Chapter 4

Memory Management

4.1 Basic memory management
4.2 Swapping
4.3 Virtual memory
4.4 Page replacement algorithms
4.6 Design issues for paging systems
4.7 Implementation issues
4.8 Segmentation

sun@hit.edu.cn

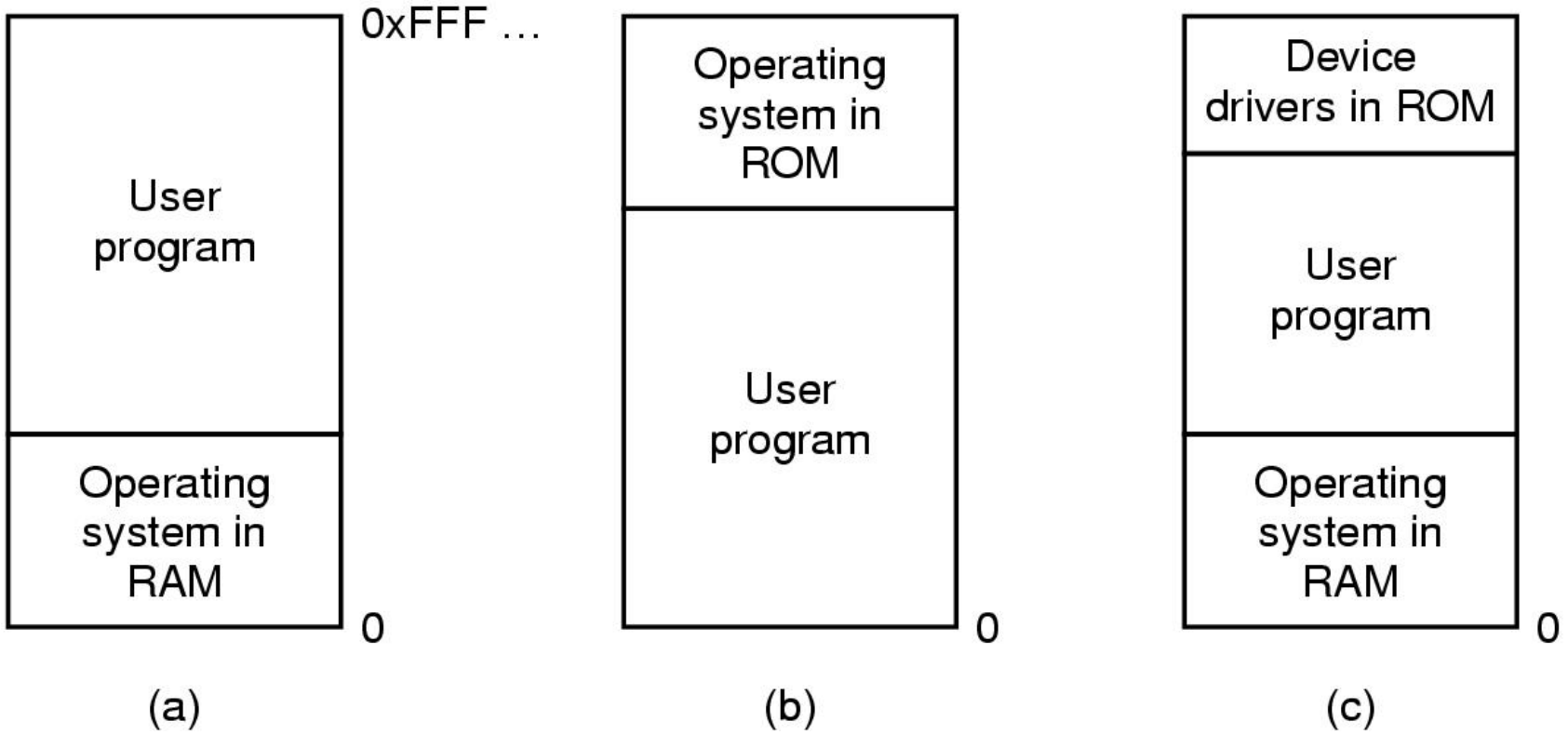
Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Memory manager handles the memory hierarchy

sun@hit.edu.cn

Basic Memory Management

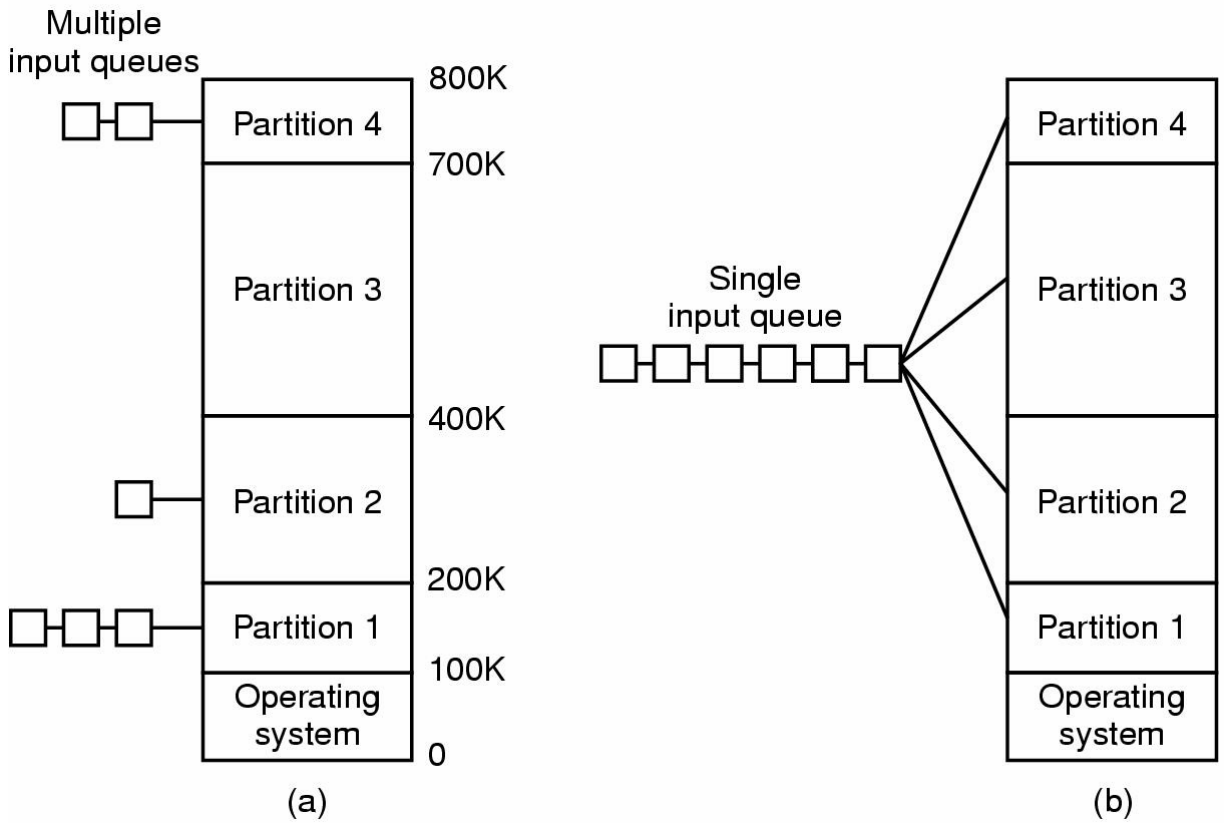
Monoprogramming without Swapping or Paging



Three simple ways of organizing memory

- an operating system with one user process

Multiprogramming with Fixed Partitions



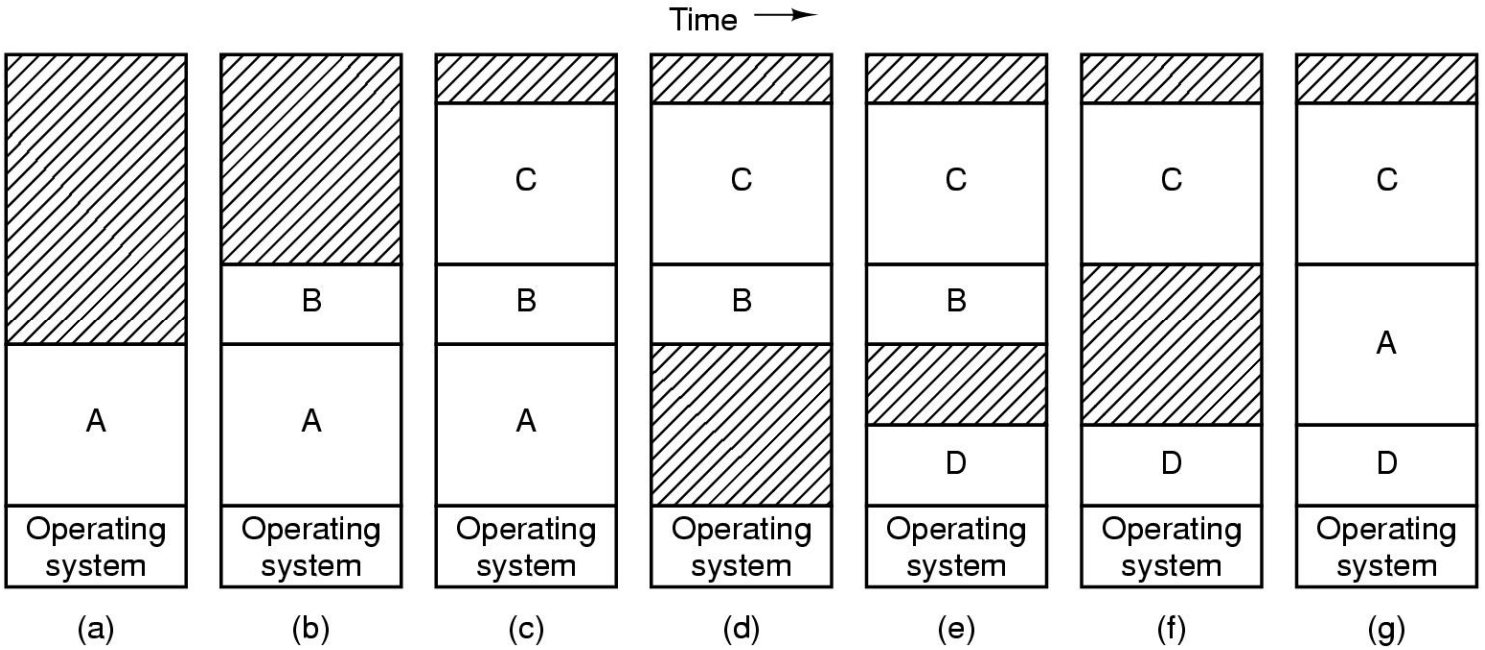
- **Fixed memory partitions**
  - **separate input queues for each partition**
  - **single input queue**

Relocation and Protection

- **Cannot be sure where program will be loaded in memory**
  - **address locations of variables, code routines cannot be absolute**
  - **must keep a program out of other processes' partitions**
- **Use base and limit values**
  - **address locations added to base value to map to physical address**
  - **address locations larger than limit value is an error**

sun@hit.edu.cn

Swapping (交换)

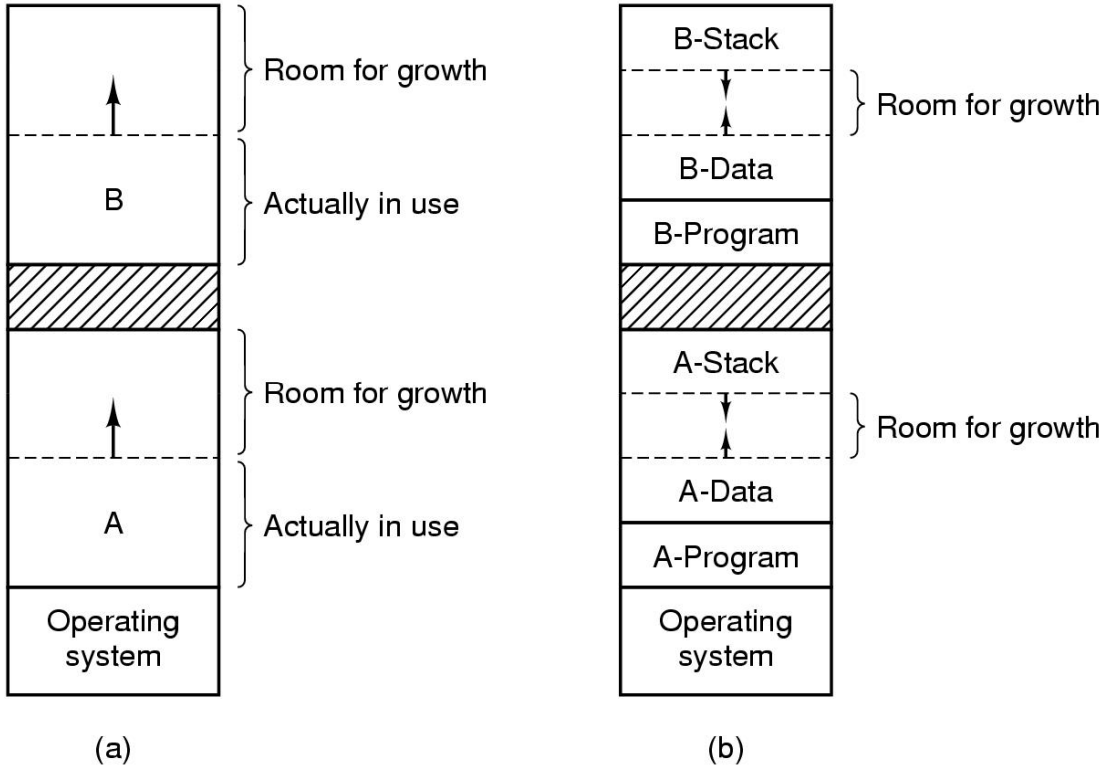


Memory allocation changes as

- processes come into memory
- leave memory

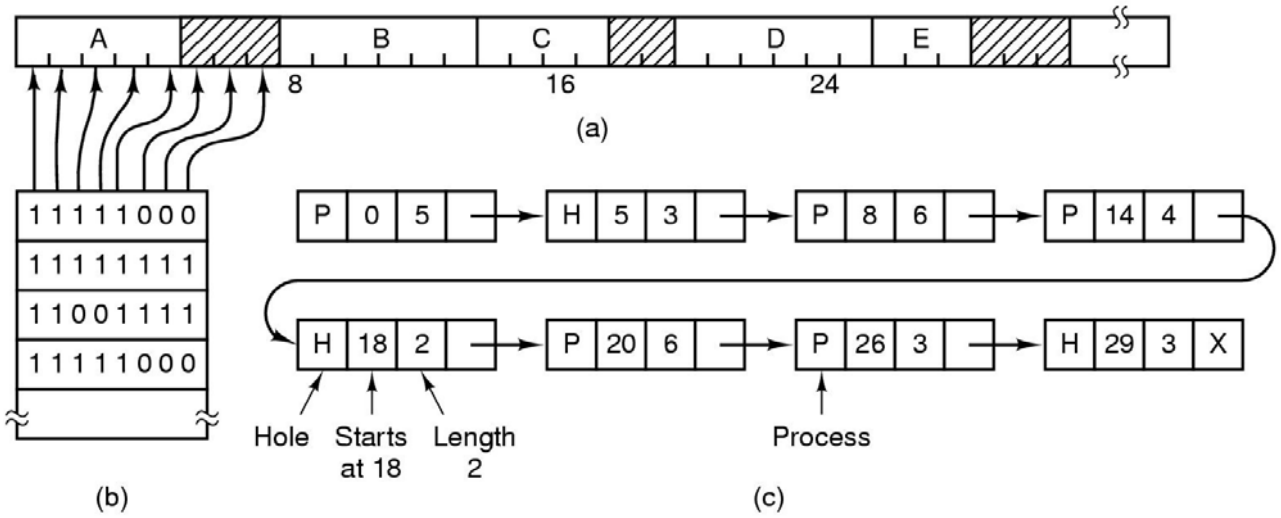
Shaded regions are unused memory

Swapping (2)



