Управување со меморија

Оперативни системи 2024

проф. д-р Димитар Трајанов,
проф. д-р Невена Ацковска,
проф. д-р Боро Јакимовски,
проф д-р Весна Димитрова,
проф. д-р Игор Мишковски,
проф. д-р Сашо Граматиков,
вонр. проф. д-р Милош Јовановиќ,
вонр. проф. д-р Ристе Стојанов,
доц. д-р Костадин Мишев

Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Управување со меморијата

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

Мемориска хиерархија

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia

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 (гранење на наредби, податоци...) во кодот мора да бидат преведени во вистински физички адреси

Адресни побарувања за некој процес

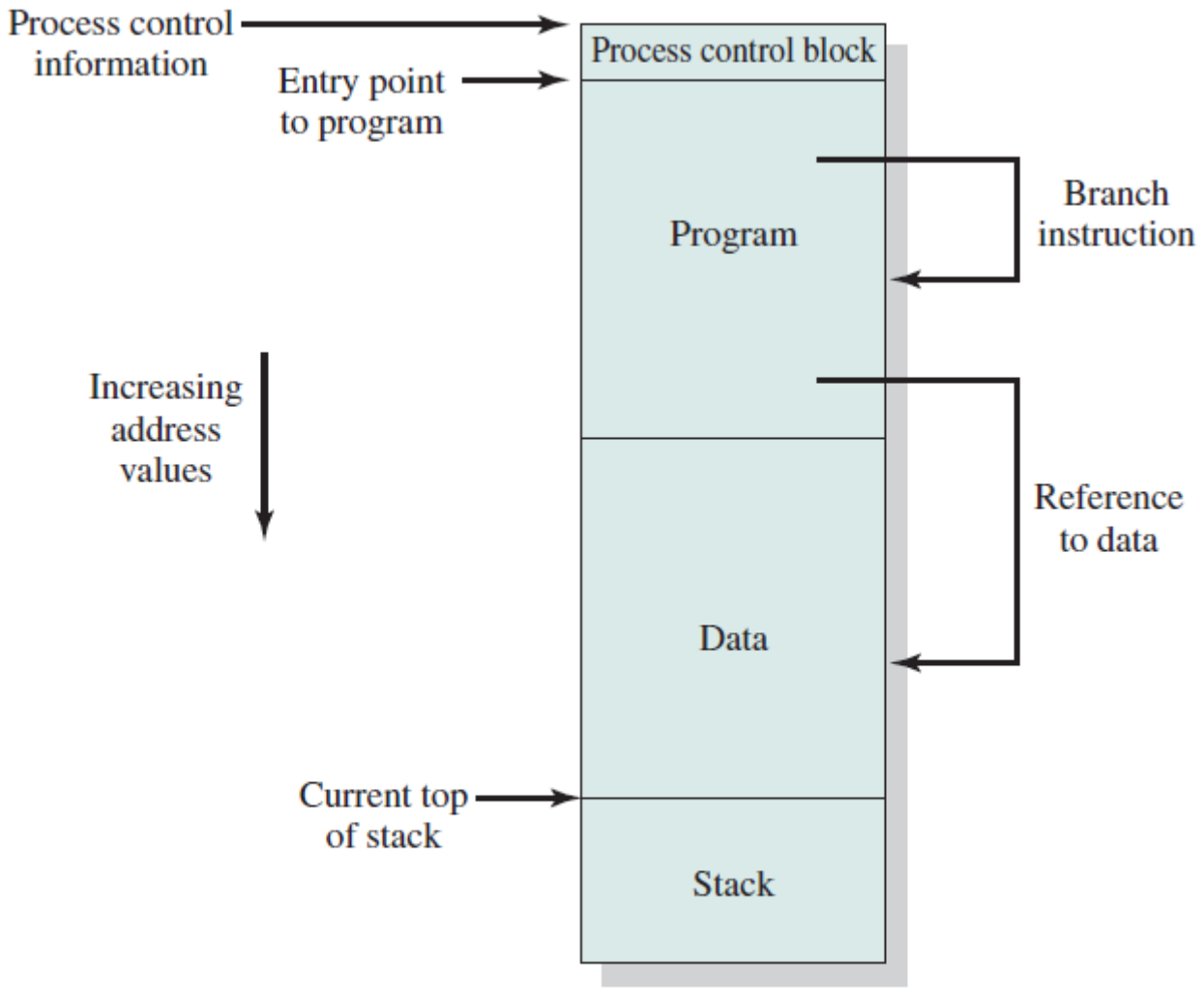


Figure 7.1 Addressing Requirements for a Process

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

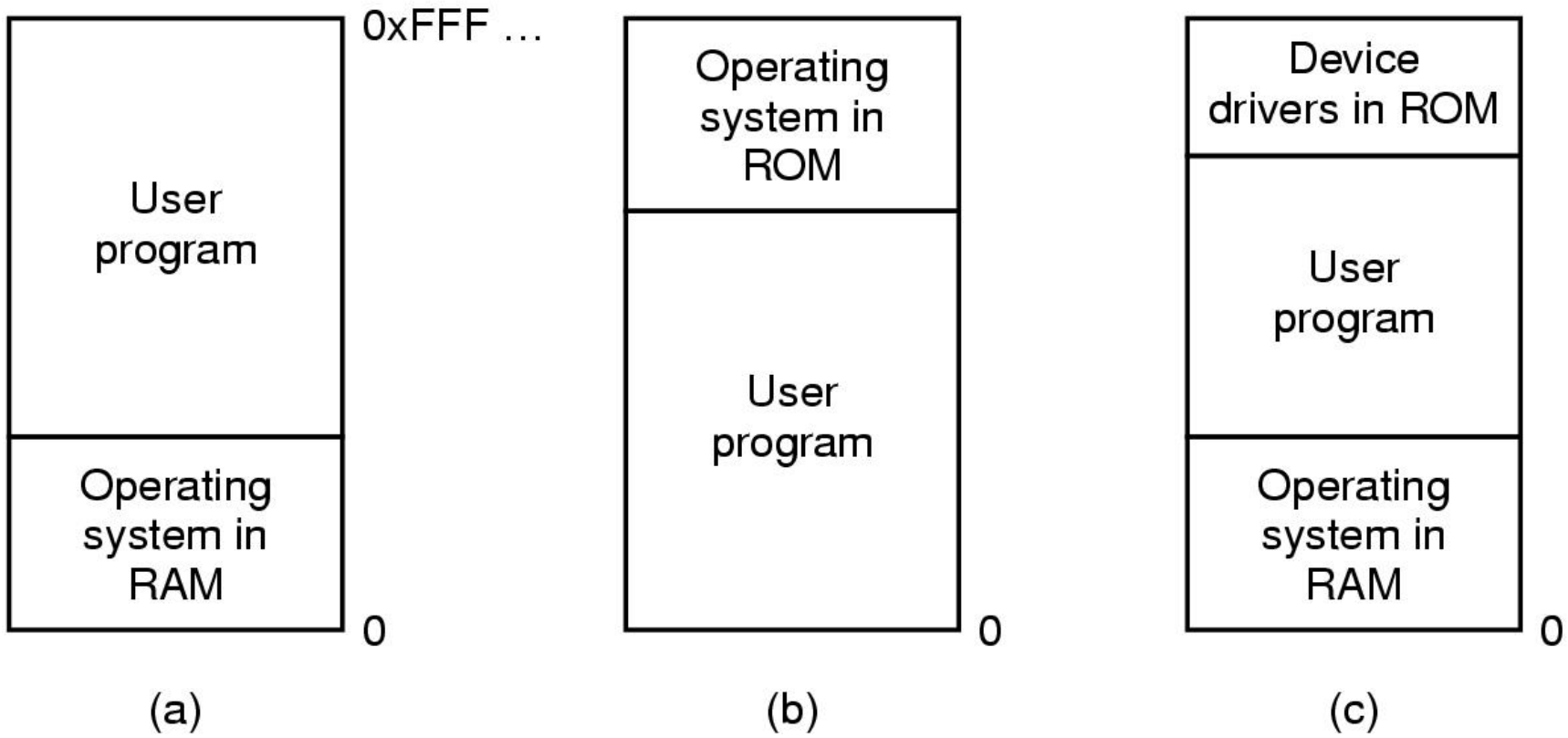
Управувањето со меморија треба да овозможи (3)

- **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



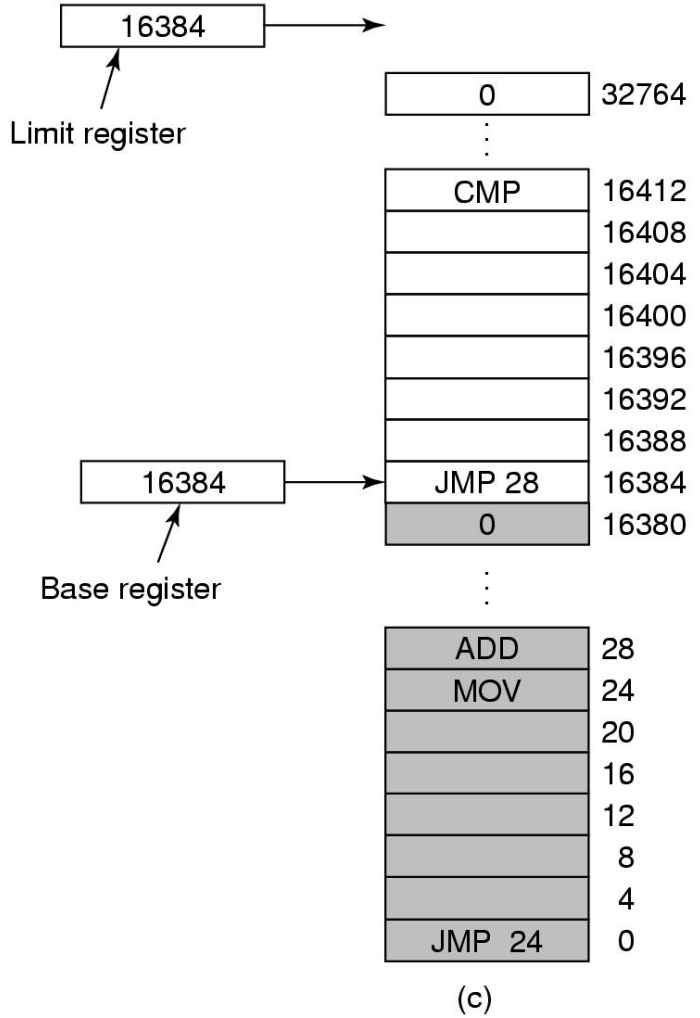
Организирање на меморијата со ОС и еден кориснички програм

