



Chapter 3

MEMORY MANAGEMENT

- Basic memory management
- Swapping
- Virtual memory
- Page replacement algorithms
- Modeling page replacement algorithms
- Design issues for paging systems
- Implementation issues
- Segmentation

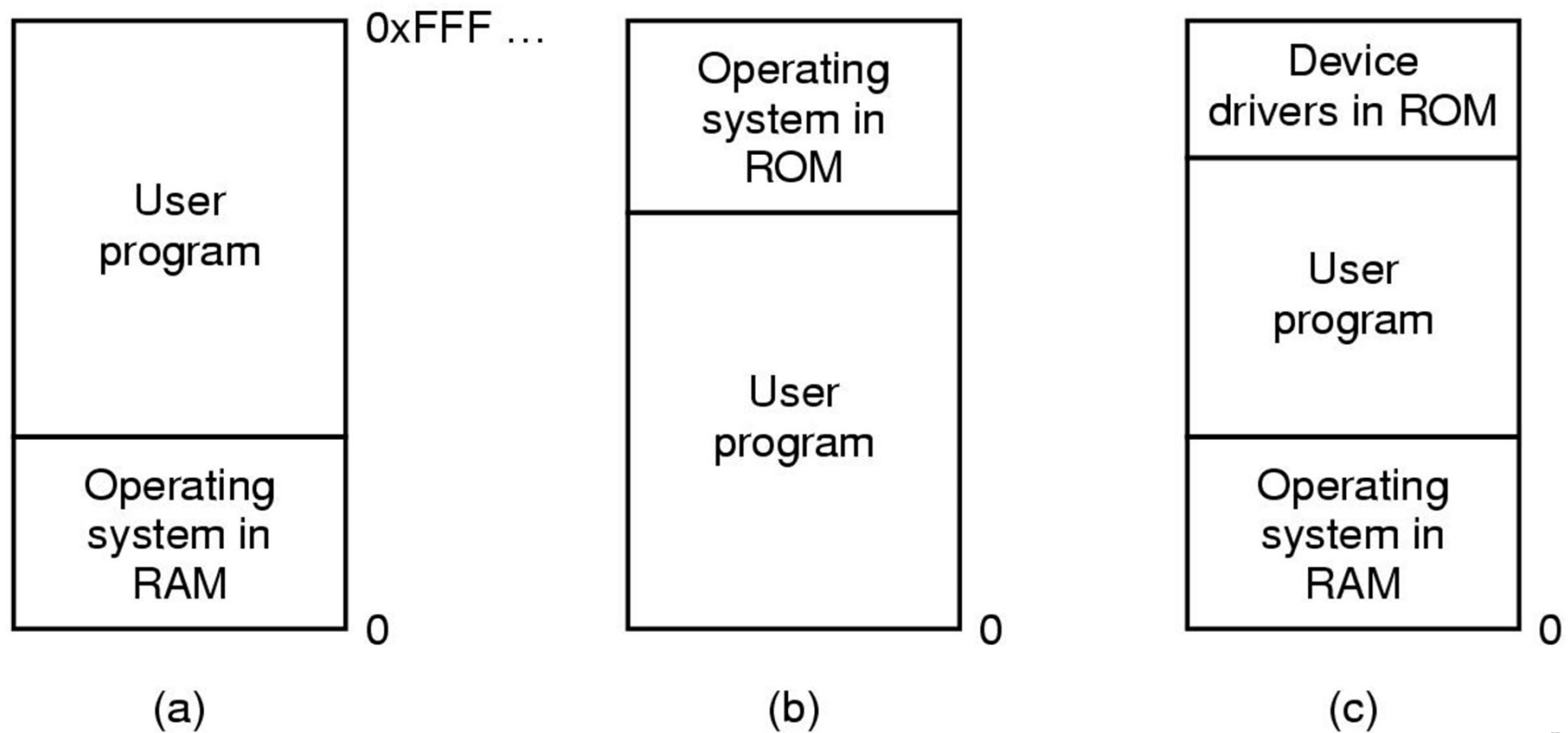


Memory Management

- 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

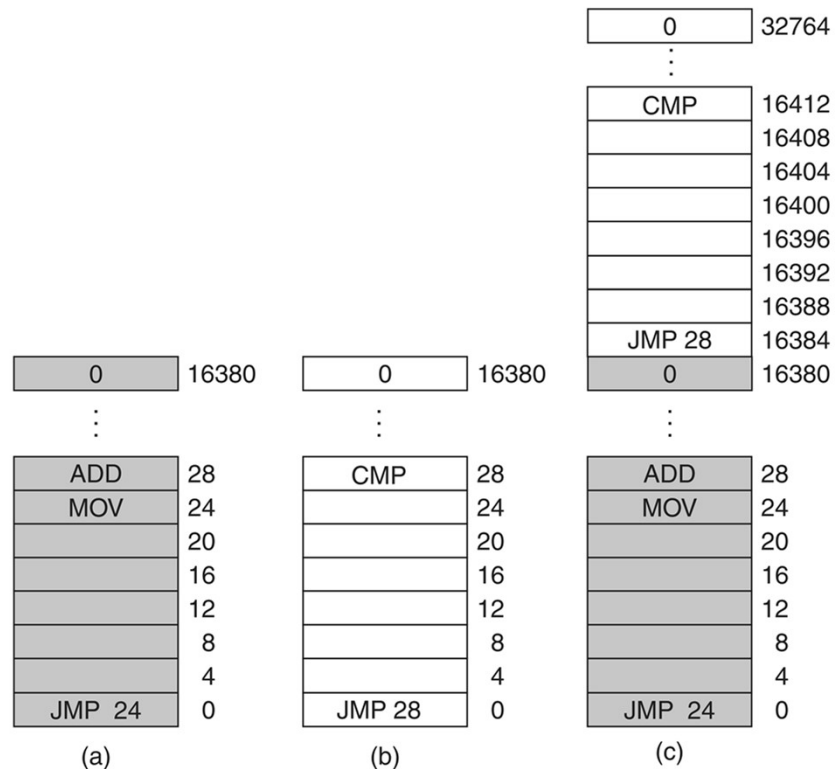
NO MEMORY ABSTRACTION

- Only one process at a time can be running



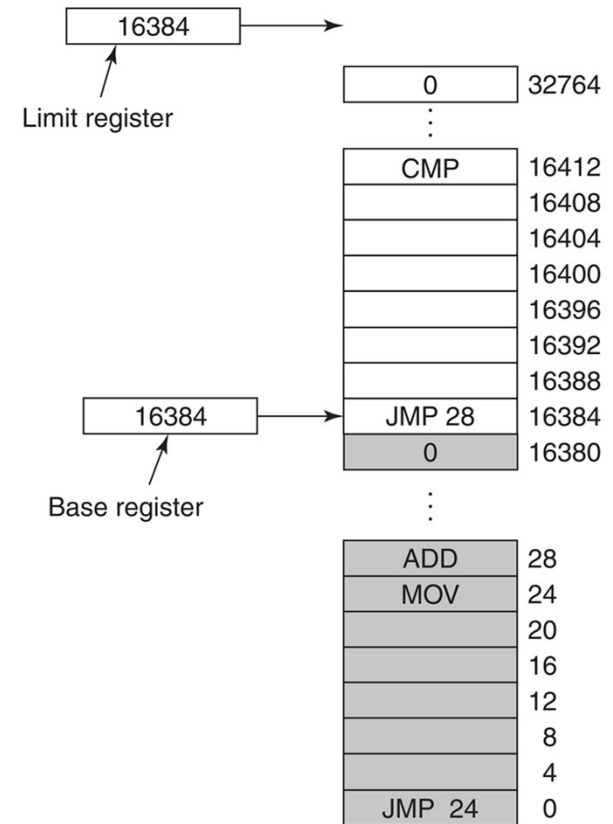
Running Multiple Programs Without a Memory Abstraction

