# Управување со меморија (Memory Management)

## Управување со меморија

- ✓ Идеалната меморија
  - ✓Голема
  - ✓ Брза
  - ✓Евтина
- ✓Ограничувања
  - ✓Faster access time, greater cost per bit
  - ✓Greater capacity, smaller cost per bit
  - ✓Greater capacity, slower access speed
- ✓Мемориска хиерархија

# Мемориска хиерархија

- Going down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access to the memory by the processor

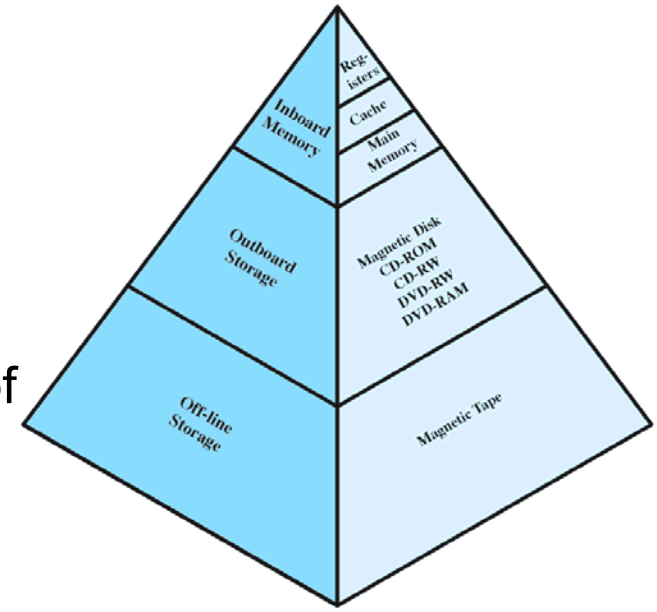
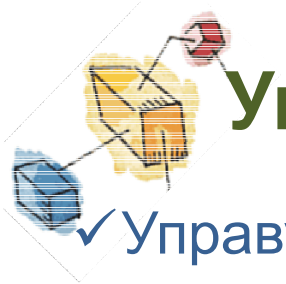


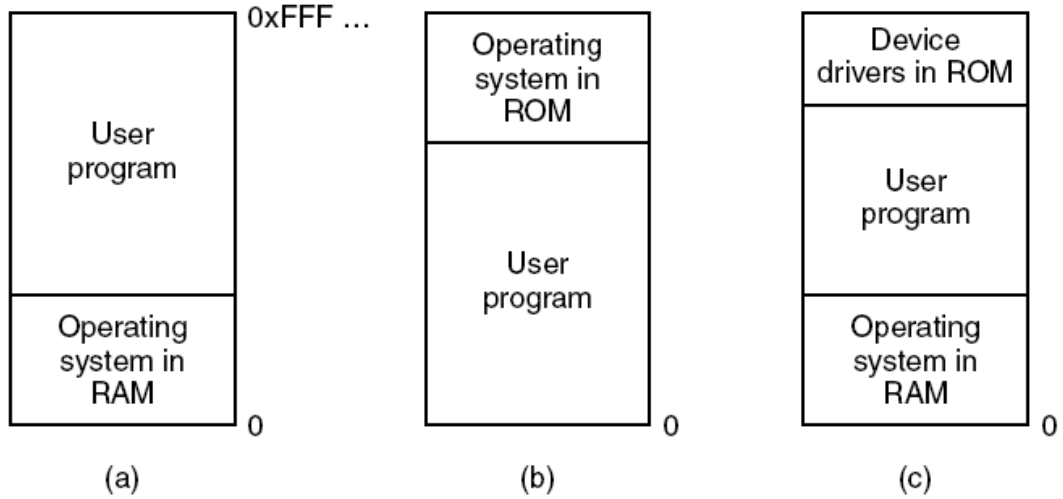

Figure 1.14 The Memory Hierarchy



# Управување со меморија

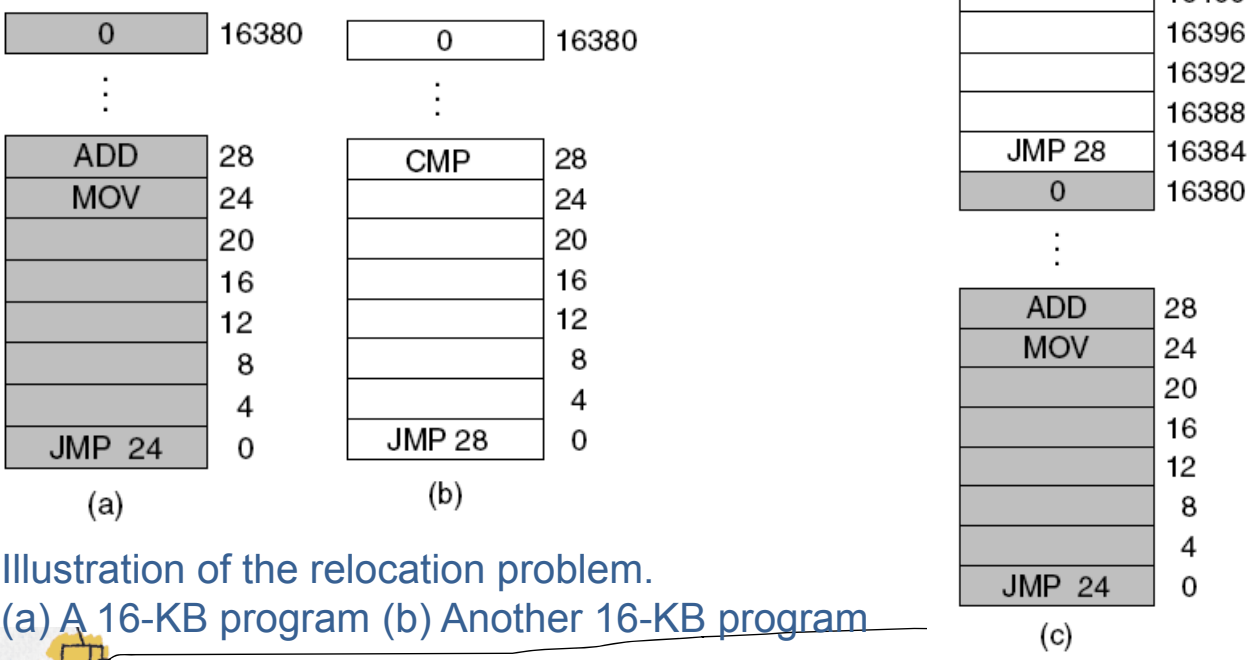
- ✓Управувач со меморијата
  - ✓Управува со мемориската хиерархија
  - ✓ Се стреми кон идеална меморија

# No Memory Abstraction

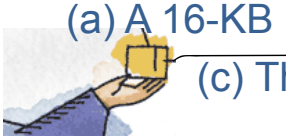


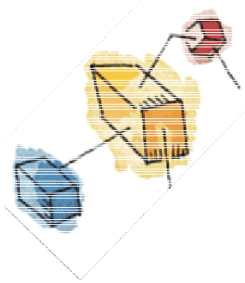

✓ Three simple ways of organizing memory with an operating system and one user process. Other possibilities also exist.

# Multiple Programs Without Memory Abstraction -Relocation problem-



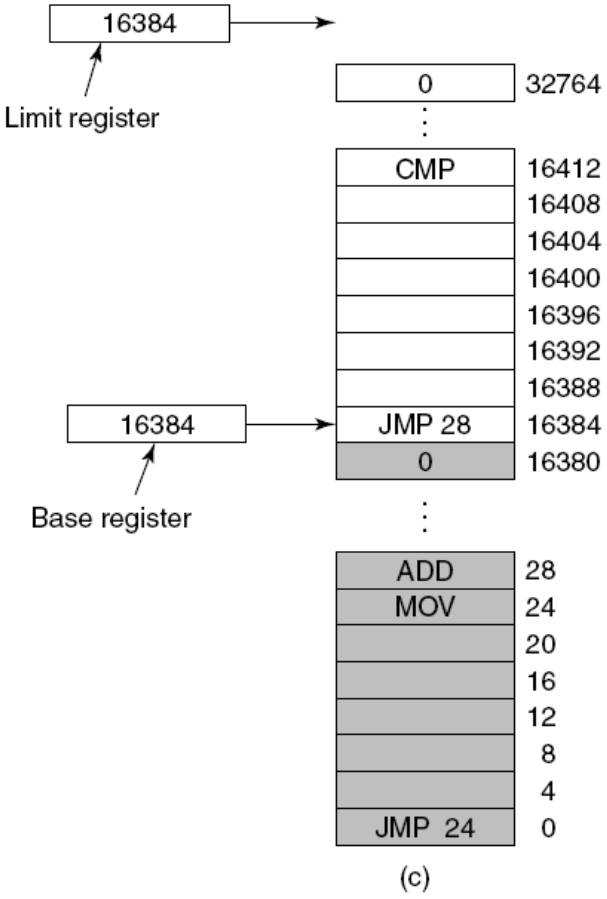

✓ Illustration of the relocation problem.
(a) A 16-KB program (b) Another 16-KB program
(c) The two programs loaded consecutively into memory.