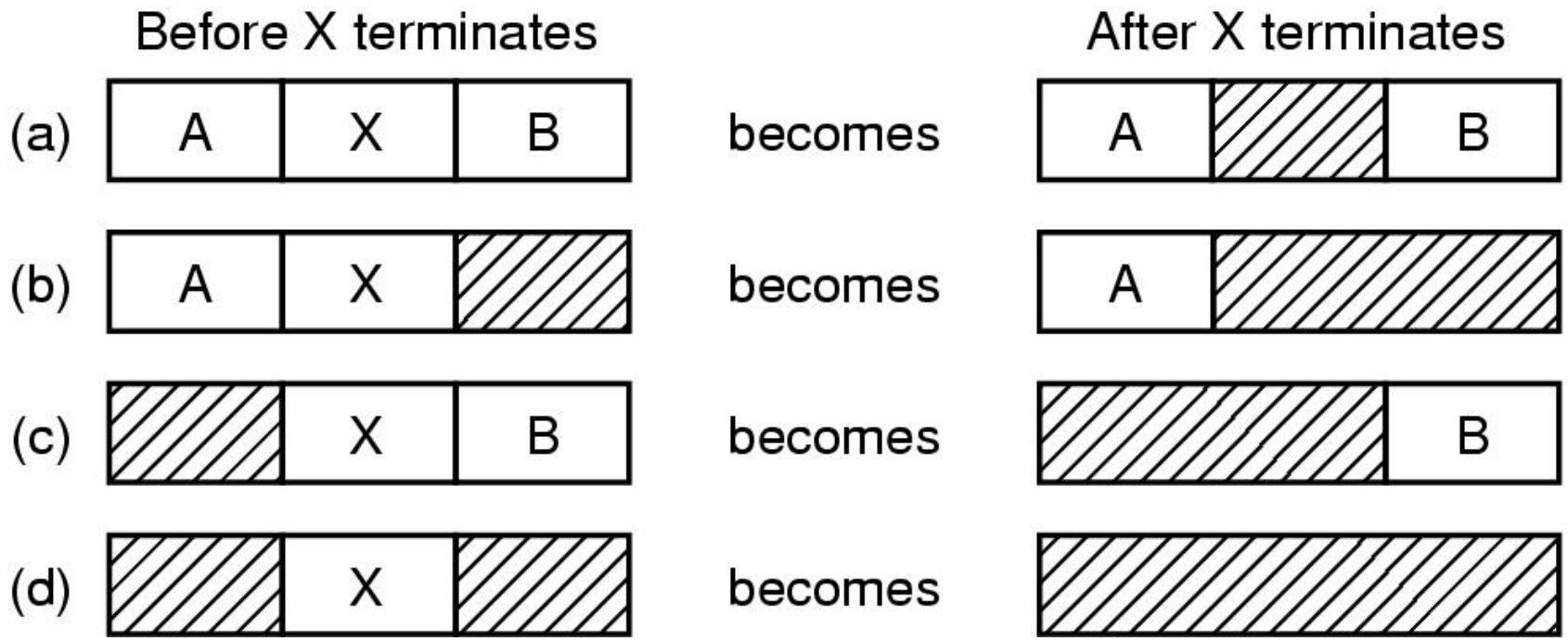
- **Allocating space for growing data segment**
- **Allocating space for growing stack & data segment**

Memory Management with Bitmaps



- **Part of memory with 5 processes, 3 holes**
  - **tick marks show allocation units**
  - **shaded regions are free**
- **Corresponding bitmap**
- **Same information as a list**

Memory Management with Linked Lists



Four neighbor combinations for the terminating process X

sun@hit.edu.cn

Memory Allocation Algorithm

- **First fit**
  - **Use the first hole that is big enough in the list**
- **Next fit**
  - **Start searching the list from the place where it left off last time**
- **Best fit**
  - **Search the entire list and takes the smallest hole that is adequate**
- **Worst fit**
  - **Always take the largest available hole**
- **Quick fit**
  - **Maintain separate lists**

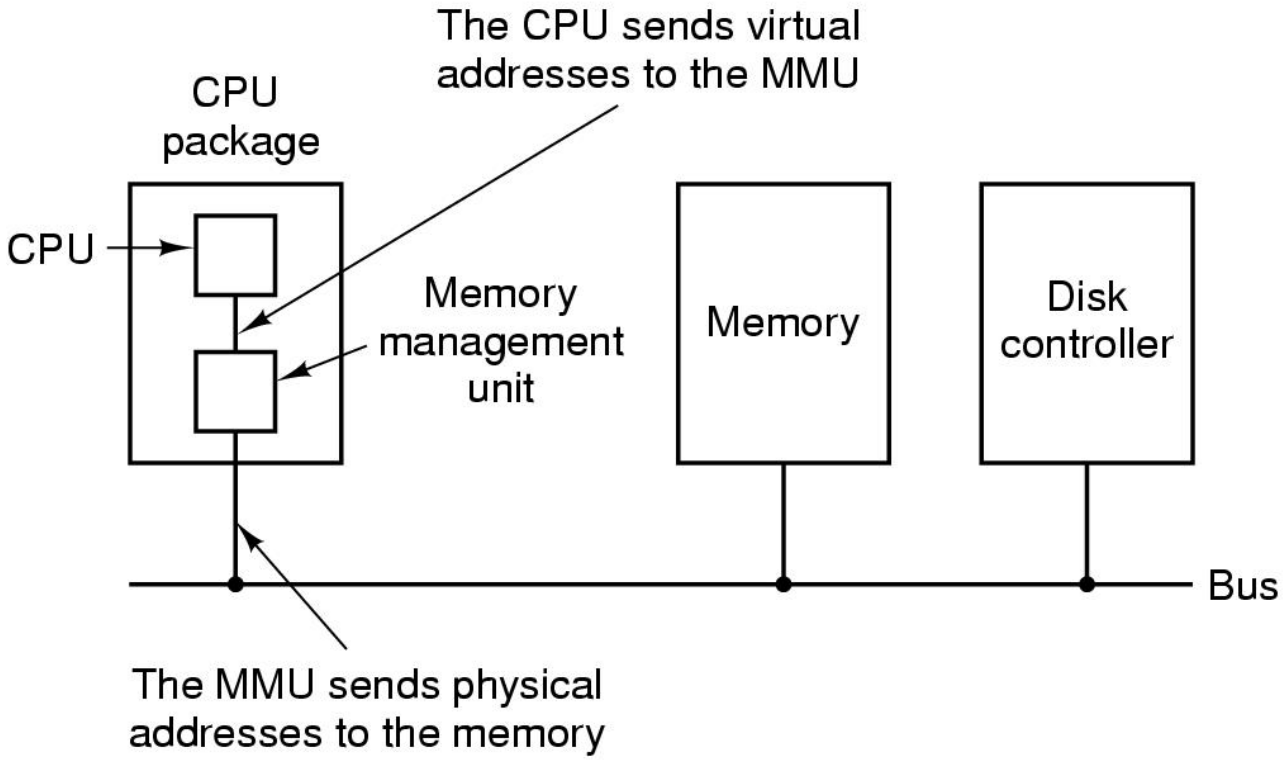
sun@hit.edu.cn

Virtual Memory

- **Overlays**
  - **Split the program into pieces**
- **Virtual memory**
  - **The OS keeps those parts of the program currently in use in main memory, and the rest on the disk**

sun@hit.edu.cn

Paging (1)



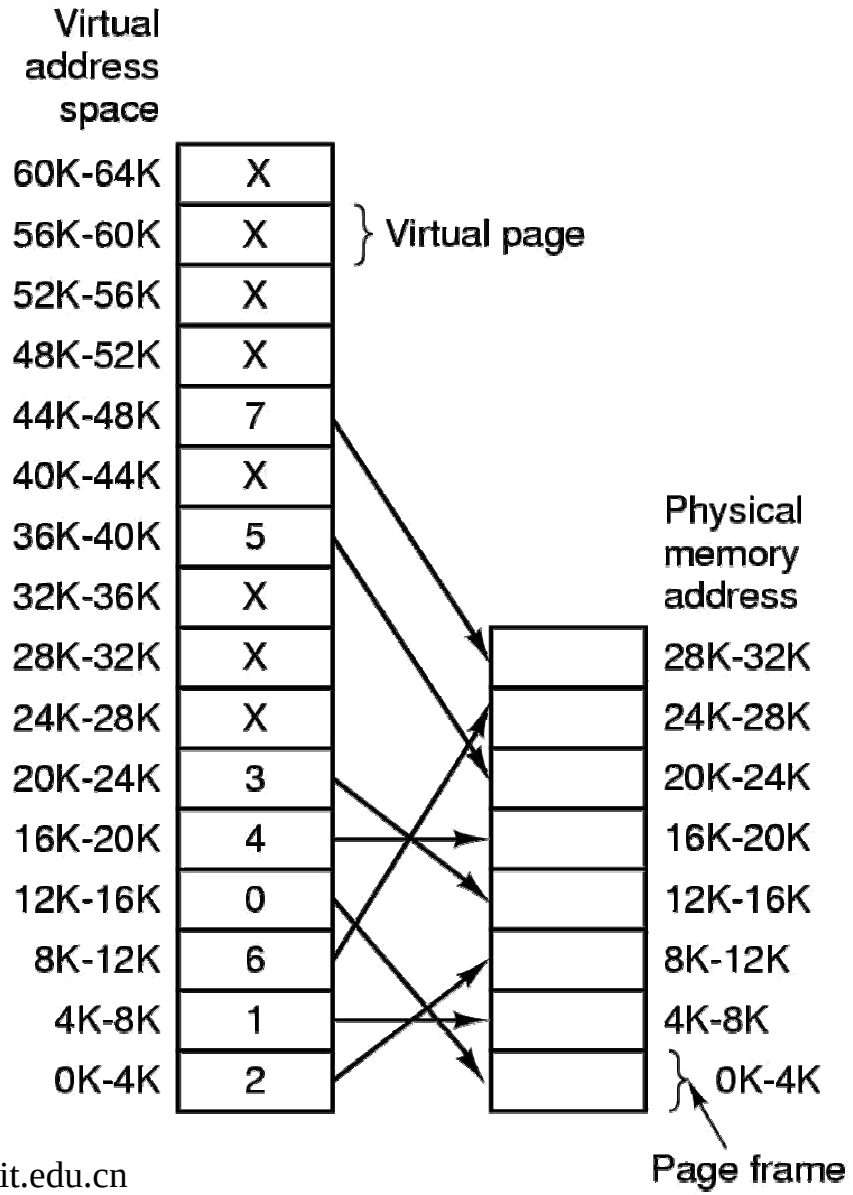
The position and function of the MMU

sun@hit.edu.cn

Paging (2)

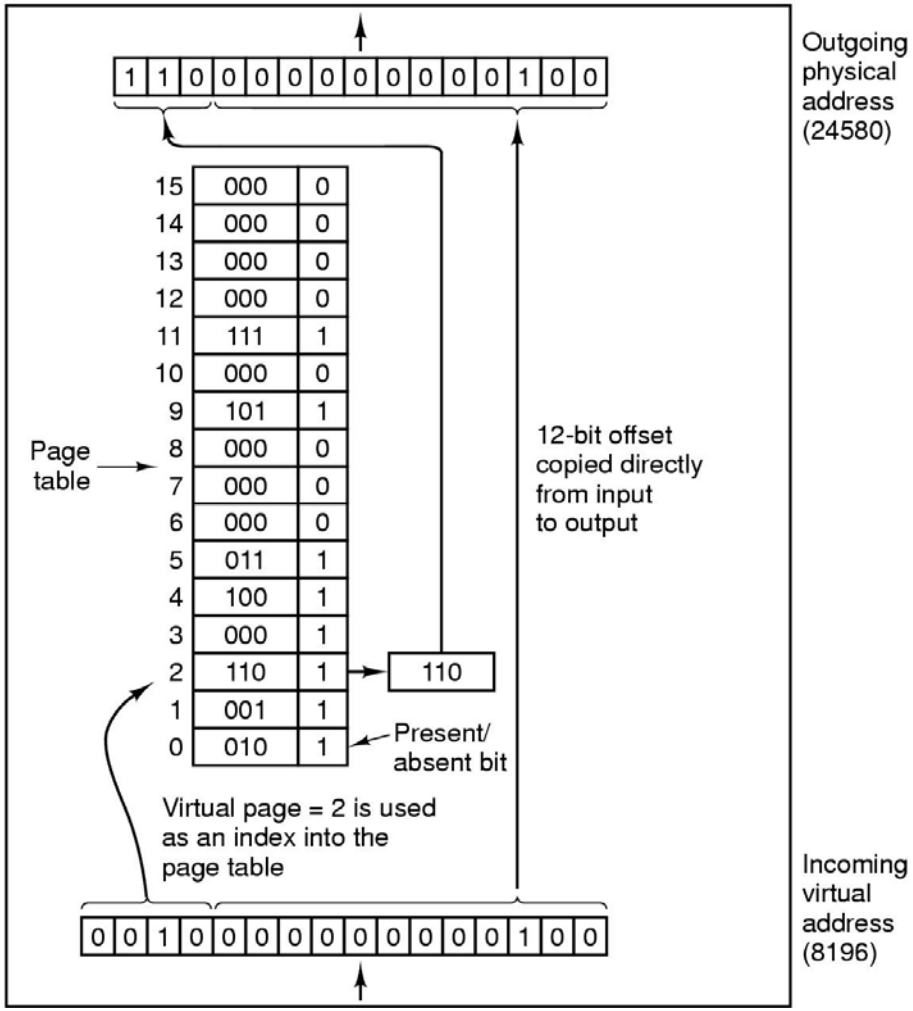
The relation between virtual addresses and physical memory addresses given by page table

```
MOV REG, 0
MOV REG, 8192
MOV REG, 20500
MOV REG, 32780
```