- IBM 360
 - Special hardware for protection
 - 4 bits for each memory block
 - static relocation



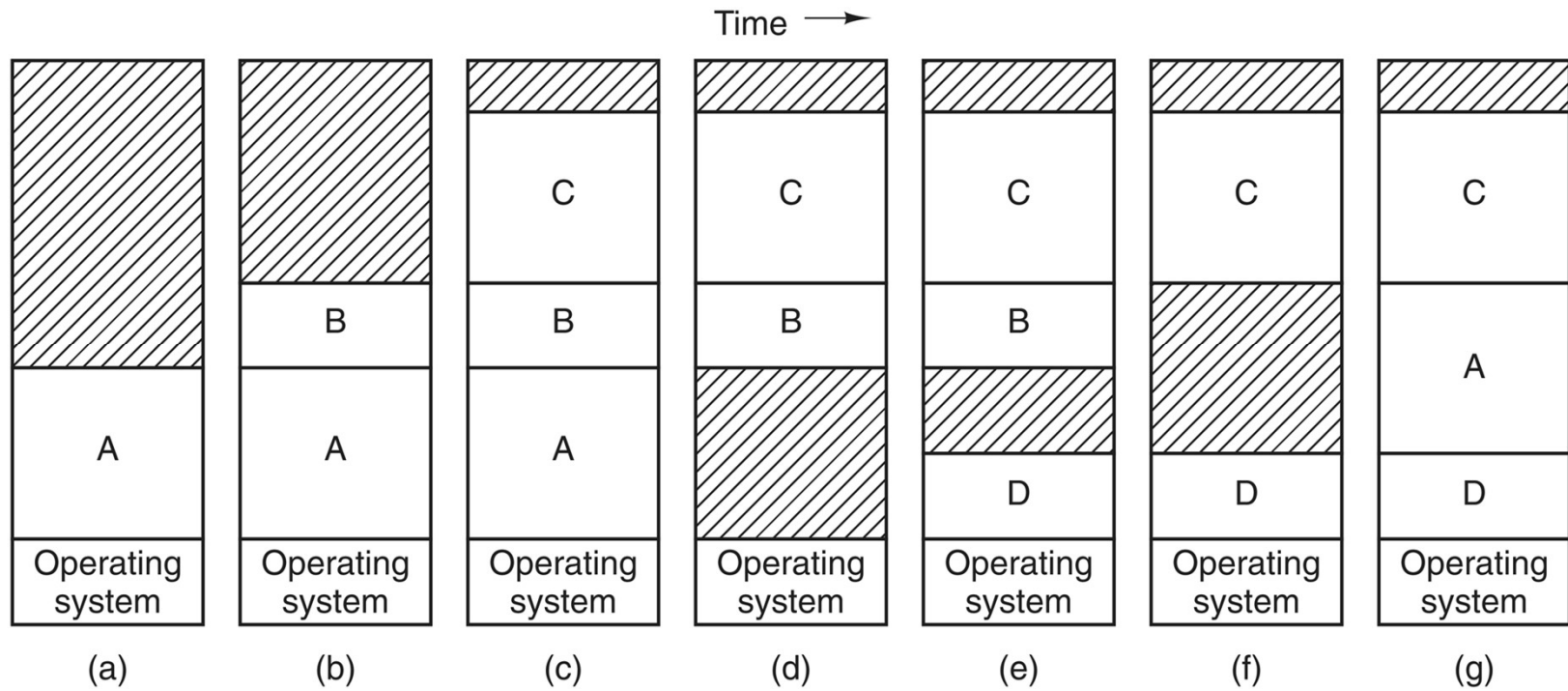
MEMORY ABSTRACTION: ADDRESS SPACES

- Two problems should be solved
 - protection
 - relocation
- Base and limit registers

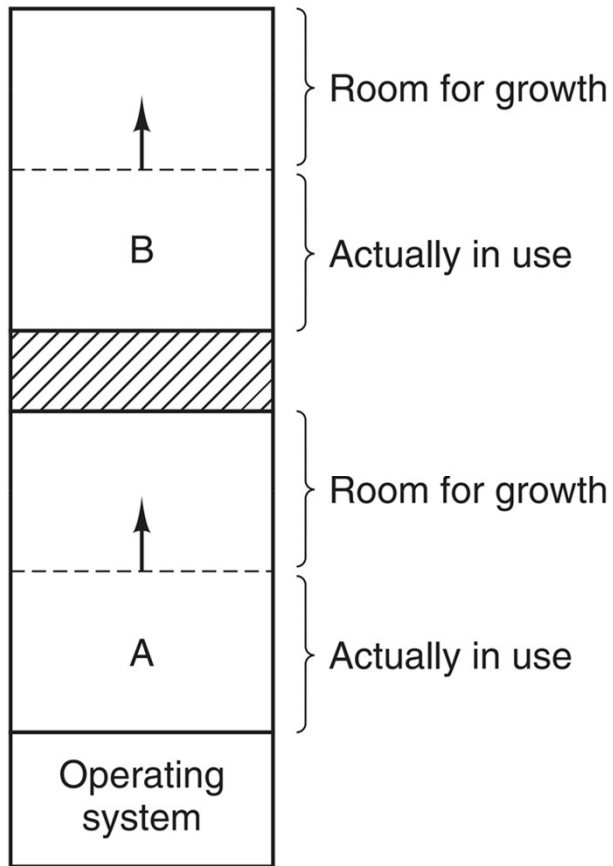


(c)