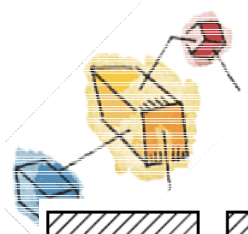




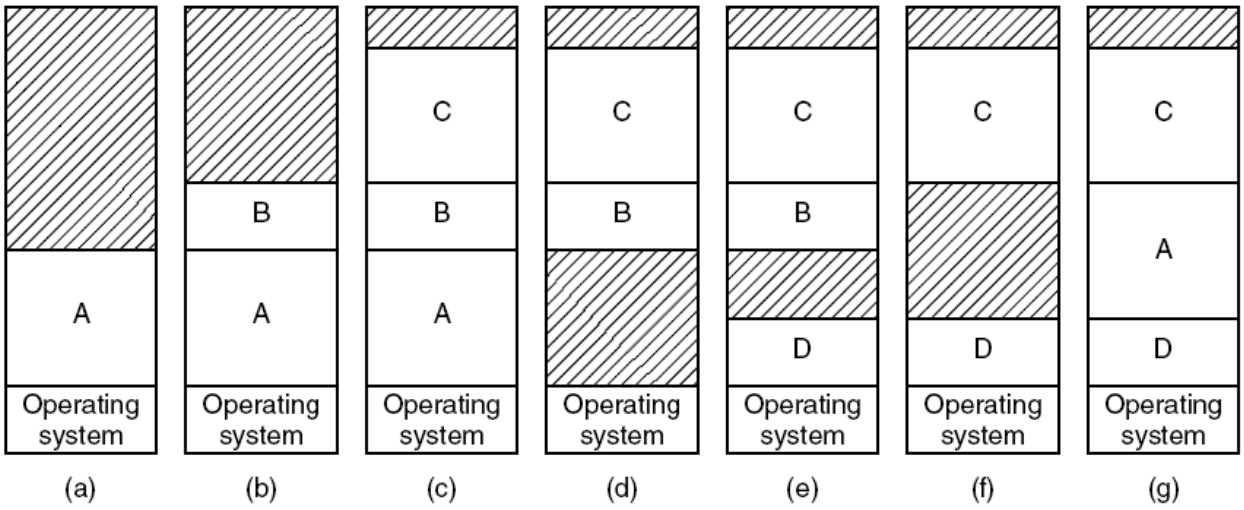
# Base and Limit Registers

- ✓ Base and limit registers can be used to give each process a separate address space.

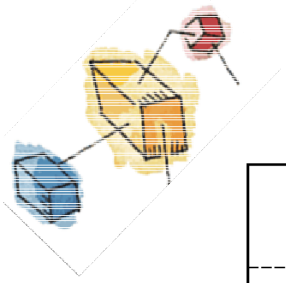


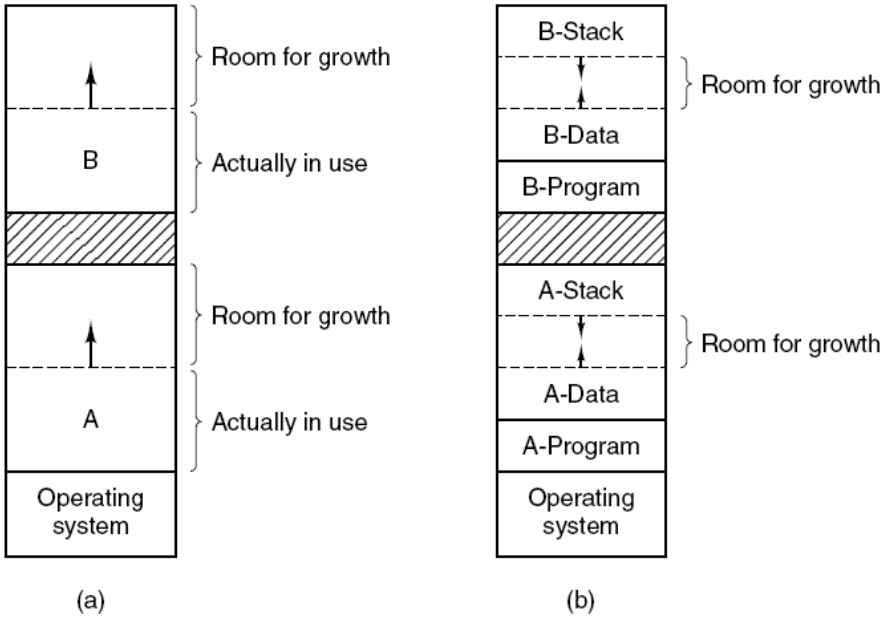
# Swapping (1)



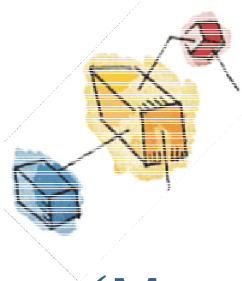

- ✓ Memory allocation changes as processes **come into memory** and **leave it**. The shaded regions are unused memory.



# Swapping (2)



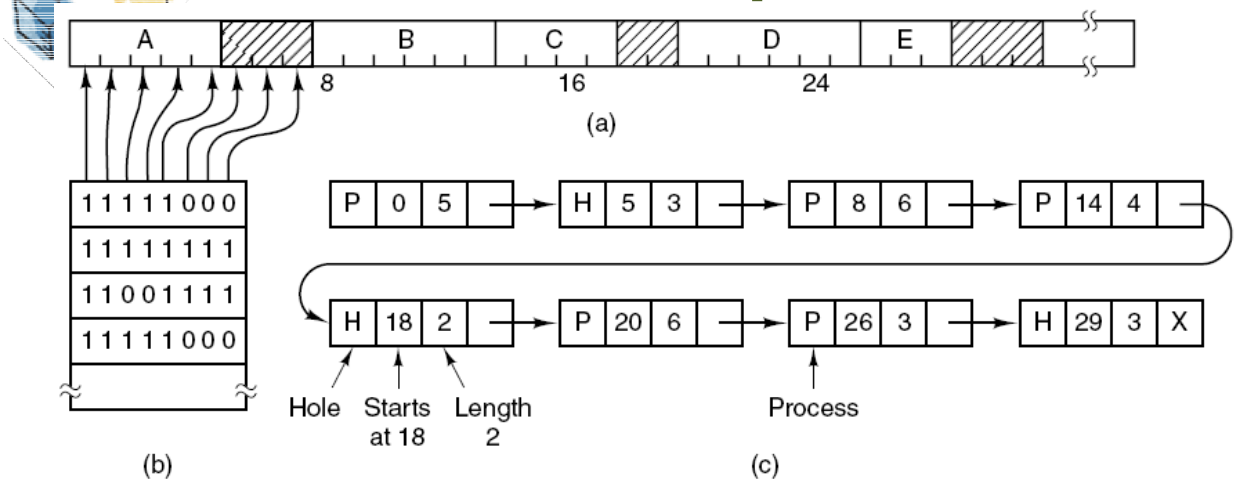

- ✓ (a) Allocating space for growing data segment.
- ✓ (b) Allocating space for growing stack, growing data segment.