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)

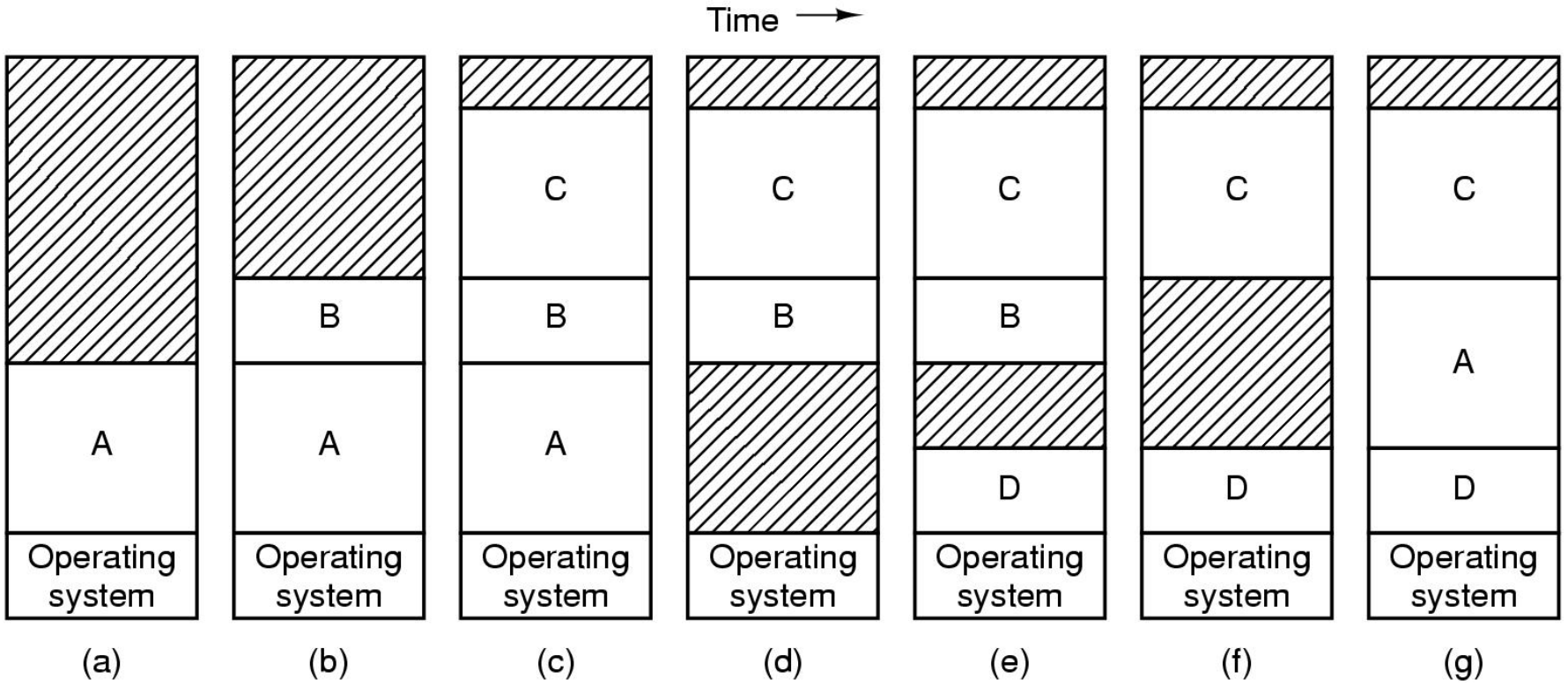
Основен и граничен регистар (Base & Limit)

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



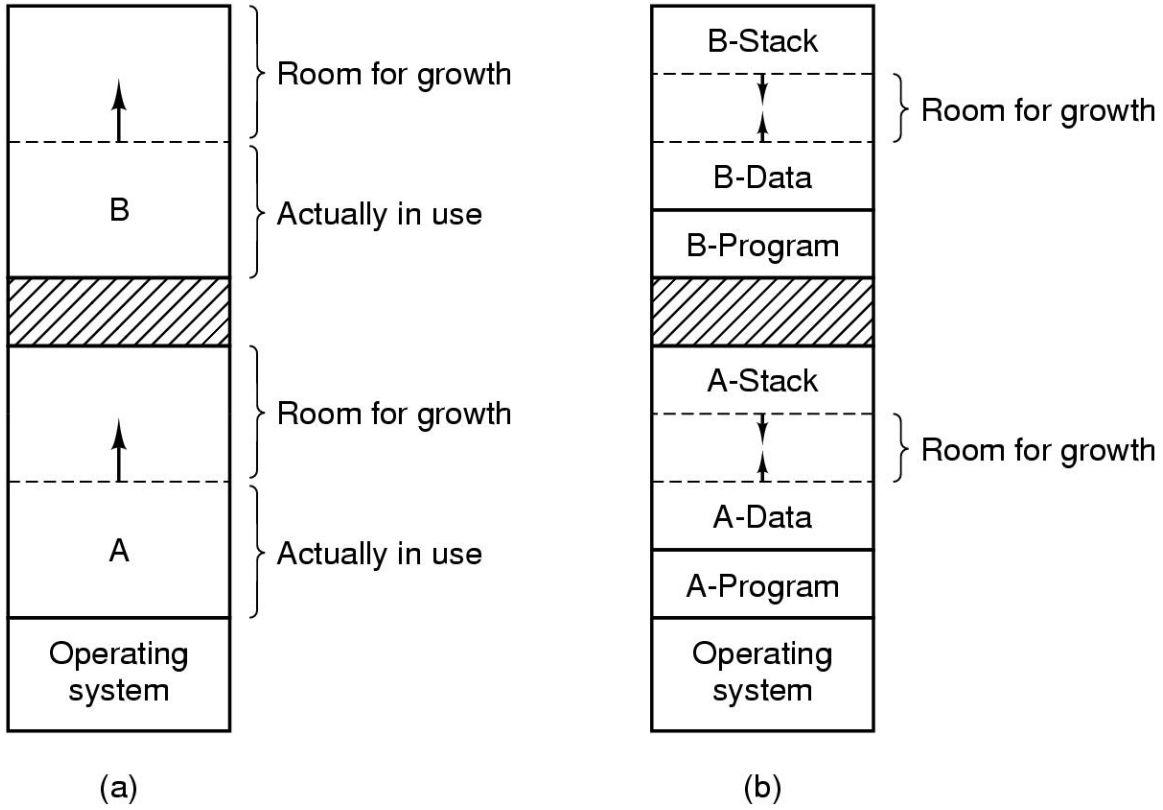
(c)

Замена на процеси од RAM (Swapping)



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

Swapping



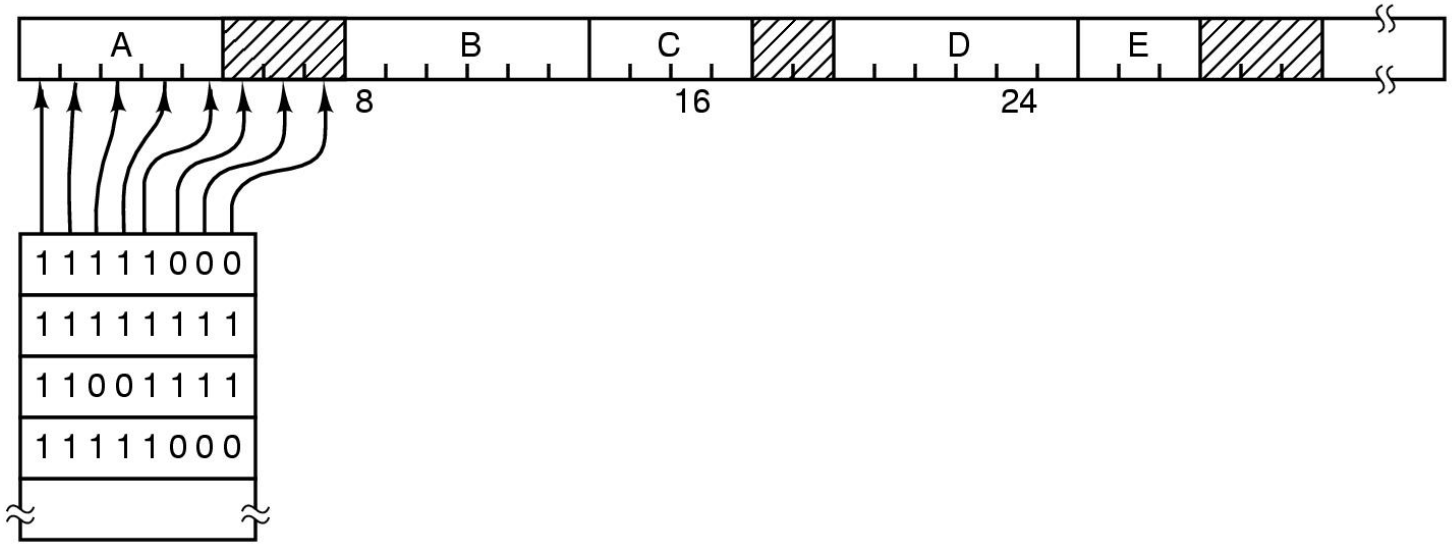
- Резервирање на простор за сегменти кои динамички може да растат

Управување со слободната меморија

- При динамичко управување со меморија ОС мора да управува и со слободниот простор