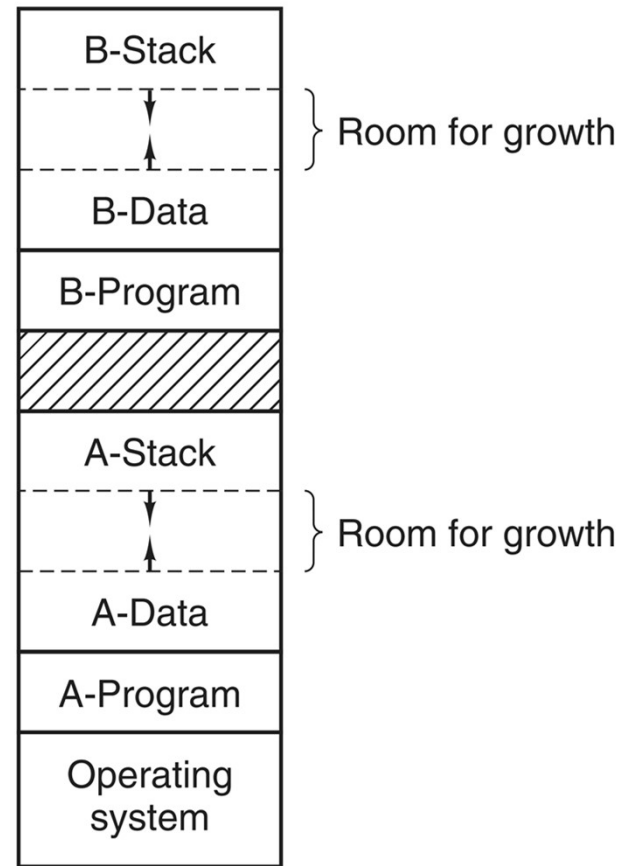
Swapping



Swapping



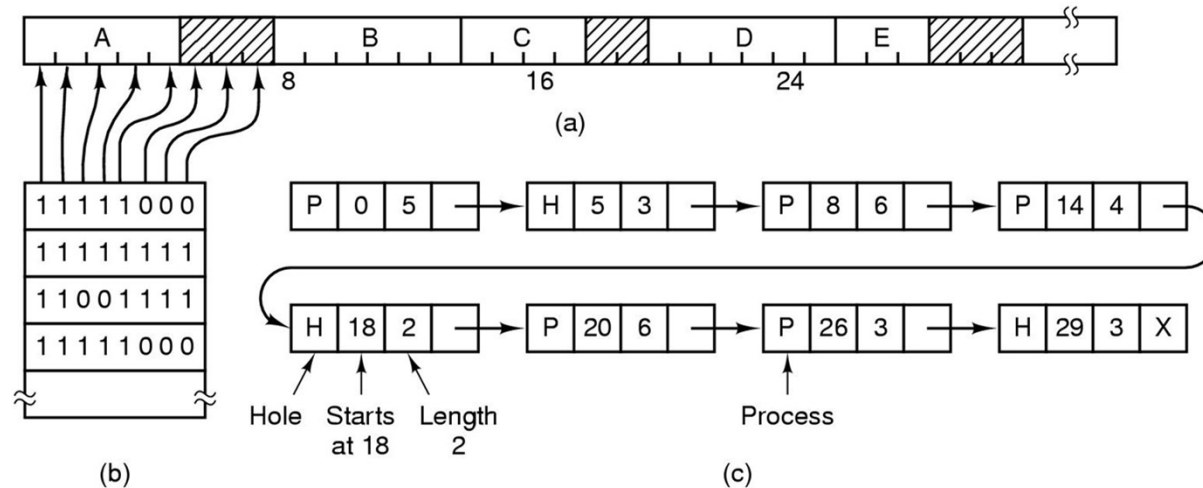
(a)



(b)