# Managing Free Memory

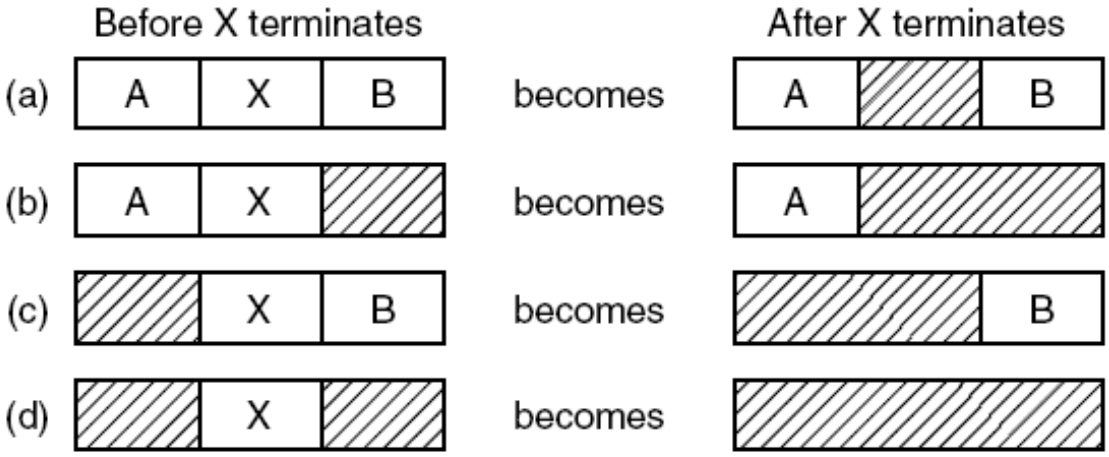
- ✓ Memory management with bitmaps
- ✓ Memory management with linked lists

# Memory Management with Bitmaps



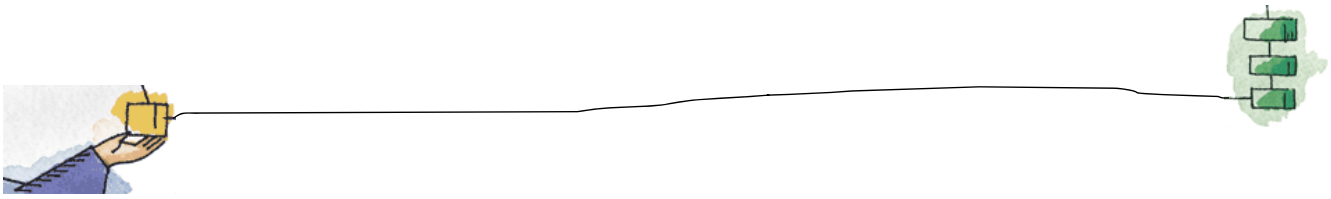

- ✓ (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free.
- ✓ (b) The corresponding bitmap.
- ✓ (c) The same information as a list.

# Memory Management with Linked Lists

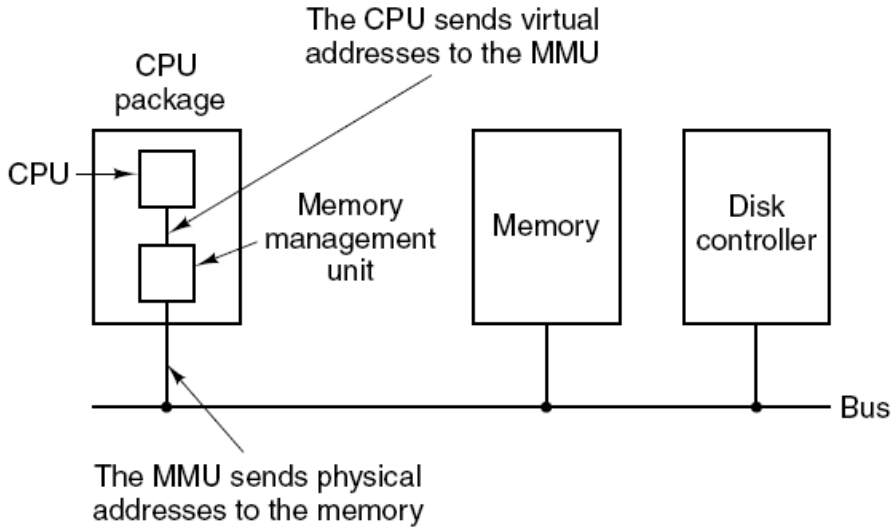


# Virtual Memory – Paging (1)

- ✓ Овозможува проширување на адресниот простор над капацитетот на главната меморија
- ✓ Меморијата се дели на страници, и само дел од нив се во главната меморија.
- ✓ Останатите се чуваат на диск.



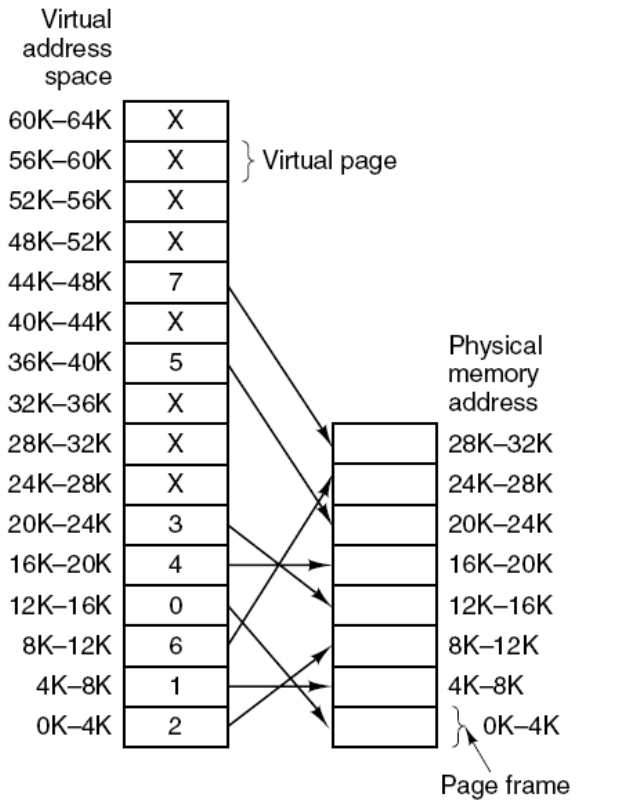
# Virtual Memory – Paging (1)




- ✓ The position and function of the MMU – shown as being a part of the CPU chip (it commonly is nowadays). Logically it could be a separate chip, was in years gone by.

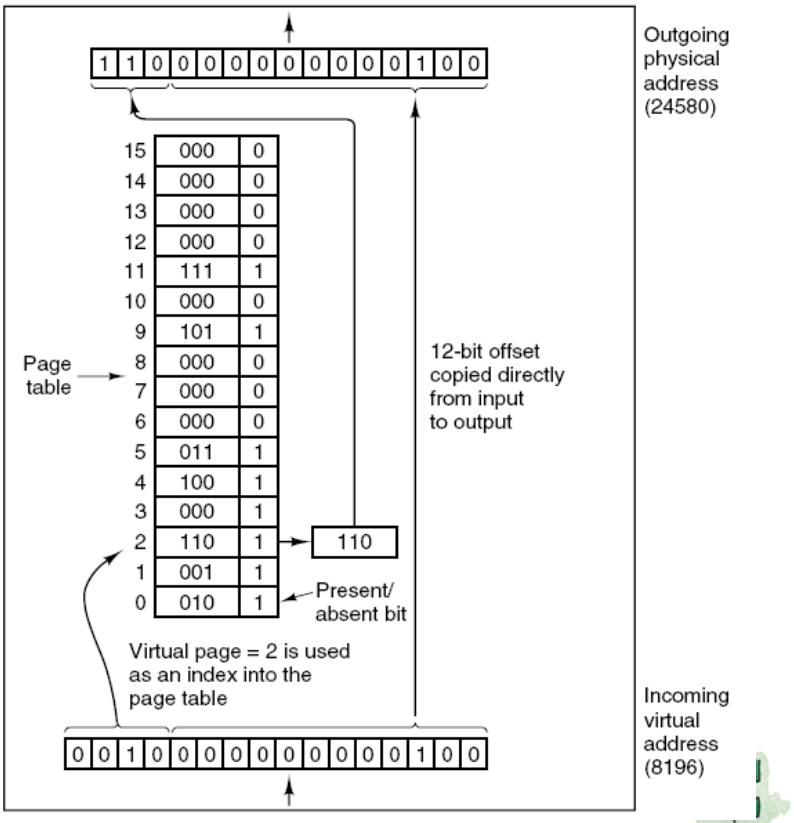
# Paging (2)

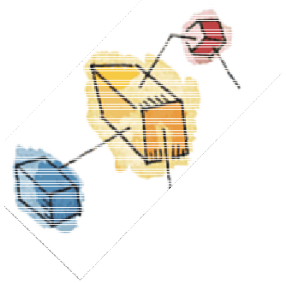
- ✓ Relation between virtual addresses and physical memory addresses given by page table.
- ✓ Every page begins on a multiple of 4096, ends 4095 addresses higher,
  - ✓ 4K–8K really means 4096–8191
  - ✓ 8K to 12K means 8192–12287.




