

# Оперативни Системи

## Управување со меморија

Вон. Проф. Д-р Димитар Трајанов

Вон. проф. Д-р Невена Ацковска

Доц. Д-р Боро Јакимовски



# Содржина

- ▶ Вовед
- ▶ Swapping
- ▶ Виртуелна меморија
- ▶ Алгоритми за замена на страници
- ▶ Дизајнирање на системот за страничење
- ▶ Имплементација на системот за страничење
- ▶ Сегментација

# Управување со меморијата

- ▶ Идеалната меморија
  - Голема
  - Брза
  - Евтина
- ▶ Мемориска хиерархија
- ▶ Управувач со меморијата
  - Управува со мемориската хиерархија во насока на градење на идеална меморија

# Зошто е важно управувањето со меморија?

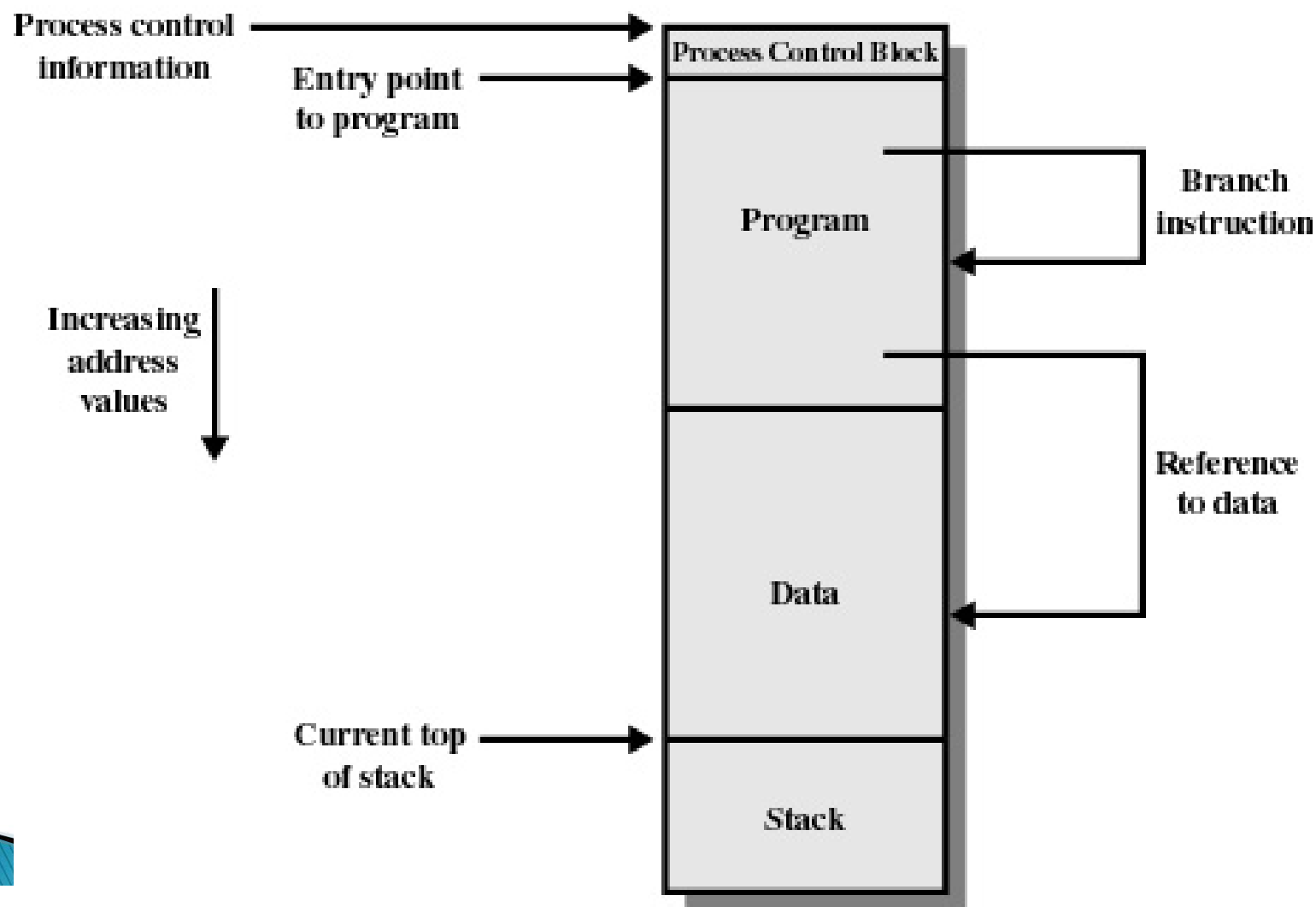
- ▶ Меморијата е важен ресурс и мора внимателно да се управува (раководи)
- ▶ Parkinson-ов закон: „Програмите се зголемуваат со тенденција да ја пополнат меморијата која им е на располагање“
- ▶ Меморијата мора да се алоцира ефикасно за да се постават што повеќе процеси во меморија
- ▶ Треба да се обезбеди да се изведе програма чијашто големина може да е поголема од реалната меморија што ни е на располагање

# Управувањето со меморија треба да овозможи (1)

## ► Релокација

- Програмерот не знае каде ќе биде сместен програмот во меморијата кога ќе биде извршуван
- Кога програмот се извршува, тој може да е сместен назад на дискот (swapped), а потоа да е вратен во главната меморија на различна локација (релоциран)
- Мемориските референци (гранање на наредби, податоци...) во кодот мора да бидат преведени во вистински физички адреси

# Адресни побарувања за некој процес



# Управувањето со меморија треба да овозможи (2)

## ► Заштита

- Процесите не смее да референцираат мемориски локации во простор на друг процес без дозвола
- Не е можно да се проверат апсолутните адреси во програмите бидејќи процесот може да биде релоциран.
- Мора да бидат проверени за време на извршување
  - ОС не може да ги предвиди сите мемориски референци кои процесот ќе ги направи

# Управувањето со меморија треба да овозможи (3)

## ► Делење (Sharing)

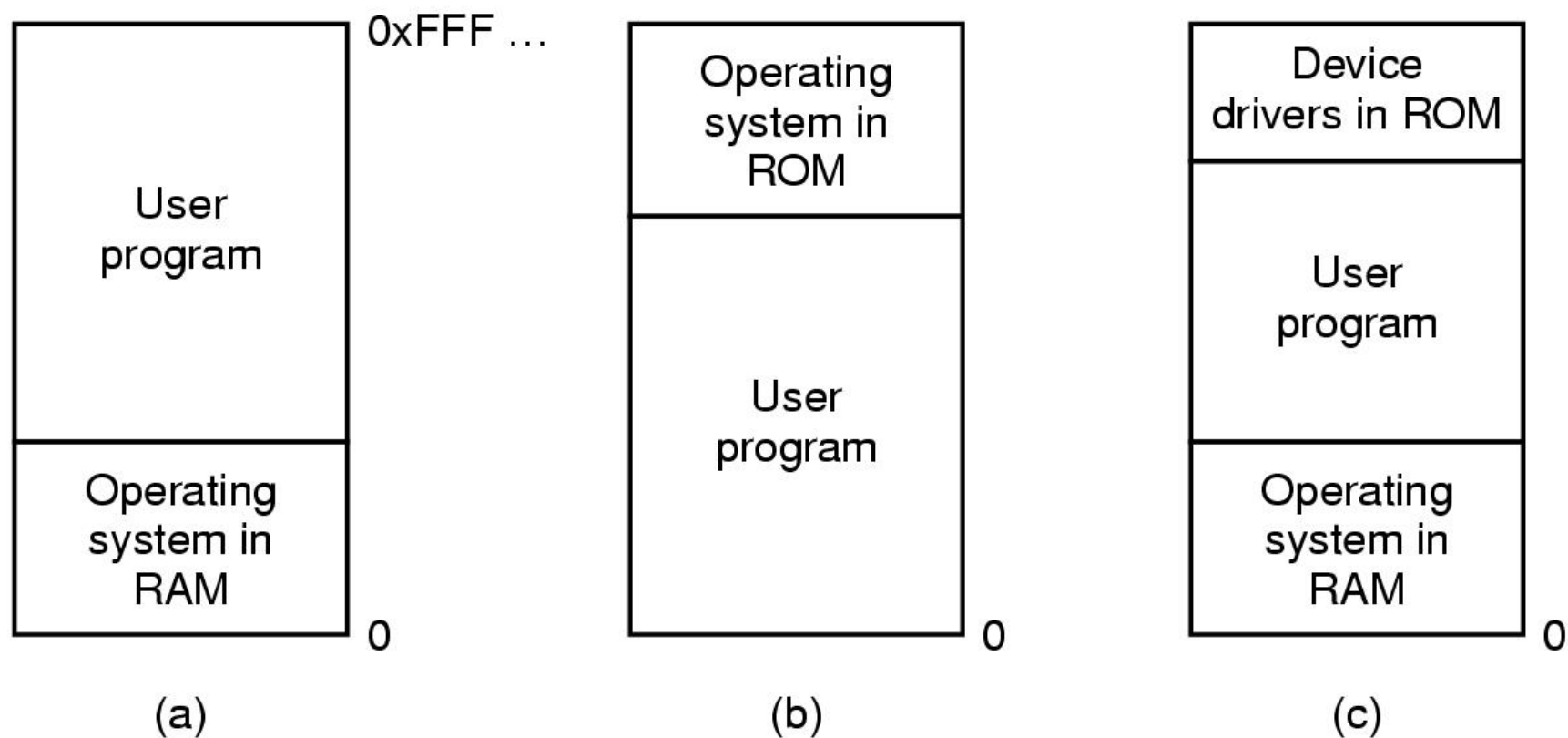
- Дозволи неколку процеси да пристапат до исто парче на меморијата
- Подобро да се дозволи секој процес да пристапи до истата копија од програмот (да чита) отколку да има своја посебна копија



# Управувач со меморија

- ▶ Делот на ОС што раководи со меморијата (**memory manager**)
  - води сметка кои делови од меморијата се користат
  - доделува – алоцира меморија на процеси
  - одзема – деалоцира меморија од процеси
  - замена (swapping) меѓу меморија и диск