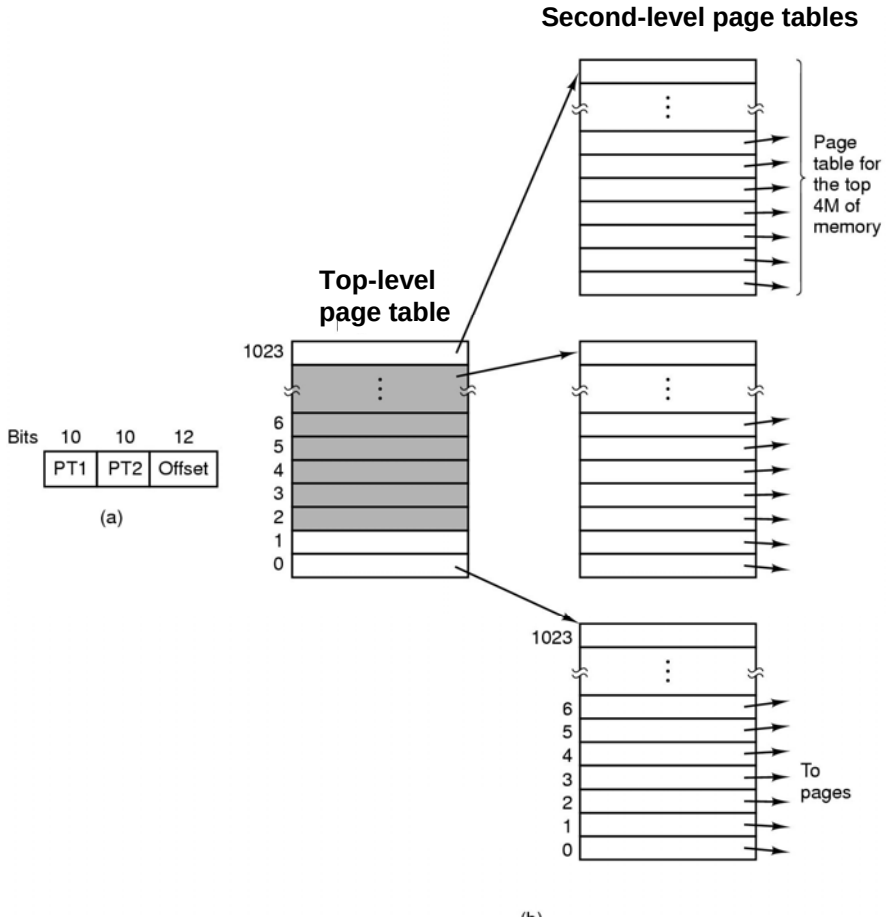
sun@hit.edu.cn

Page Tables (1)



Internal operation of MMU with 16 4 KB pages

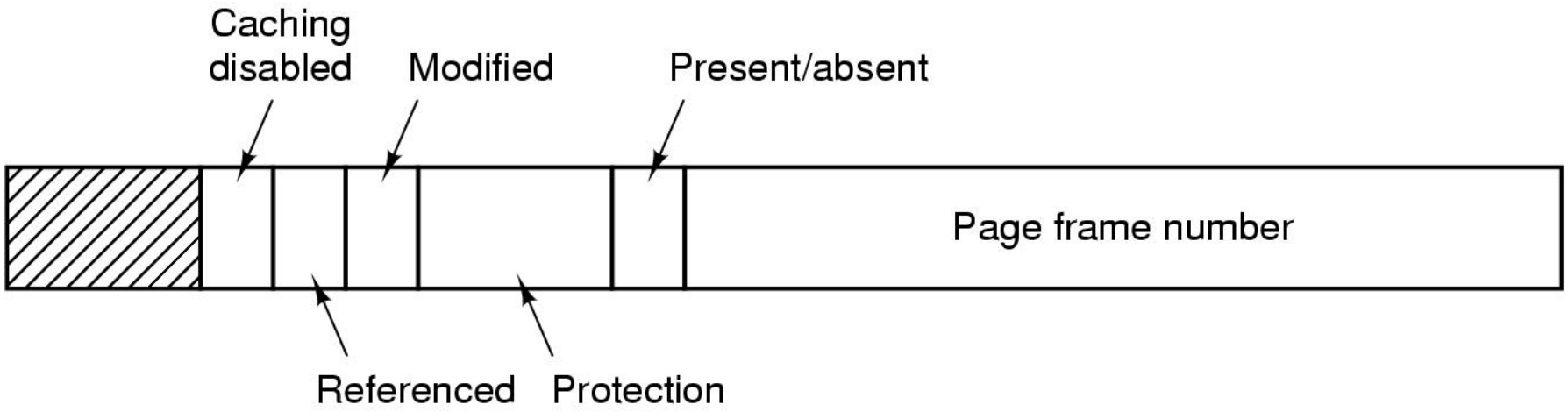
Page Tables (2)



- 32 bit address with 2 page table fields
- Two-level page tables

sun@hit.edu.cn

Page Tables (3)



Typical page table entry

sun@hit.edu.cn

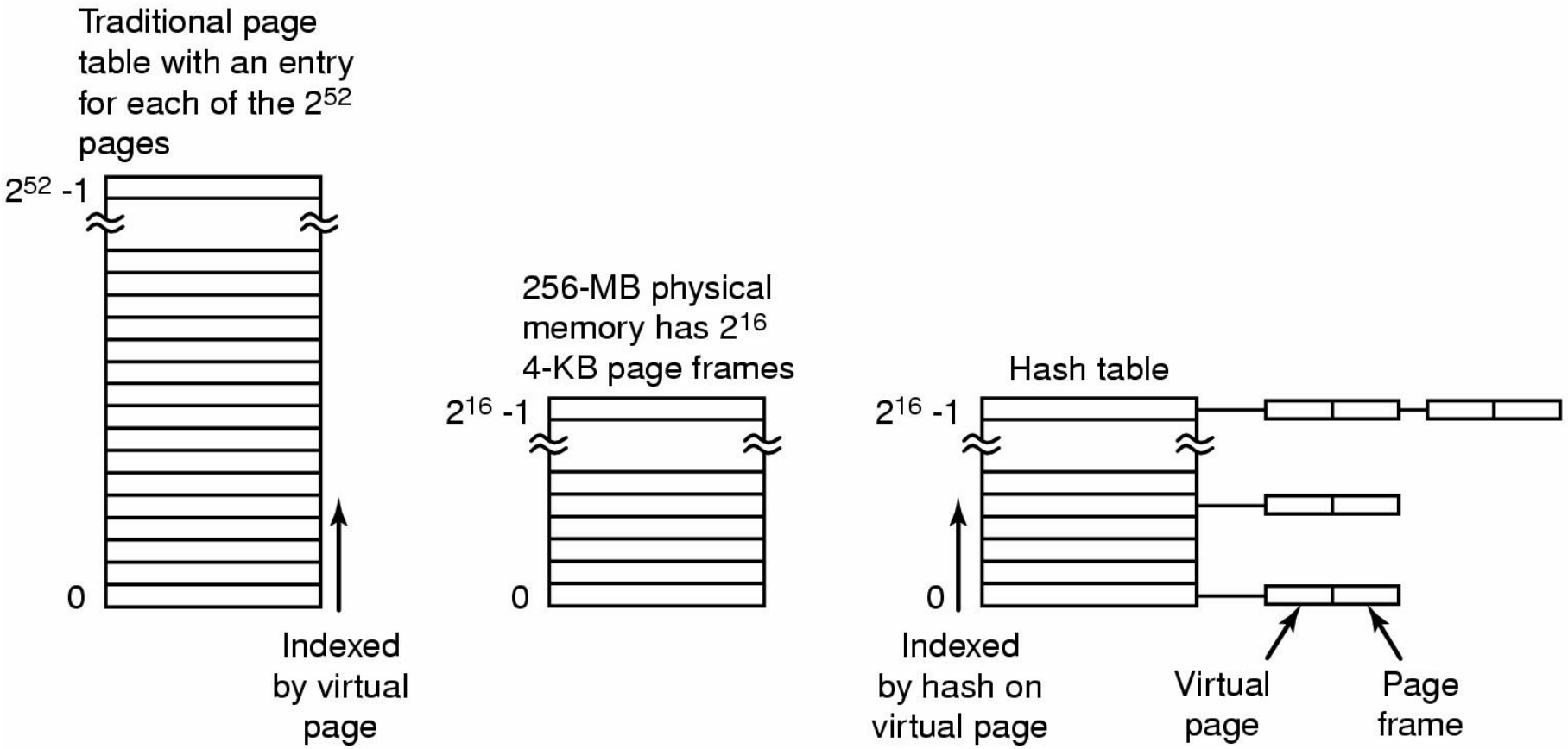
TLBs – Translation Lookaside Buffers (转换检测缓冲器)

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

A TLB to speed up paging

sun@hit.edu.cn

Inverted Page Tables



Comparison of a traditional page table with an inverted page table

sun@hit.edu.cn

Page Replacement Algorithms

- **Page fault forces choice**
  - **which page must be removed**
  - **make room for incoming page**
- **Modified page must first be saved**
  - **unmodified just overwritten**
- **Better not to choose an often used page**
  - **will probably need to be brought back in soon**

sun@hit.edu.cn

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable
- Estimate by …
  - logging page use on previous runs of process
  - although this is impractical

sun@hit.edu.cn

Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- NRU removes page at random
  - from lowest numbered non empty class
- NRU is easy to understand, moderately efficient to implement, and gives an adequate performance

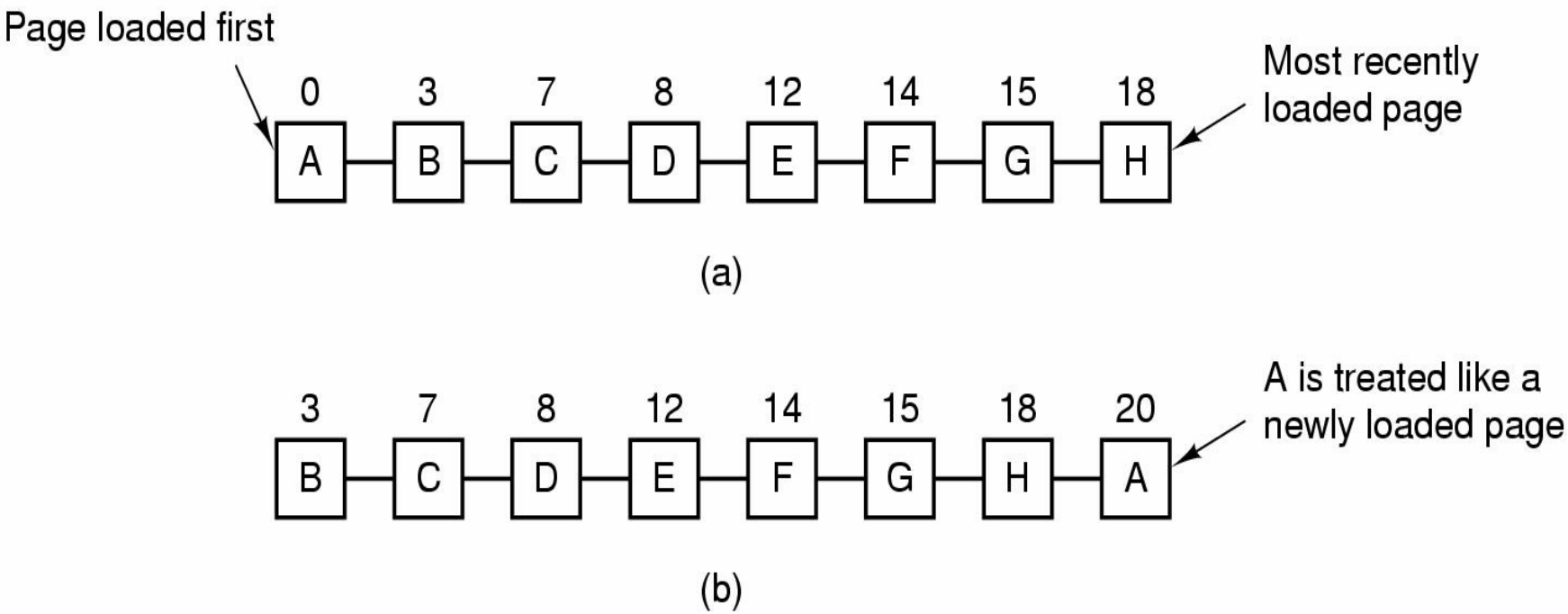
sun@hit.edu.cn

FIFO Page Replacement Algorithm

- **Maintain a linked list of all pages**
  - **in order they came into memory**
- **Page at beginning of list replaced**
- **Disadvantage**
  - **page in memory the longest may be often used**

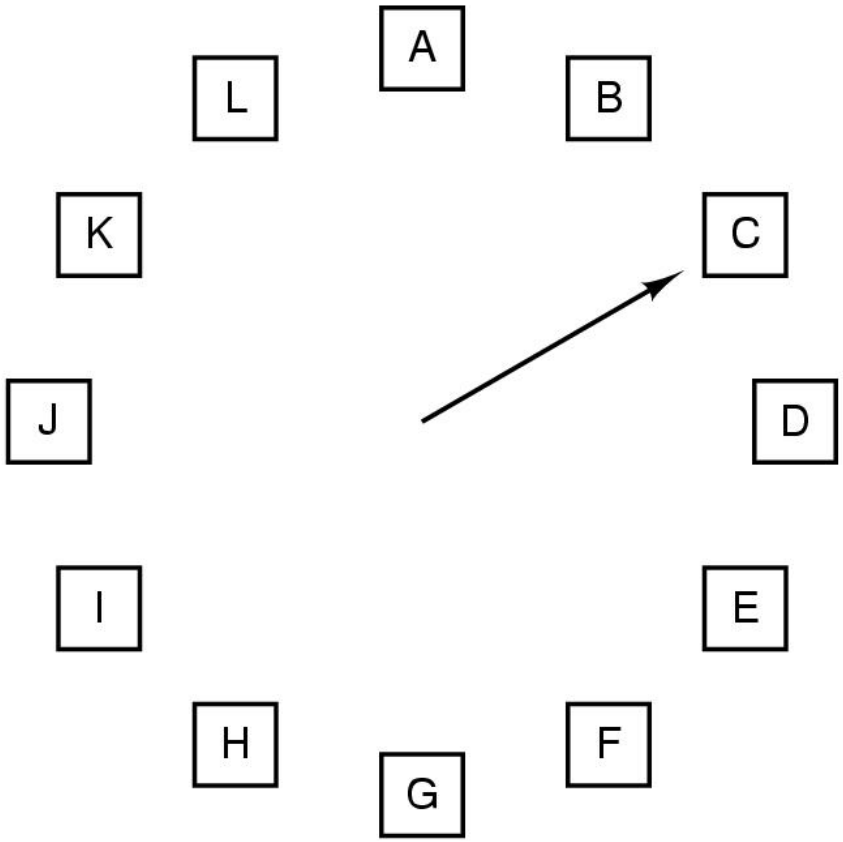
sun@hit.edu.cn

Second Chance Page Replacement Algorithm



- **Operation of a second chance**
  - **pages sorted in FIFO order**
  - **Page list if fault occurs at time 20, *A* has *R* bit set (numbers above pages are loading 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