# Paging (

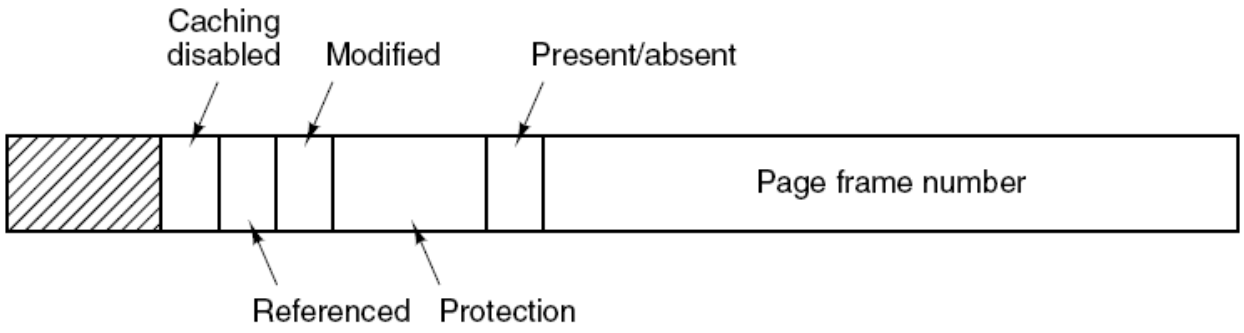
- ✓ The internal operation of the MMU with 16 4-KB pages.

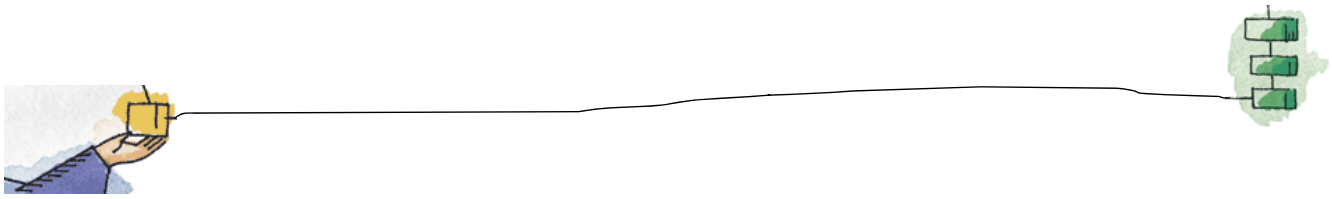




# Structure of Page Table Entry

✓A typical page table entry.

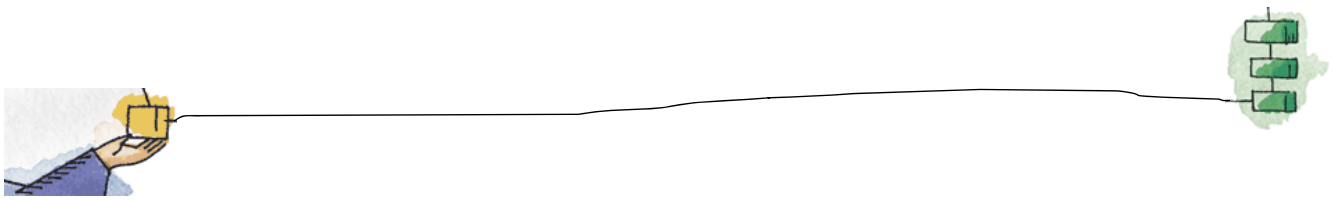


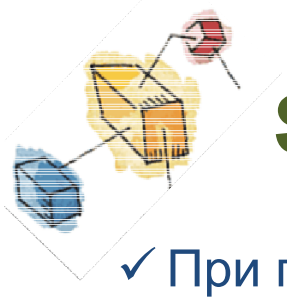



# Speeding Up Paging

✓Paging implementation issues

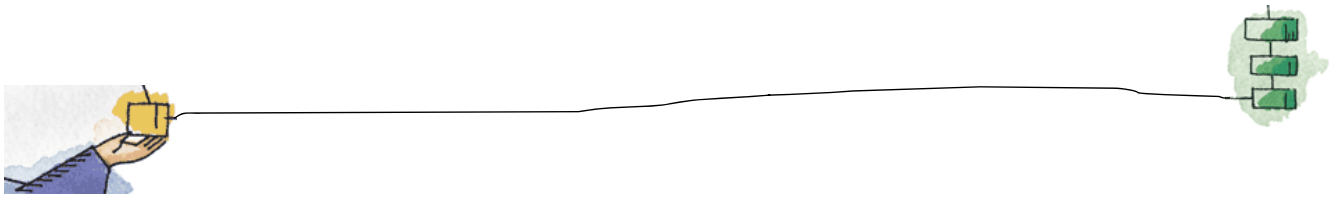
1. The mapping from virtual address to physical address must be fast.
2. If the virtual address space is large, the page table will be large.





# Speeding Up Paging

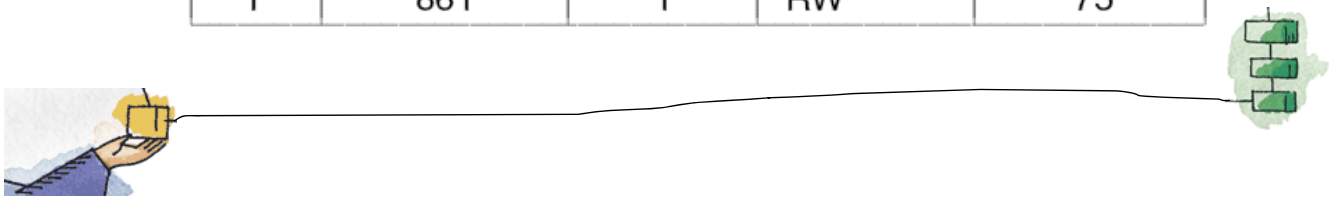
- ✓ При пристап до меморија мора да се пристапи и до табелата на страници за да се креира физичката адреса
- ✓ TLB претставува концепт на кеширање на најчесто користените редови од табелата на страници
  - ✓ На пример Pentium може да чува 64 реда од табелата на страници во процесорот

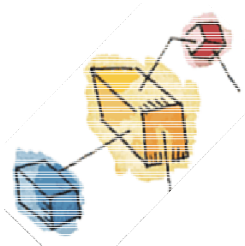


# Translation Lookaside Buffers

- ✓ A TLB to speed up paging.

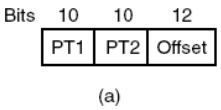
| Valid | Virtual page | Modified | Protection | Page frame |
|---|---|---|---|---|
| 1 | 140 | 1 | RW | 31 |
| 1 | 20 | 0 | R X | 38 |
| 1 | 130 | 1 | RW | 29 |
| 1 | 129 | 1 | RW | 62 |
| 1 | 19 | 0 | R X | 50 |
| 1 | 21 | 0 | R X | 45 |
| 1 | 860 | 1 | RW | 14 |
| 1 | 861 | 1 | RW | 75 |





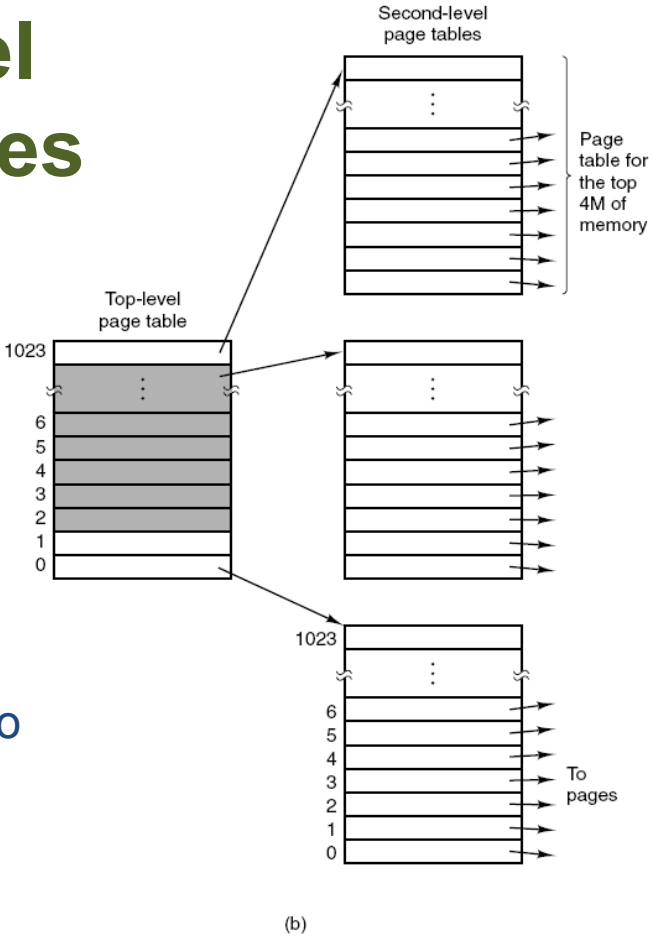
# Multilevel Page Tables

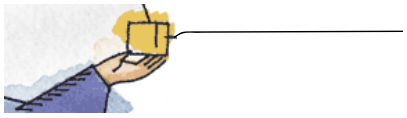
- ✓ Поради проблемот со огромните табели на станици во пракса се користат повеќе нивовски табели

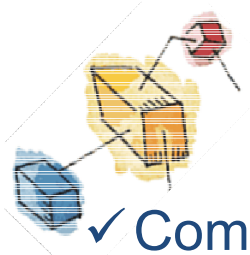



- ✓ 1.000.000 страници за 32 битен адресен простор и 4KB станици

- ✓ (a) A 32-bit address with two page table fields.
- ✓ (b) Two-level page tables.