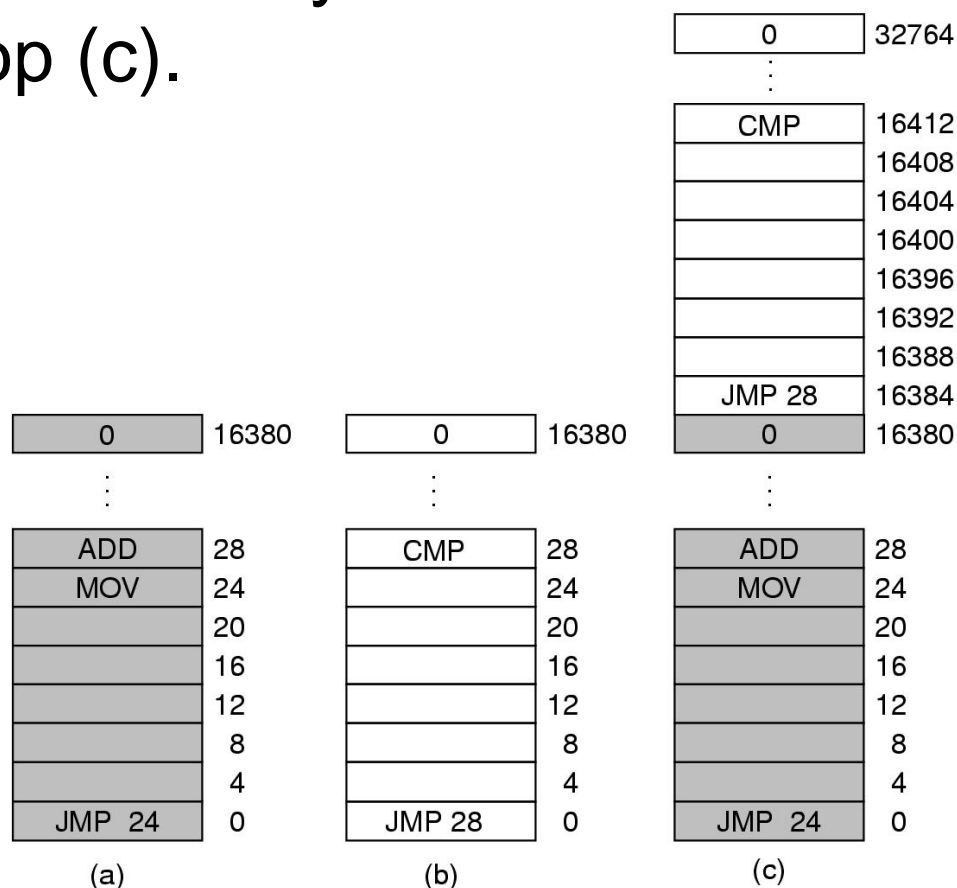
# Без мемориска апстракција



Организирање на меморијата со ОС и еден кориснички програм

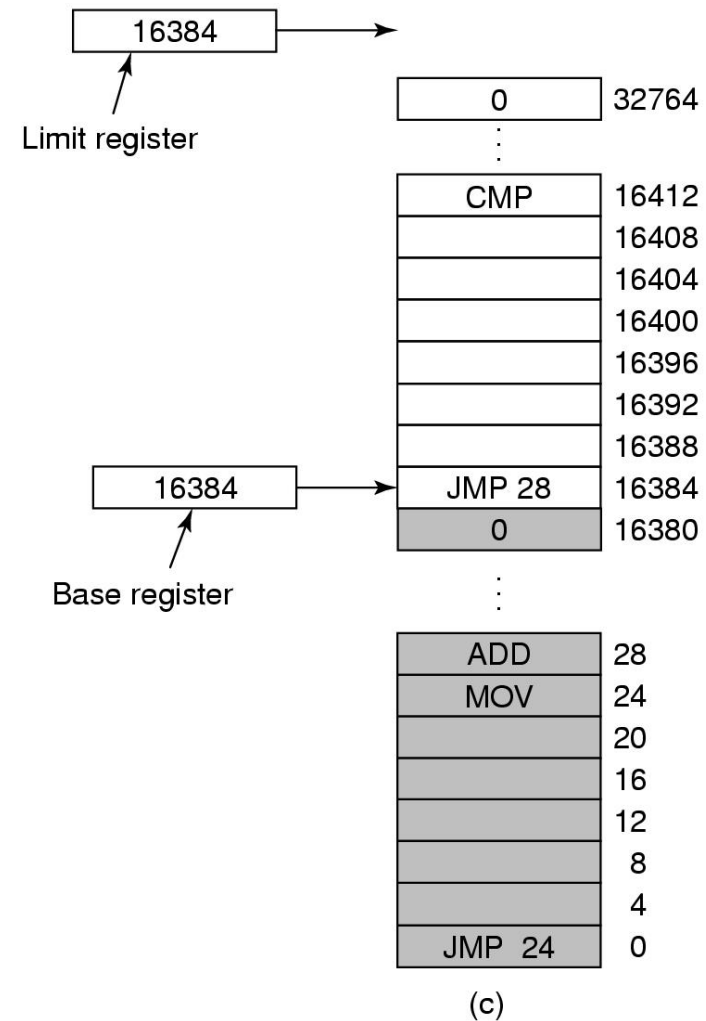
# Извршување на повеќе програми без мемориска апстракција

- ▶ Проблем на реалокација: Имаме две програми (a) и (b) и истите ги чуваме во ист адресен простор (c).
- ▶ Решение е статичка релокација (додај го првиот алоциран бајт на секоја адреса во програмата)

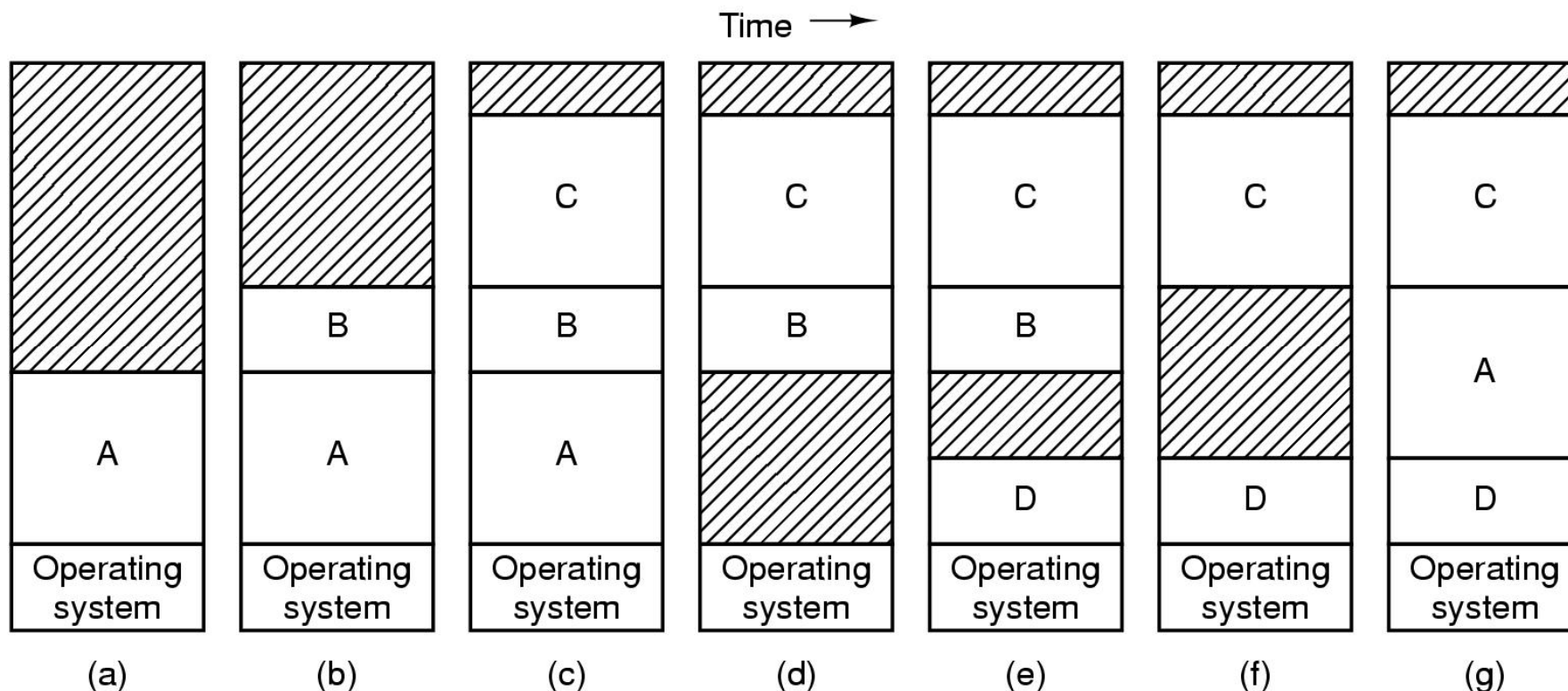


# Основен и граничен регистар (Base & Limit)

- ▶ Base и limit регистрите се користат за на секој процес да му се додели различен адресен простор

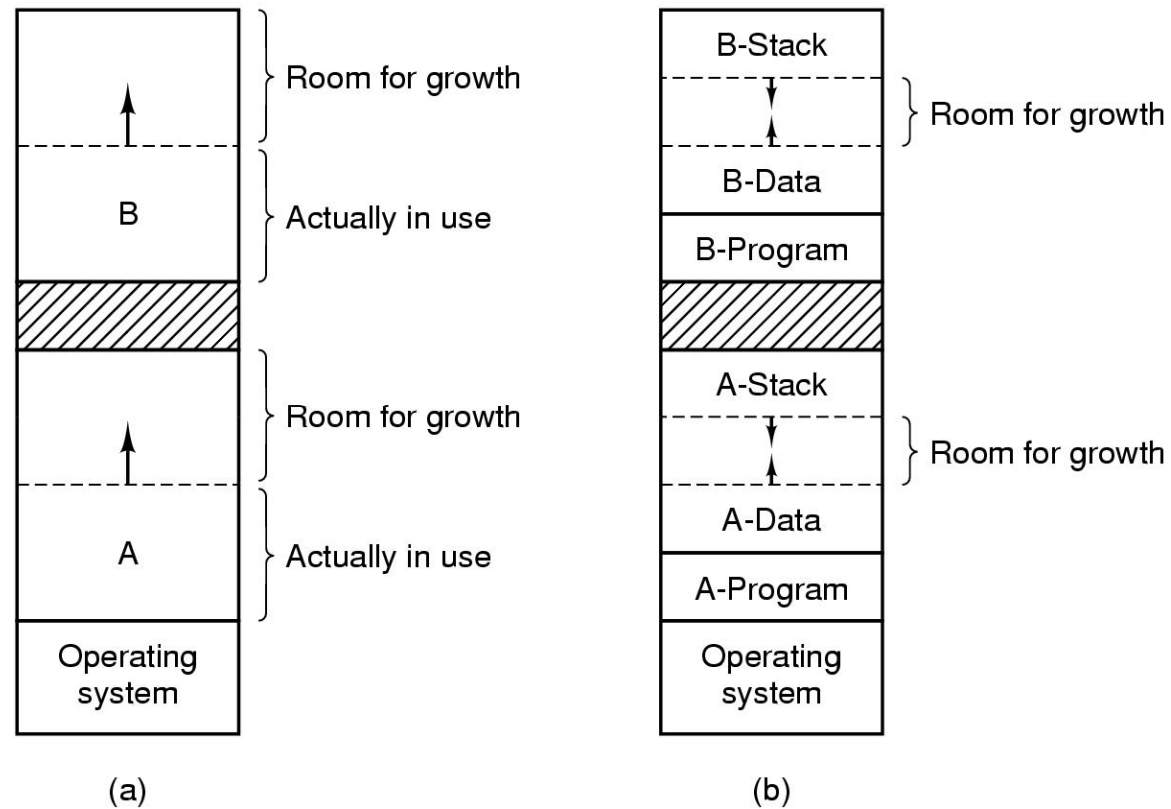


# Замена на процеси од RAM (Swapping)



- ▶ Зафатеноста на меморијата се менува со
  - Доаѓањето на процес во меморијата
  - Отстранувањето на процес од меморијата