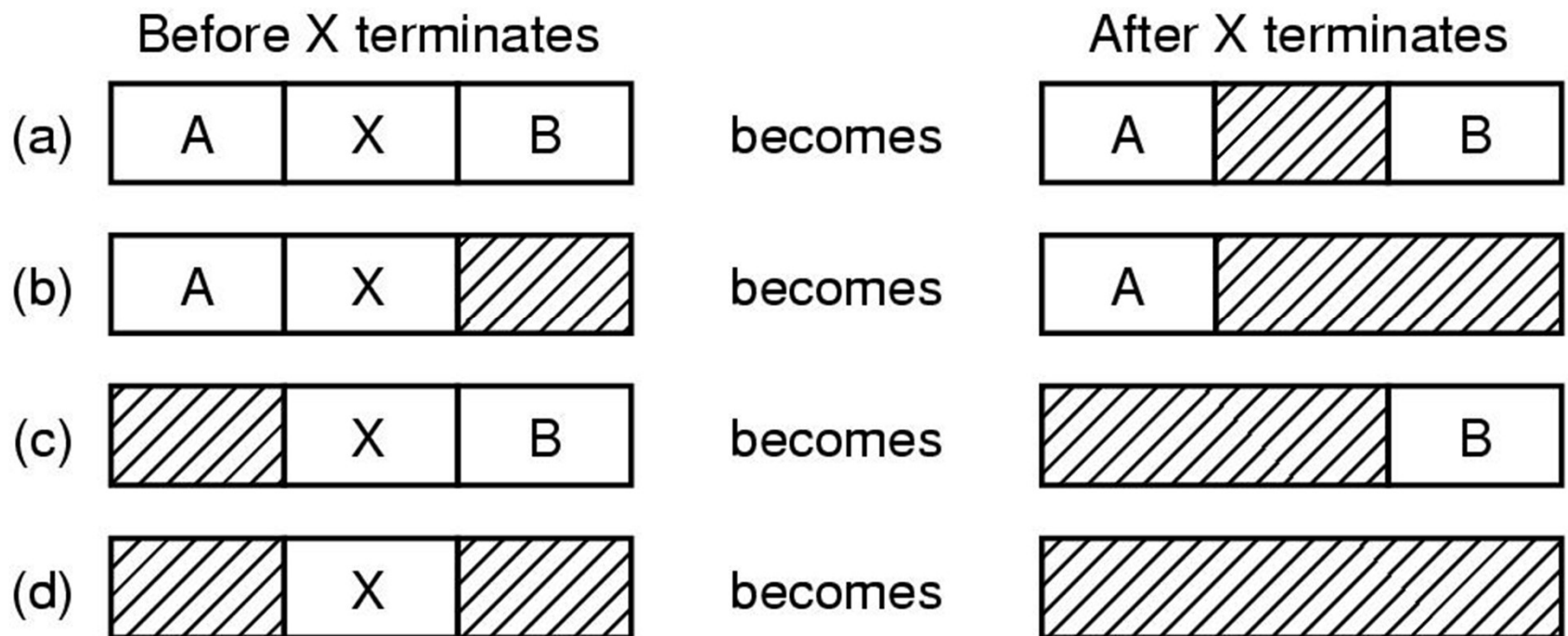
Memory Management with Bit Maps

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



Memory Management with Linked Lists

- Four neighbor combinations for the terminating process X



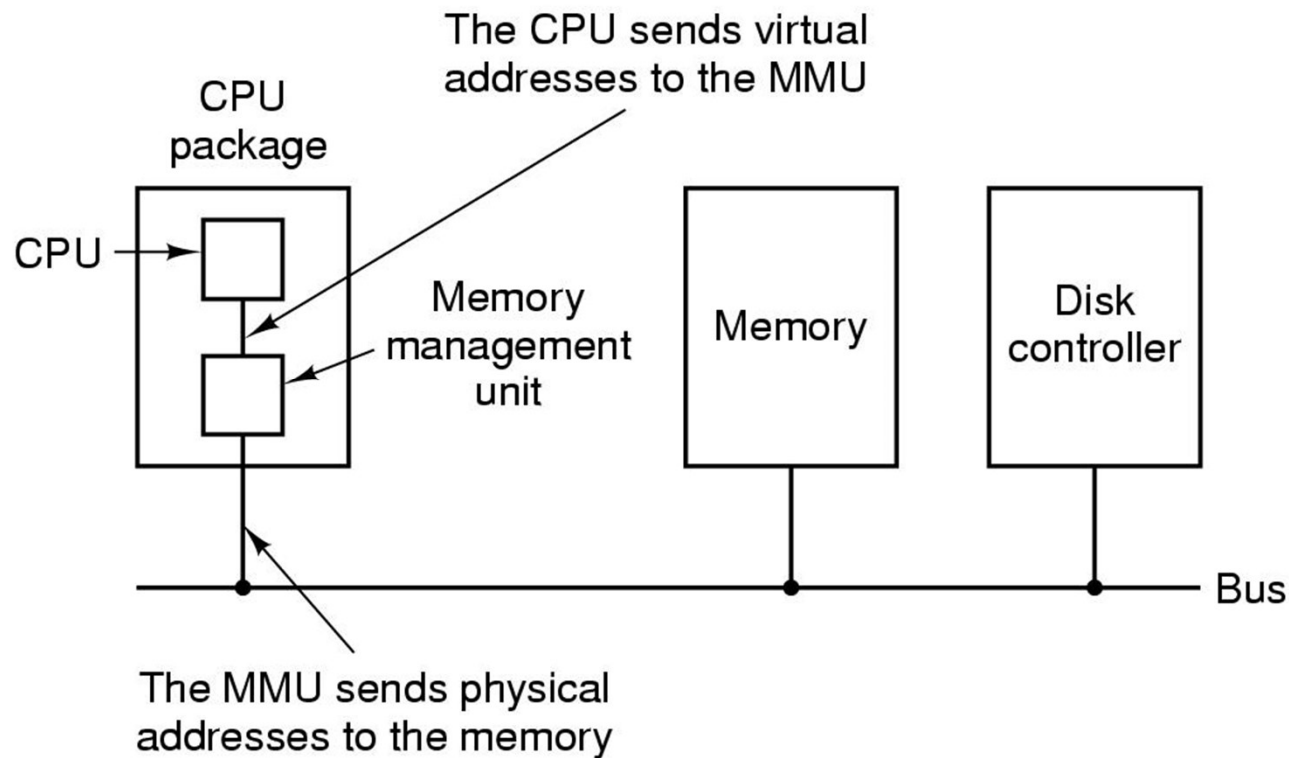


Memory Management with Linked Lists

- Algorithms for allocating memory for new processes
 - First fit
 - Next fit
 - Best fit
 - Worst fit

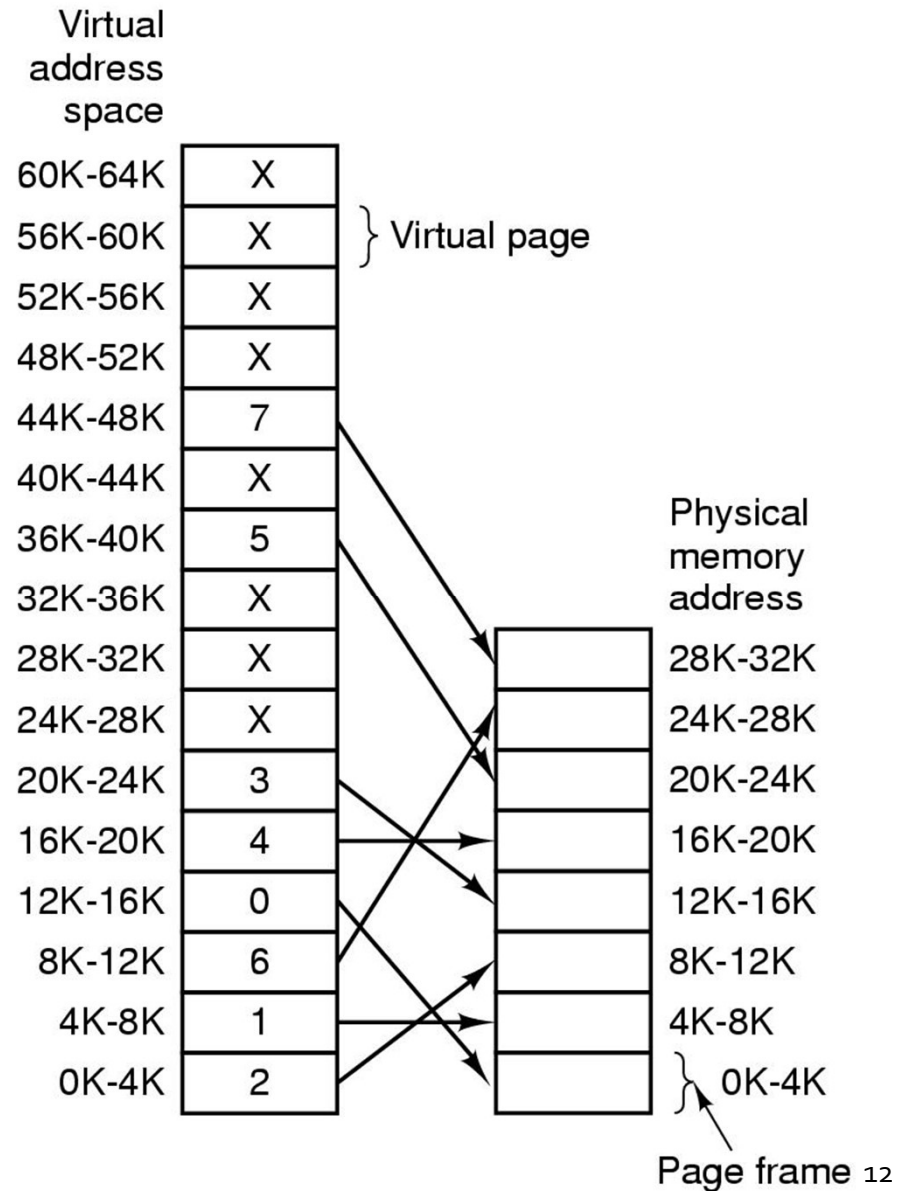
Virtual Memory Paging (1)