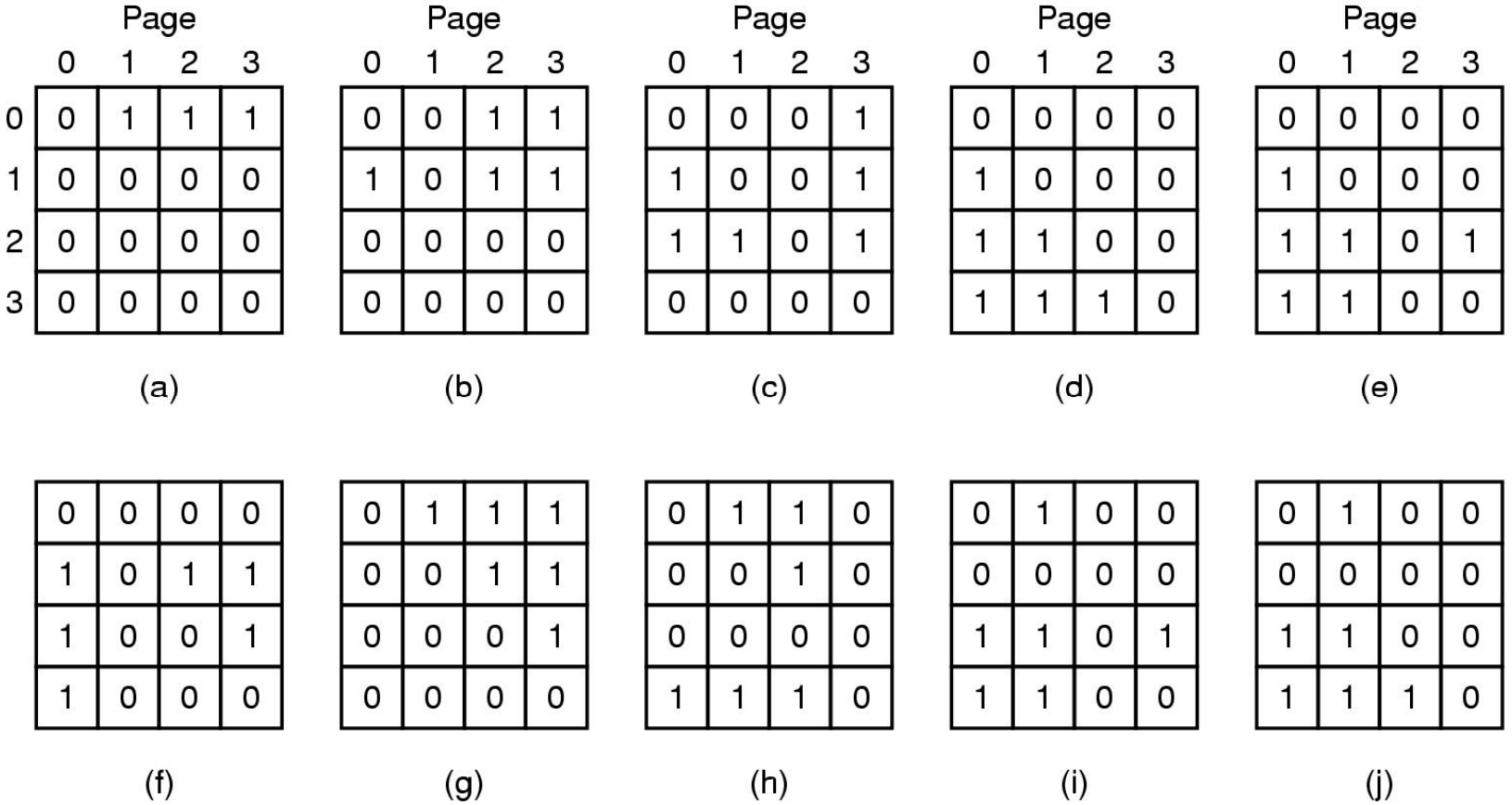
sun@hit.edu.cn

Least Recently Used (LRU)

- **Assume pages used recently will used again soon**
  - **throw out page that has been unused for longest time**
- **Must keep a linked list of pages by software**
  - **most recently used at front, least at rear**
  - **update this list every memory reference !!**
- **Alternatively keep counter in each page table entry by hardware**
  - **choose page with lowest value counter**
  - **periodically zero the counter**

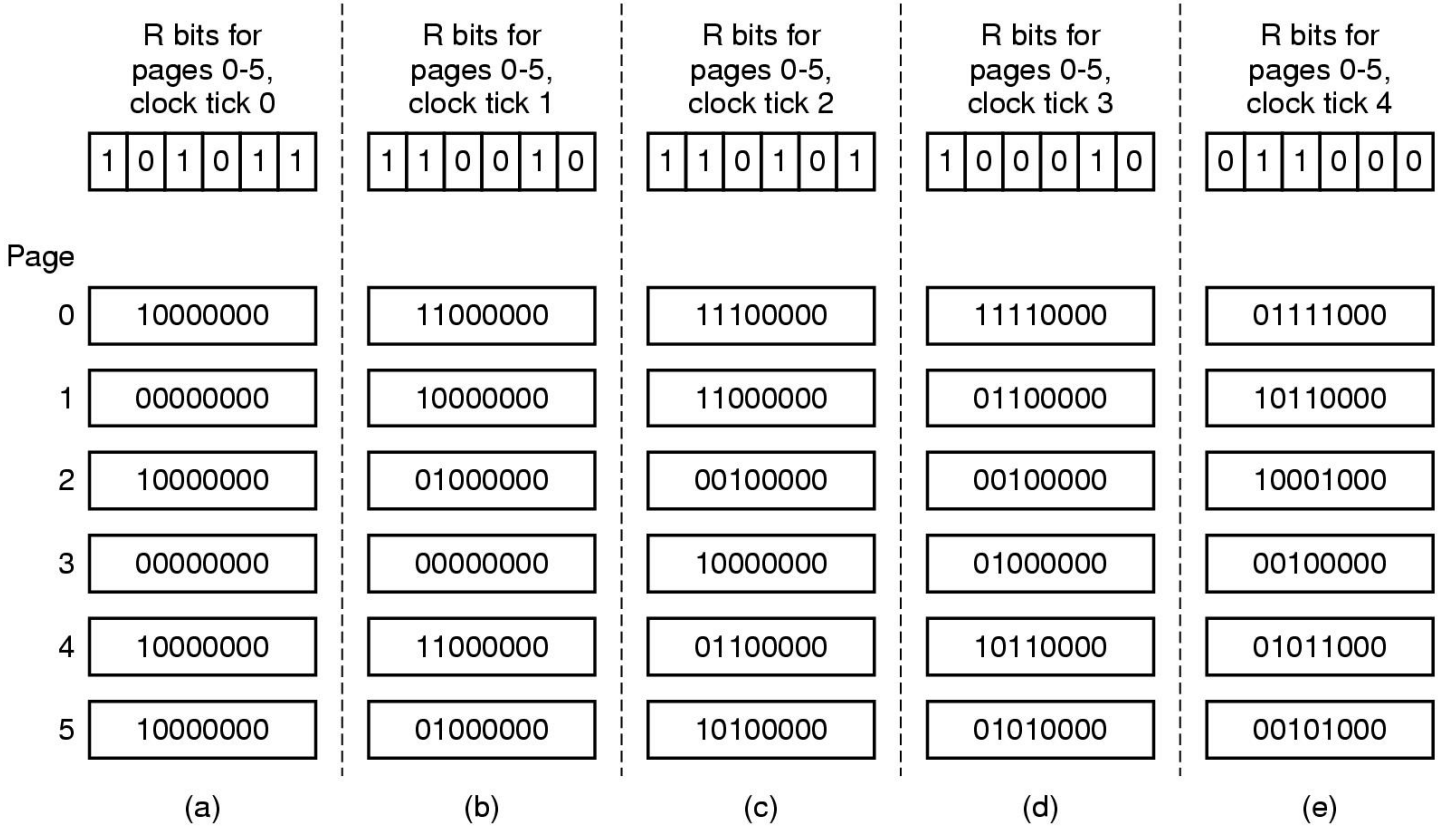
sun@hit.edu.cn

A Second Hardware LRU



LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

Simulating LRU in Software
NFU and Aging



- **The aging algorithm simulates LRU in software**
- **Note 6 pages for 5 clock ticks, (a) – (e)**

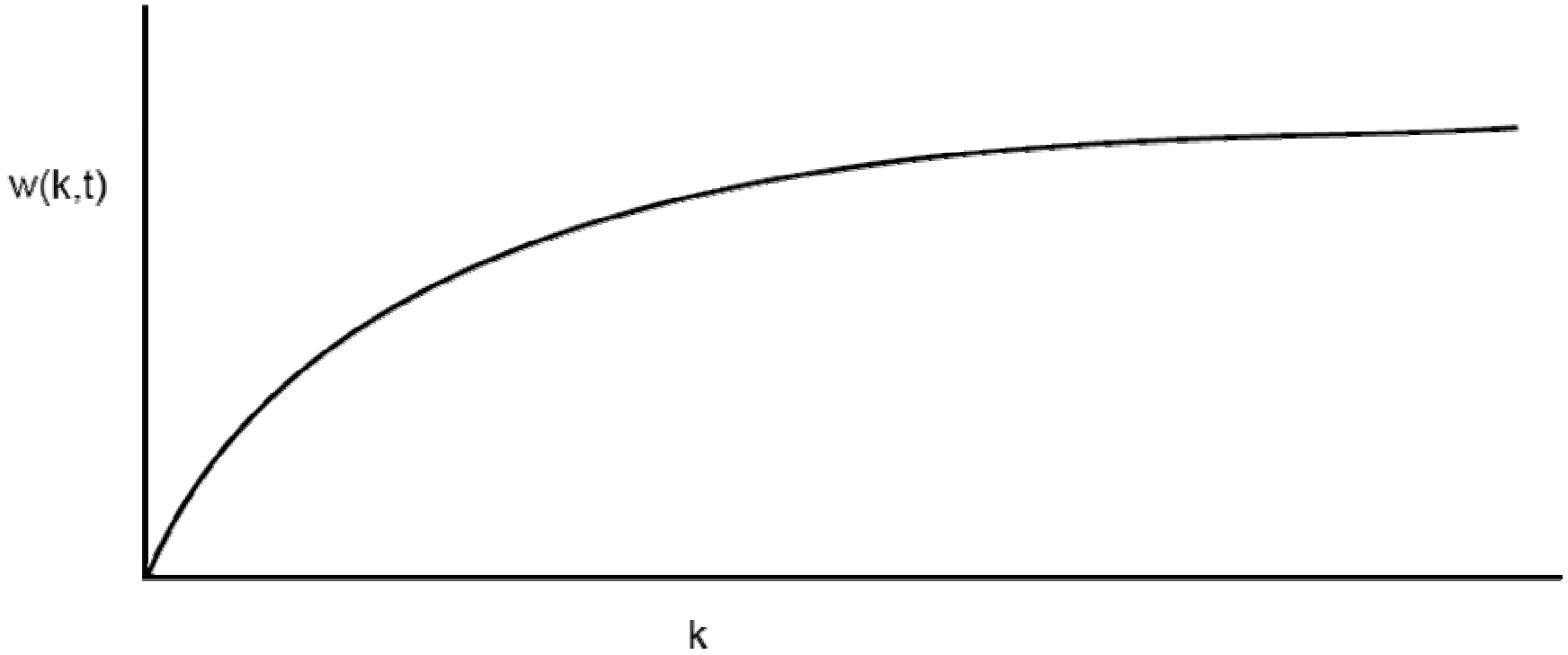
sun@hit.edu.cn

The Working Set Page Replacement Algorithm (1)

- **Demand paging**
  - **pages are loaded only on demand, not in advance**
- **Locality of reference**
  - **the process references only a relatively small fraction of its pages**
- **Working set**
  - **the set of pages that a process is currently using**

sun@hit.edu.cn

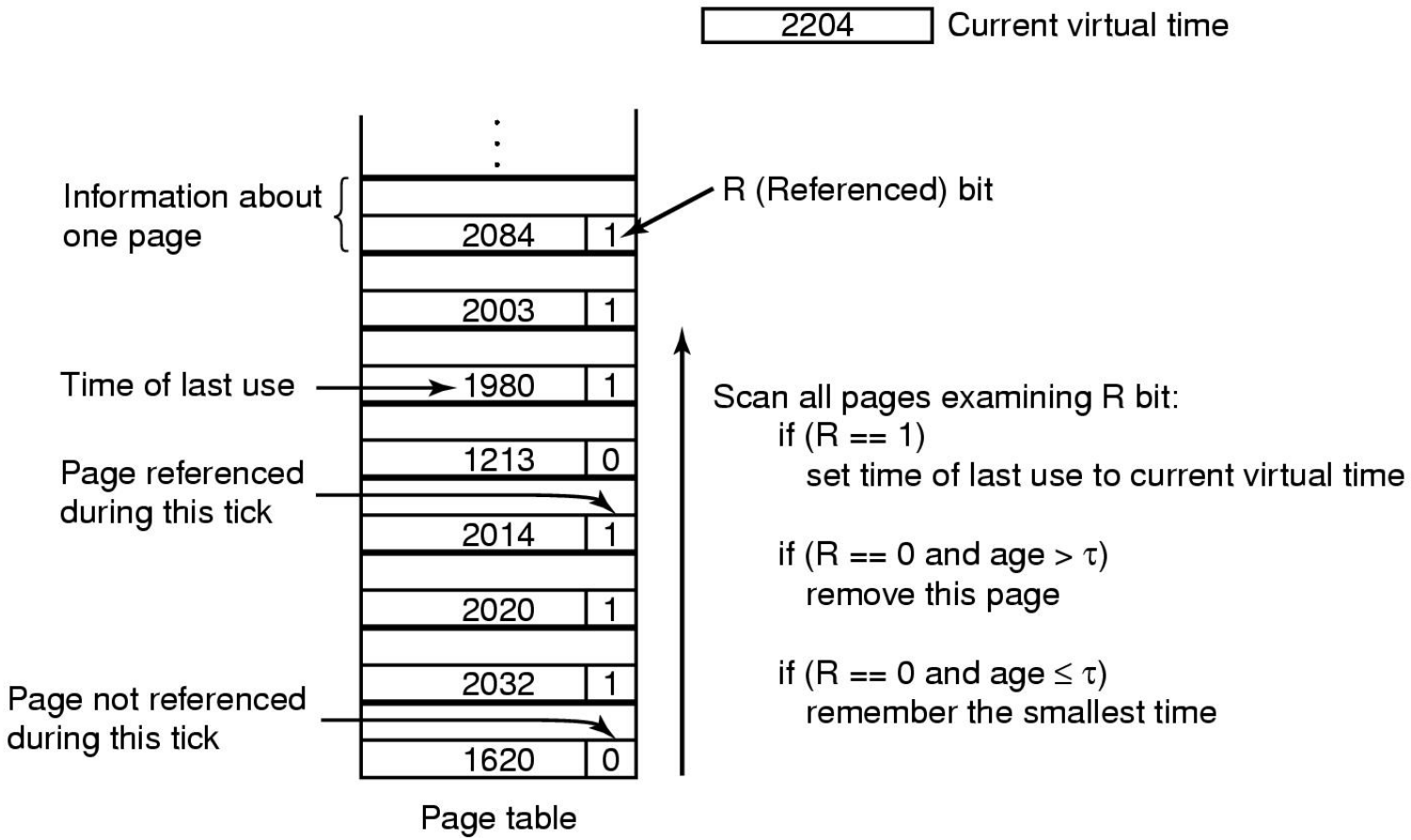
The Working Set Page Replacement Algorithm (2)