# Swapping



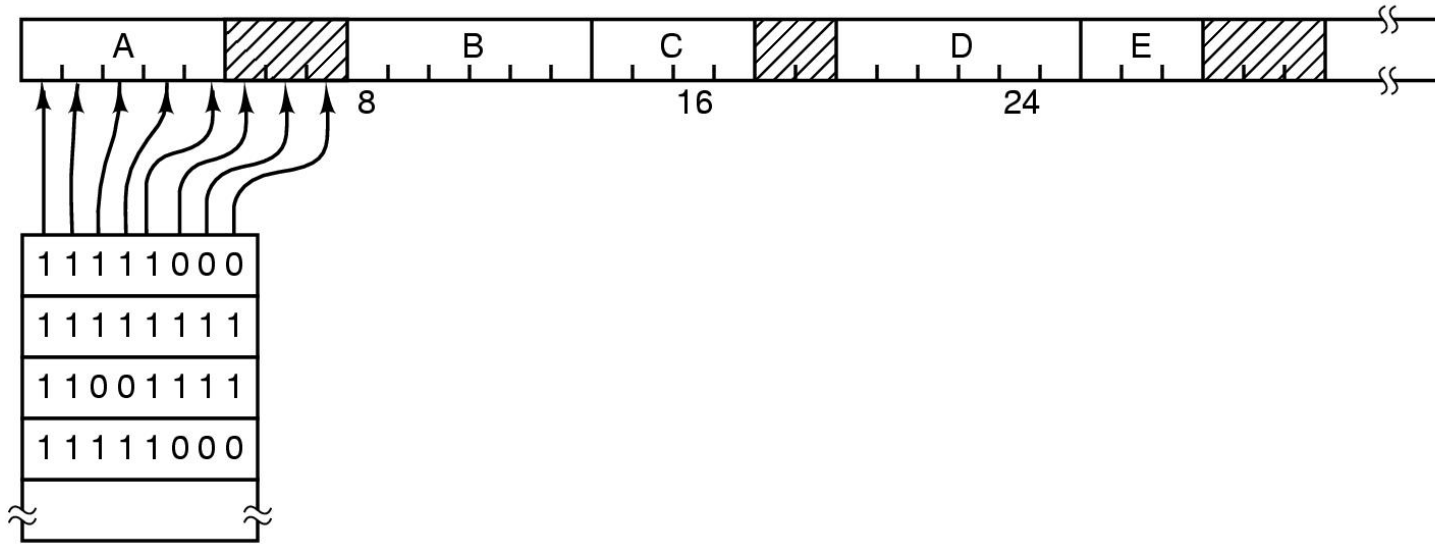
- Резервирање на простор за сегменти кои динамички може да растат

# Управување со слободната меморија

- ▶ При динамичко управување со меморија ОС мора да управува и со слободниот простор
- ▶ Две методи
  - Битмапи
  - Поврзани листи



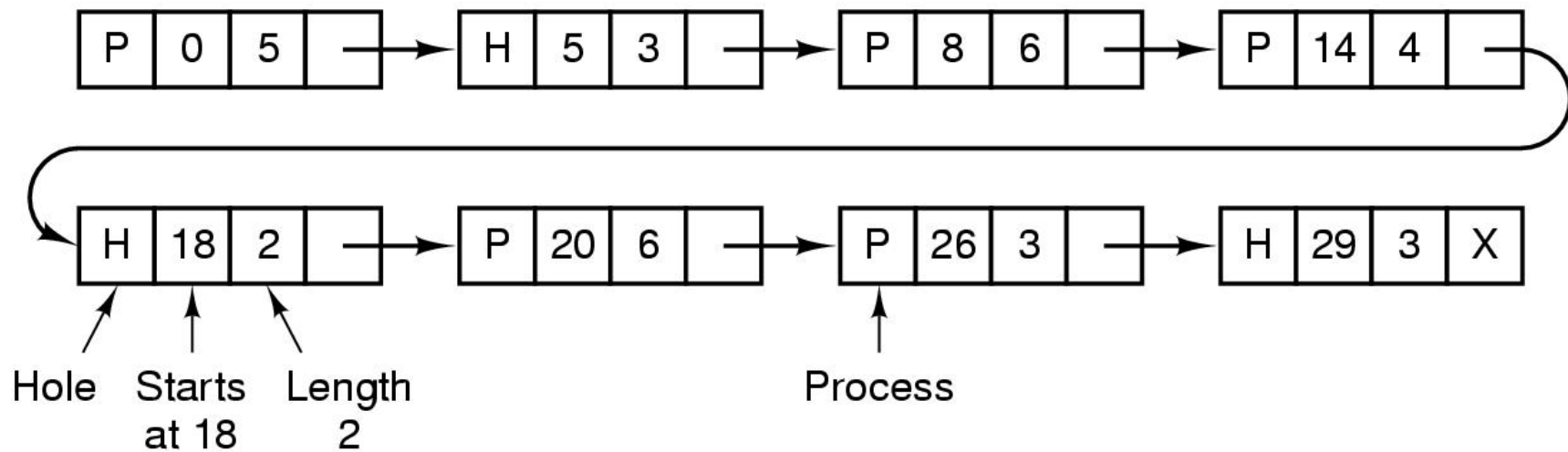
# Управување на меморијата со бит мапи



- ▶ Меморијата се дели на алокациски единици
- ▶ За секоја алокациска единица постои еден бит кој покажува дали е зафатена

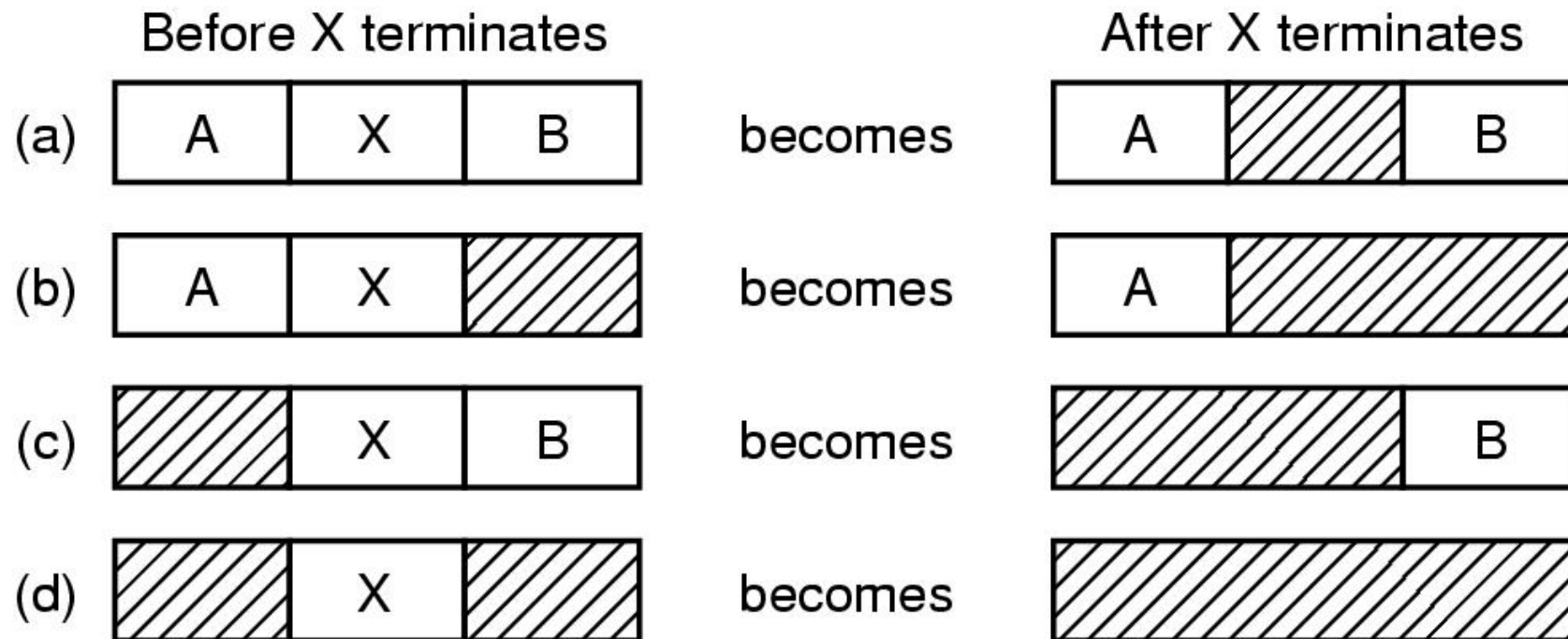


# Управување на меморијата со ЛИСТИ



- ▶ Меморијата се дели на алокациски единици
- ▶ За секој блок од меморија зафатен од процес и за секој слободен блок постои јазол кој го покажува неговиот почеток и должина
- ▶ Начин на алокација на нов блок
  - FIRST FIT
  - NEXT FIT
  - BEST FIT
  - WORST FIT
  - QUICK FIT

