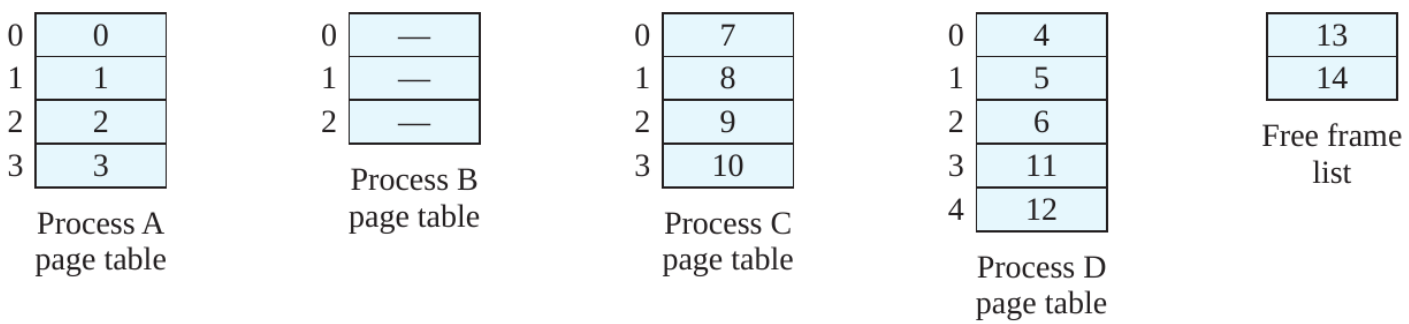
# Straničenje

## Problem

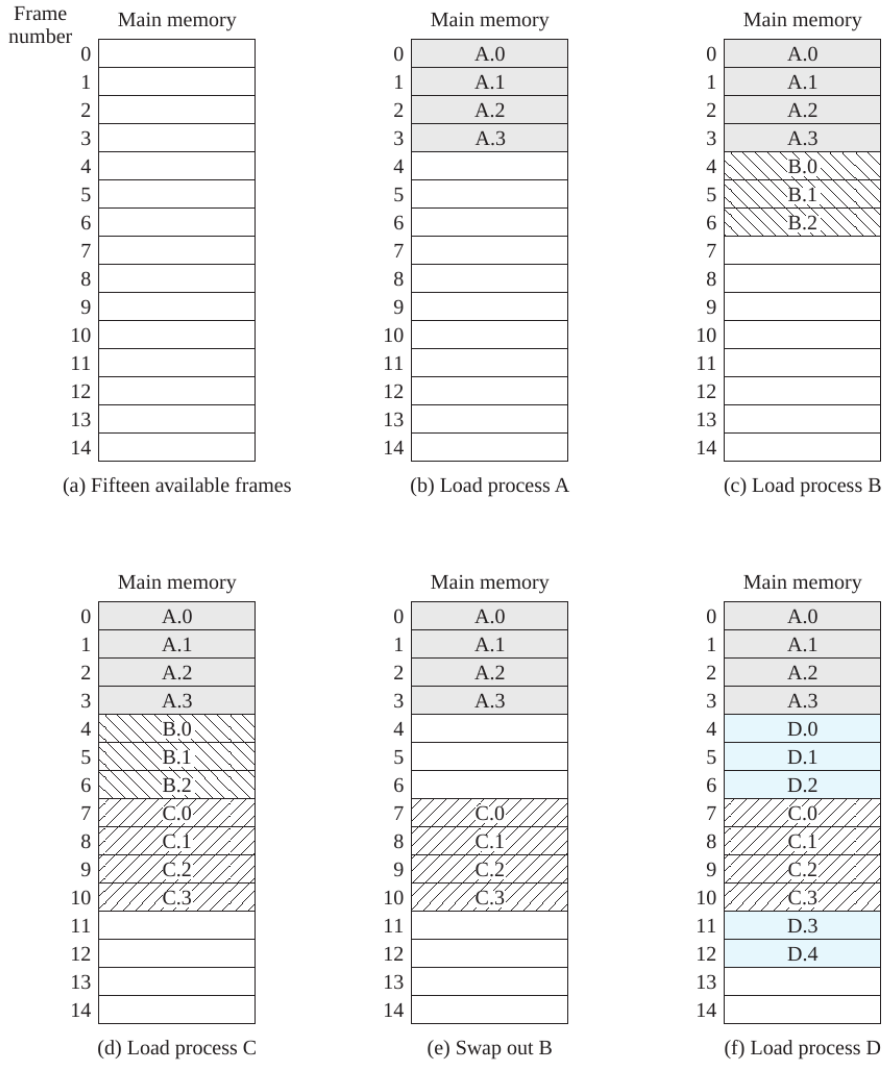
Podela na particije, bilo fiksno, bilo dinamički je **neefikasno** u smislu unutrašnje, odnosno spoljašnje fragmentacije.