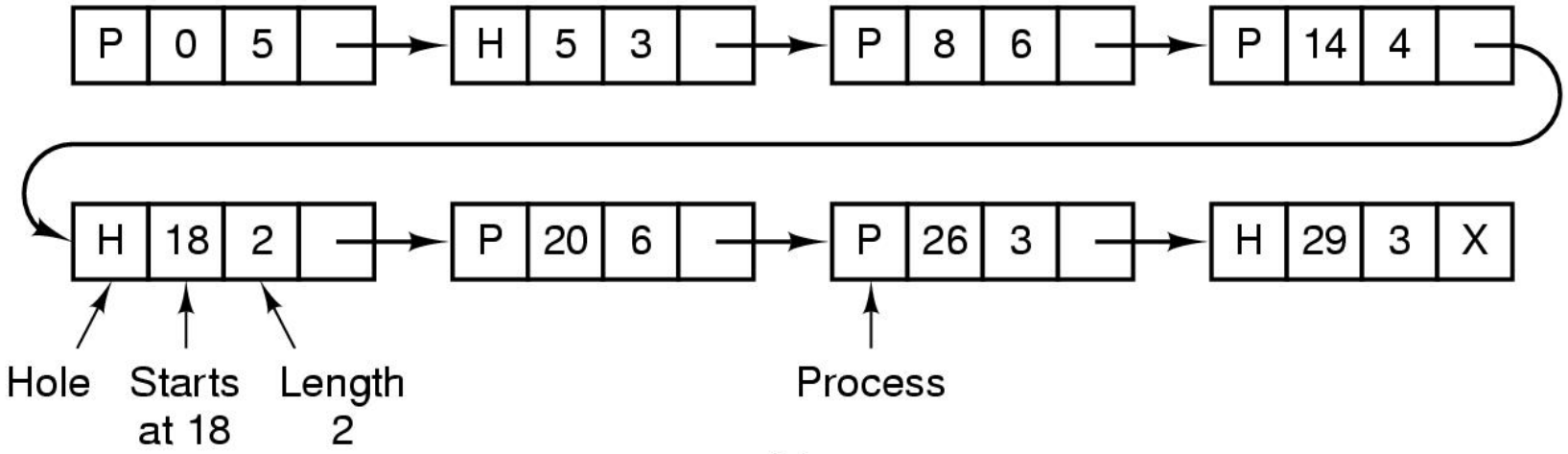
- Две методи
  - Битмапи
  - Поврзани листи

Управување на меморијата со бит мапи



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

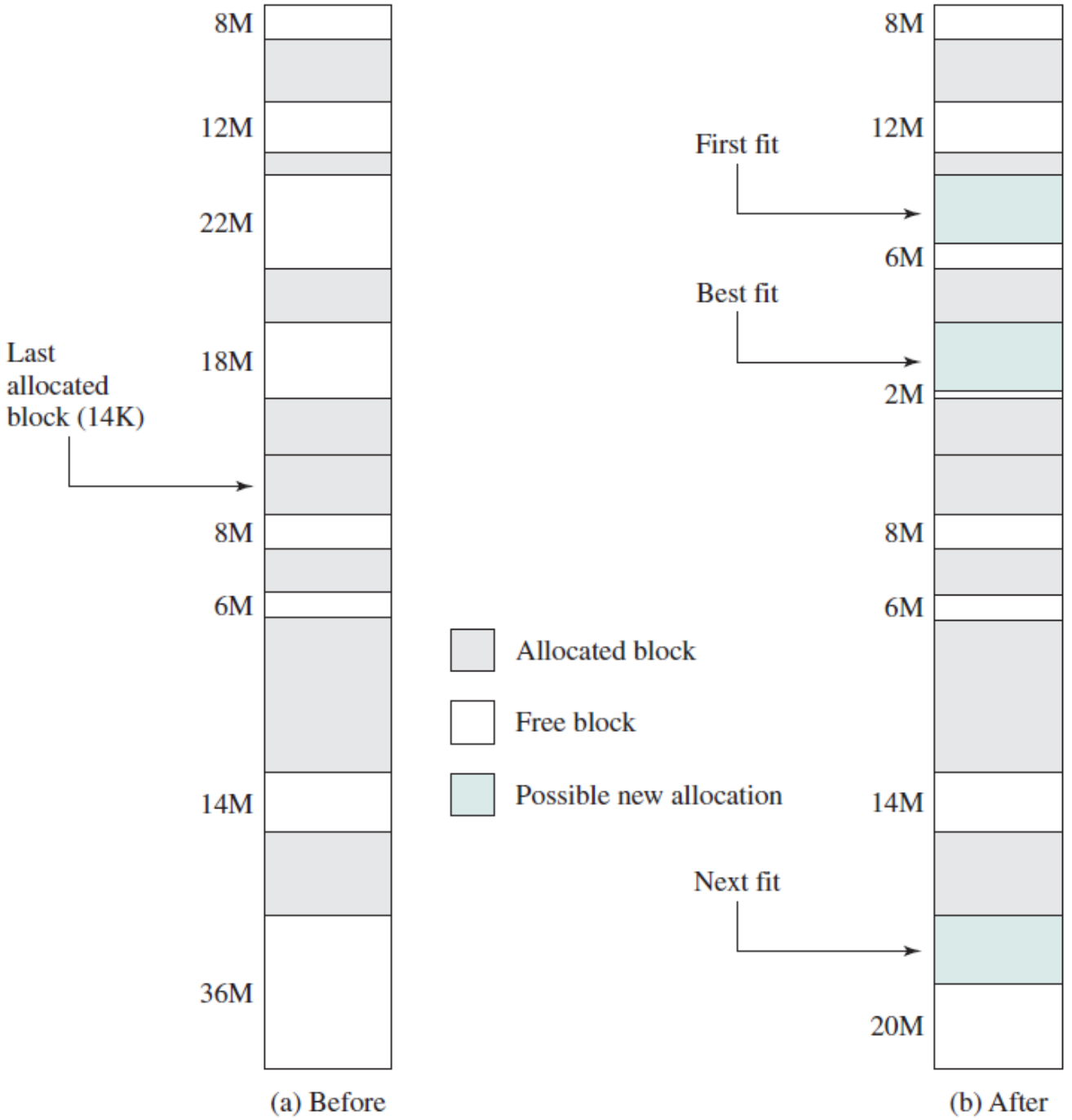
Управување на меморијата со листи



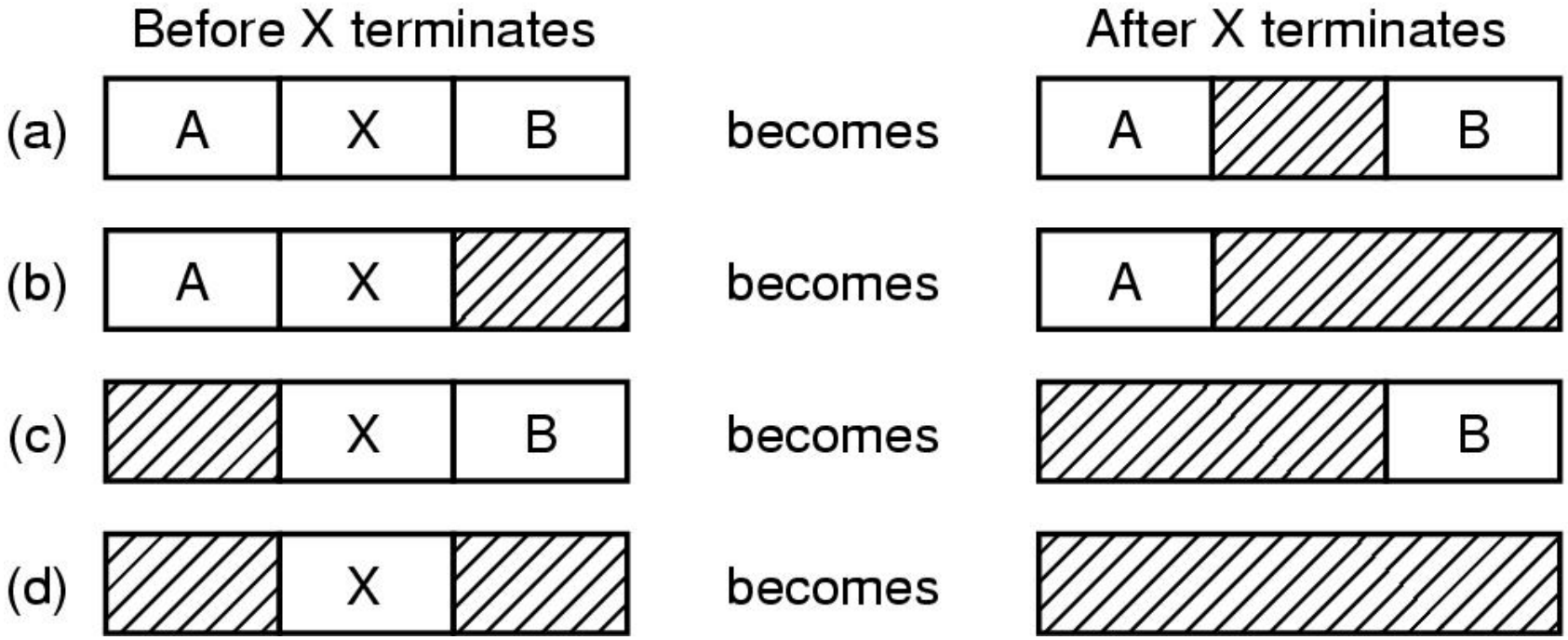
- Меморијата се дели на алокациски единици
- За секој блок од меморија зафатен од процес и за секој слободен блок постои јазол кој го покажува неговиот почеток и должина
- Начин на алокација на нов блок
  - FIRST FIT
  - NEXT FIT
  - BEST FIT
  - QUICK FIT