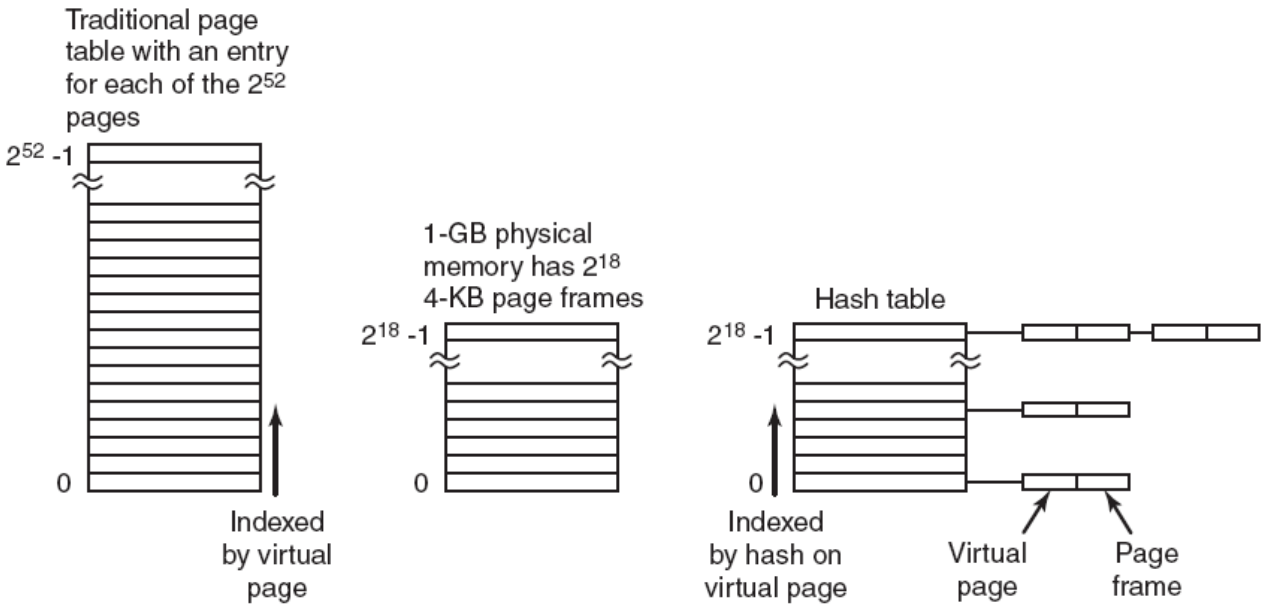




# Inverted Page Tables

- ✓ Comparison of a traditional page table with an inverted page table.



# Алгоритми за замена на страница

- ✓ При настанувањето на Page Fault треба се избере која страница да се отстрани за да се направи место за новата страница
- ✓ Променетите страници мора да се снимат на диск
- ✓ Алгоритам за оптимална замена на страници
  - ✓ Треба да се замени страницата која би била потребна најдалеку во иднината
  - ✓ Оптимално но невозможно да се направи

# Алгоритми за замена на страница

- ✓ Optimal page replacement algorithm
- ✓ Not recently used page replacement
- ✓ First-In, First-Out page replacement
- ✓ Second chance page replacement
- ✓ Clock page replacement
- ✓ Least recently used page replacement
- ✓ Working set page replacement
- ✓ WSClock page replacement

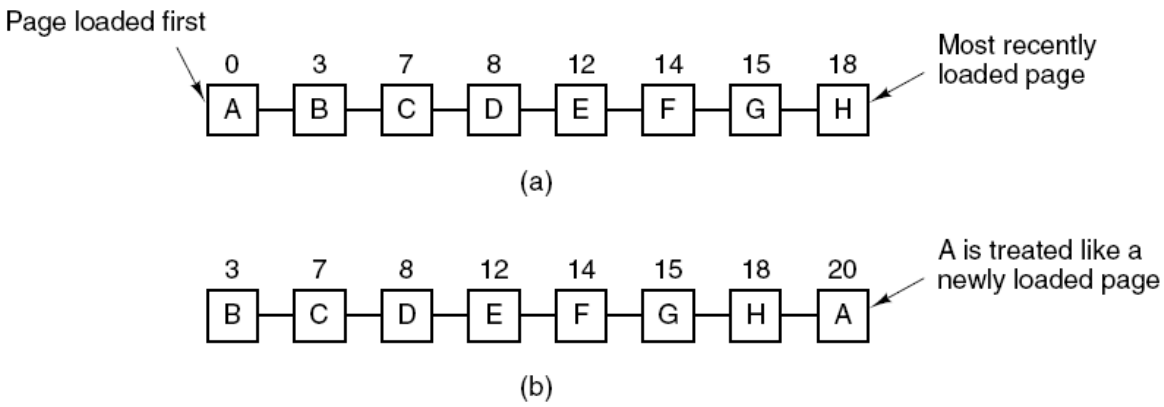
# Not recently used (NRU)

- ✓ За секоја страница има Reference bit и Modified bit
  - ✓ Битовите се поставуваат кога се пристапува или се модифицира станицата
  - ✓ Периодично овие битови се бришат
- ✓ Страниците се класифицираат во класи на следниов начин
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- ✓ NRU ги отстранува страниците по случаен избор почнувајќи од најниската не празна класа

# First-In First-Out (FIFO)

- ✓ Се чува поврзана листа од сите страници
  - ✓ Тие се подредуваат по редоследот по кој доаѓаат во меморијата
- ✓ Се отстранува страницата која е на почетокот на листата
- ✓ Недостаток
  - ✓ Страницата која е надолго време во меморијата може да биде нај користена

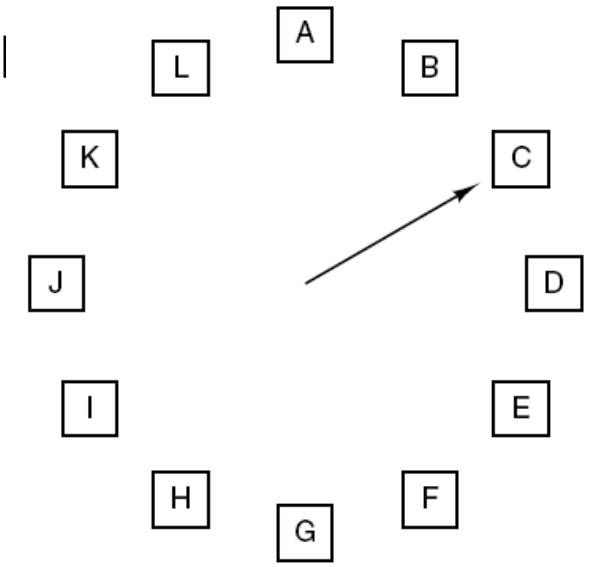
# Second Chance Algorithm




(a) Pages sorted in FIFO order. Ако R битот е поставен страницата се става на крај и се поробува да се отстрани наредната страница која има R=0

(b) Page list if a page fault occurs at time 20 and A has its R bit set. The numbers above the pages are their load times.