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

sun@hit.edu.cn

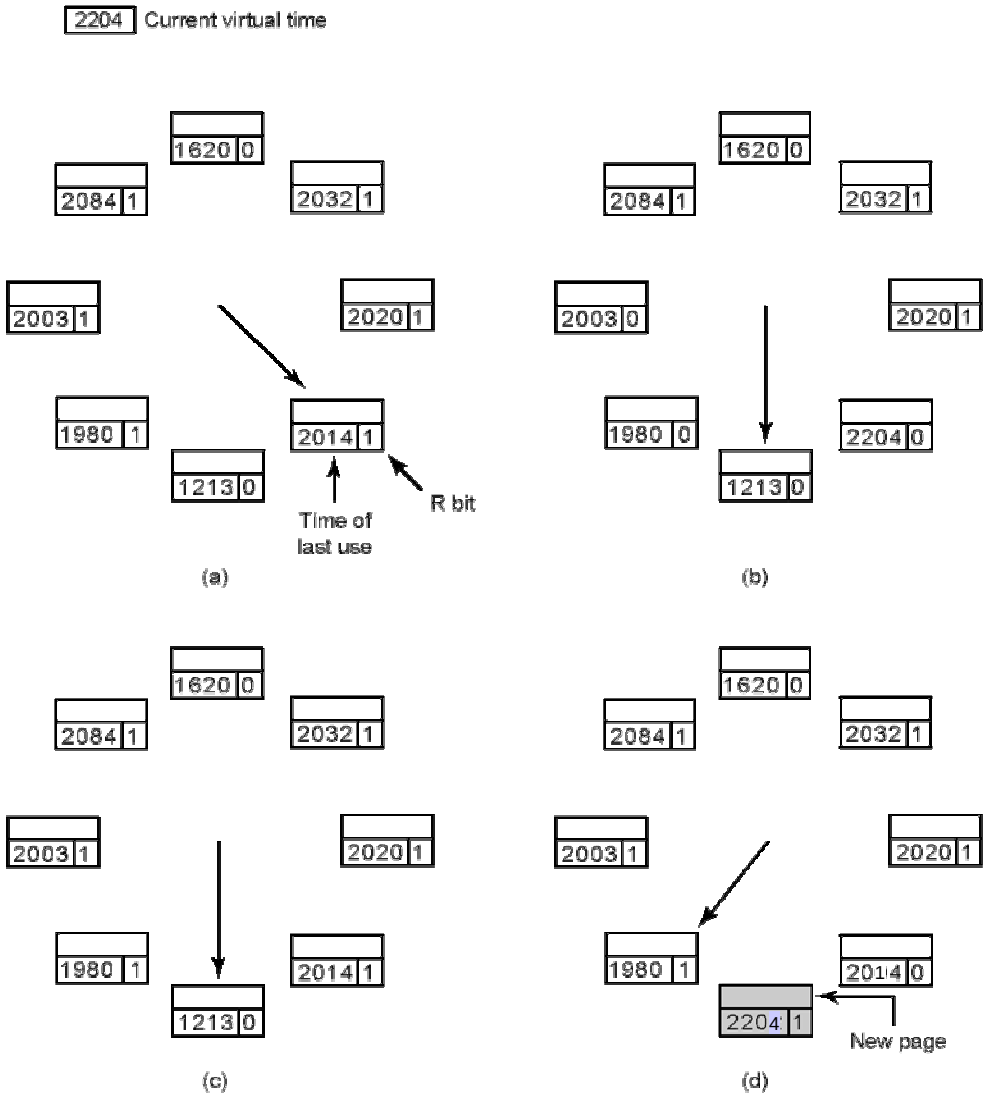
The Working Set Page Replacement Algorithm (3)



The working set algorithm

sun@hit.edu.cn

The WSClock Page Replacement Algorithm



sun@hit.edu.cn

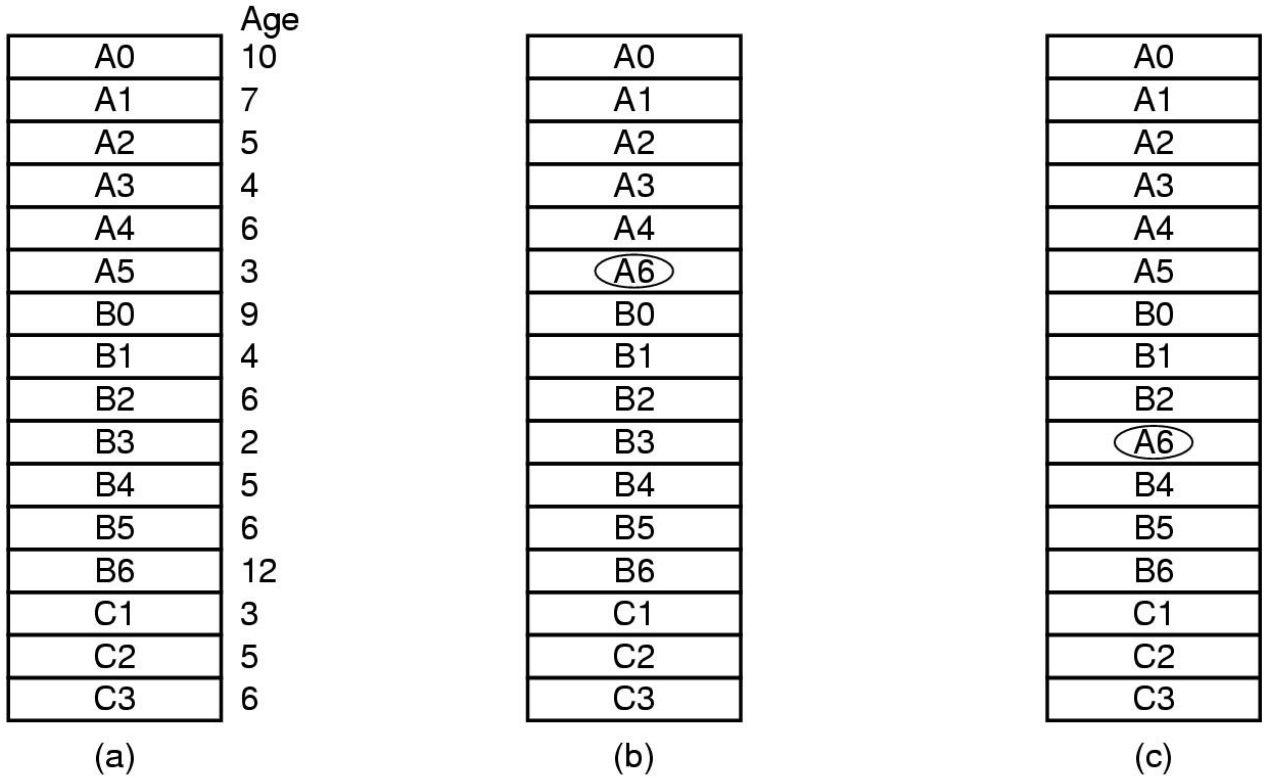
Review of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
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

sun@hit.edu.cn

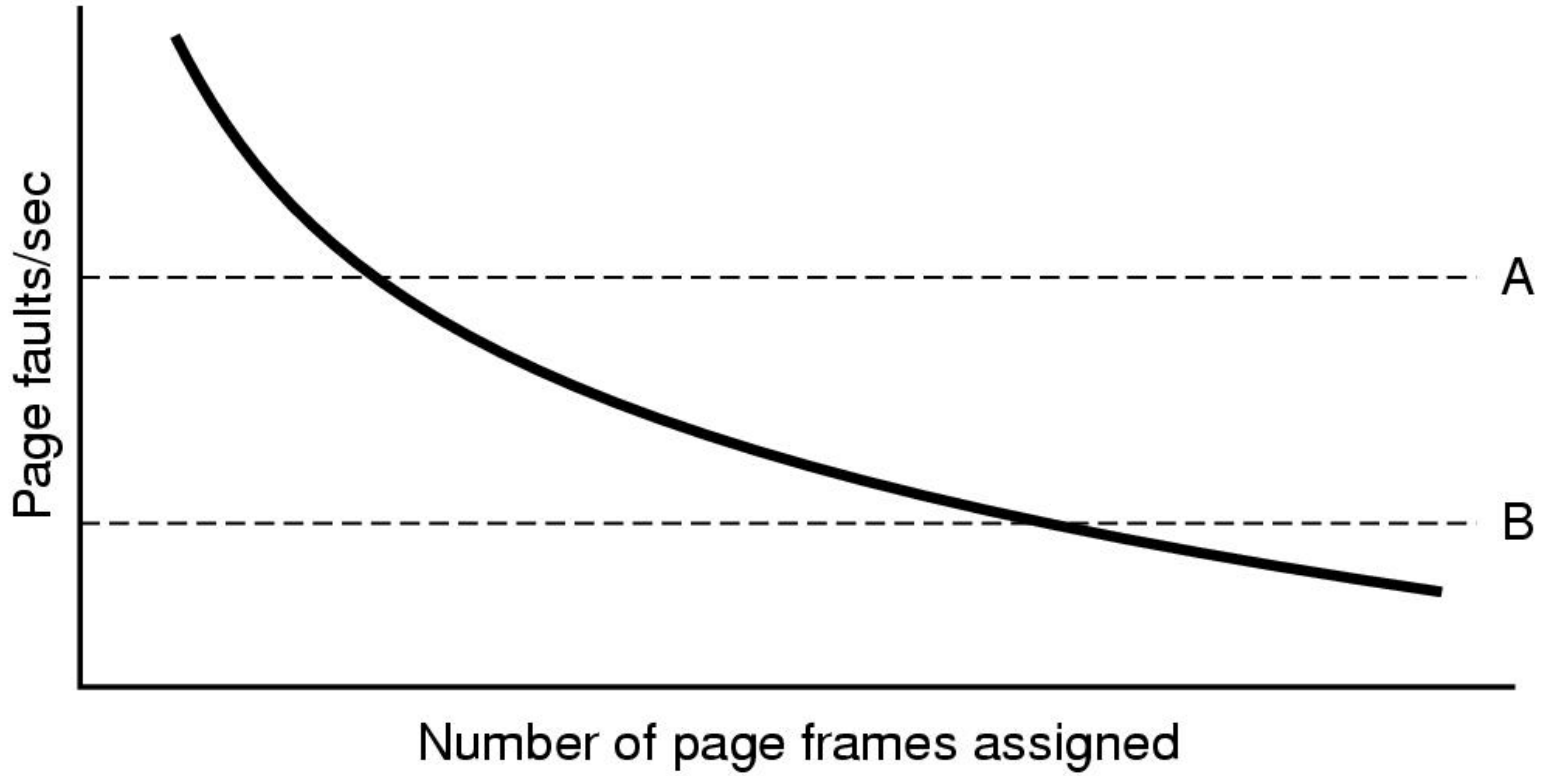
Design Issues for Paging Systems

Local versus Global Allocation Policies



a) Original configuration
b) Local page replacement
c) Global page replacement

Page Fault Frequency (PFF)



Page fault rate as a function of the number of page frames assigned

sun@hit.edu.cn

Load Control

- **Despite good designs, system may still thrash**
- **When PFF algorithm indicates**
  - **some processes need more memory**
  - **but no processes need less**
- **Solution :**
  Reduce number of processes competing for memory
  - **swap one or more to disk, divide up pages they held**
  - **reconsider degree of multiprogramming**

sun@hit.edu.cn

Page Size (1)

Small page size

- **Advantages**
  - **less internal fragmentation**
  - **better fit for various data structures, code sections**
  - **less unused program in memory**
- **Disadvantages**
  - **programs need many pages, larger page tables**
  - **waste swapping time**

sun@hit.edu.cn

Page Size (2)

- **Overhead due to page table and internal fragmentation**

- **Where**
  - **s = average process size in bytes**
  - **p = page size in bytes**
  - **e = page entry**