Примери

Мемориска конфигурација пред и по алоцирање на 16 Mbyte блок



Процес на спојување на слободните меморискиблокови

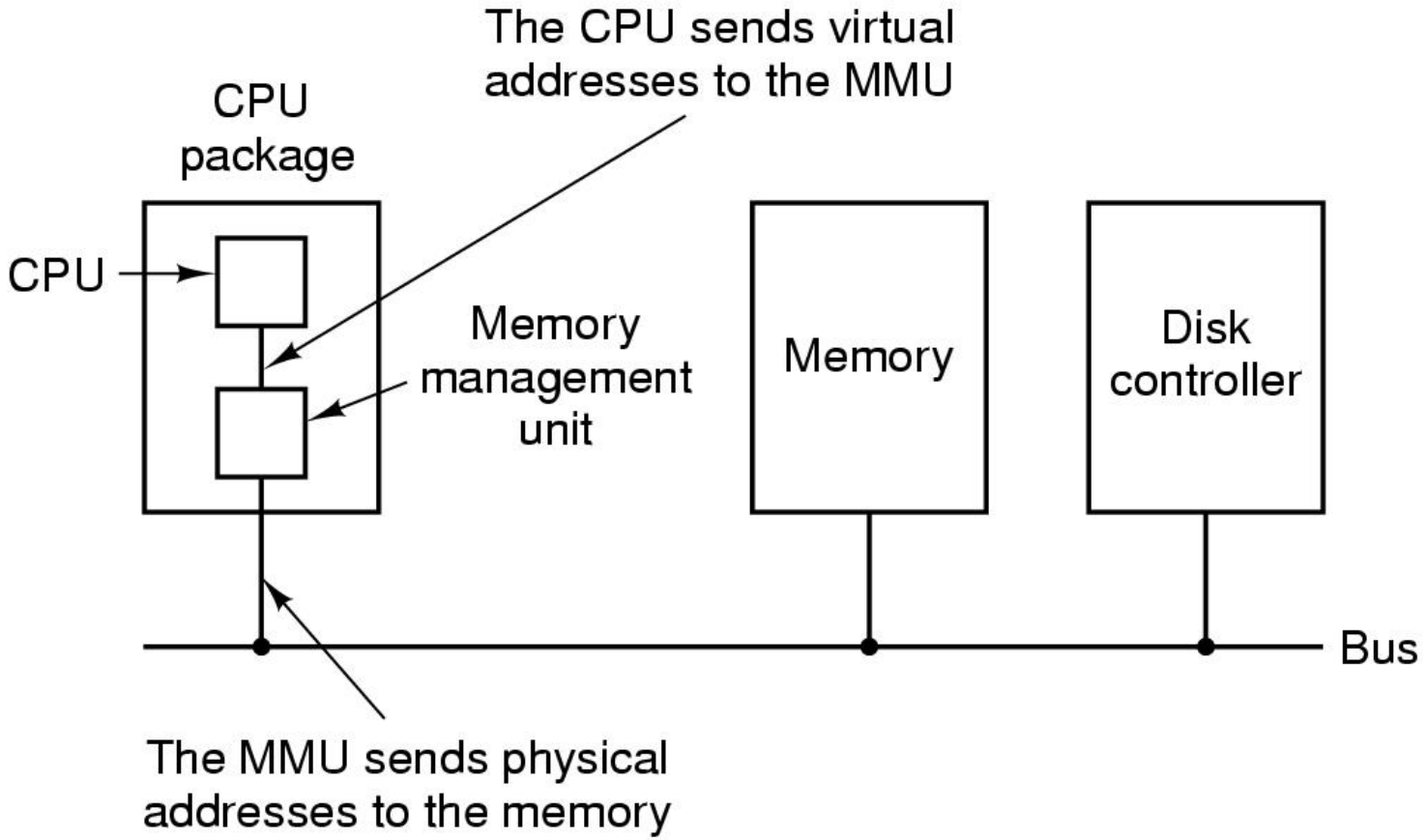


- Приказ на спојувањето на слободните меморискиблокови ако даден блок X се ослободи