- The position and function of the MMU



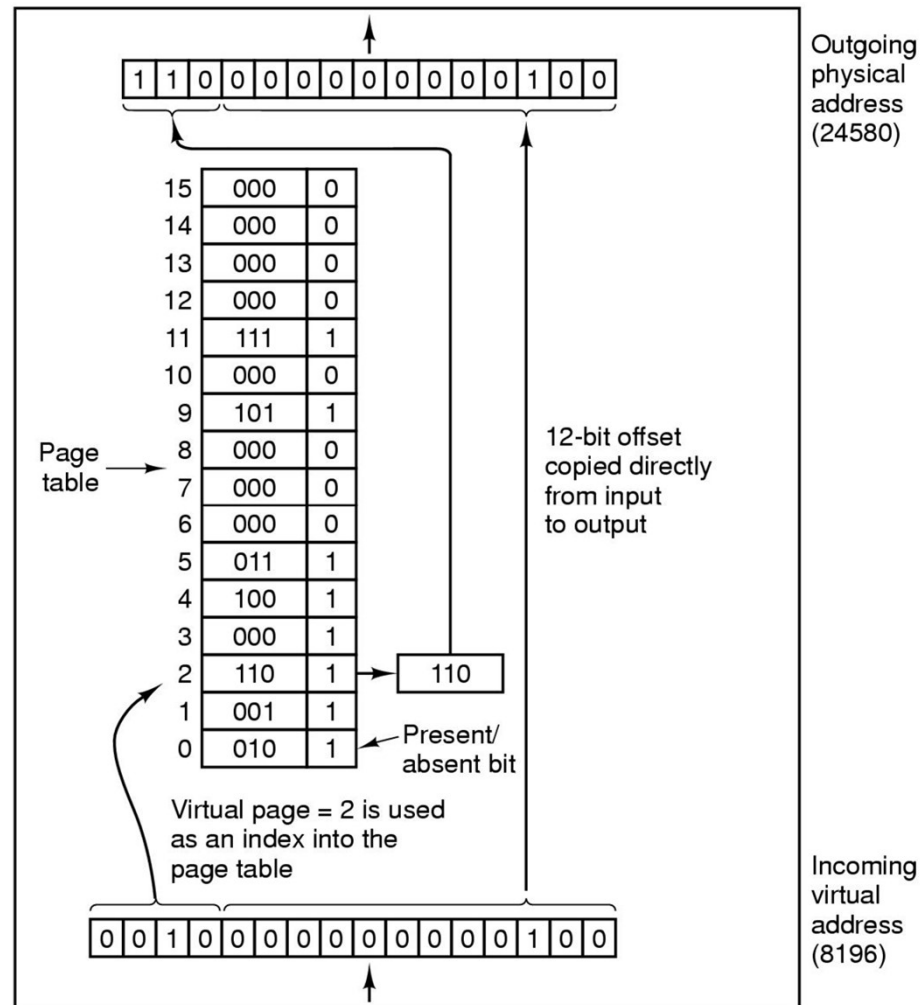
Paging (2)

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



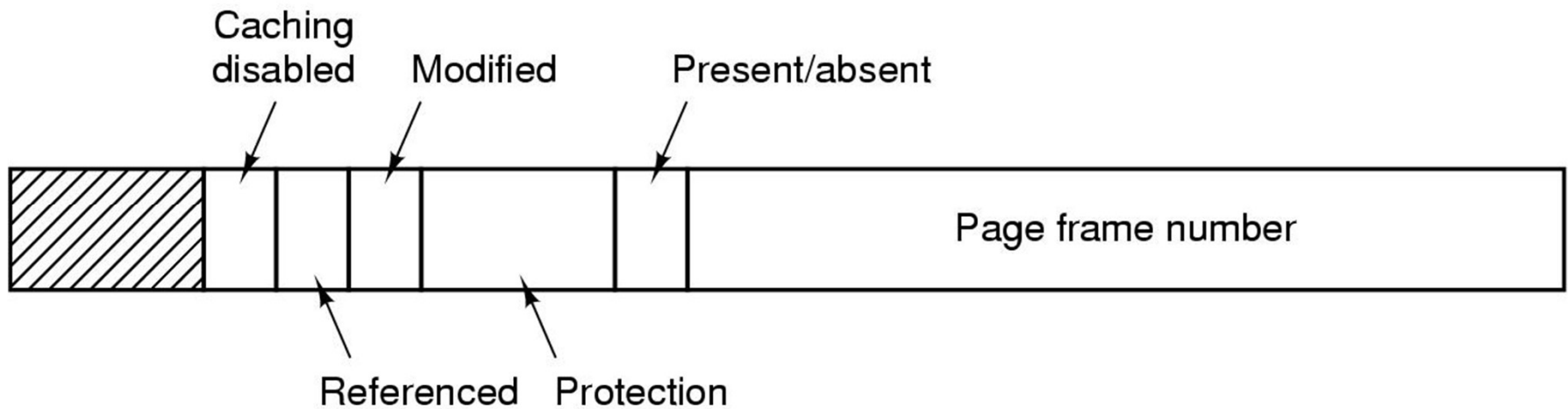
Page Tables (1)

- Internal operation of MMU with 16 4 KB pages



Page Tables (2)

- Typical page table entry



TLBs - Translation Lookaside Buffers

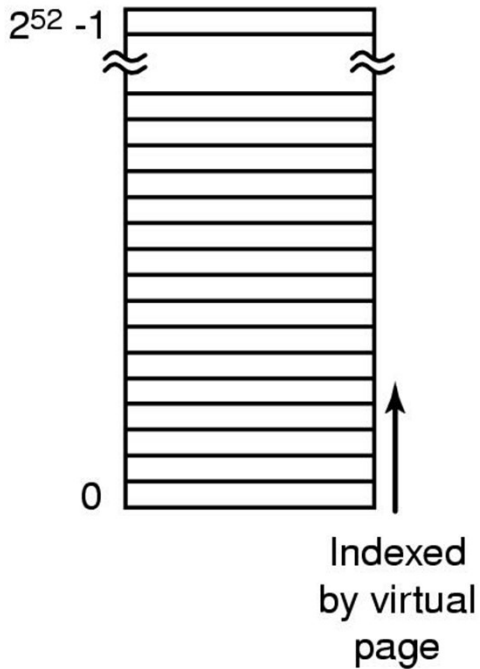
- A TLB to speed up paging

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

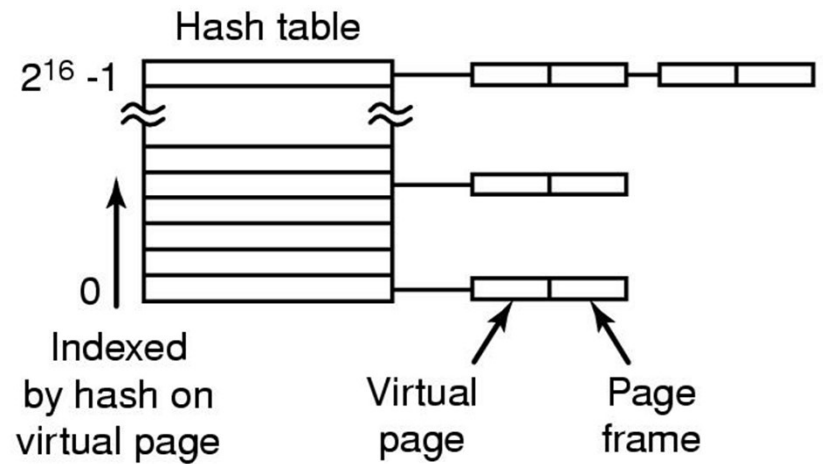
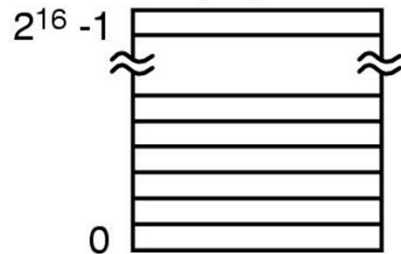
Page Tables for Large Memories