# The Clock Page Replacement Algorithm




When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

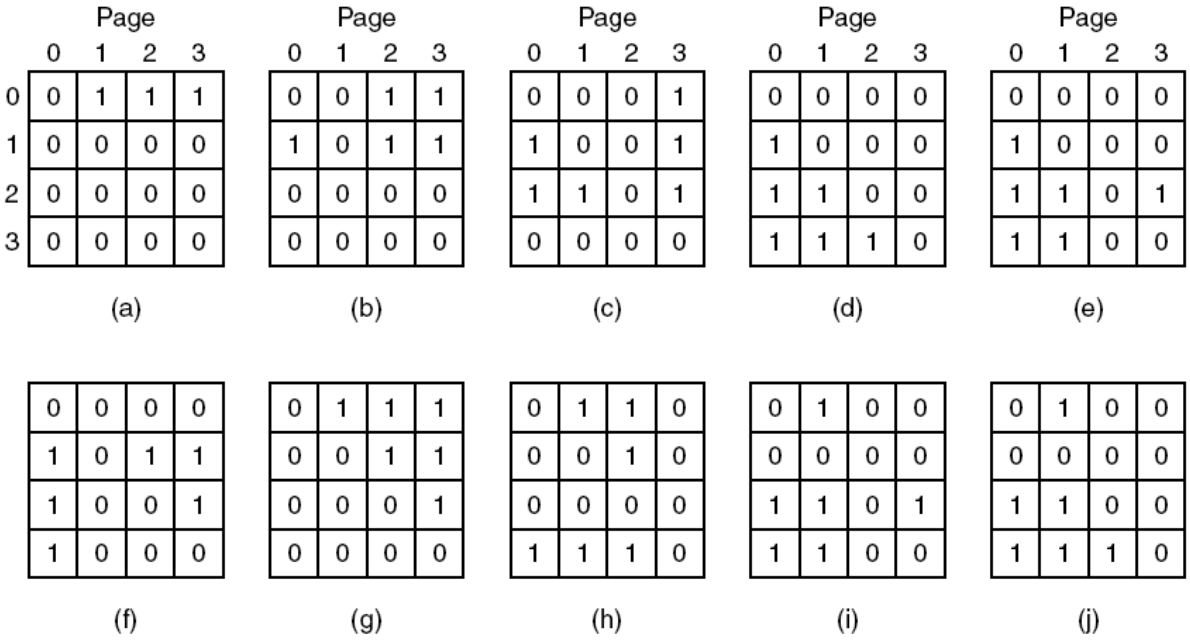
R = 0: Evict the page

R = 1: Clear R and advance hand

# Least Recently Used (LRU)

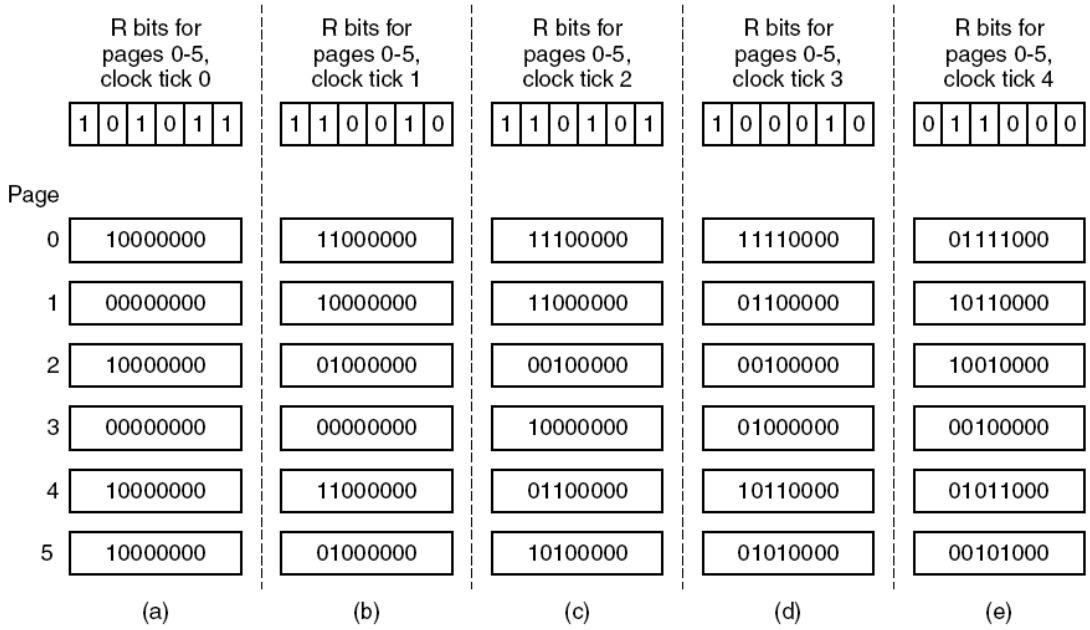
- Се претпоставува дека станиците кои се користени неодамна ќе се користат пак
  - Се отфрлаат страниците кои не се користени најдолго време
  - Да се чува сортирана листа од страници врз основа на бројот на пристапи до нив
- Решение 1
  - Листата да се ажурира на секој пристап до меморијата
  - За секоја страница се чува бројач кој се зголемува при секој пристап до неа
- Решение 2
  - Периодично се ресетира бројачот

# LRU Page Replacement



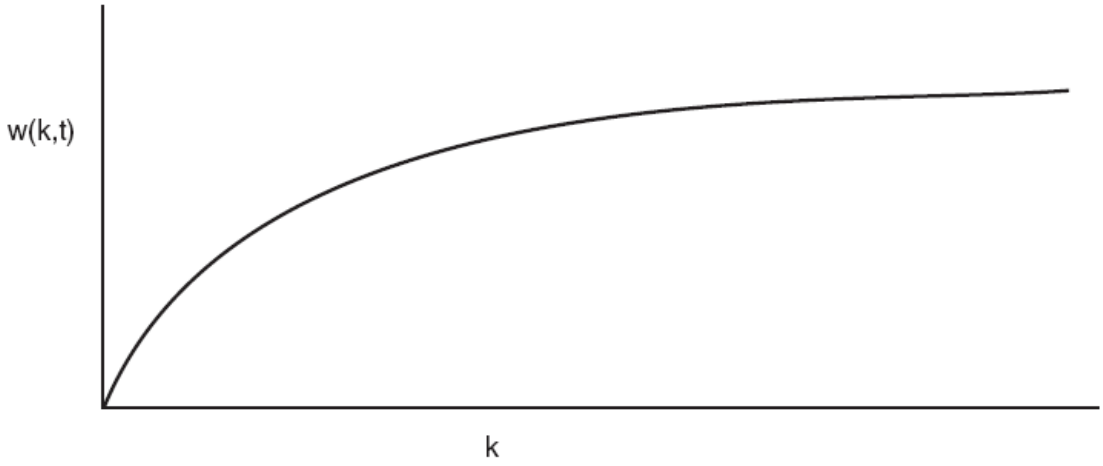

- LRU using a matrix when pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

# Simulating LRU in Software

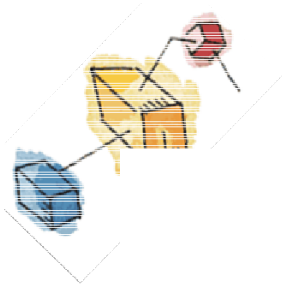



- ✓ The aging algorithm simulates LRU in software.
- ✓ Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

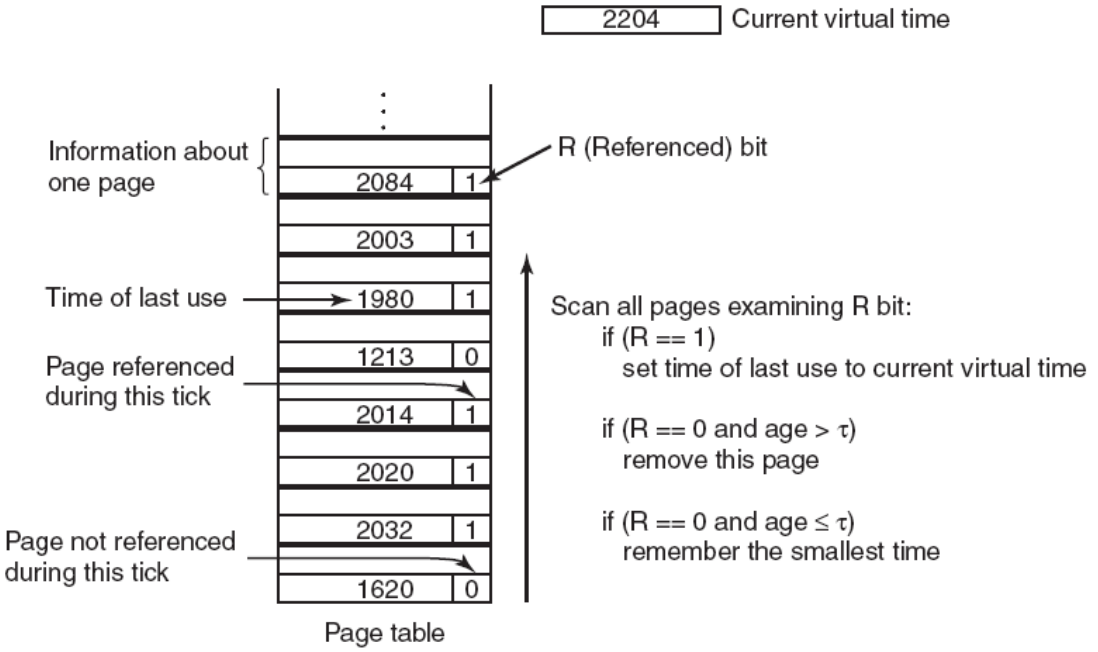
# Working Set Page Replacement (1)




The working set is the set of pages used by the k most recent memory references. The function w(k, t) is the size of the working set at time t.