# Процес на спојување на слободните мемориски блокови

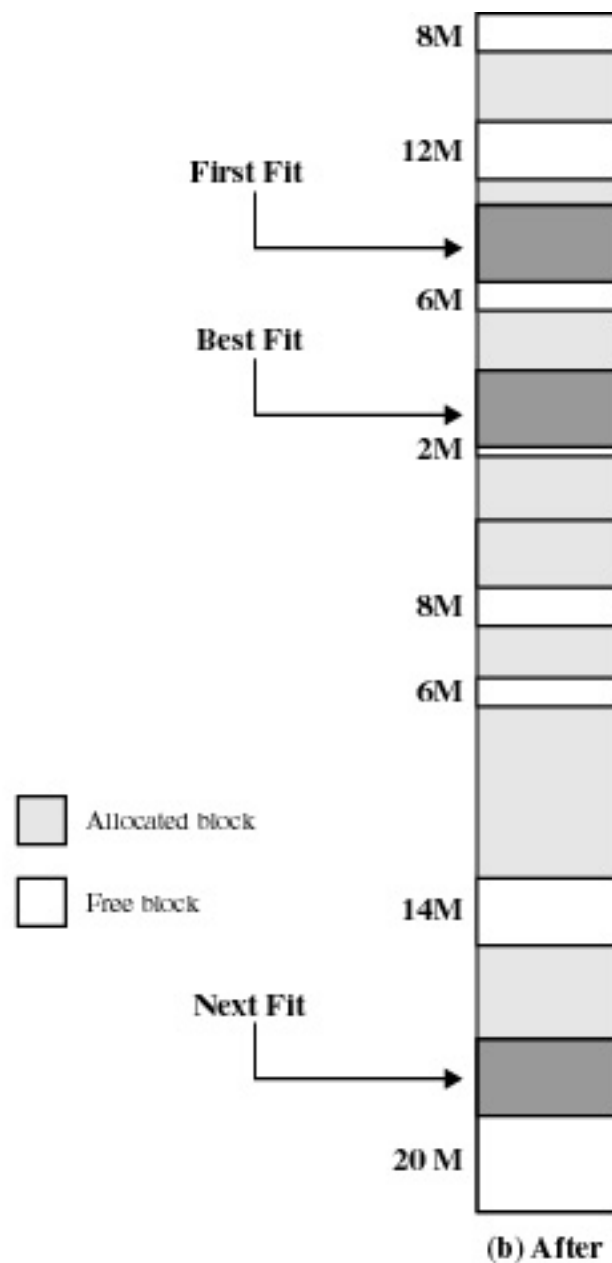
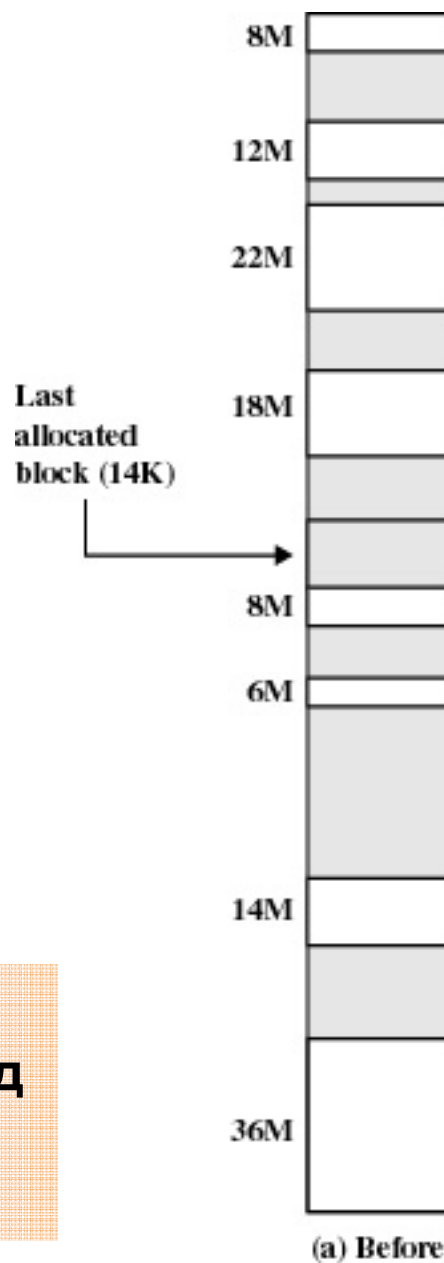


- ▶ Приказ на спојувањето на слободните мемориски блокови ако даден блок X се ослободи

# Примери

## First, Next, Best fit

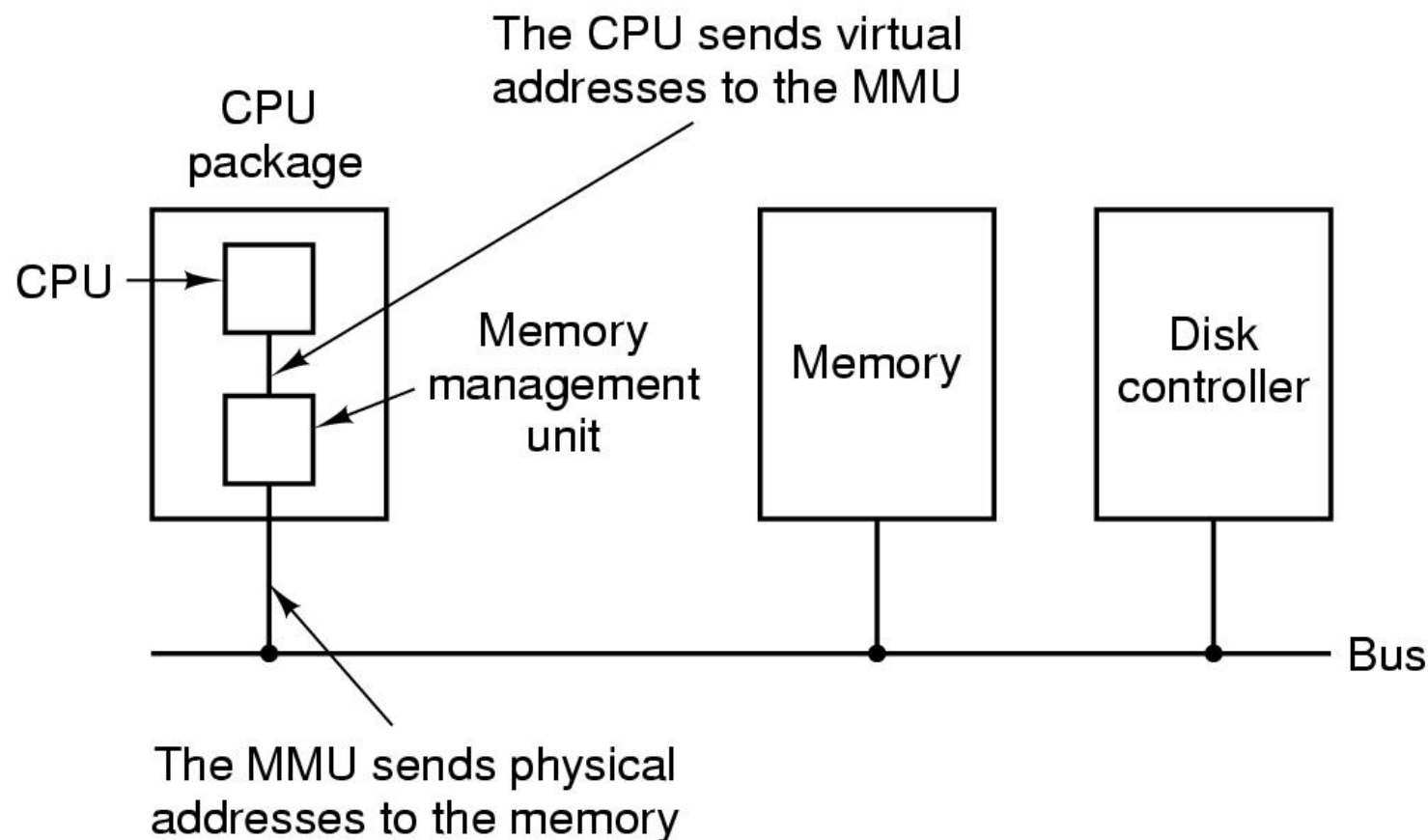
Мемориска  
конфигурација пред  
и по алоцирање на  
16 Mbyte блок



# Виртуелна меморија

- ▶ Овозможува проширување на адресниот простор над капацитетот на главната меморија
- ▶ Главна идеја – секој програм има свој адресен простор кој се дели на делчиња наречени страници.
- ▶ Страниците се мапираат во физичката меморија, но не секоја страница мора да е во RAM кога ќе е потребна
  - Кога не е се прави акција на преземање на делот од програмата од диск (и повторно да ја изврши наредбата која претходно се обидел да ја изврши)

# Виртуелна меморија



# Страничење

- ▶ Подели ја меморијата во мали, еднакво големи делчиња и секој процес подели го на парчиња исто толку големи
- ▶ Делчињата на процесите се викаат страници, а делчињата од меморијата се викаат рамки
- ▶ ОС одржува табела на страници за секој процес
  - Ја содржи локацијата на рамката за секоја страница на процесот
  - Секоја мемориска референца се состои од број на страница и офсет (поместување) во самата страница

| Frame number | Main memory |
|--------------|-------------|
| 0            |             |
| 1            |             |
| 2            |             |
| 3            |             |
| 4            |             |
| 5            |             |
| 6            |             |
| 7            |             |
| 8            |             |
| 9            |             |
| 10           |             |
| 11           |             |
| 12           |             |
| 13           |             |
| 14           |             |

(a) Fifteen Available Frames

| Frame number | Main memory |
|--------------|-------------|
| 0            | A.0         |
| 1            | A.1         |
| 2            | A.2         |
| 3            | A.3         |
| 4            |             |
| 5            |             |
| 6            |             |
| 7            |             |
| 8            |             |
| 9            |             |
| 10           |             |
| 11           |             |
| 12           |             |
| 13           |             |
| 14           |             |

(b) Load Process A

| Frame number | Main memory |
|--------------|-------------|
| 0            | A.0         |
| 1            | A.1         |
| 2            | A.2         |
| 3            | A.3         |
| 4            | B.0         |
| 5            | B.1         |
| 6            | B.2         |
| 7            |             |
| 8            |             |
| 9            |             |
| 10           |             |
| 11           |             |
| 12           |             |
| 13           |             |
| 14           |             |