- Inverted Page Tables

Traditional page table with an entry for each of the 2^{52} pages



256-MB physical memory has 2^{16} 4-KB page frames





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



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



Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
 - bits are set when page is referenced, modified
- Pages are classified
 - Class 0: not referenced, not modified.
 - Class 1: not referenced, modified.
 - Class 2: referenced, not modified.
 - Class 3: referenced, modified.
- NRU removes page at random
 - from lowest numbered non empty class

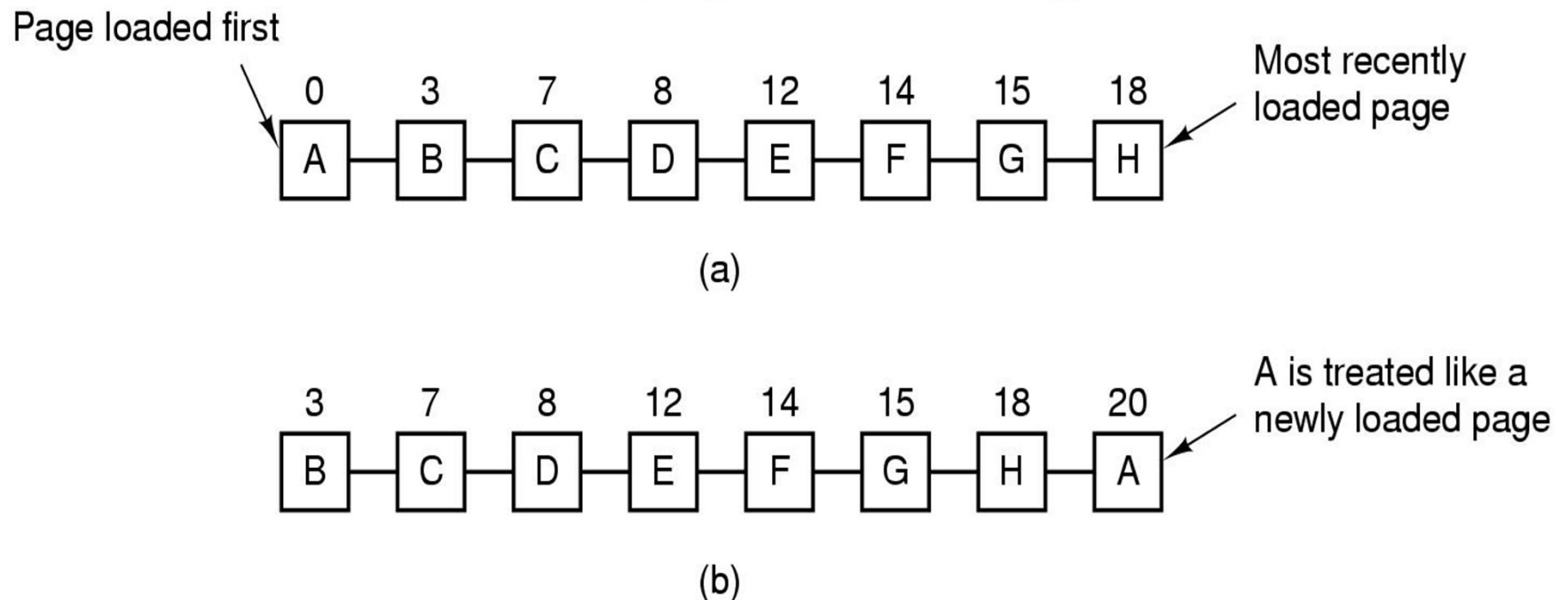


FIFO Page Replacement Algorithm

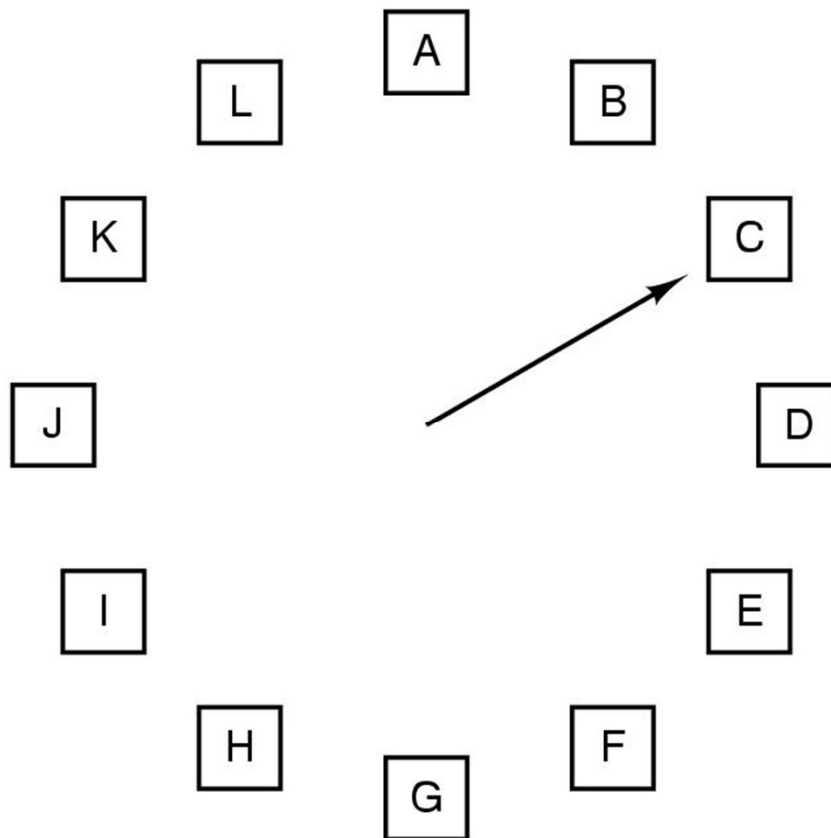
- Maintain a linked list of all pages
 - in order they came into memory
- Page at beginning of list replaced
- Disadvantage
 - page in memory the longest may be often used