## Novi pristup - straničenje

- Delovi procesa, koji se zovu **stranice (pages)**, dodeljuju se odgovarajućim raspoloživim delovima memorije koji se zovu **okviri (frames)**.
- Mora postojati evidencija o dodeljenim okvirima. Za tu svrhu se koristi **tabela stranica**



# Primer straničenja




(a) Fifteen available frames

(b) Load process A

(c) Load process B

(d) Load process C

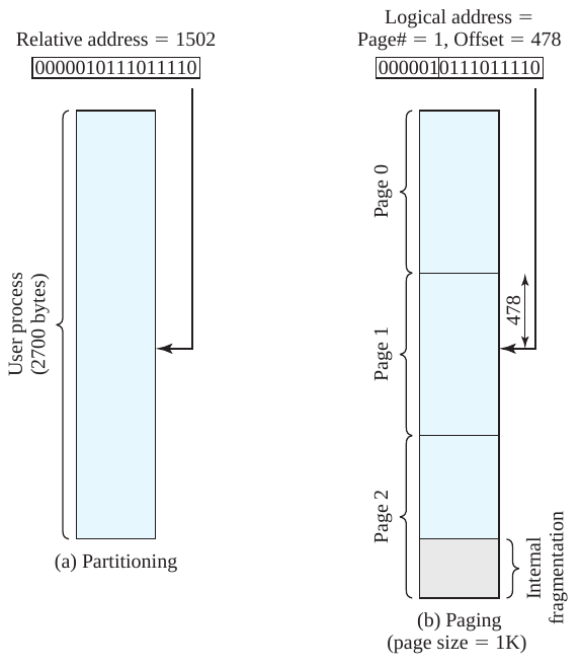
(e) Swap out B

(f) Load process D

# Logičke adrese kod particionisanja i straničenja

## Objašnjenje primera

Primer na slici prikazuje situaciju kada imamo 16-bitne adrese (ukupno 64K), koje su podeljene na stranice od po 1K. Relativna adresa 1502 je binarno 0000010111011110. Znači, 10 bitova za adresu u 6 bitova za indeks stranice. Adresa 1502 odgovara pomeraju za 478 (0111011110) na stranici 1 (000001).