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

page table space

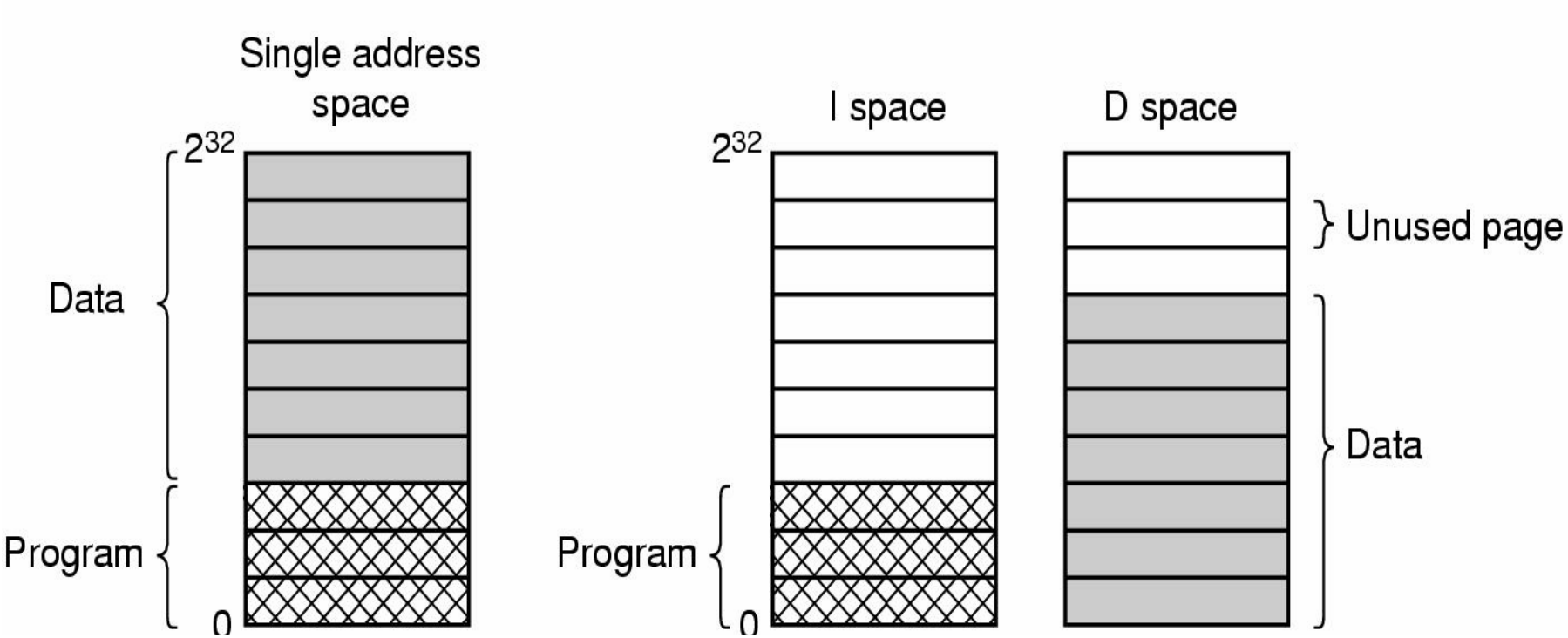
internal fragmentation

Optimized when

$$p = \sqrt{2se}$$

sun@hit.edu.cn

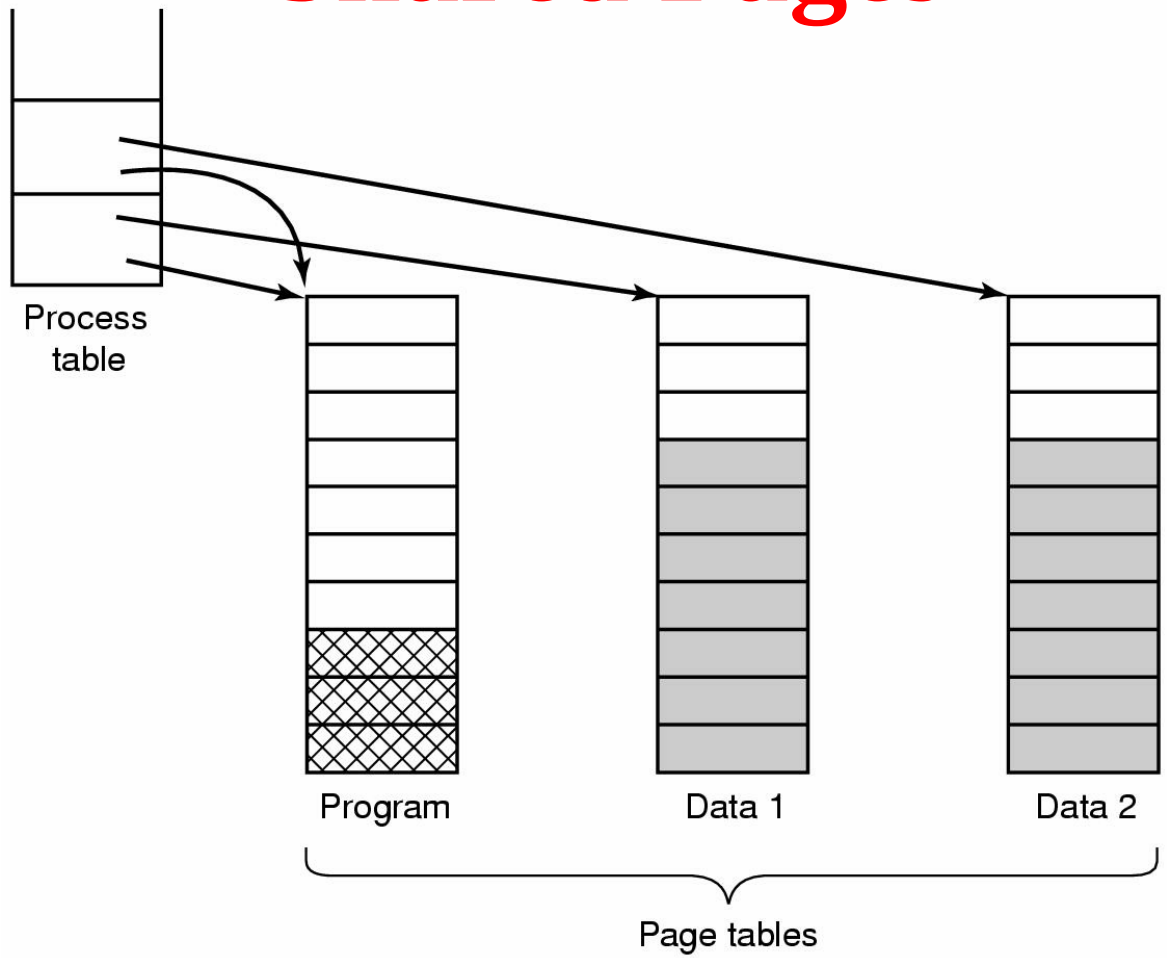
Separate Instruction and Data Spaces



- One address space
- Separate I and D spaces

sun@hit.edu.cn

Shared Pages



Two processes sharing the same program sharing its page table

sun@hit.edu.cn

Cleaning Policy

- Need for a background process, paging daemon
  - periodically inspects state of memory
- When too few frames are free
  - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
  - as regular page replacement algorithm but with diff ptr

Implementation Issues

Operating System Involvement with Paging

Four times when OS involved with paging

- **Process creation**
  - determine program size
  - create page table
  - initialize swap area
- **Process execution**
  - MMU reset for new process
  - TLB flushed
- **Page fault time**
  - determine virtual address causing fault
  - swap target page out, needed page in
- **Process termination time**
  - release page table, pages, disk space
  - shared pages can only be released by the last process using them

sun@hit.edu.cn

Page Fault Handling

- Hardware traps to kernel
- General registers saved
- OS determines which virtual page needed
- OS checks validity of address, seeks page frame
- If selected frame is dirty, write it to disk
- OS brings schedules new page in from disk
- Page tables updated
- Faulting instruction backed up to when it began
- Faulting process scheduled
- Registers restored
- Program continues

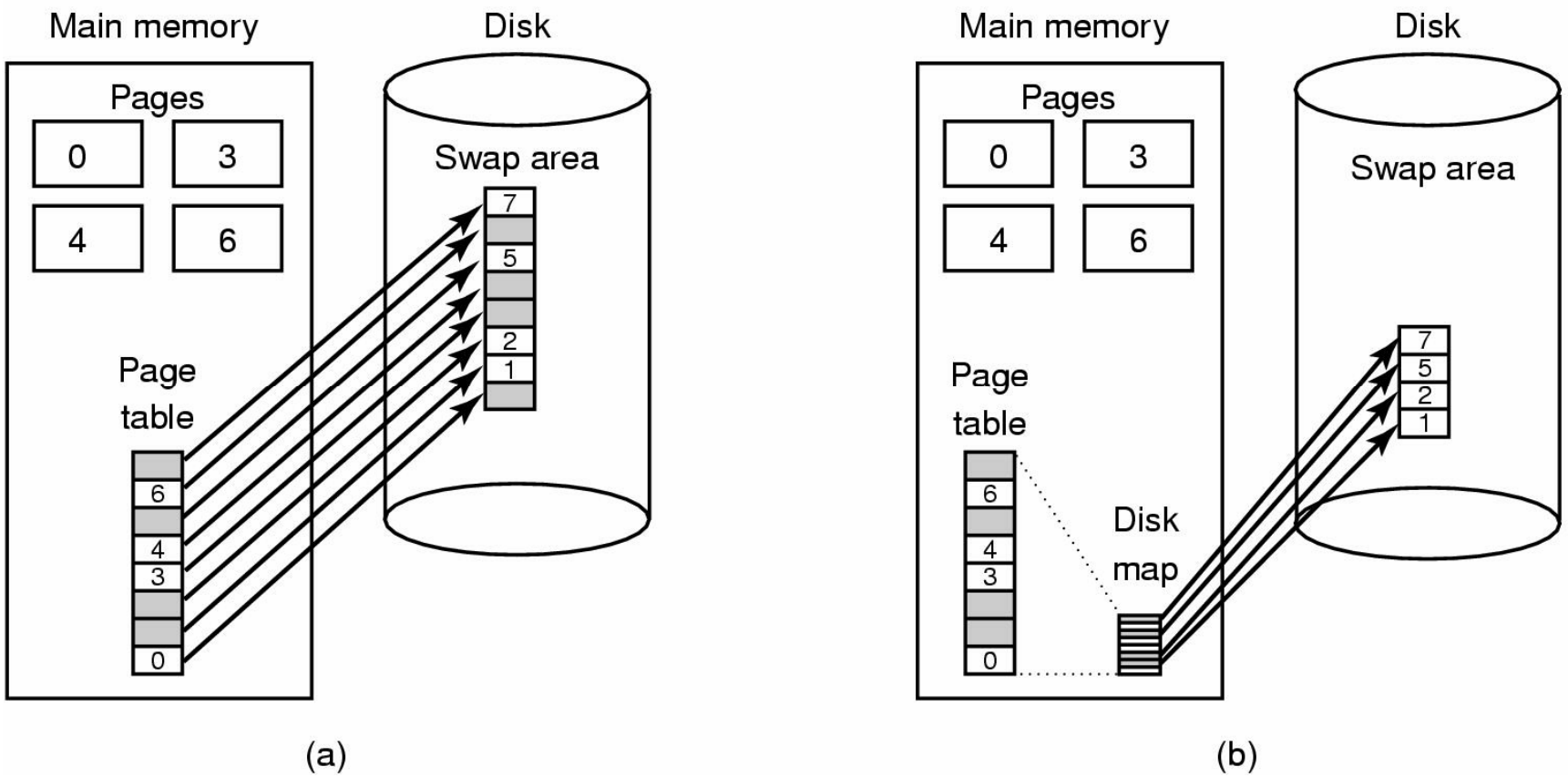
sun@hit.edu.cn

Locking Pages in Memory

- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first proc may be chosen to be paged out
- Solutions
  - need to specify some pages locked
  - do all I/O to kernel buffer then copy data to pages

sun@hit.edu.cn

Backing Store

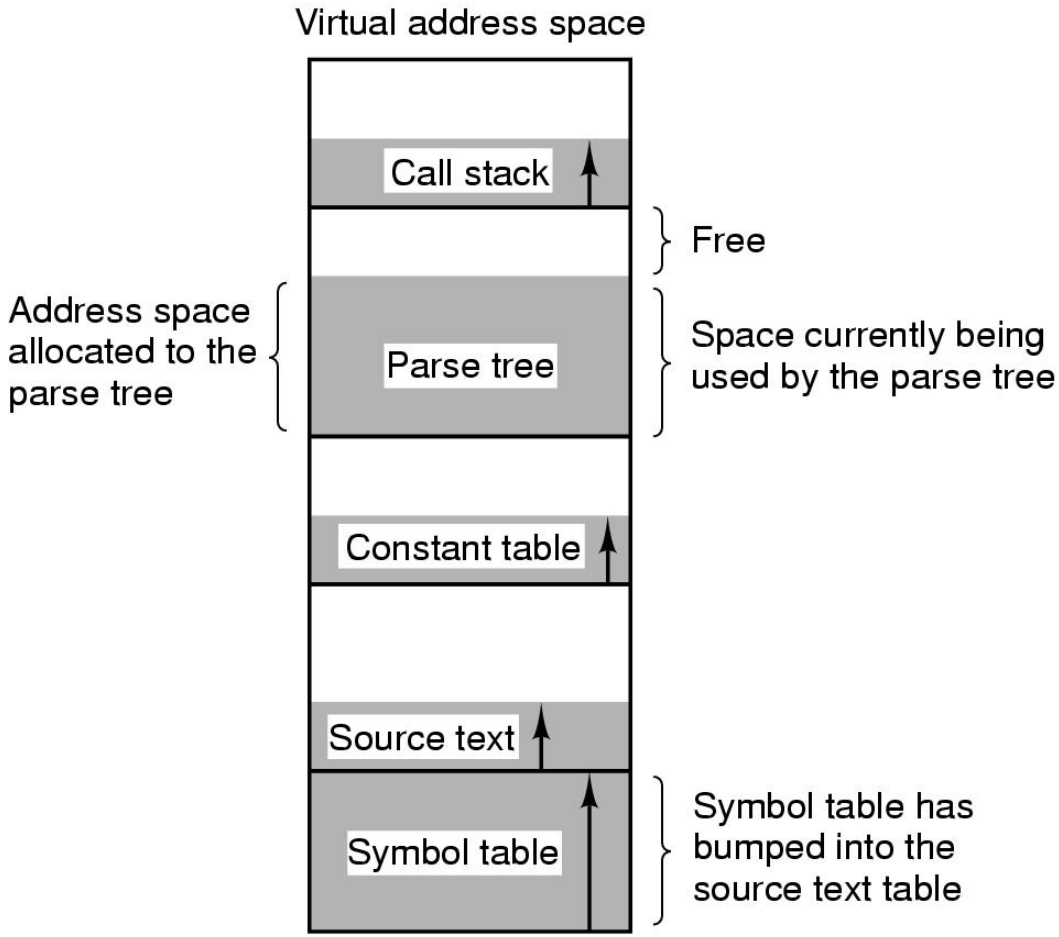


(a) Paging to static swap area

(b) Backing up pages dynamically

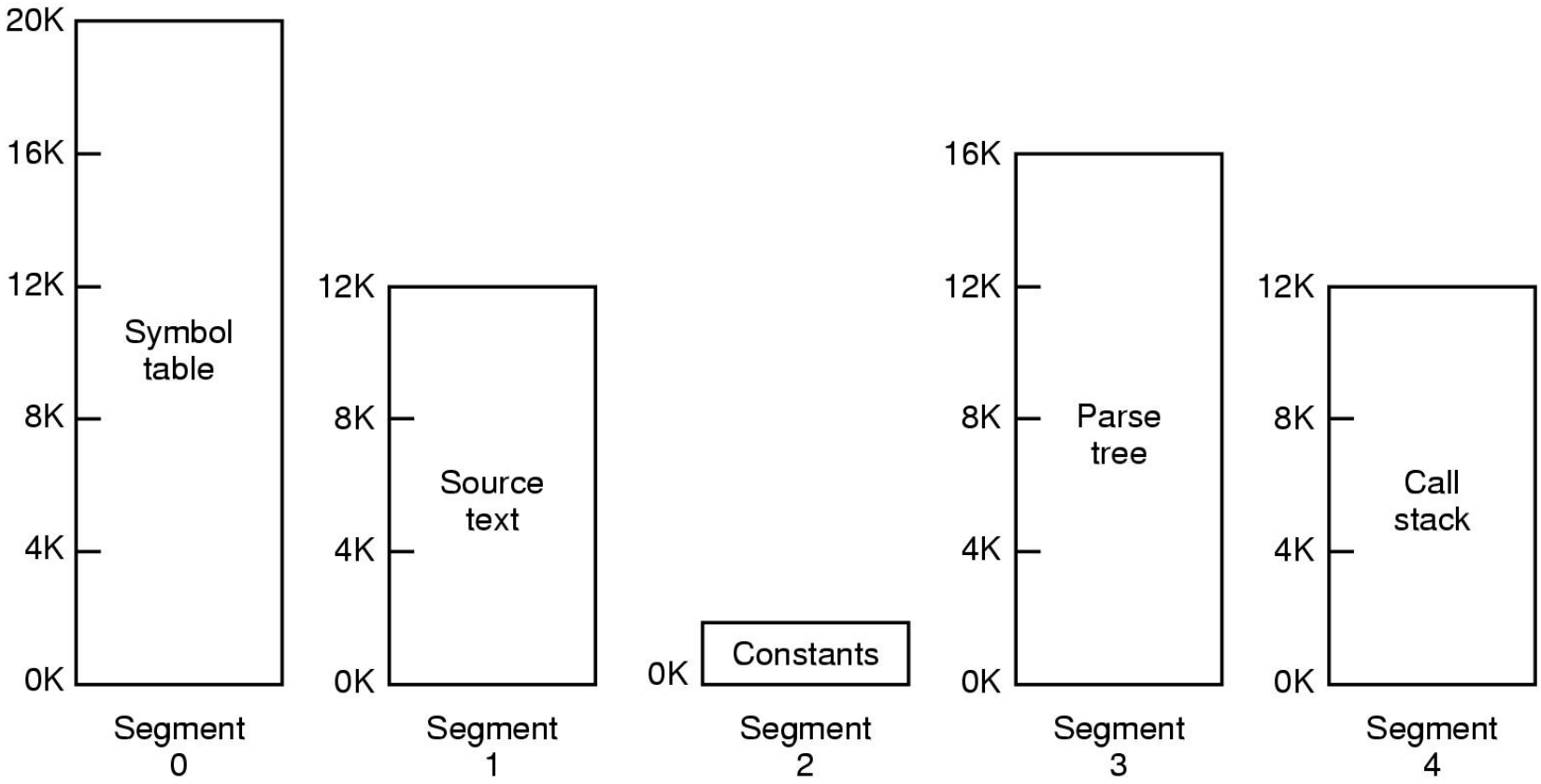
sun@hit.edu.cn

Segmentation (1)