# Working Set Page




The working set algorithm.

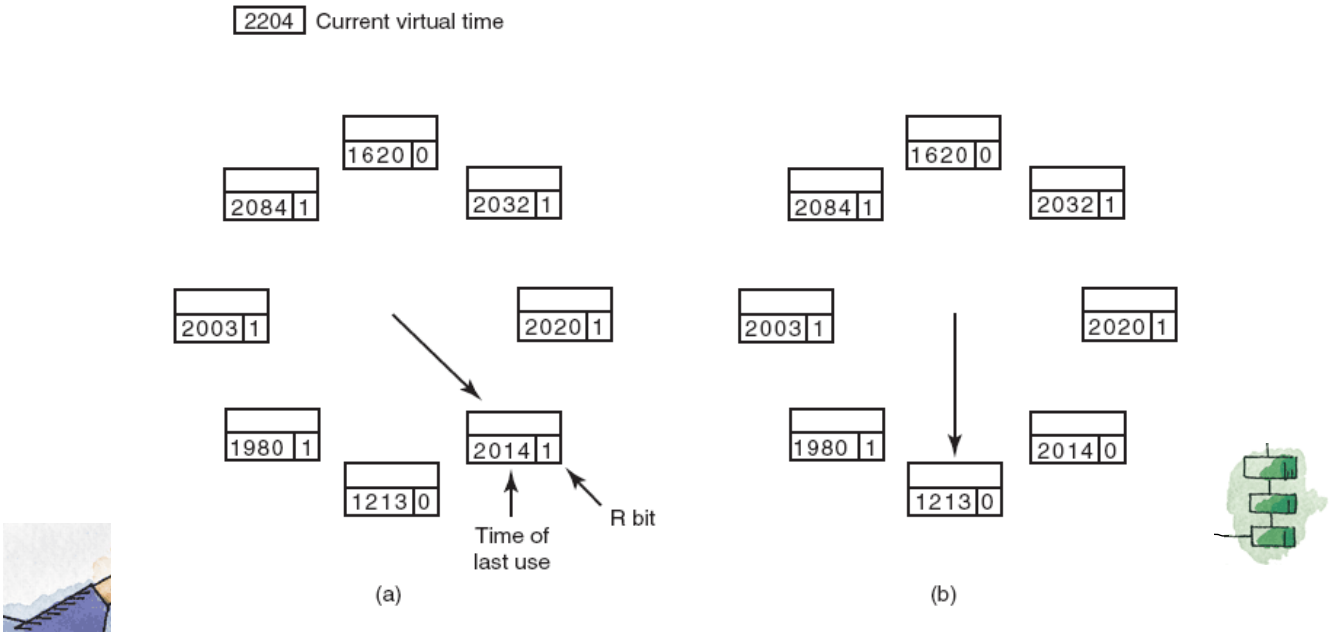
# The WSClock Page Replacement Algorithm (1)

When the hand comes all the way around to its starting point there are two cases to consider:

1. At least one write has been scheduled.
2. No writes have been scheduled.

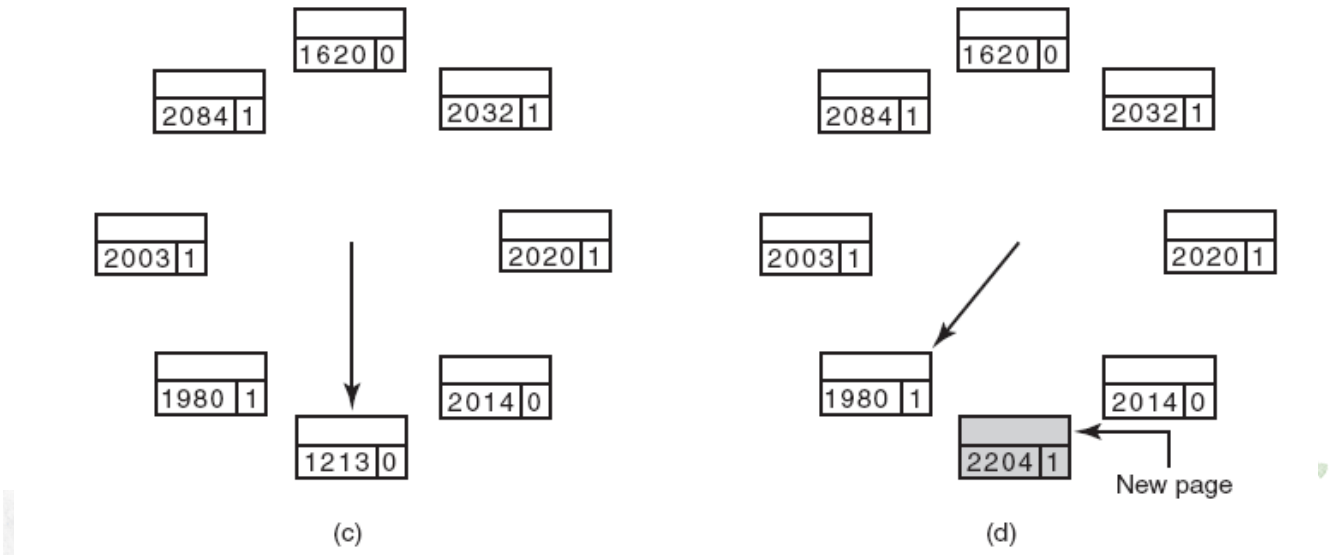
# The WSClock Page Replacement Algorithm (2)

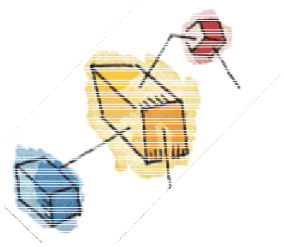
Operation of the WSClock algorithm. (a) and (b) give an example of what happens




# The WSClock Page Replacement Algorithm (3)

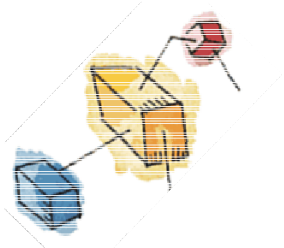
Operation of the WSClock algorithm. (c) and (d) give an example of R = 0.