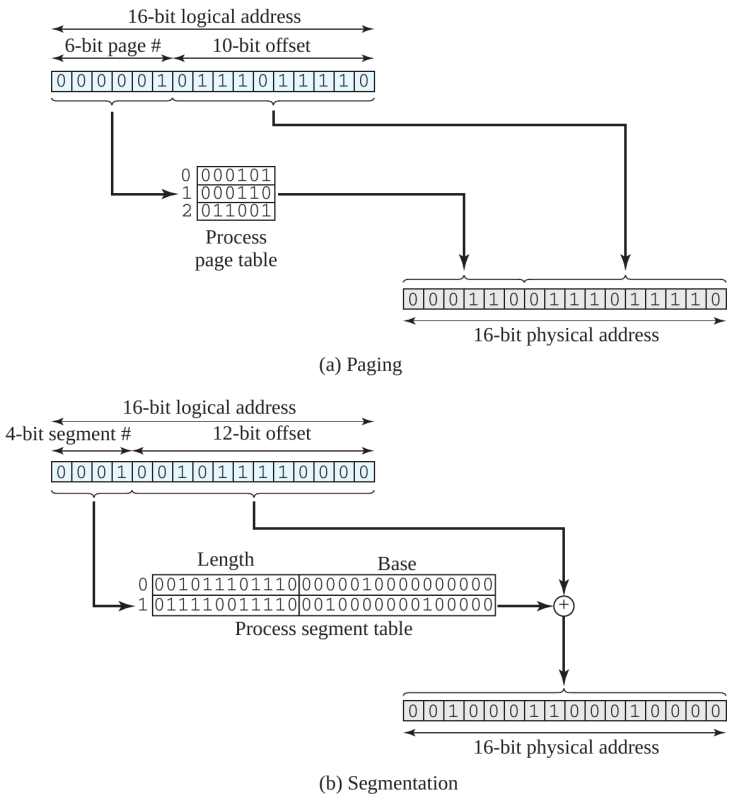
# Segmentacija

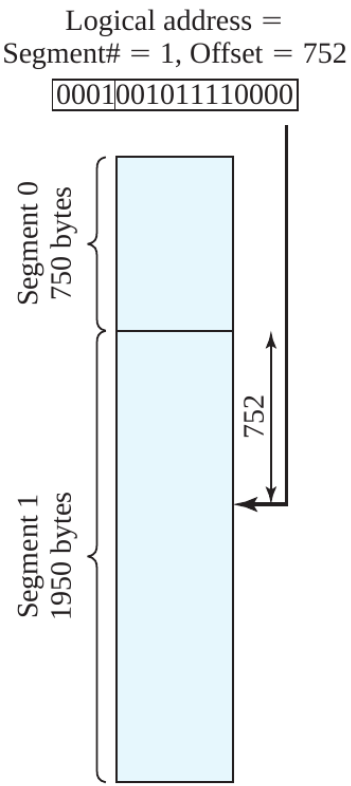
- Korisnički program može da se podeli tako što se kod i podaci raščlane na izvestan broj segmenata.
- Dok je straničenje obično nevidljivo za programera, segmentacija je vidljiva i predstavlja dodatnu pogodnost kod **modularnog programiranja**
- Logička adresa se sastoji iz dva dela, **broja segmenta** i **pomeraja**
- Pošto segmenti nisu jednake veličine, proračun efektivne adrese je malo složeniji, uz referenciranje tabele segmenata

# Segmentacija - primer

## Objašnjenje primera

Gore je dato formiranje fizičke adrese kod straničenja, a dole kod segmentacije. Kod segmentacije je uzet 4-bitni broj segmenta i 12-bitni pomeraj.



# Sigurnosni problemi

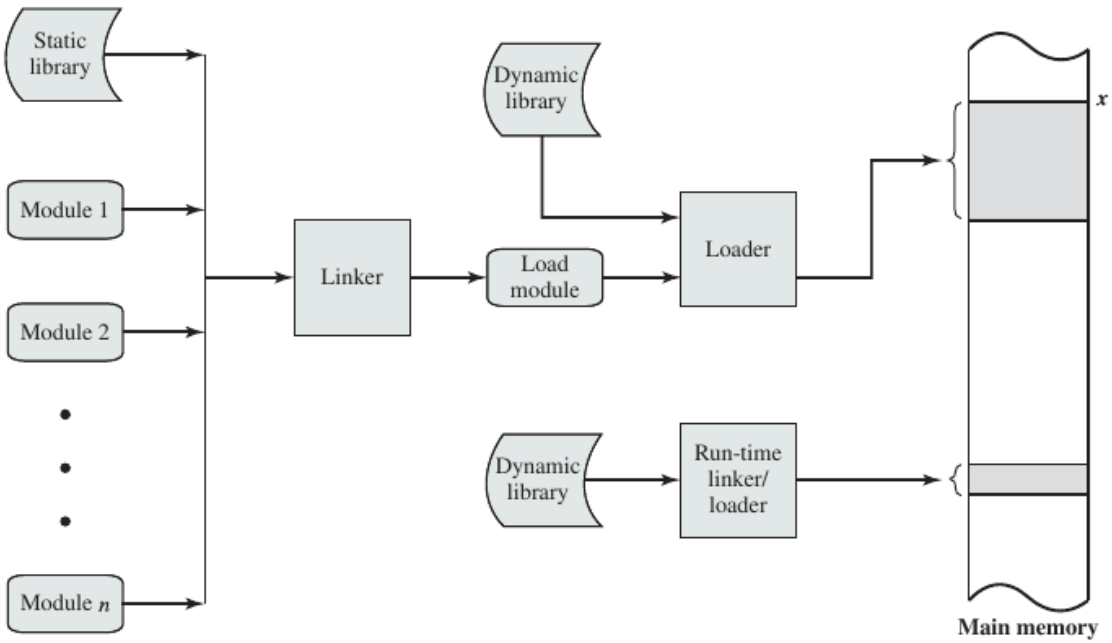
*Buffer overflow (buffer overrun)*

```
int main (int argc, char *argv[ ])
{
  int valid=FALSE;
  char str1[8];
  char str2[8];
  strcpy(str1,"START");
  gets(str2);
  if (strncmp(str1,str2,8)==0)
    valid=TRUE;
  printf("buffer: str1(%s),str2(%s),valid(%d) \n", str1, str2, valid);
}
```