- One-dimensional address space with growing tables
- One table may bump into another

Segmentation (2)



Allows each table to grow or shrink, independently

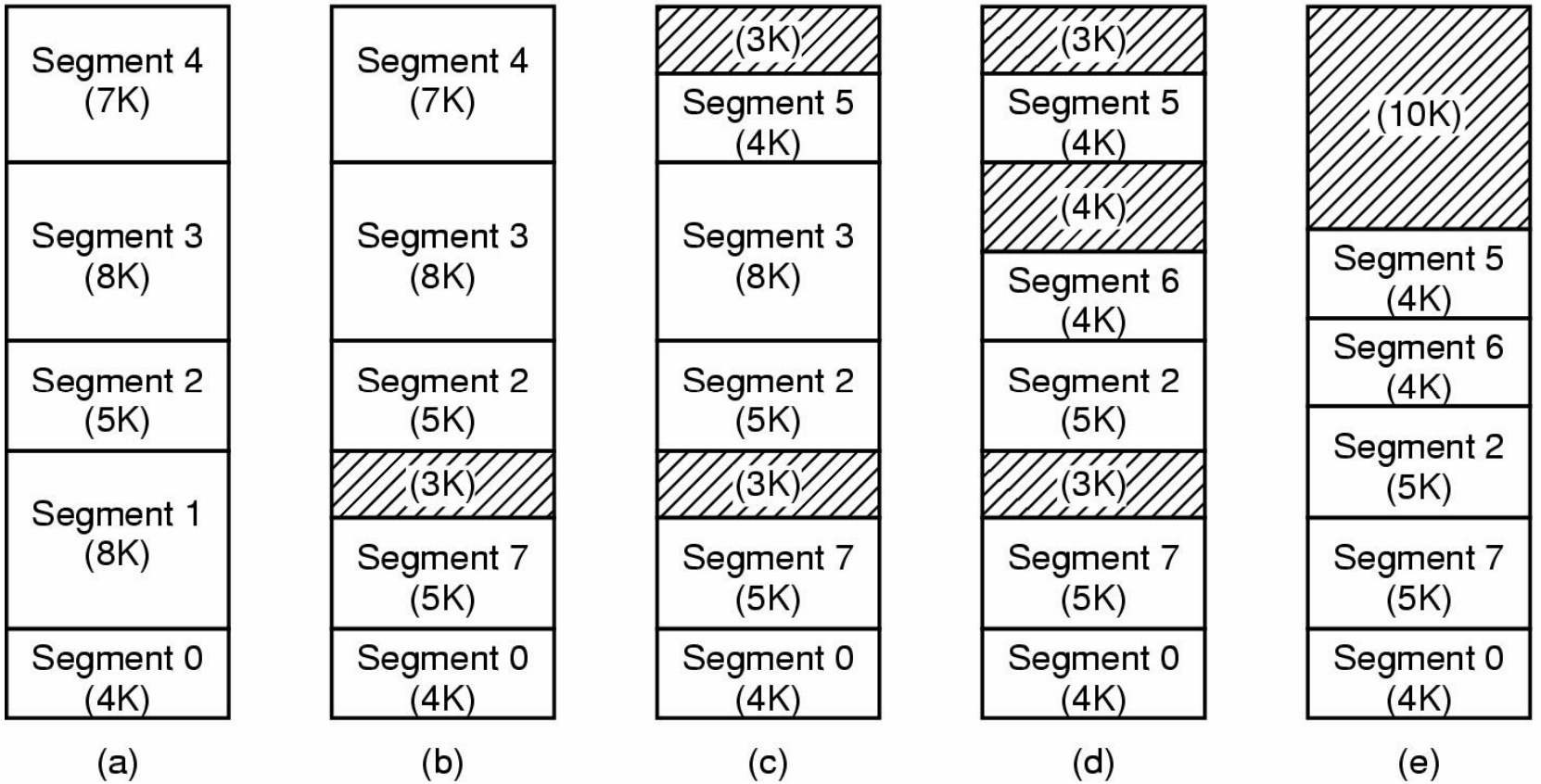
sun@hit.edu.cn

Segmentation (3)

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Comparison of paging and segmentation

Implementation of Pure Segmentation

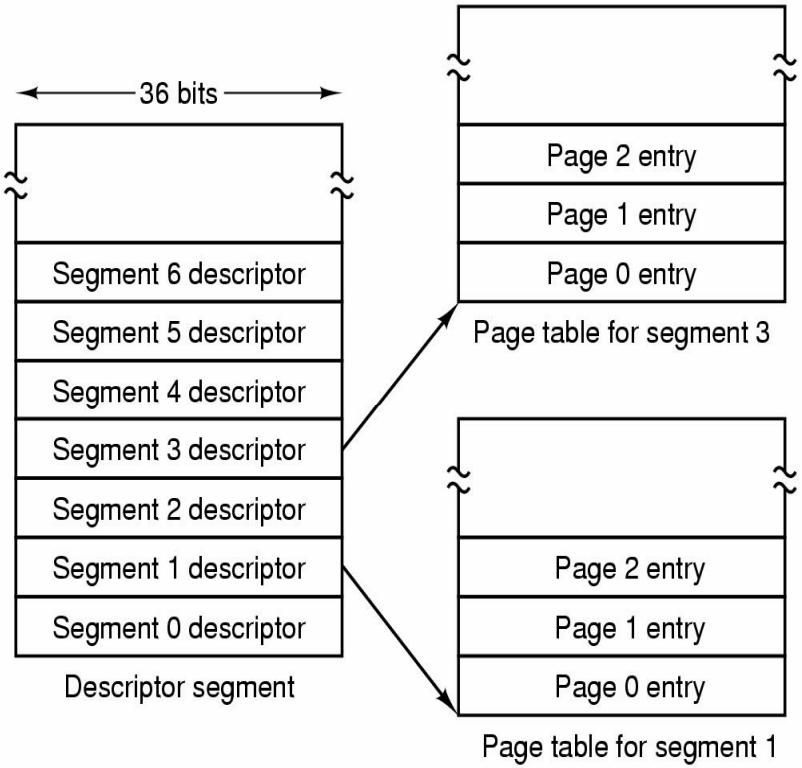


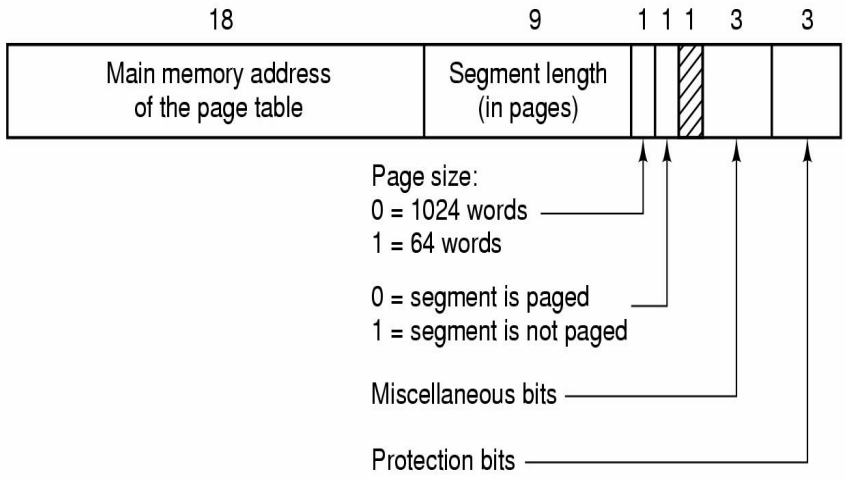
(a)-(d) Development of checkerboarding

(e) Removal of the checkerboarding by compaction

sun@hit.edu.cn

Segmentation with Paging: MULTICS (1)

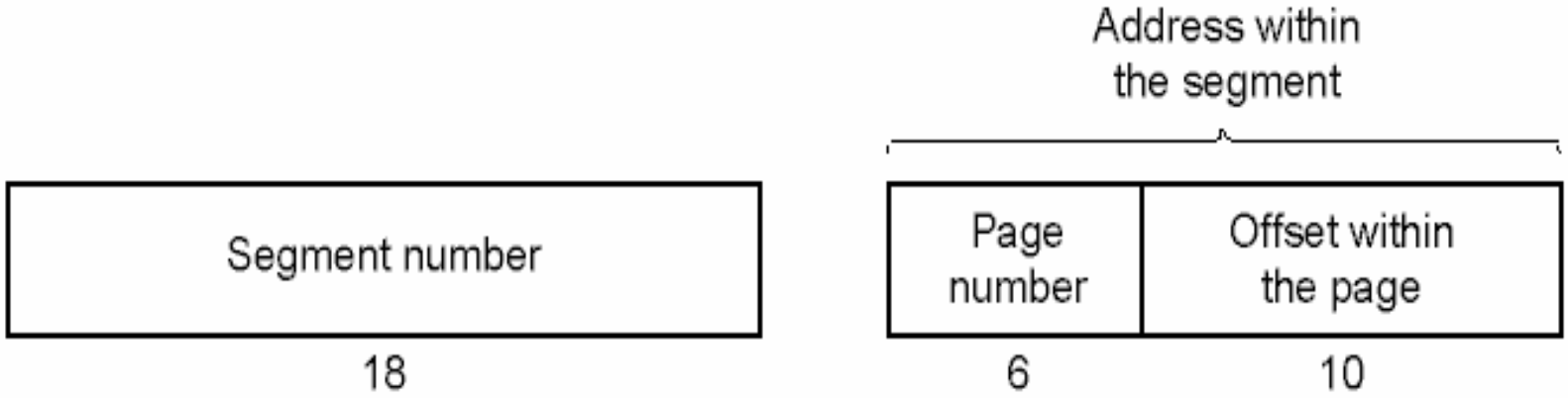




- **Descriptor segment points to page tables**
- **Segment descriptor – numbers are field lengths**

sun@hit.edu.cn

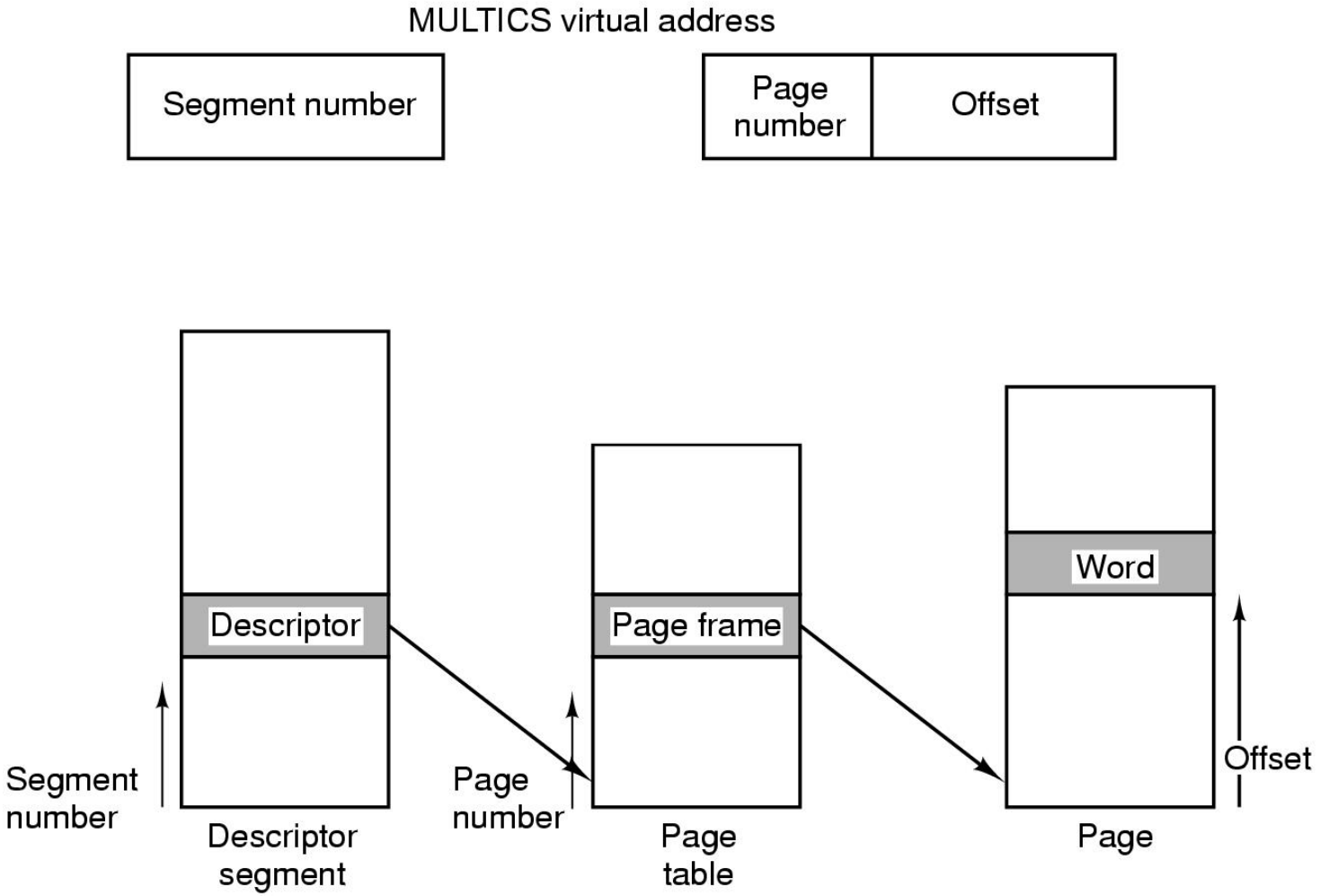
Segmentation with Paging: MULTICS (2)



A 34-bit MULTICS virtual address

sun@hit.edu.cn

Segmentation with Paging: MULTICS (3)



Conversion of a 2-part MULTICS address into a main memory address

Segmentation with Paging: MULTICS (4)

Comparison field: Segment number	Comparison field: Virtual page	Page frame	Protection	Age	Is this entry used?
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

- **Simplified version of the MULTICS TLB**
- **Existence of 2 page sizes makes actual TLB more complicated**

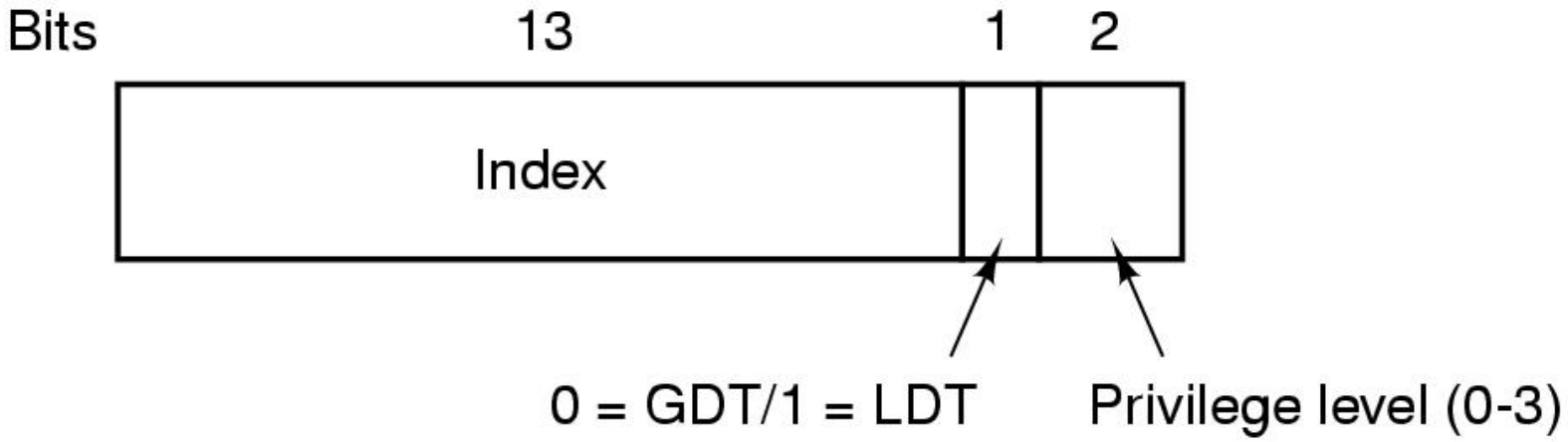
Segmentation with Paging: Pentium

- The Pentium has
  - 16K independent segments, each up to 1G 32-bit words
  - two tables:
    - LDT (Local Descriptor Table)
    - GDT (Global Descriptor Table)