(c) Load Process B

| Frame number | Main memory |
|--------------|-------------|
| 0            | A.0         |
| 1            | A.1         |
| 2            | A.2         |
| 3            | A.3         |
| 4            | B.0         |
| 5            | B.1         |
| 6            | B.2         |
| 7            | C.0         |
| 8            | C.1         |
| 9            | C.2         |
| 10           | C.3         |
| 11           |             |
| 12           |             |
| 13           |             |
| 14           |             |

(d) Load Process C

| Frame number | Main memory |
|--------------|-------------|
| 0            | A.0         |
| 1            | A.1         |
| 2            | A.2         |
| 3            | A.3         |
| 4            |             |
| 5            |             |
| 6            |             |
| 7            | C.0         |
| 8            | C.1         |
| 9            | C.2         |
| 10           | C.3         |
| 11           |             |
| 12           |             |
| 13           |             |
| 14           |             |

(e) Swap out B

| Frame number | Main memory |
|--------------|-------------|
| 0            | A.0         |
| 1            | A.1         |
| 2            | A.2         |
| 3            | A.3         |
| 4            | D.0         |
| 5            | D.1         |
| 6            | D.2         |
| 7            | C.0         |
| 8            | C.1         |
| 9            | C.2         |
| 10           | C.3         |
| 11           | D.3         |
| 12           | D.4         |
| 13           |             |
| 14           |             |

(f) Load Process D





# ... А табелите на страници се

|   |   |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

Process A  
page table

|   |   |
|---|---|
| 0 | — |
| 1 | — |
| 2 | — |

Process B  
page table

|   |    |
|---|----|
| 0 | 7  |
| 1 | 8  |
| 2 | 9  |
| 3 | 10 |

Process C  
page table

|   |    |
|---|----|
| 0 | 4  |
| 1 | 5  |
| 2 | 6  |
| 3 | 11 |
| 4 | 12 |

Process D  
page table

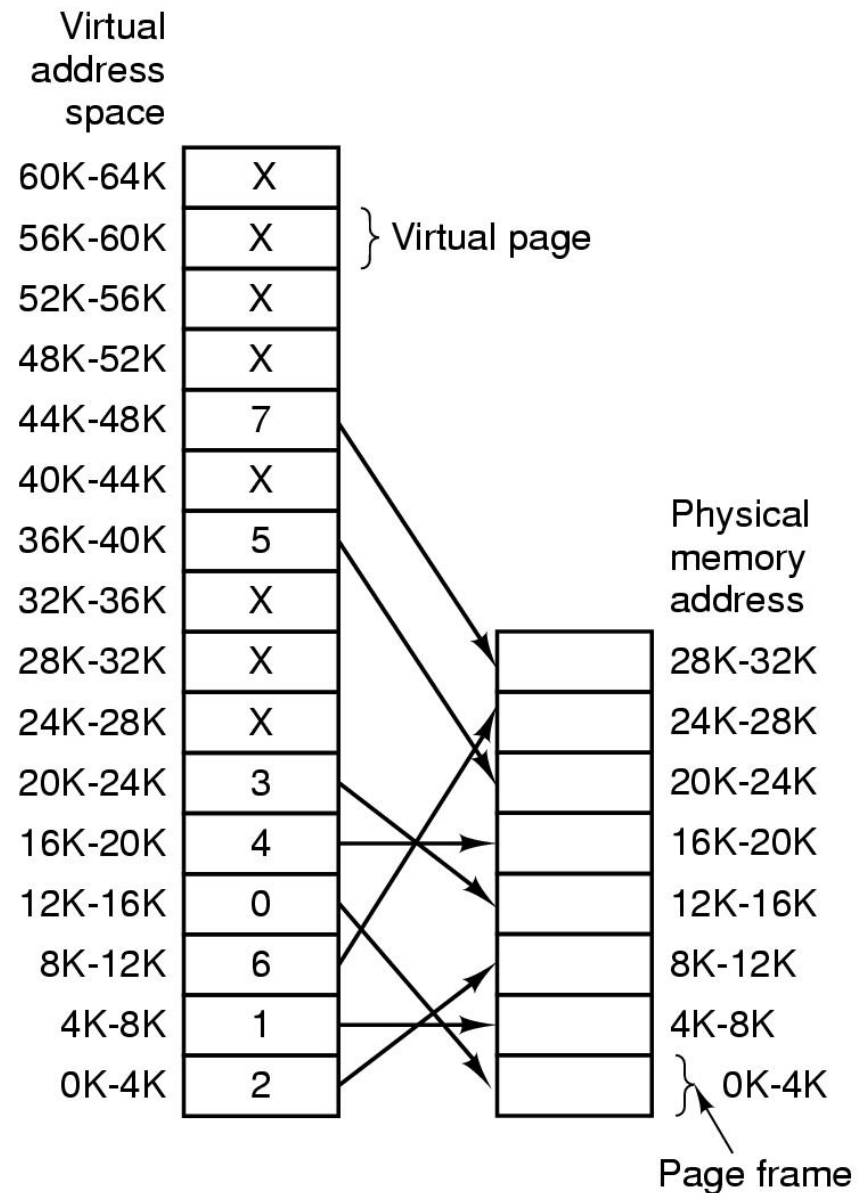
|    |
|----|
| 13 |
| 14 |

Free frame  
list



# Страничење

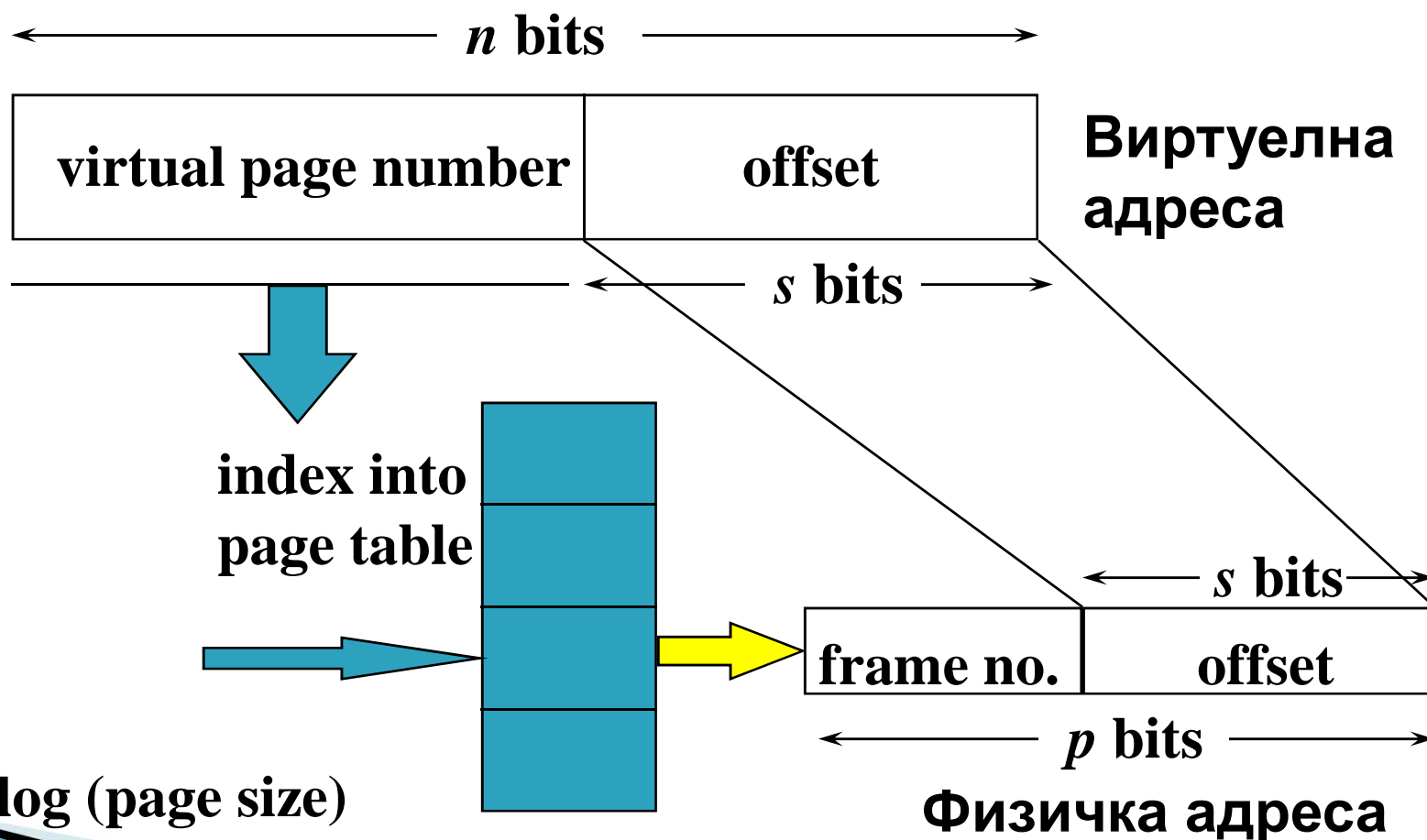
- ▶ Се користи табела на страници која врши мапирање на виртуелната во физичка адреса