# Summary of Page Replacement Algorithms

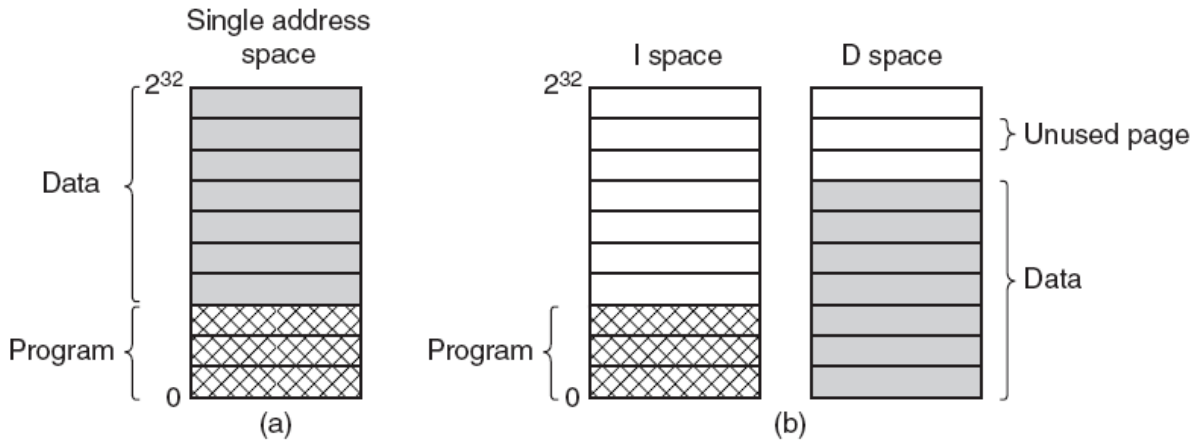
| Algorithm | Comment |
|---|---|
| Optimal | Not implementable, but useful as a benchmark |
| NRU (Not Recently Used) | Very crude approximation of LRU |
| FIFO (First-In, First-Out) | Might throw out important pages |
| Second chance | Big improvement over FIFO |
| Clock | Realistic |
| LRU (Least Recently Used) | Excellent, but difficult to implement exactly |
| NFU (Not Frequently Used) | Fairly crude approximation to LRU |
| Aging | Efficient algorithm that approximates LRU well |
| Working set | Somewhat expensive to implement |
| WSClock | Good efficient algorithm |



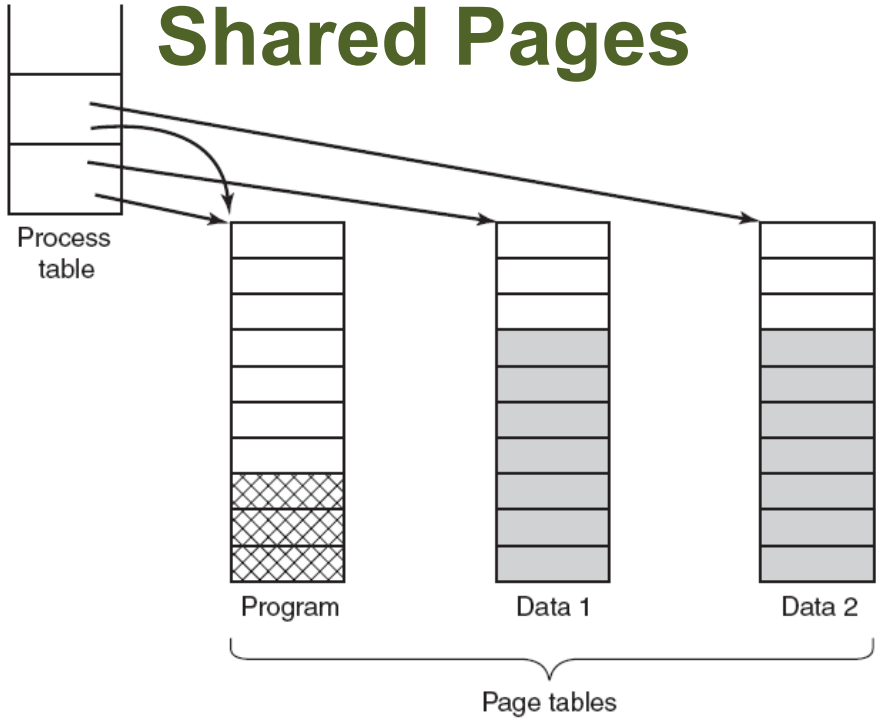
# Design issues for Paging Systems

- ✓ Глобална и локална замена
- ✓ Контрола на оптоварувањето
- ✓ Големина на страниците
- ✓ Поделба на адресните простори за податоци и прогрми
- ✓ Споделени страници
- ✓ Споделени библиотеки

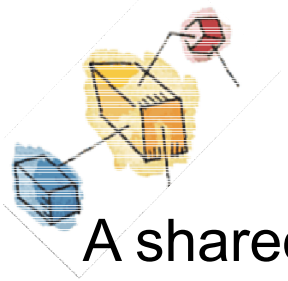
# Separate Instruction and Data Spaces




# Shared Pages

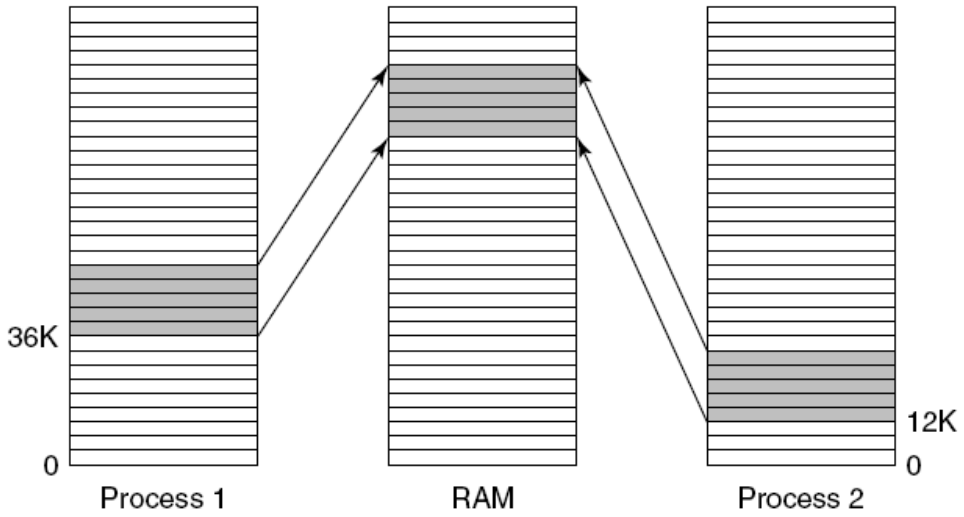


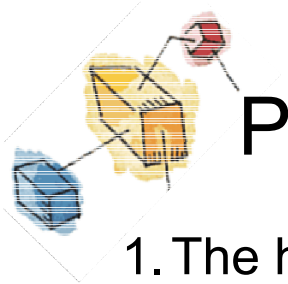

Two processes sharing the same program sharing its page table.



# Shared Libraries

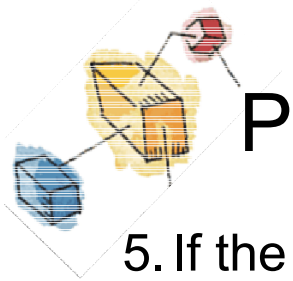
A shared library being used by two processes.



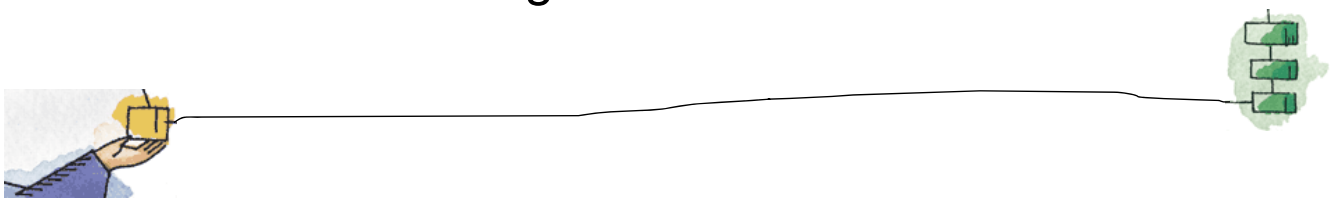
# Page Fault Handling (1)

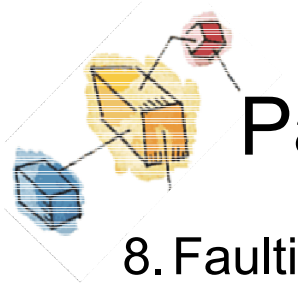
1. The hardware traps to the kernel, saving the program counter on the stack.
2. An assembly code routine is started to save the general registers and other volatile information.
3. The operating system discovers that a page fault has occurred, and tries to discover which virtual page is needed.
4. Once the virtual address that caused the fault is known, the system checks to see if this address is valid and the protection consistent with the access



# Page Fault Handling (2)

5. If the page frame selected is dirty, the page is scheduled for transfer to the disk, and a context switch takes place.
6. When page frame is clean, operating system looks up the disk address where the needed page is, schedules a disk operation to bring it in.
7. When disk interrupt indicates page has arrived, page tables updated to reflect position, frame marked as being in normal state.





# Page Fault Handling (3)

8. Faulting instruction backed up to state it had when it began and program counter reset to point to that instruction.
9. Faulting process scheduled, operating system returns to the (assembly language) routine that called it.
10. This routine reloads registers and other state information and returns to user space to continue execution, as if no fault had occurred.