- Load selectors for the segments into the machine's six segment registers

sun@hit.edu.cn

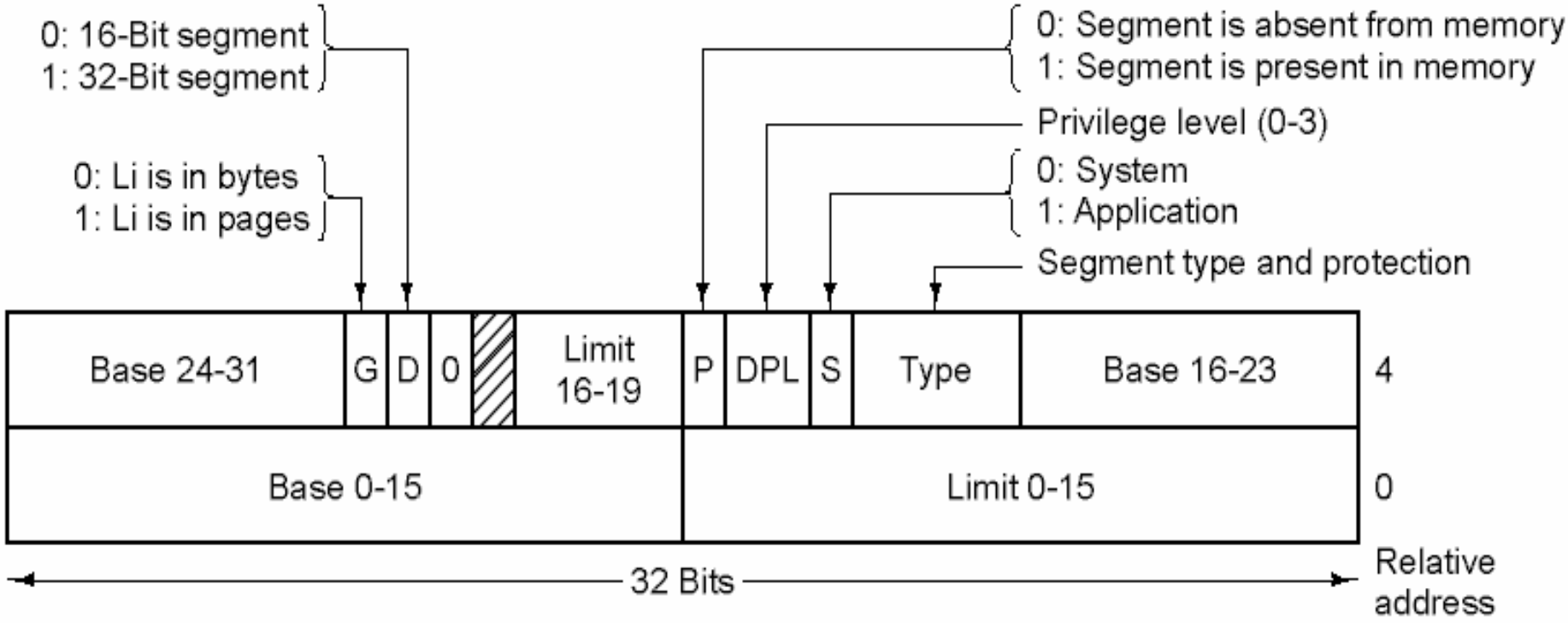
A Pentium selector



- **The value of the selector can *not* be zero**

sun@hit.edu.cn

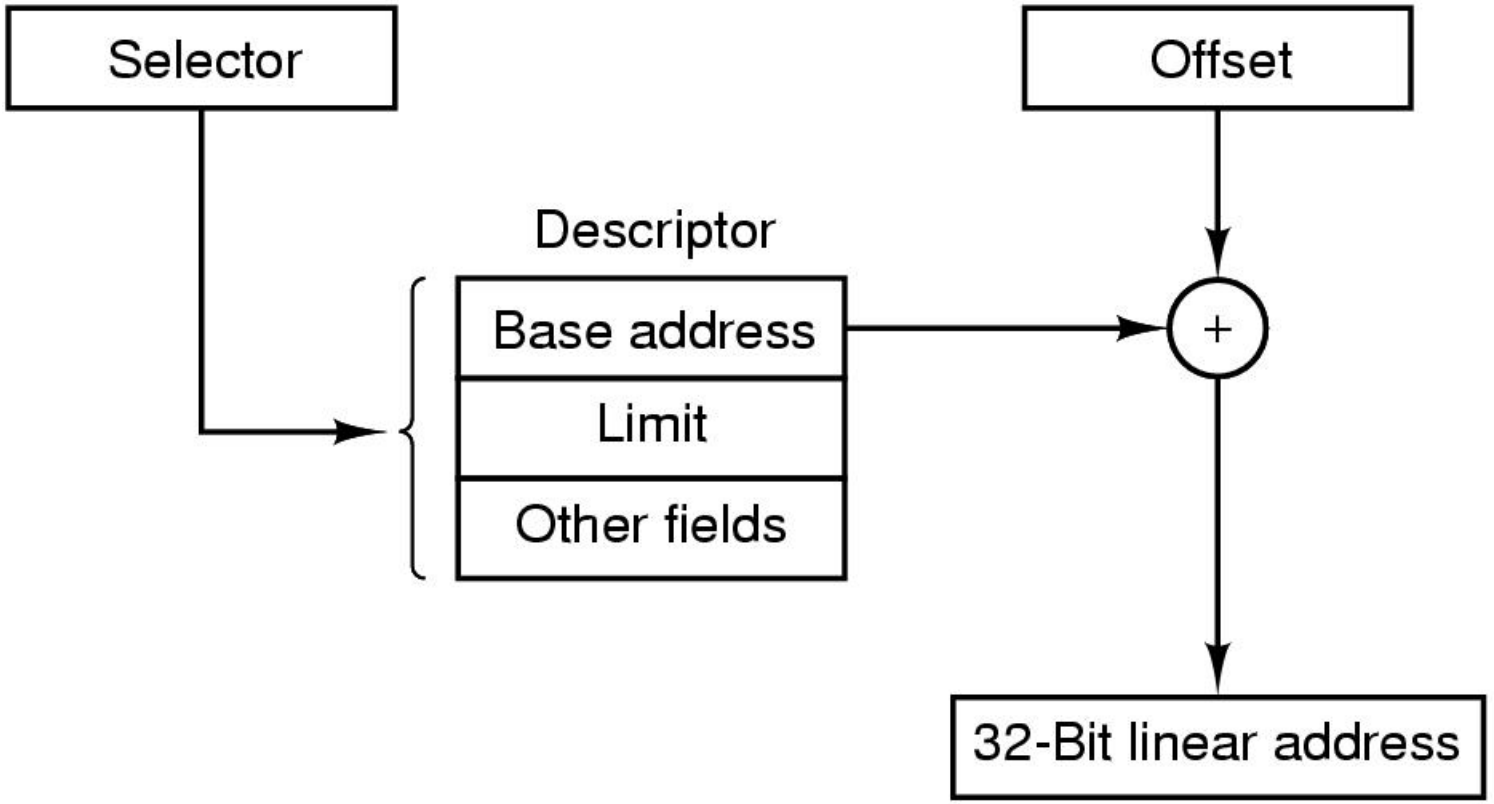
Pentium code segment descriptor



- **Data segments descriptor differ slightly**

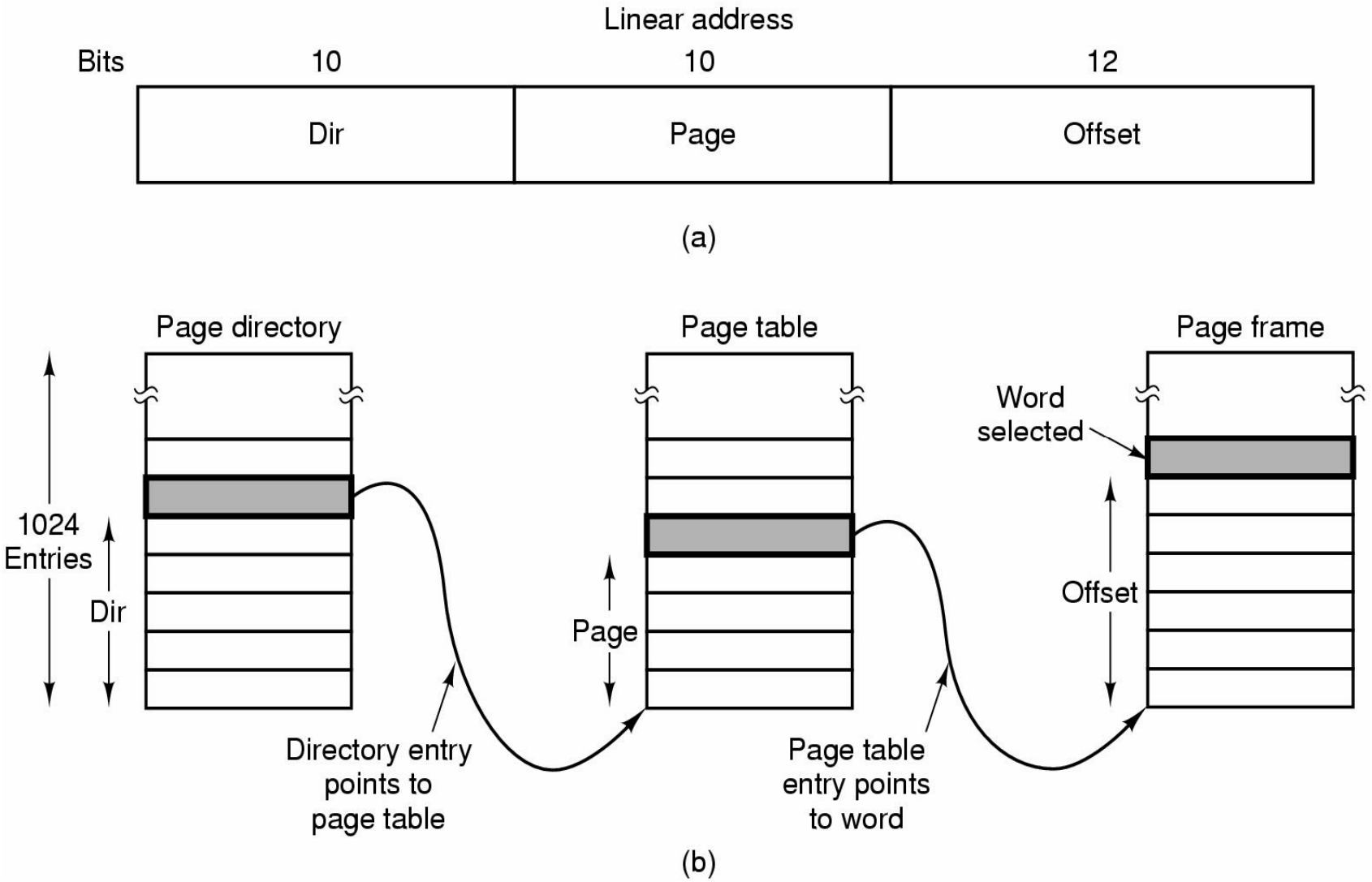
sun@hit.edu.cn

Conversion of a (selector, offset) pair to a linear address



sun@hit.edu.cn

Mapping of a linear address onto a physical address

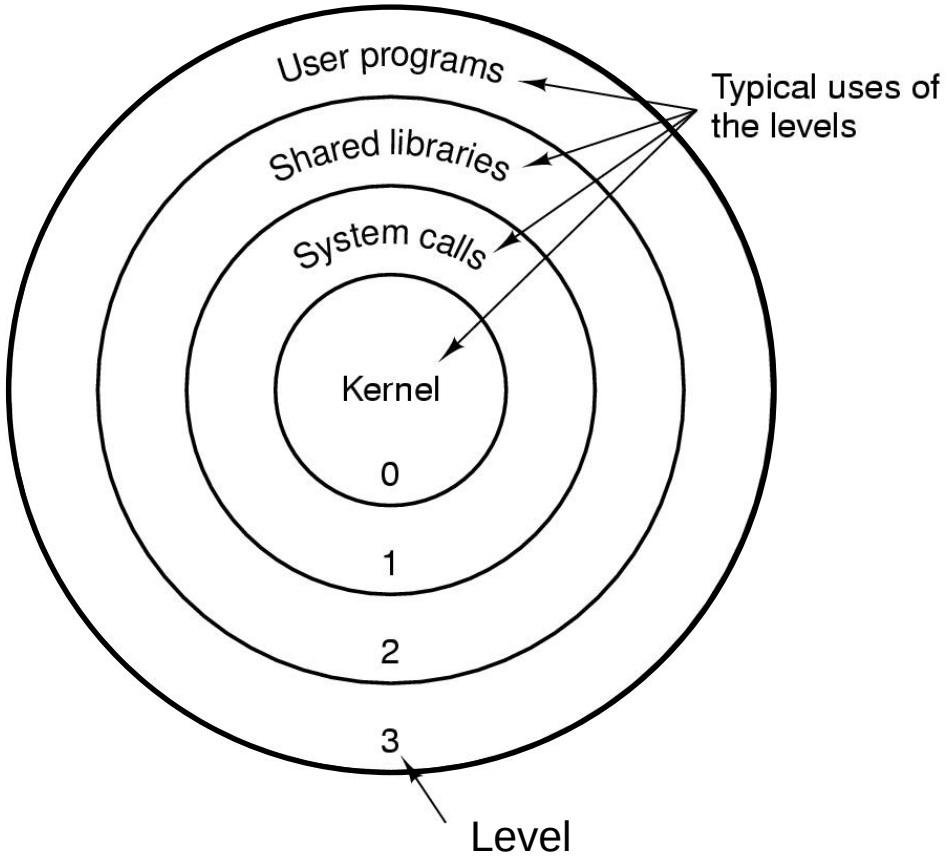


If you only need paging

- **Set up all the segment registers with the same selector**
- **The selector's descriptor has *Base=0* and *Limit* set to the maximum**
- **The instruction offset will then be the linear address**
- **All current operating systems for the Pentium work this way besides OS/2**
- **How about need segmentation only?**

sun@hit.edu.cn

Protection on the Pentium



sun@hit.edu.cn