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

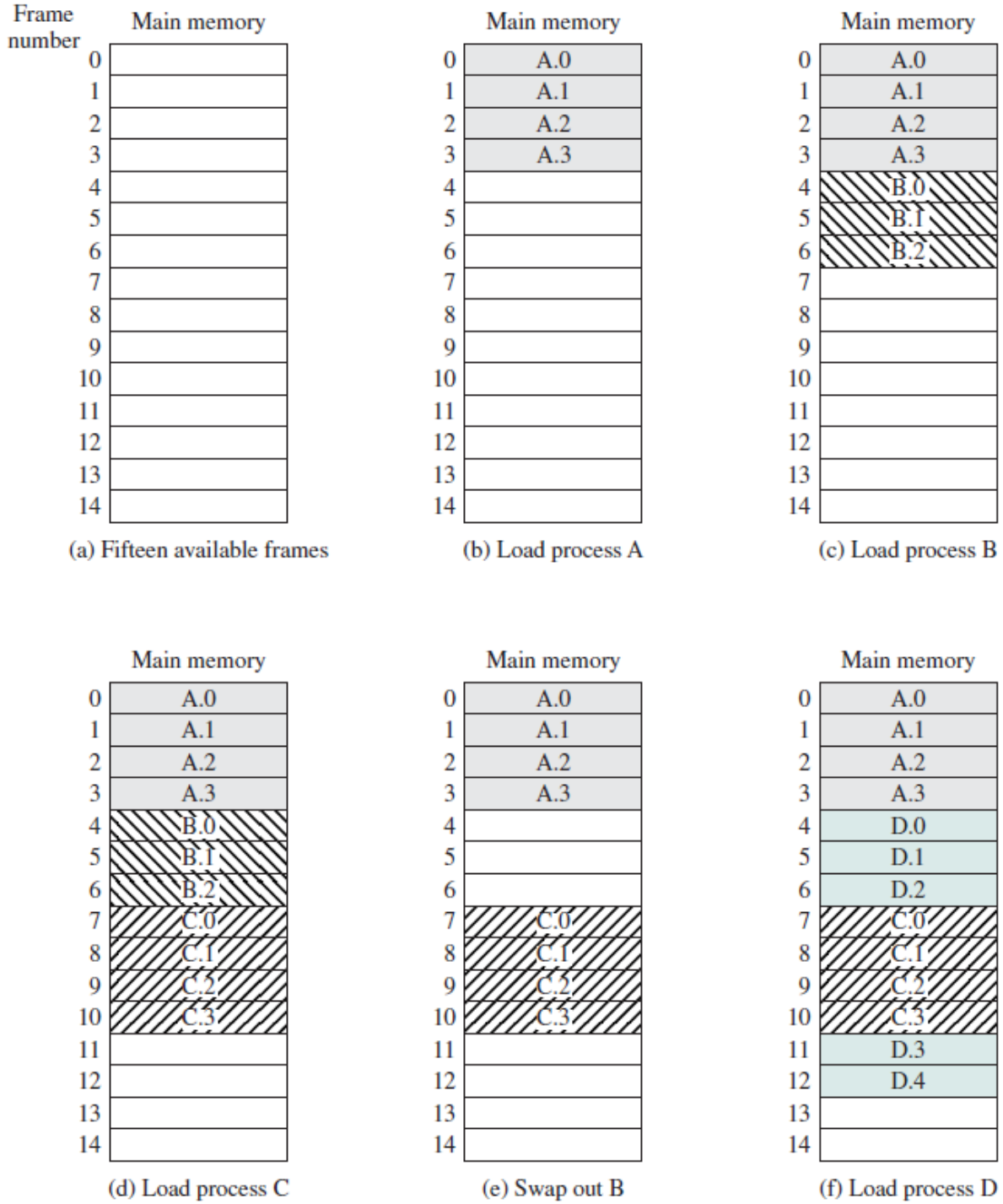


Страничење

- 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



(a) Fifteen available frames

(b) Load process A

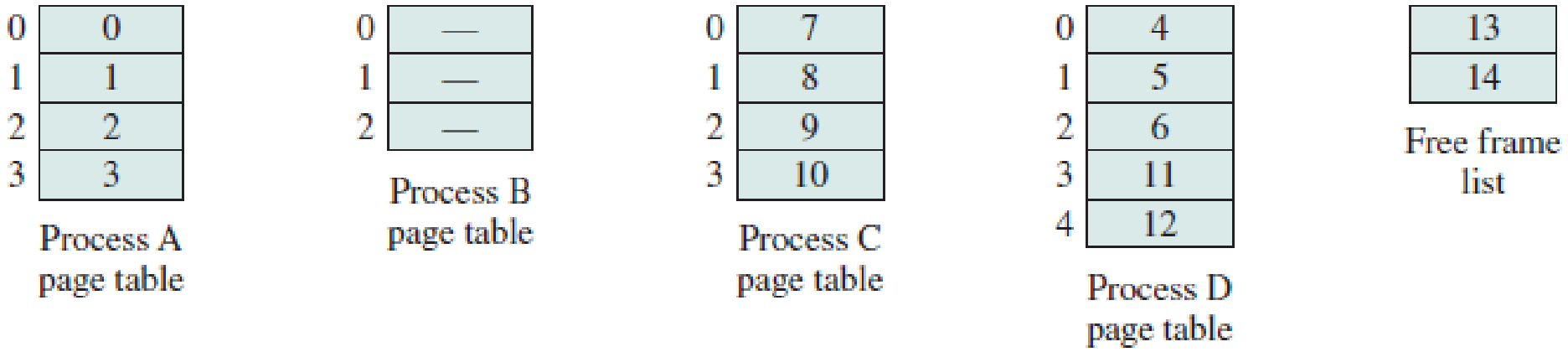
(c) Load process B

(d) Load process C

(e) Swap out B

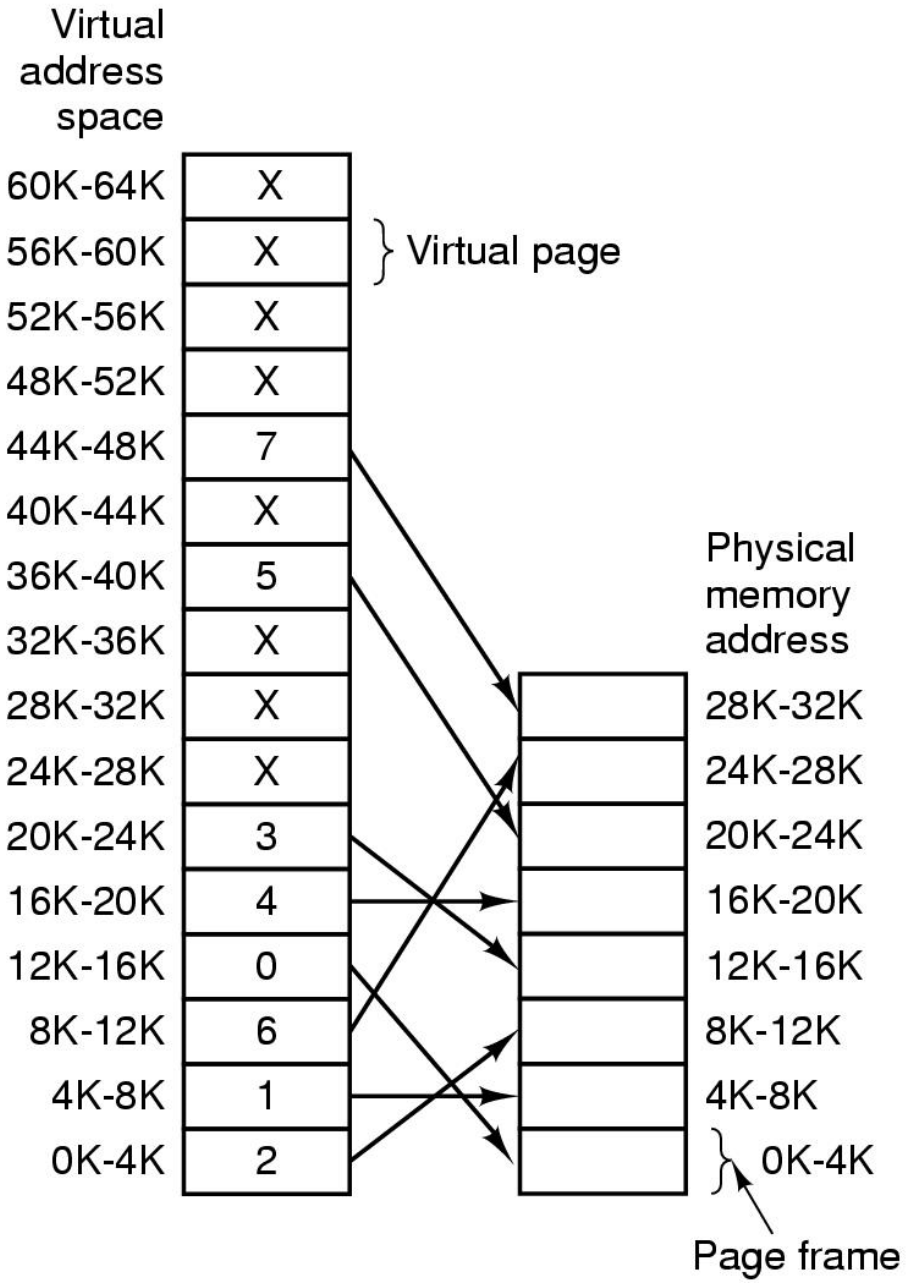
(f) Load process D

… А табелите на страници се



Страничење

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

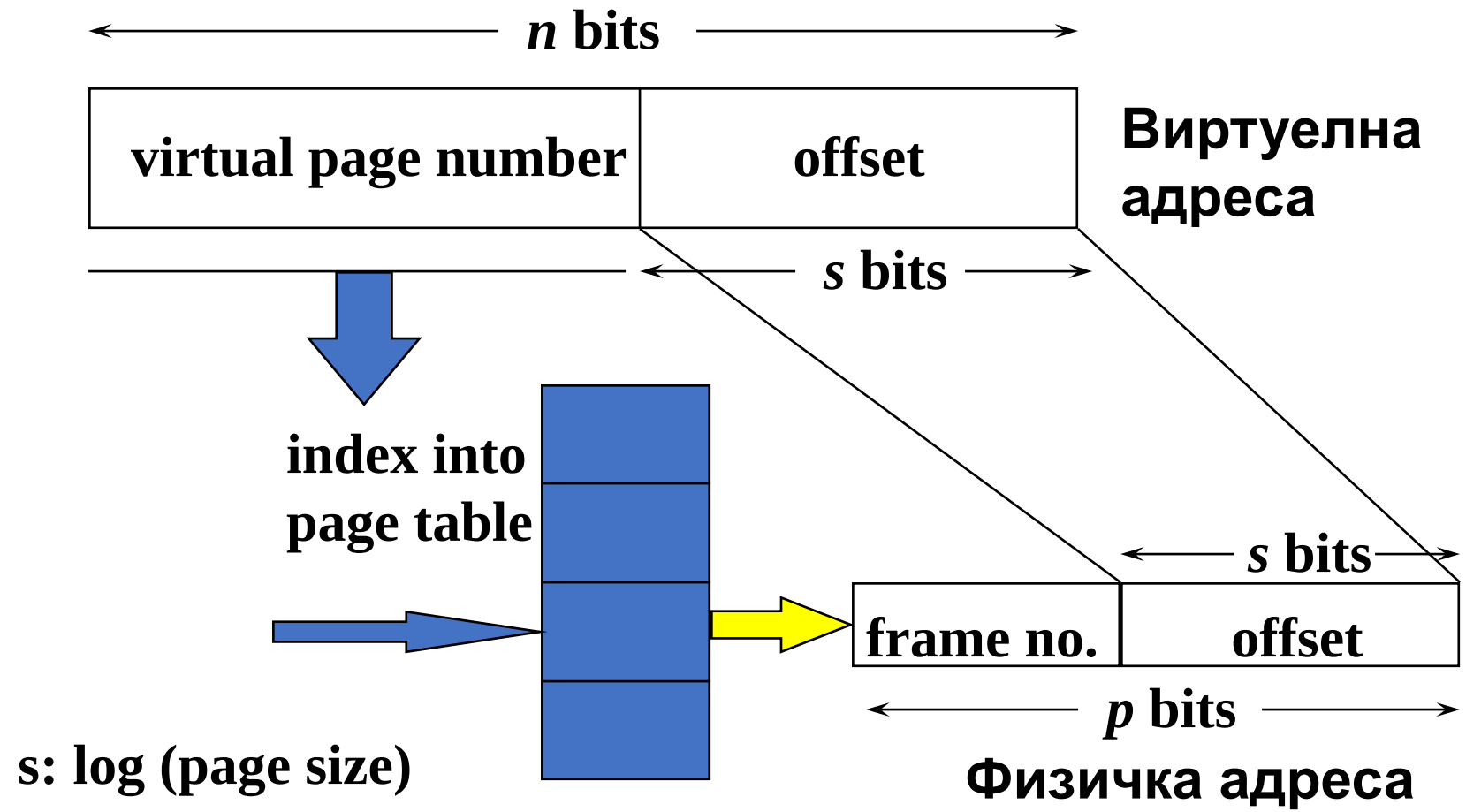


Делови на виртуелна адреса

- Секоја адреса генерирана од CPU (виртуелна адреса) се дели на 2 дела
  - број на страница (page number) - **p**
  - поместување во страницата (page offset) – **d**

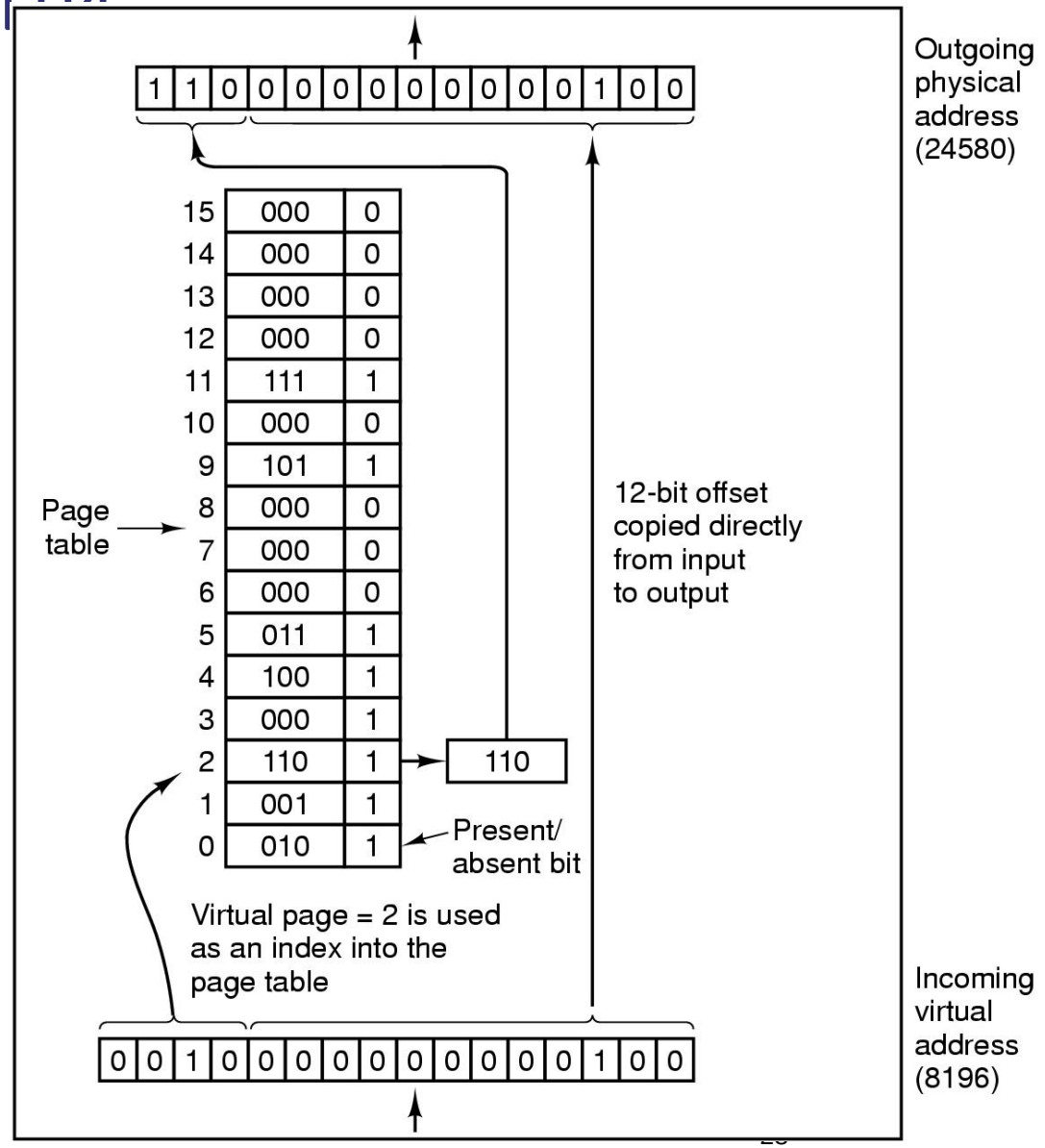
- Бројот на страницата се користи како индекс во **табелата на страници** (page table)
  - содржи **базна адреса** на секоја страница во физичката меморија
  - **базната адреса** се комбинира со **поместувањето** на страницата за да се дефинира физичката адреса што се испраќа на мемориската единица