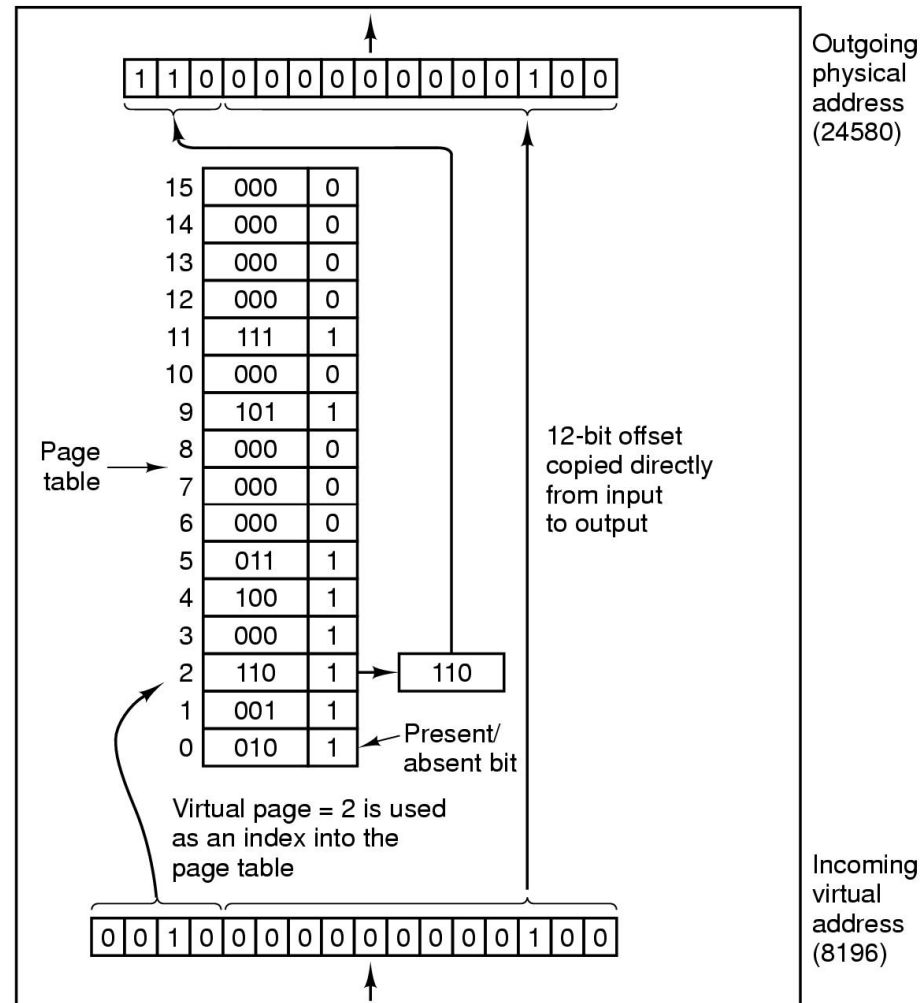
# Делови на виртуелна адреса

- ▶ Секоја адреса генерирана од CPU (виртуелна адреса) се дели на 2 дела
  - број на страница (page number) – **p**
  - поместување во страницата (page offset) – **d**
- ▶ Бројот на страницата се користи како индекс во **табелата на страници** (page table)
  - содржи **базна адреса** на секоја страница во физичката меморија
  - **базната адреса** се комбинира со **поместувањето** на страницата за да се дефинира физичката адреса што се испраќа на мемориската единица

# Пресликување од виртуелна во реална адреса

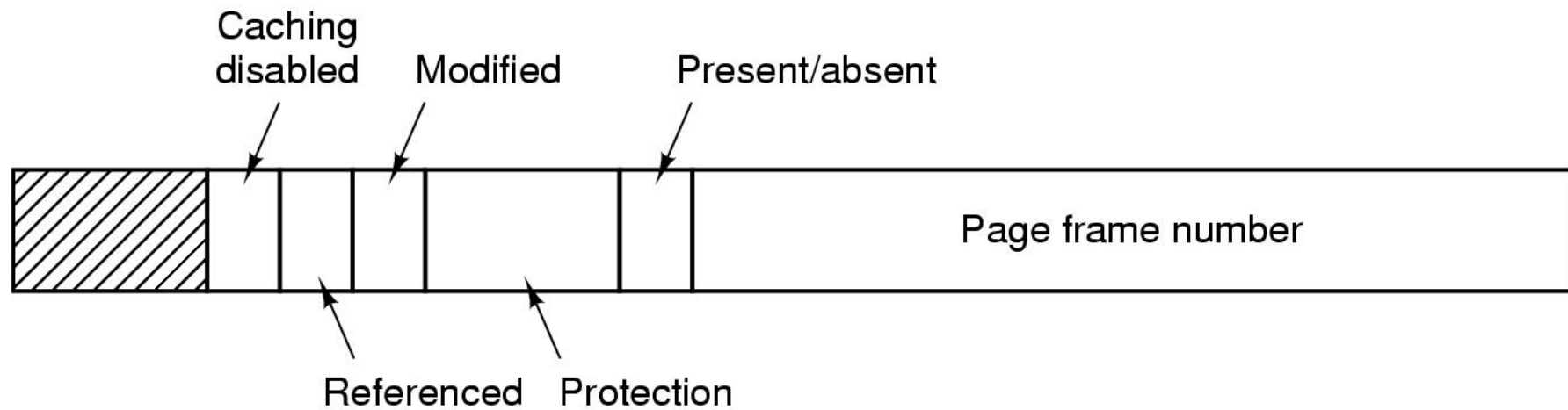


# Табела на страници

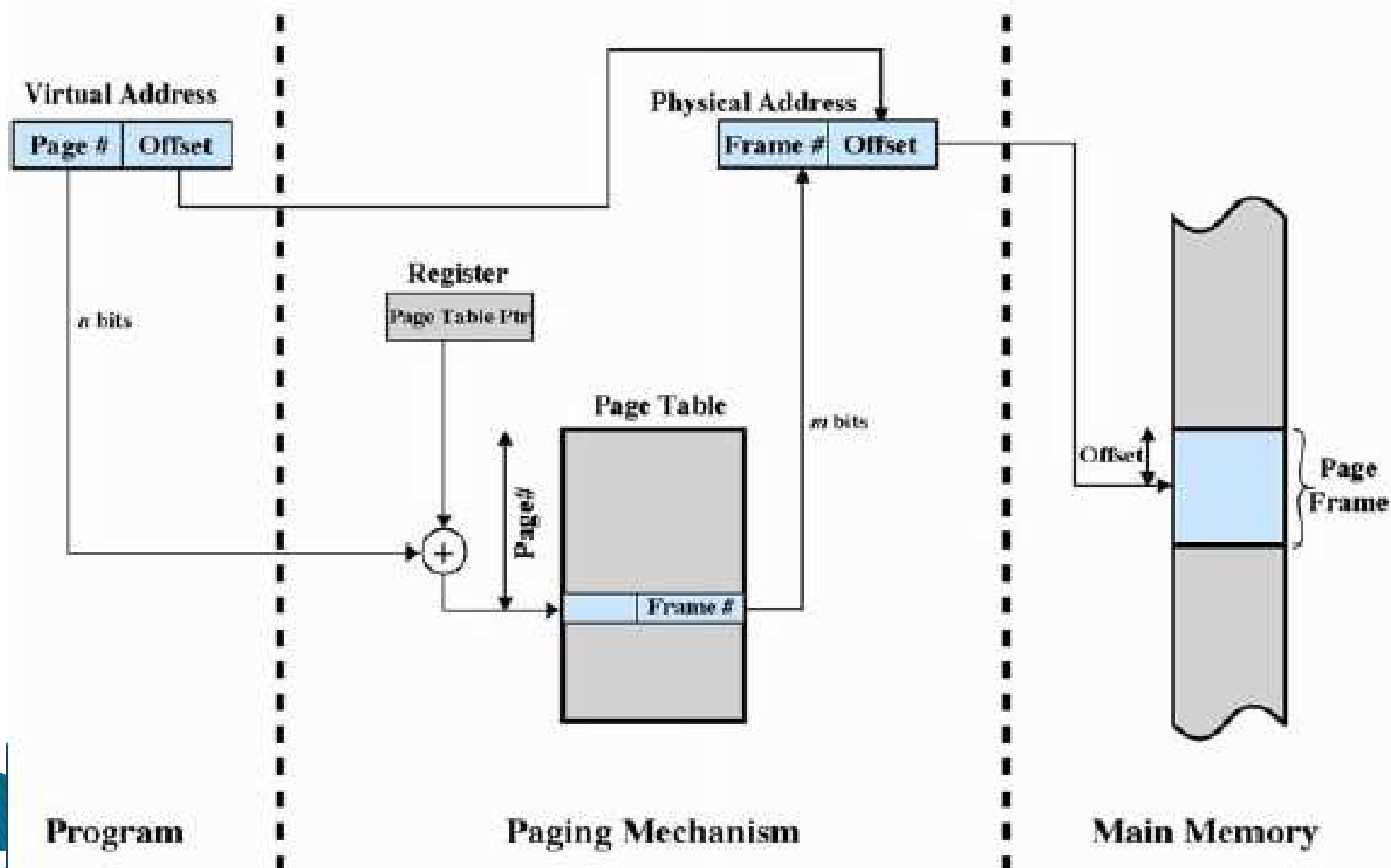


MMU со 16 страници со по 4 KB

# Типичен ред од табелата на страници



# Преведување на адреса во систем со страничење



Program

Paging Mechanism

Main Memory

# Пример (MMU)

- ▶ Големина на страница 8 KB =  $8 \cdot 1024 = 8192$  B
- ▶ Наредба (читање од меморија во регистар)

**MOVE REG, 104852**

поместување

$(104852 = 12 \cdot 8192 + 6548)$  ←

- ▶ Се проверува 13-тата редица во табела на страници
- ▶ Ако  $p = 0$  (Present/absent bit)
  - page fault (страницата не е во рамка)
- ▶ Ако  $p = 1$ 
  - се чита бројот на рамката во физичката меморија за 13-тата виртуелна страница – во случајов нека е 7
- ▶ Содржината на регистерот се праќа во мемориската локација  
 $7 \cdot 8192 + 6548 = 63892$  преку магистралата
- ▶ (Меморијата не знае ништо за оваа активност на MMU, туку само гледа барање за запис во локација 63892)

# TLBs – Translation Lookaside Buffers

- ▶ При пристап до меморија може потребно е често да се пристапува до табелата на страници за да може да се креира физичката адреса
- ▶ TLB претставува концепт на кеширање на најчесто користените редови од табелата на страници
- ▶ На пример Pentium може да чува 64 реда од табелата на страници во процесорот