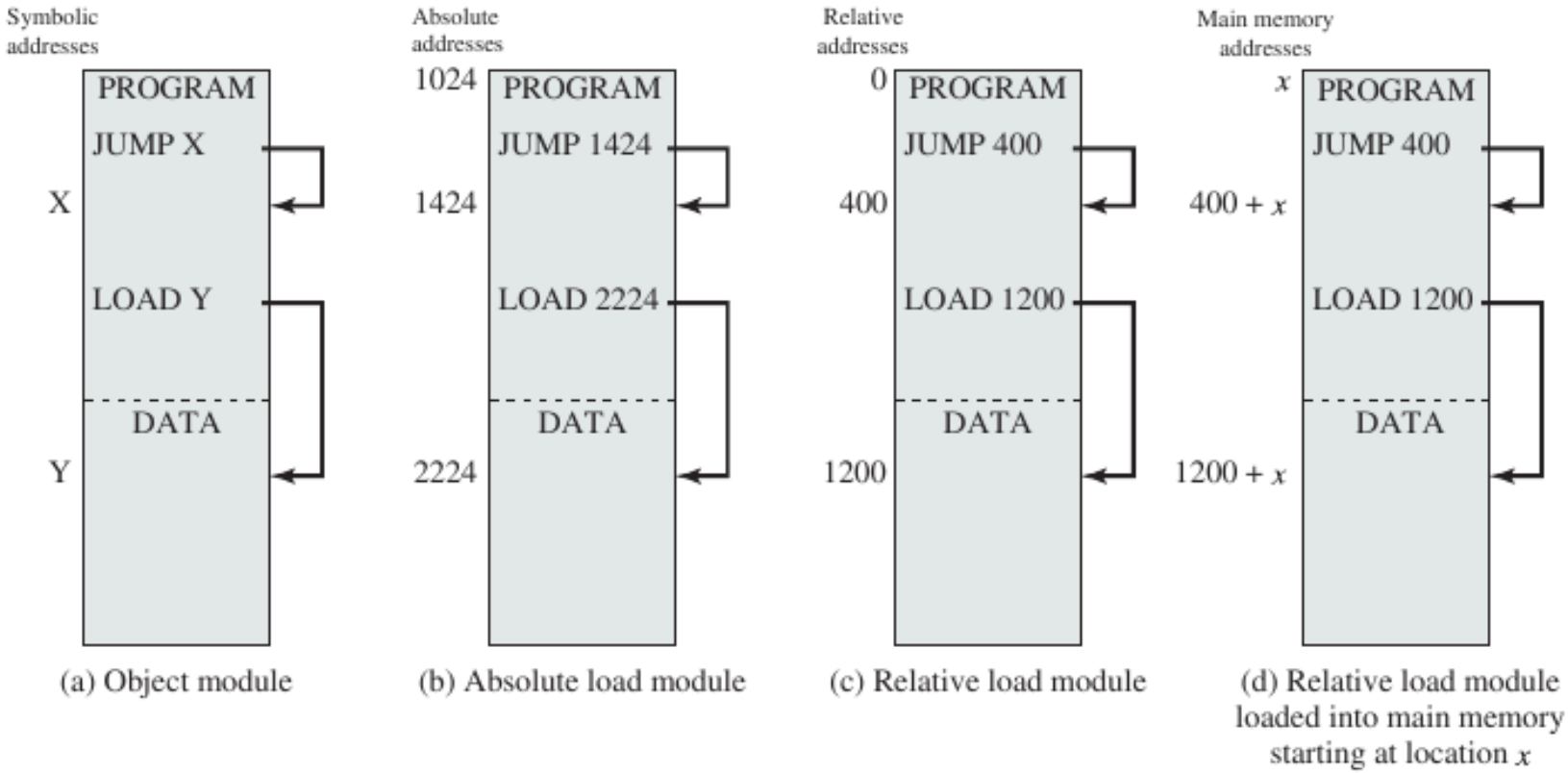
```
$ gcc -o primer primer.c -m32
$ ./primer
START
buffer: str1(START),str2(START),valid(1)
$ ./primer
STARTSTART
buffer: str1(RT),str2(STARTSTART),valid(0)
```

# Učitavanje i povezivanje (*Loading & Linking*)

1. Apsolutno učitavanje
2. Relativno učitavanje
3. Dinamičko učitavanje u vreme izvršavanja



# Apsolutno i relativno učitavanje




(a) Object module (b) Absolute load module (c) Relative load module (d) Relative load module loaded into main memory starting at location $x$

# Funkcija povezivanja (*Linker*)

1. **Statičko povezivanje**
2. **Dinamičko povezivanje** u vreme učitavanja i u vreme izvršavanja
   - Postaje lakše da se ugrade promene modula
   - Automatsko deljenje koda među procesima
   - Proširivanje funkcionalnosti OS-a