Пресликување од виртуелна во реална адреса

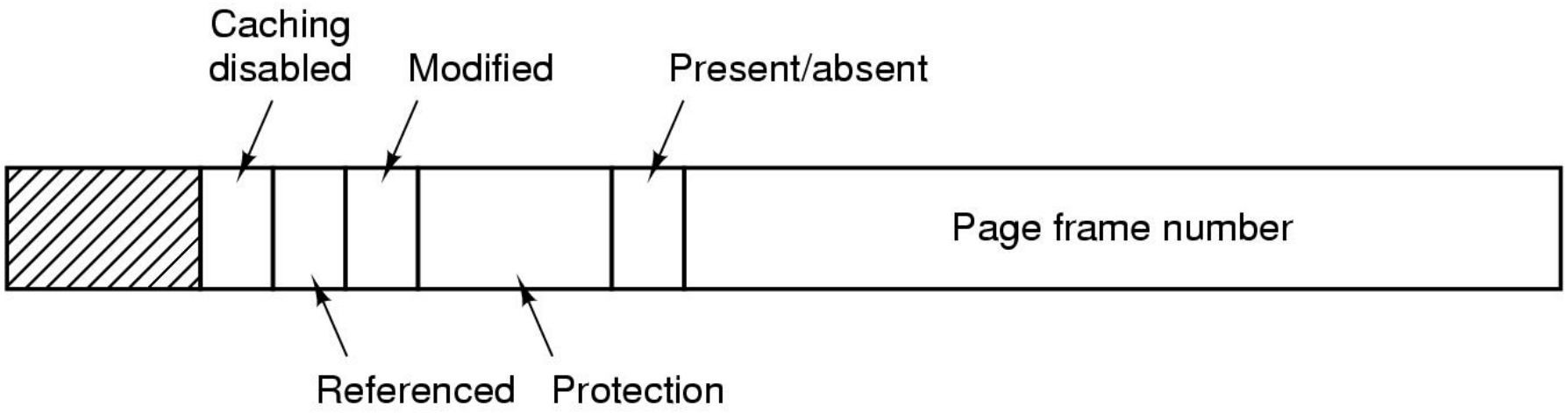


Табела на страници

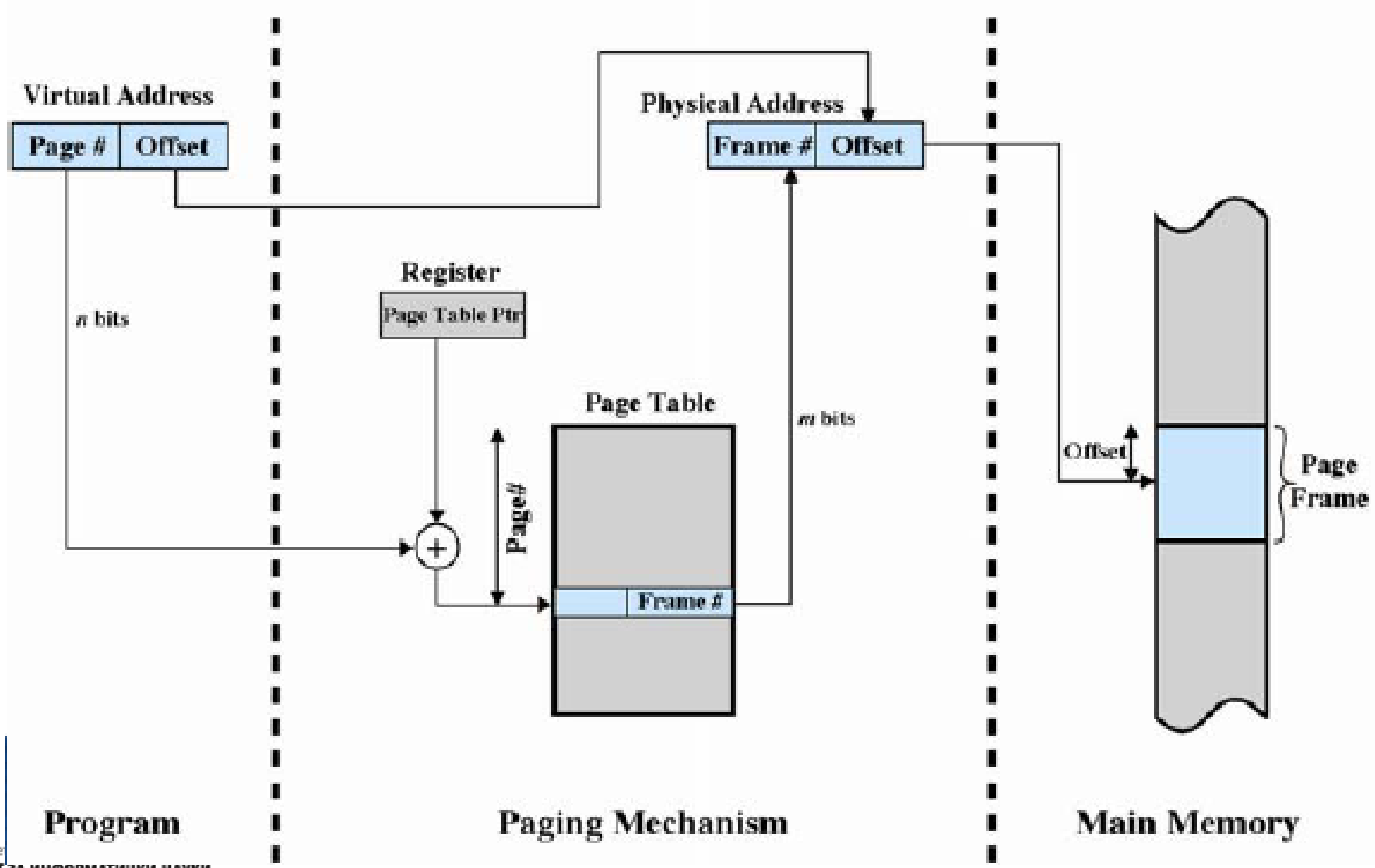
MMU со 16 страници со по 4 KB



Типичен ред од табелата на страници



Преведување на адреса во систем со страничење



Пример (MMU)

поместување

- Големина на страница **8 KB = 8 · 1024=8192 B**
- Наредба (читање од меморија во регистар)

  MOVE REG, 104852

  (104852 = **12**·8192 + **6548**) 0001 1001 1001 1001 0100
- 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 ·8192 + **6548** = 63892 преку магистралата 00 1111 1001 1001 0100
- (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, 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
- Пример:
  - **Intel Core i7 (Nehalem)**
    - L1 податочен TLB содржи 64 ставки
    - L1 инструкциски TLB содржи 128 ставки
    - L2 содржи податочни и инструкциски мапирања и има до 512 ставки

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.