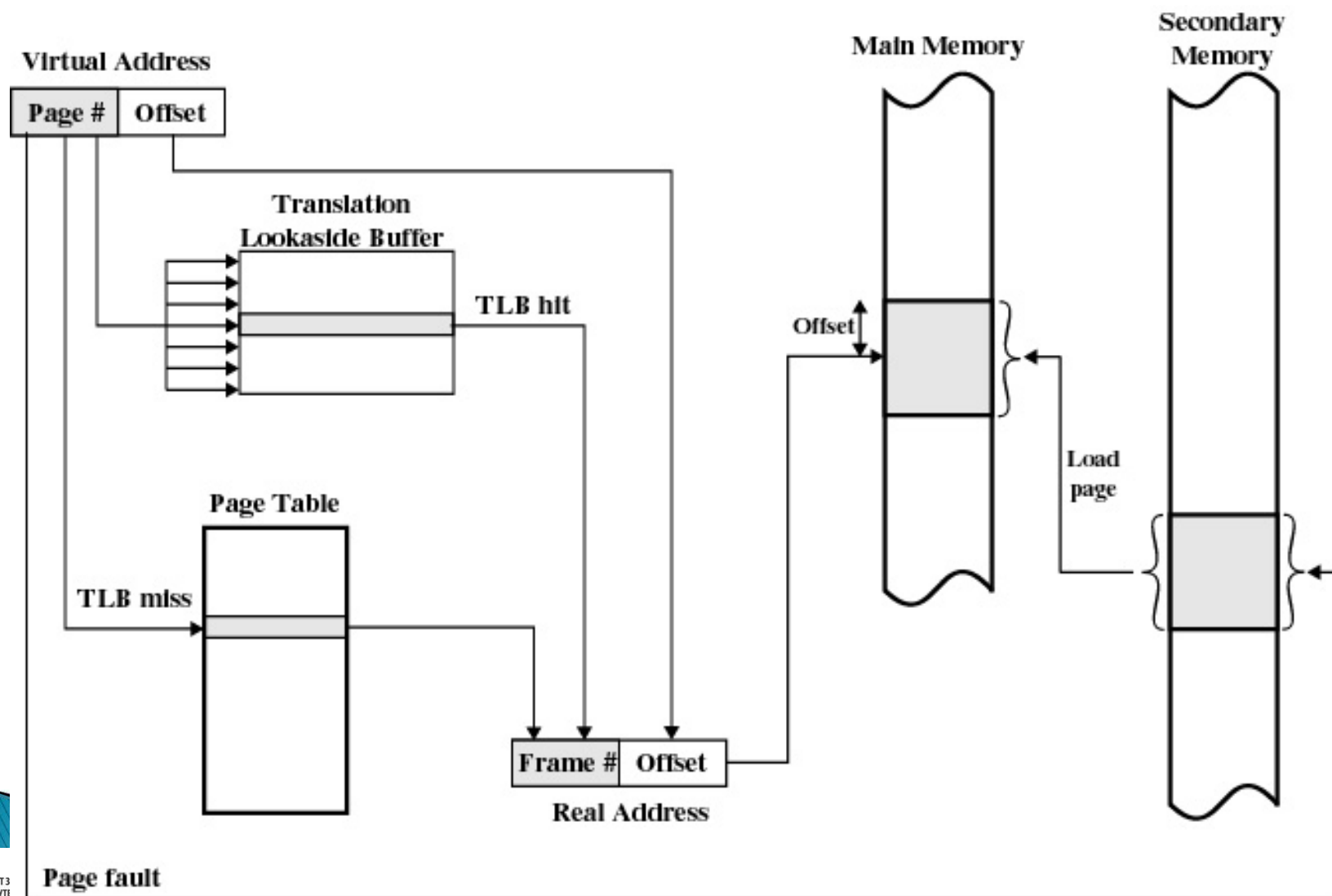
# Translation Lookaside Buffer

- ▶ Секоја референца на виртуелната меморија може да предизвика 2 пристапи до физичката меморија
  - еден за да се земе влезот од табелата на страници
  - еден за да се земе податокот
- ▶ За да се ублажи овој проблем, се воведува брз кеш за да во него се сместат некои податоци од табелата на страници
  - се нарекува **TLB – Translation Lookaside Buffer**
- ▶ TLB ги зачувува најскоро користените влезови од табелата на страници.
  - Ги зачувува броевите на виртуелните страници и пресликувањето за нив.

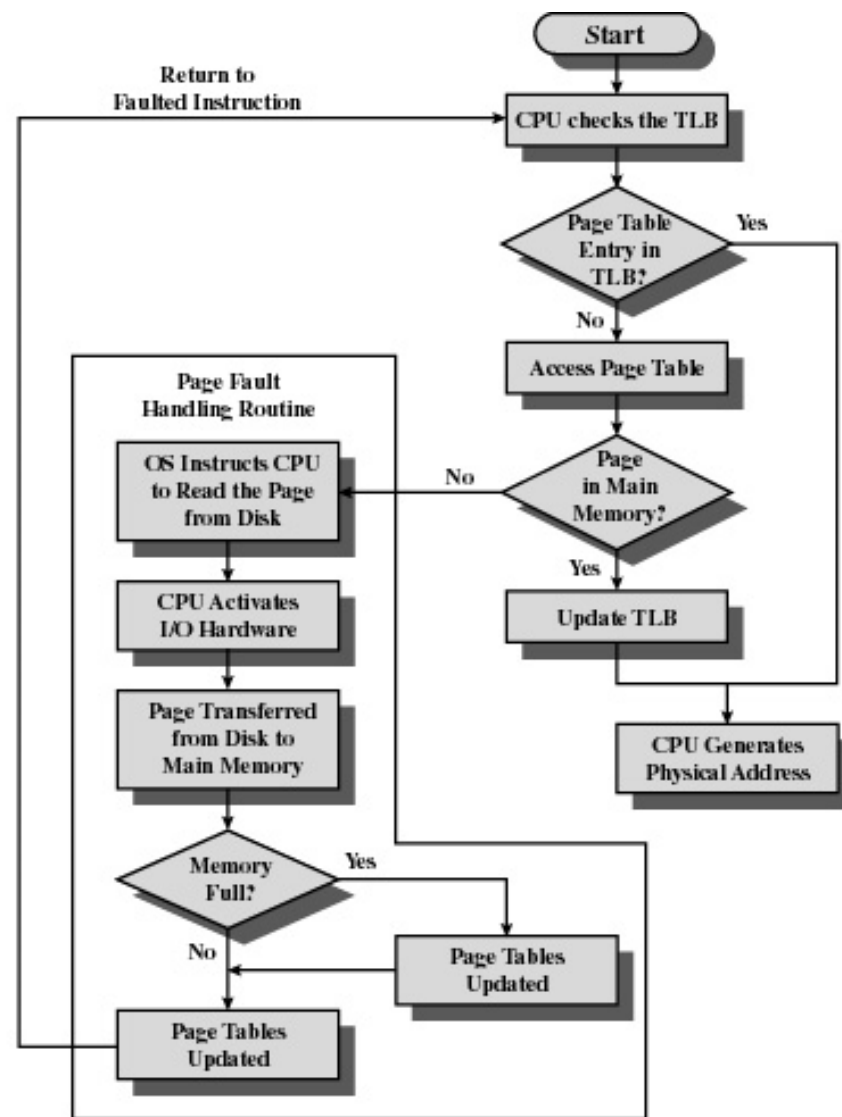
# Функционирање на TLB

- ▶ За дадена виртуелна адреса, процесорот го пребарува TLB
- ▶ Ако влезот од табелата на страници е присутен (имаме погодок, *hit*), бројот на рамката се презема оттаму и се формира реалната адреса
- ▶ Ако влезот од табелата на страници не е присутен во TLB (немаме погодок, *miss*), се користи бројот на страница за да се индексира табелата на страници, а TLB се ажурира за да го вклучи новиот влез

# Користење на TLB



# Операции на страничење и TLB




# Пример за изглед на TLB

| Valid | Virtual page | Modified | Protection | Page frame |
|-------|--------------|----------|------------|------------|
| 1     | 140          | 1        | RW         | 31         |
| 1     | 20           | 0        | R X        | 38         |
| 1     | 130          | 1        | RW         | 29         |
| 1     | 129          | 1        | RW         | 62         |
| 1     | 19           | 0        | R X        | 50         |
| 1     | 21           | 0        | R X        | 45         |
| 1     | 860          | 1        | RW         | 14         |
| 1     | 861          | 1        | RW         | 75         |

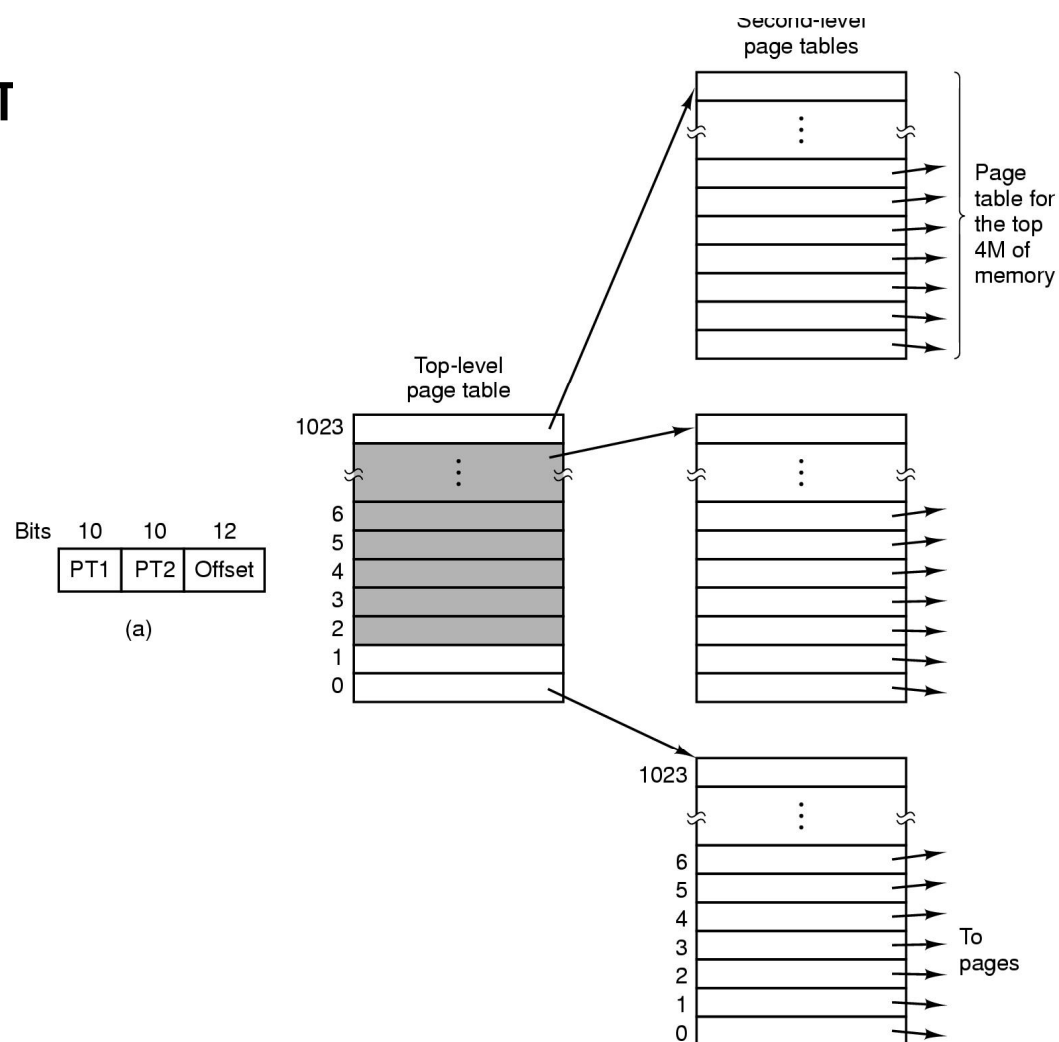
# Перформанси

- ▶ Делот од мемориските референци кои можат да бидат задоволени од асоцијативната меморија (TLB), се нарекува рата на погодување
- ▶ Колку е поголема, толку се подобри перформансите
- ▶ Пример:
  - 100 nsек за пристап до табела на страници
  - 20 nsек за пристап до TLB
  - 90% рата на погодување