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

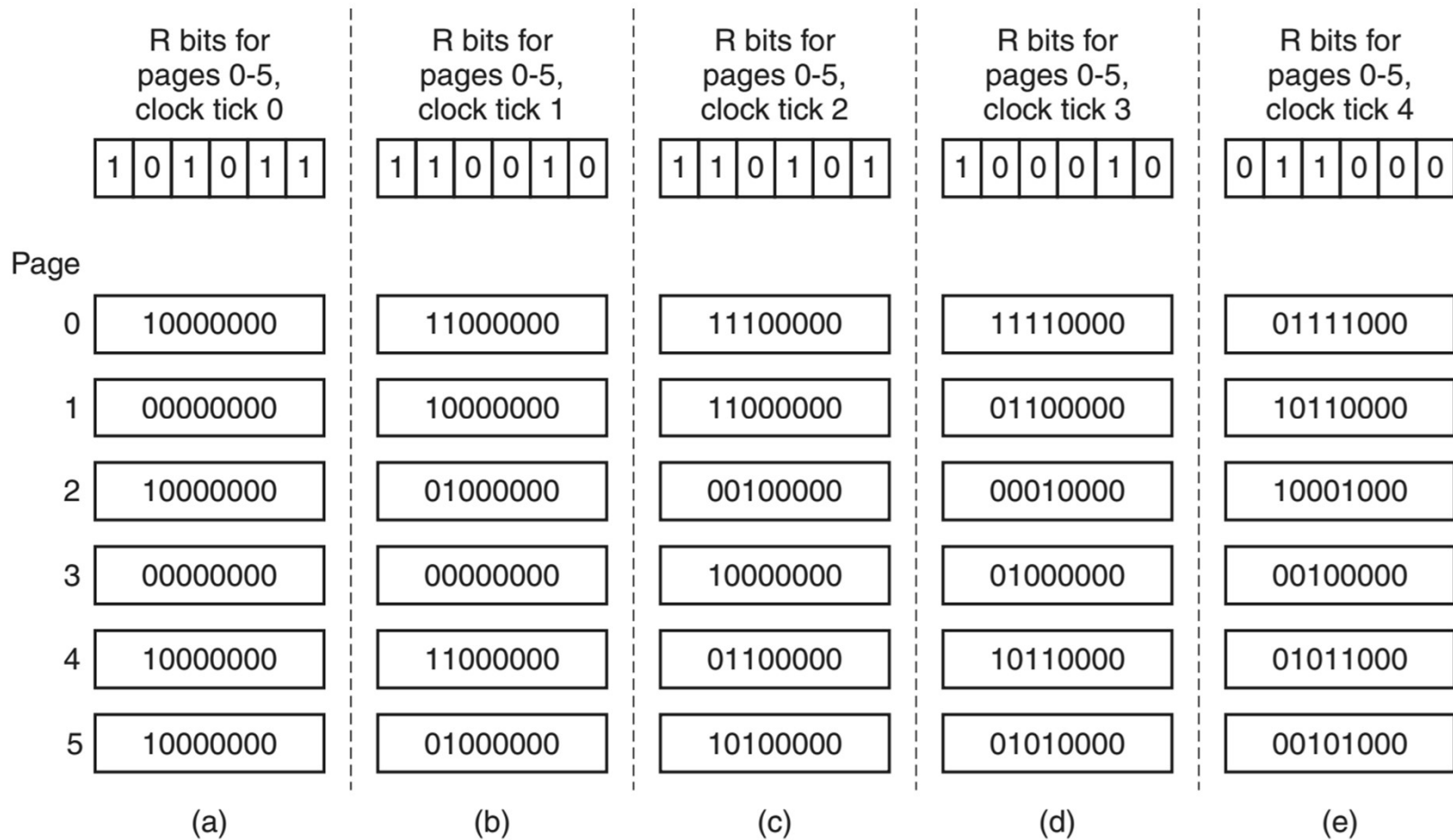


Least Recently Used (LRU)