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

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

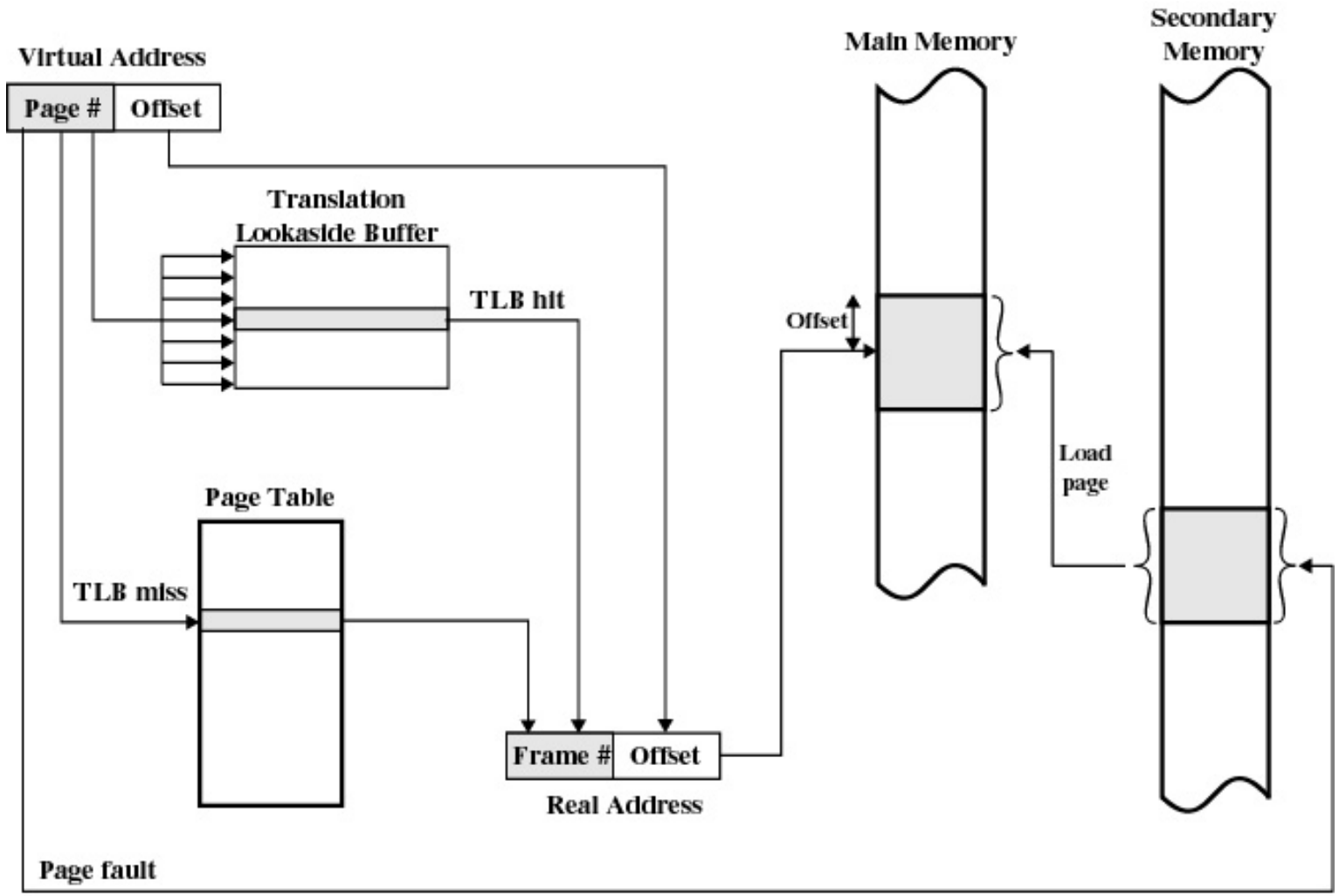


Figure 8.7 Use of a Translation Lookaside Buffer

Операции на страничење и TLB

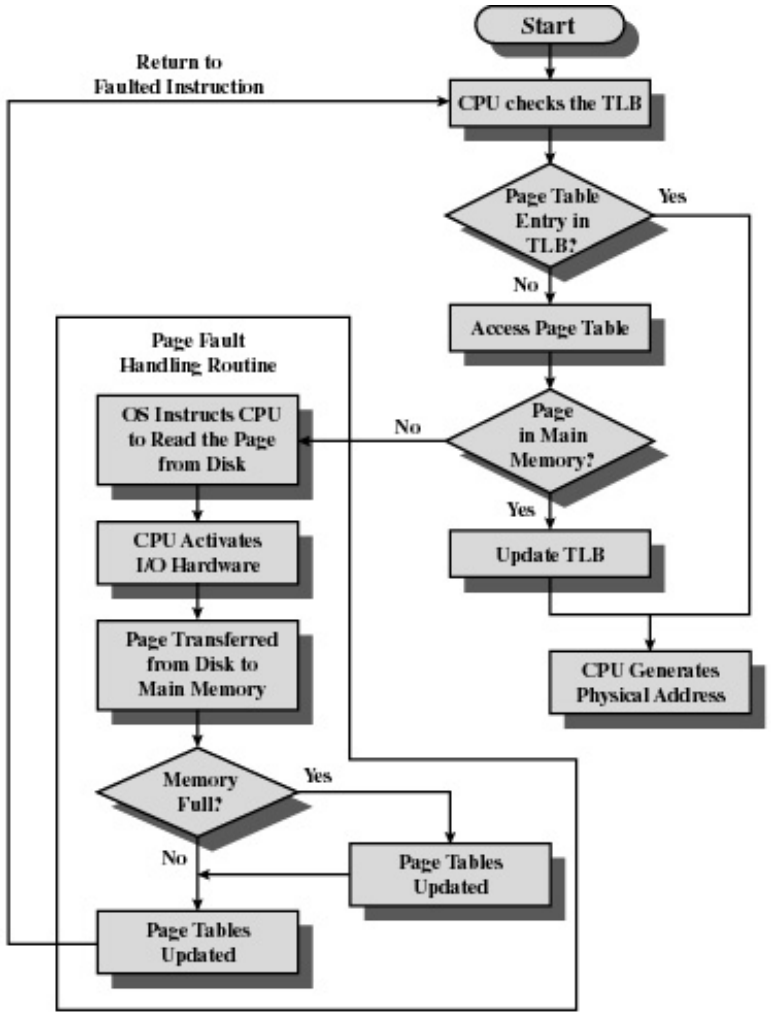


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Перформанси

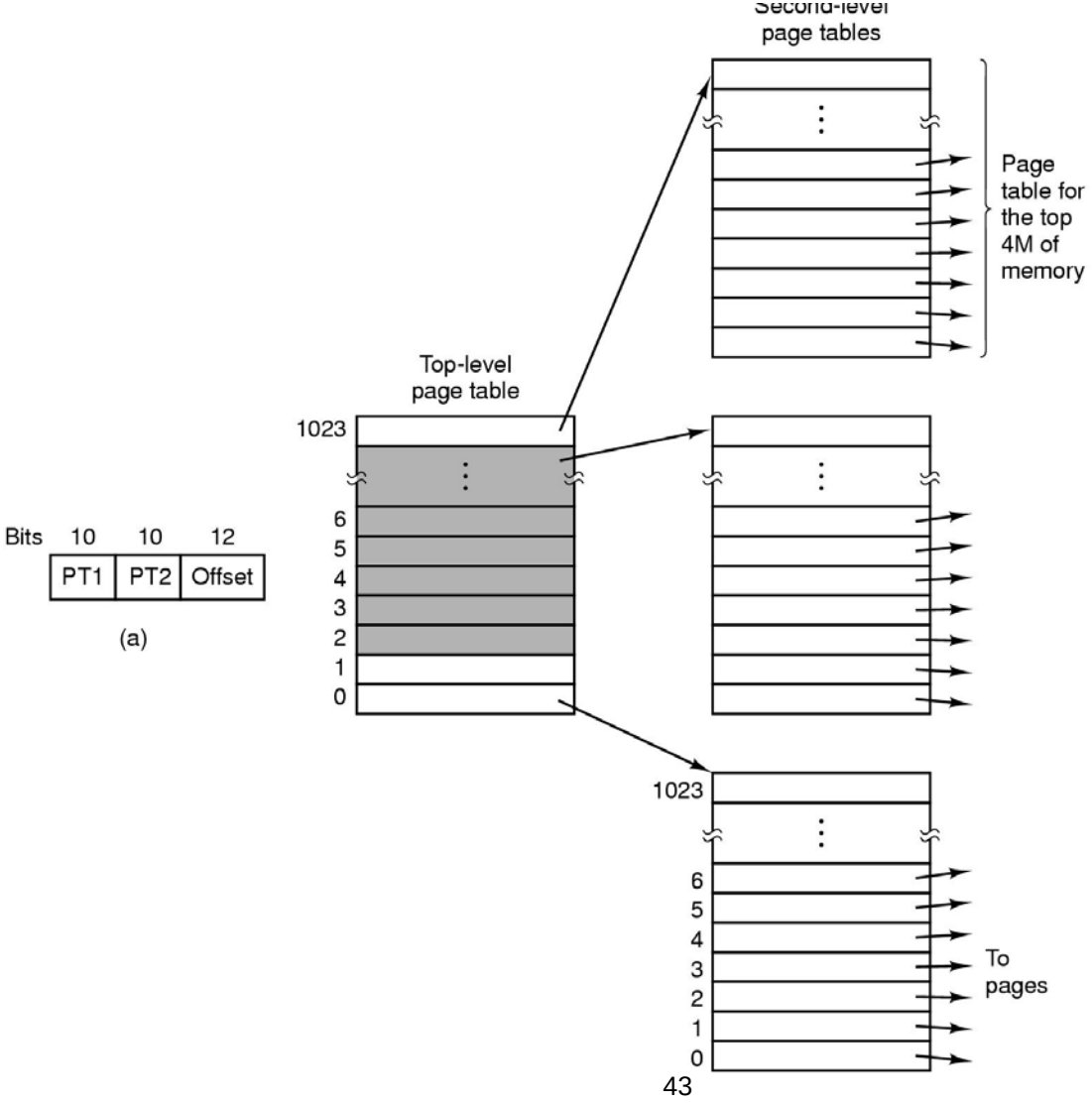
- Делот од мемориските референци кои можат да бидат задоволени од асоцијативната меморија (TLB), се нарекува стапка на погодување
- Колку е поголема, толку се подобри перформансите
- Пример:
  - 100 nsec за пристап до табела на страници
  - 20 nsec за пристап до TLB
  - 90% стапка на погодување