$$0.9 * 20 + 0.1 * 100 = 28 \text{ nsек просечно време за пристап}$$

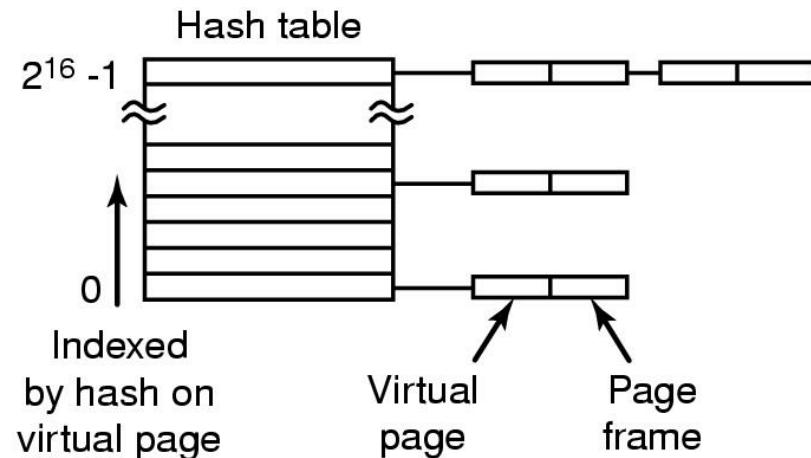
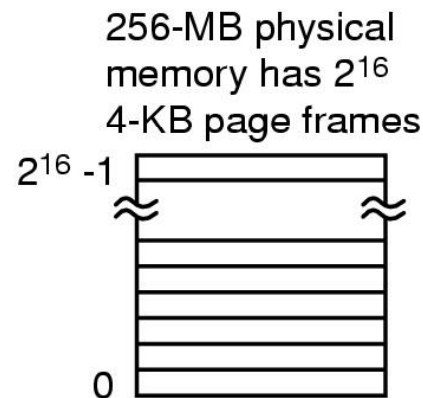
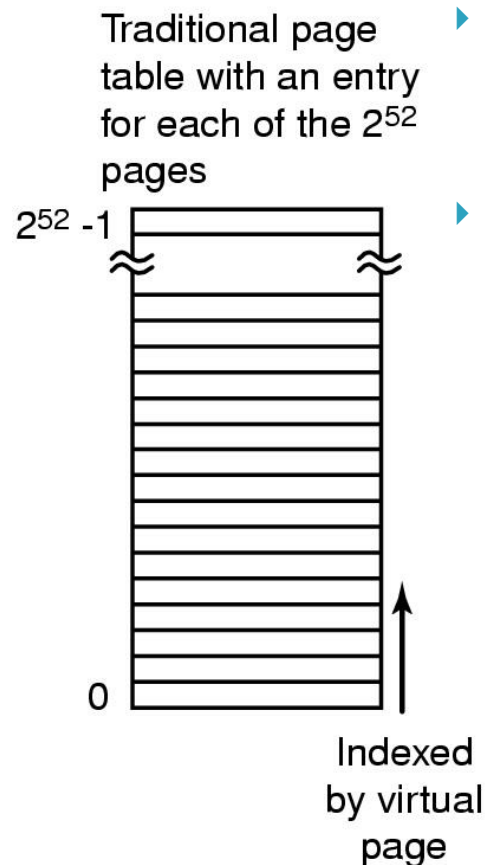
# Двонививски табели на страници

- ▶ Поради проблемот со огромните табели на страници во пракса се користат повеќе нивовски табели
- ▶ 1.000.000 страници за 32 битен адресен простор и 4KB страници



# Инвертирани табели на страници

- ▶ За 64 битна адреса таблата на страници треба да има  $2^{52}$  редови
- ▶ За секоја рамка од физичката меморија има по еден ред во кој се информациите на која виртуелна страница одговара.
- ▶ За да се конвертира адресата мора да се пребара целата табела и да се пронајде бараната виртуелна страница. Пребарувањето може да се забрза со HASH табели
- ▶ Процесот може да се забрза со TLB





# Инвертирана табела на страници

## Мотив:

- ▶ Табелите на страници зависат од големината на виртуелната меморија
- ▶ Виртуелниот адресен простор станува поголем:
  - 64–битен, со 4KB големина на страница:  
 $2^{52}$  влезови во табела ( 4 квадрилиони) ...

# Инвертирана табела на страници

## Идеја:

- ▶ Табелата на страници да се организира со осврт на физичката меморија, наместо виртуелната меморија
- ▶ Глобална табела на страници индексирана со броеви на рамките и секој влез содржи виртуелна адреса и PID
- ▶  $i$ -ти влез содржи информација за страната (виртуелна) која се наоѓа во рамката  $i$  од физичката меморија
- ▶ Користи TLB за редукција на потребата од пристап до табелата на страници
- ▶ Ако TLB промаши: барај низ табелата  $\langle \text{virtual address, PID} \rangle$ 
  - Физичката адреса се добива од индексот (frame number)

# Структура на ИТС

- ▶ Индексирана со бројот на рамката
- ▶ Влезот содржи виртуелни адреси и PID што ја користат таа рамка (ако има)
- ▶ Ги содржи и другите битови

|     |              |   |   |   |   |   |   |
|-----|--------------|---|---|---|---|---|---|
| pid | Virtual addr | v | w | r | x | f | m |
| pid | Virtual addr | v | w | r | x | f | m |
| pid | Virtual addr | v | w | r | x | f | m |

**v: valid bit (0 - page fault)**

**w: write bit**

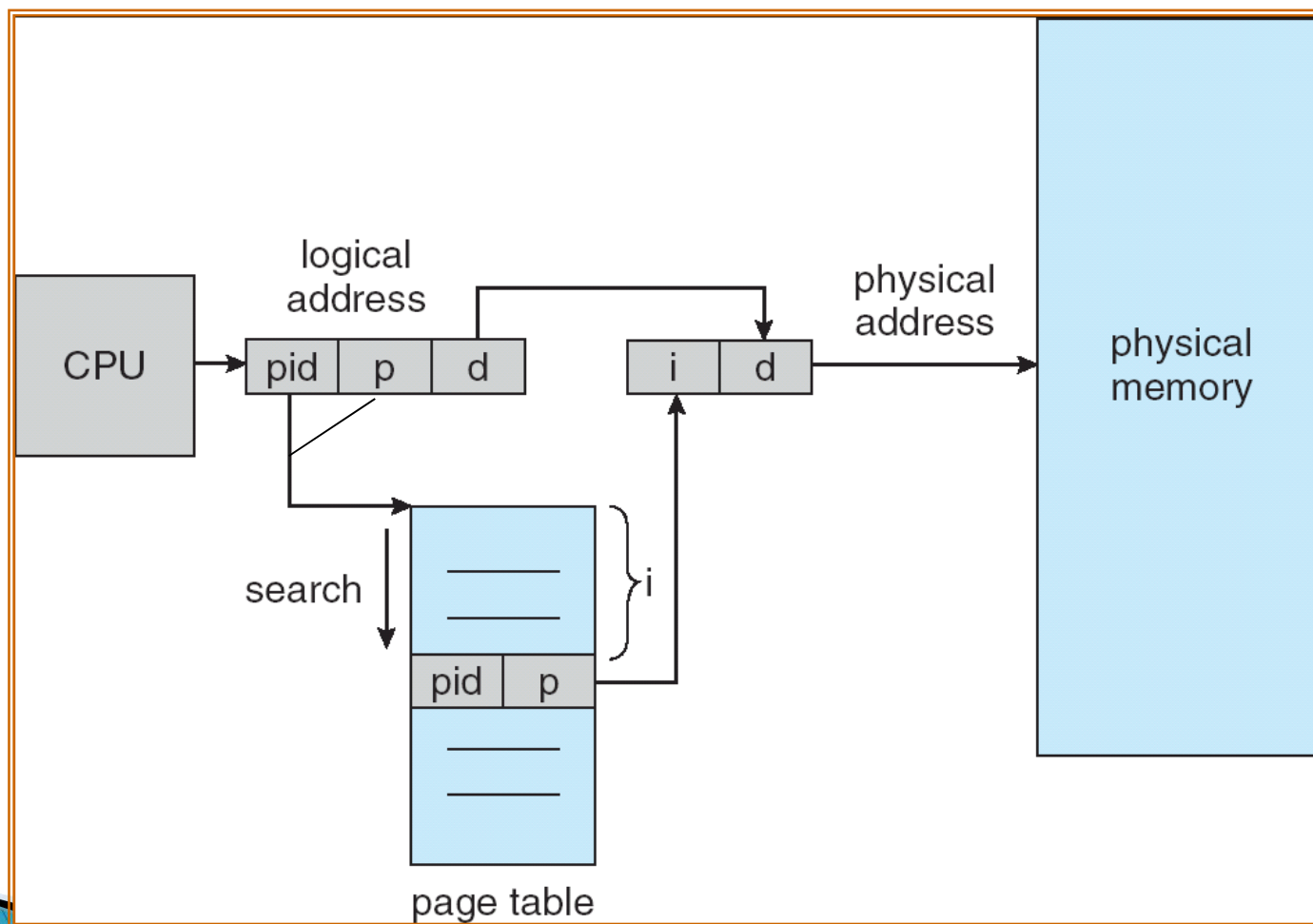
**r: read bit**

**x: execute bit (rare)**

**f: reference bit (for PR algor.)**

**m: modified bit (if 1 write to disk)**

# Пресликување од виртуелна во реална адреса



# HASH табели