0.9 * 20 + 0.1 * 100 = 28 nsec просечно време за пристап

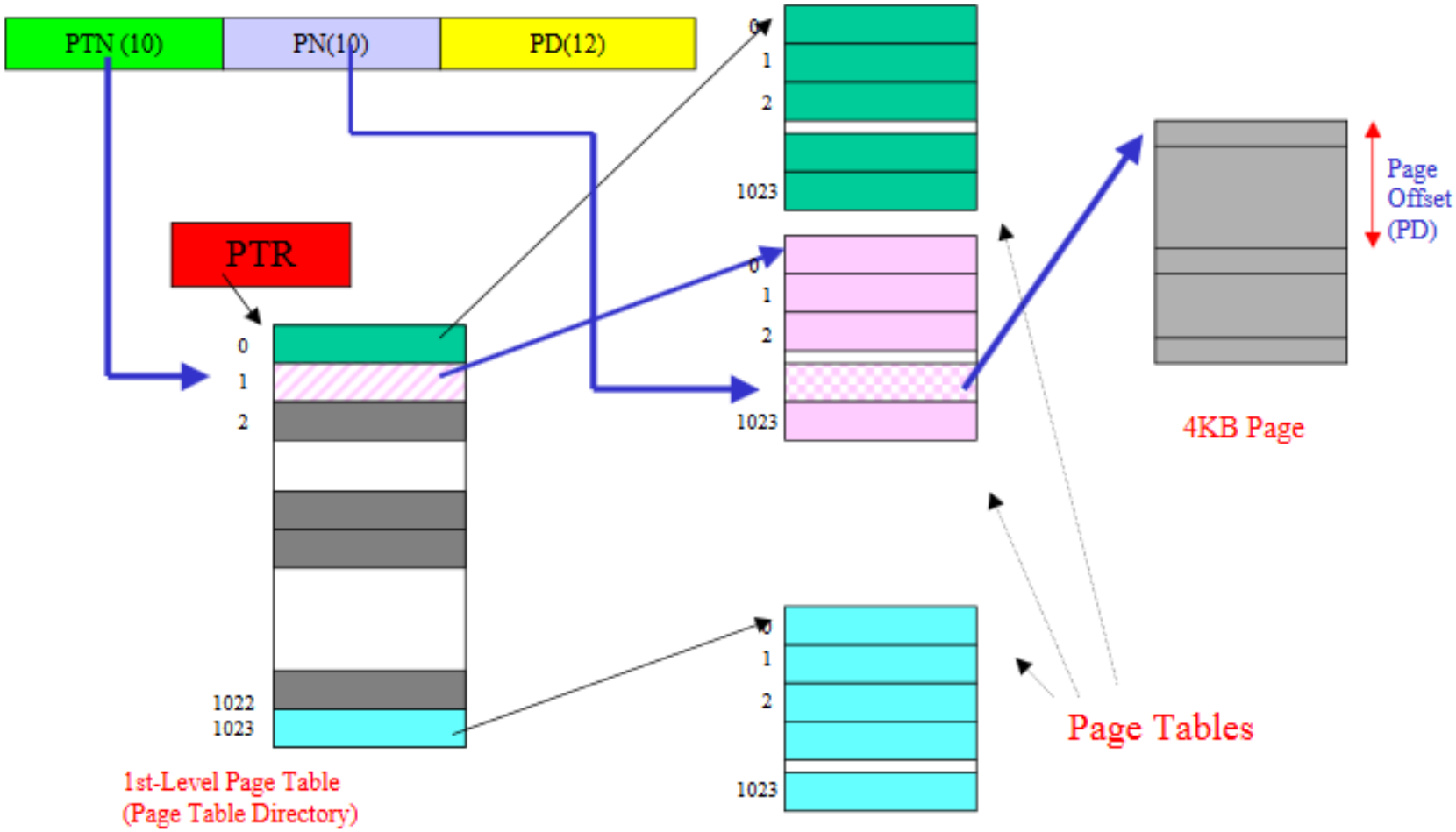
Двонивоовски табели на страници

- Поради проблемот со огромните табели на станици, во пракса се користат повеќе нивовски табели

- 1.000.000 страници за 32 битен адресен простор и 4КВ станици



Двонивоовски табели на страници



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 <virtual address, PID>
  - Физичката адреса се добива од индексот (frame number)

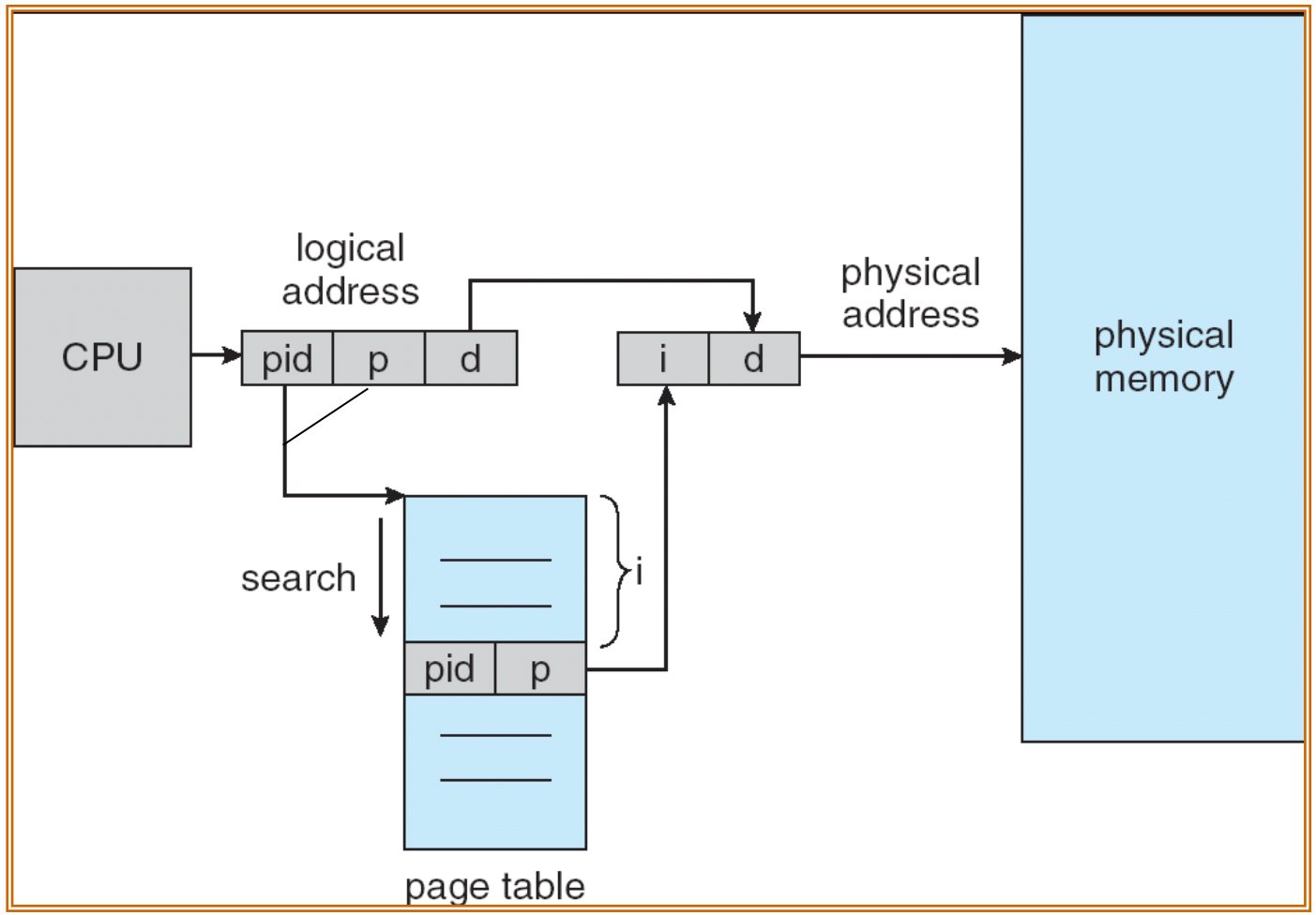
Структура на ИТС

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

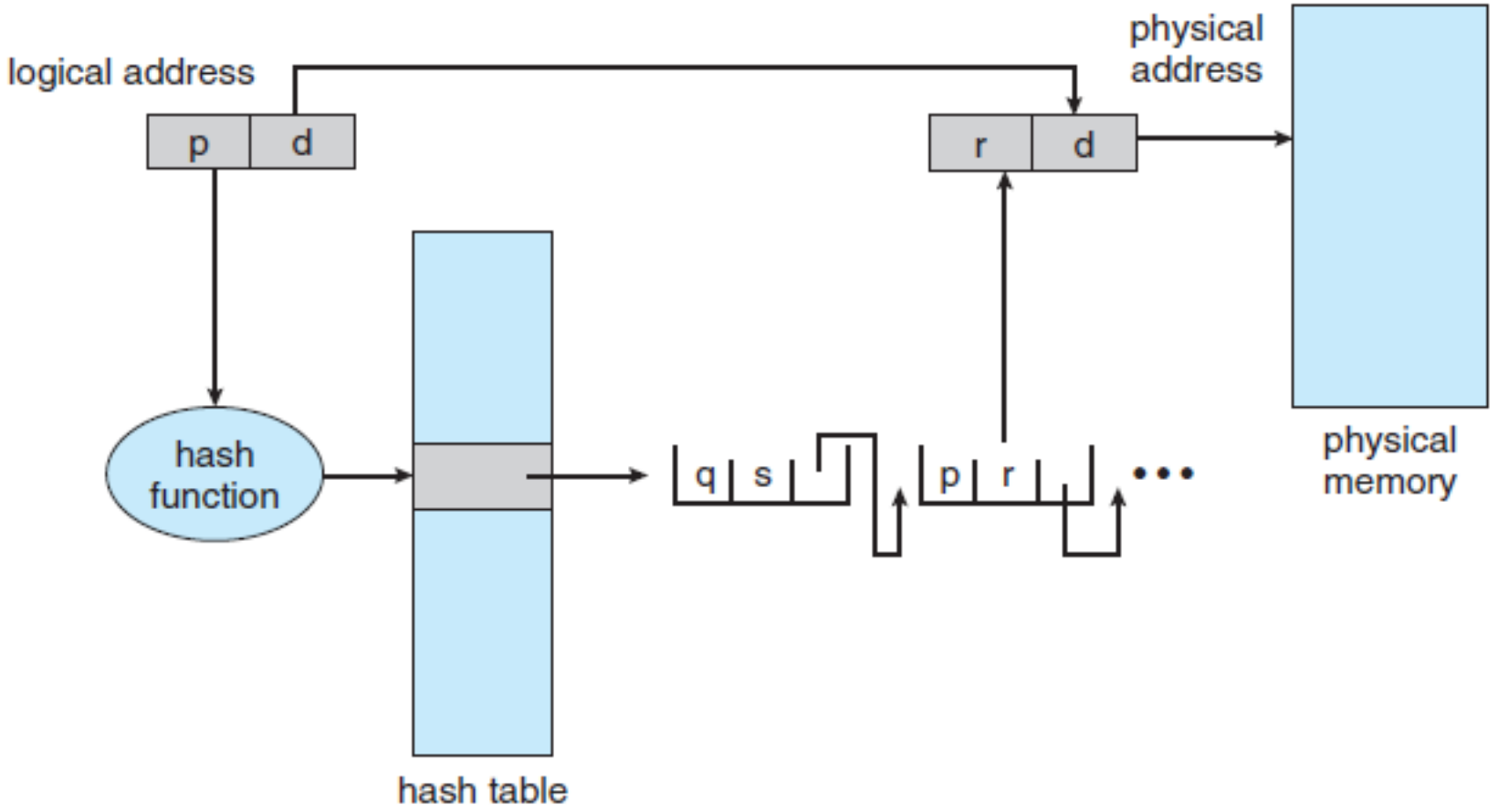
pid	Virtual addr	v	w	r	x	f	m
pid	Virtual addr	v	w	r	x	f	m
pid	Virtual addr	v	w	r	x	f	m

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


logical address
p
d
physical address
r
d
hash function
hash table
q
s
p
r
physical memory

End-to-end Core i7 Address Translation