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

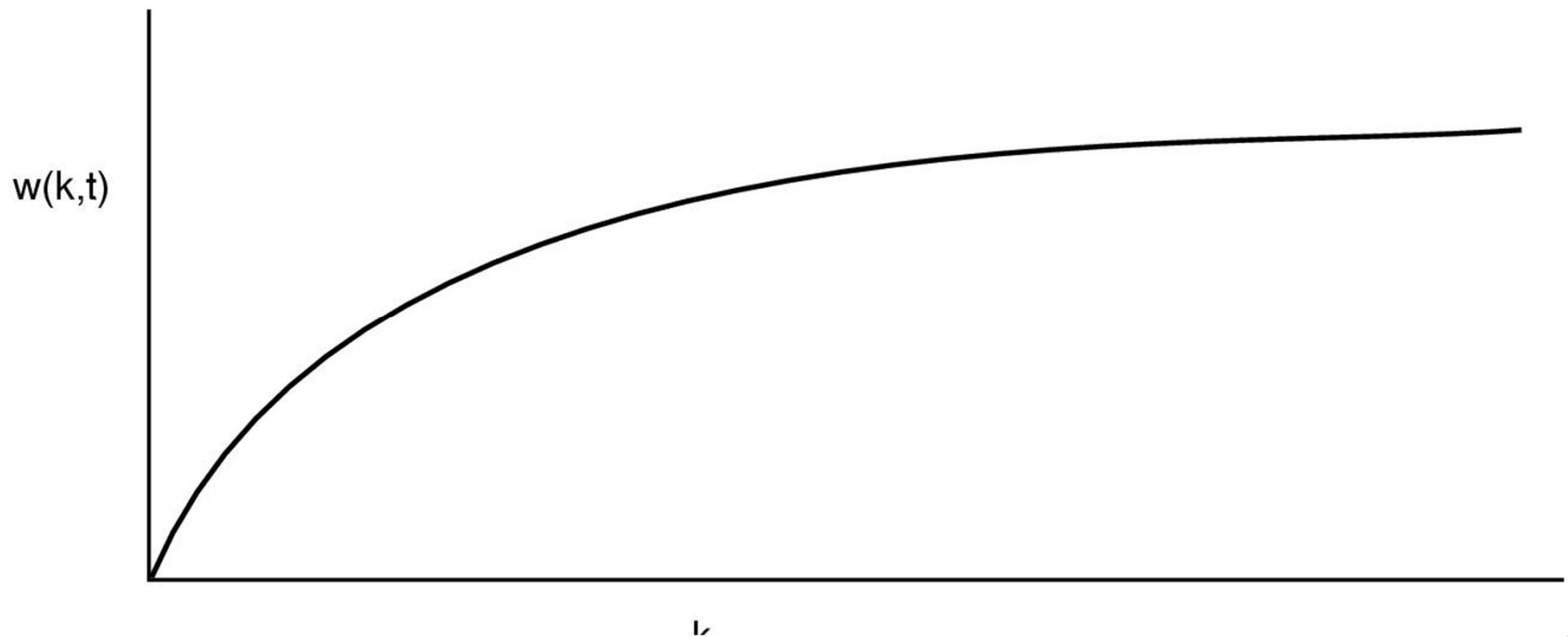
Simulating LRU in Software

- The aging algorithm



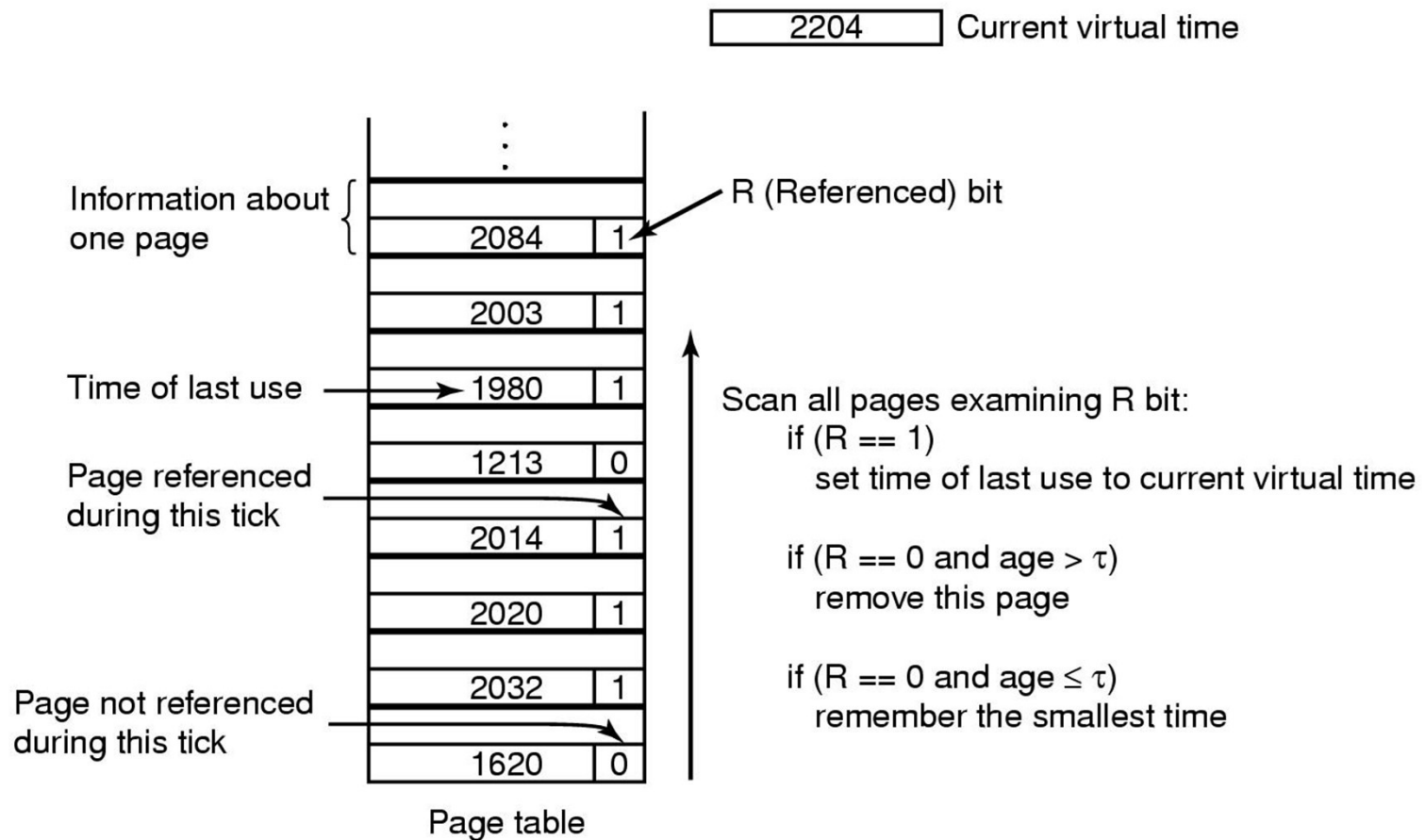
The Working Set Page Replacement Algorithm (1)

- 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



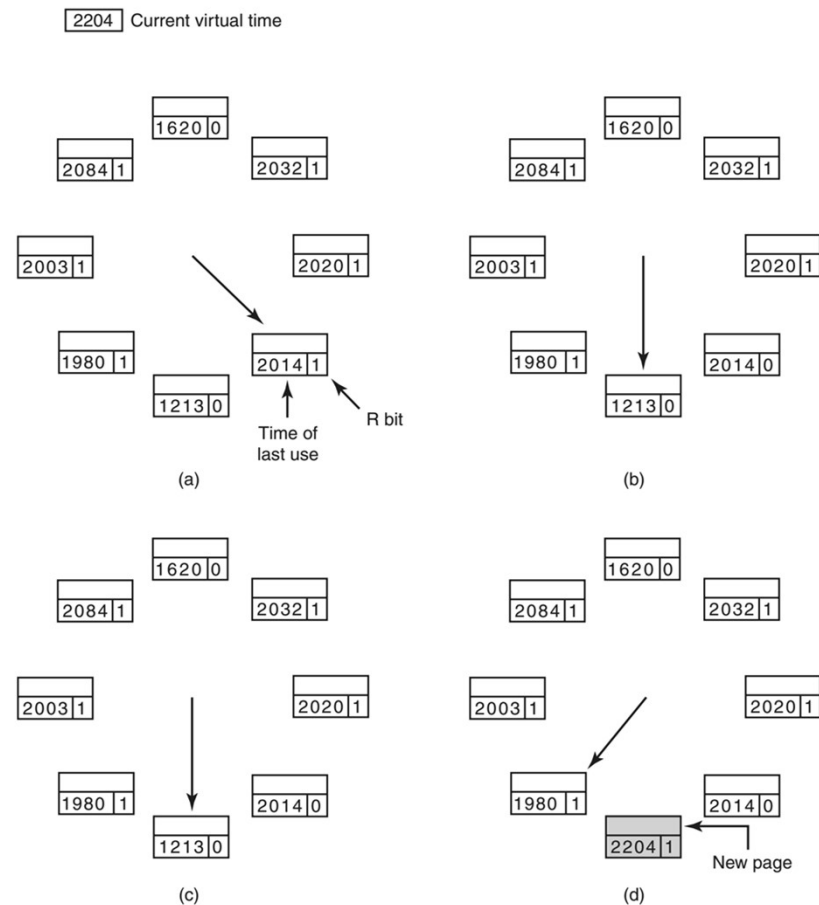
The Working Set Page Replacement Algorithm (2)

- The working set algorithm



The WSClock Page Replacement Algorithm

- Operation of the WSClock algorithm

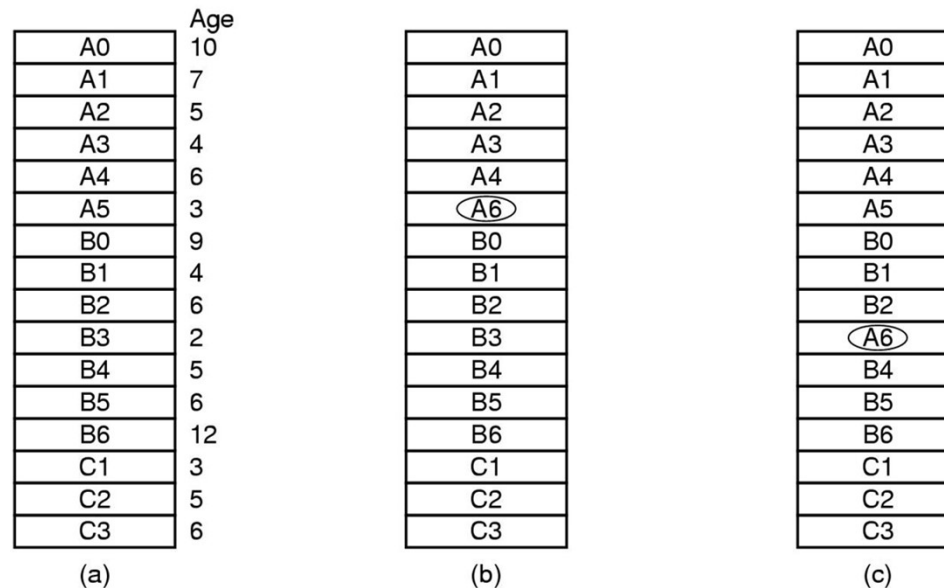


Review of Page Replacement Algorithms

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

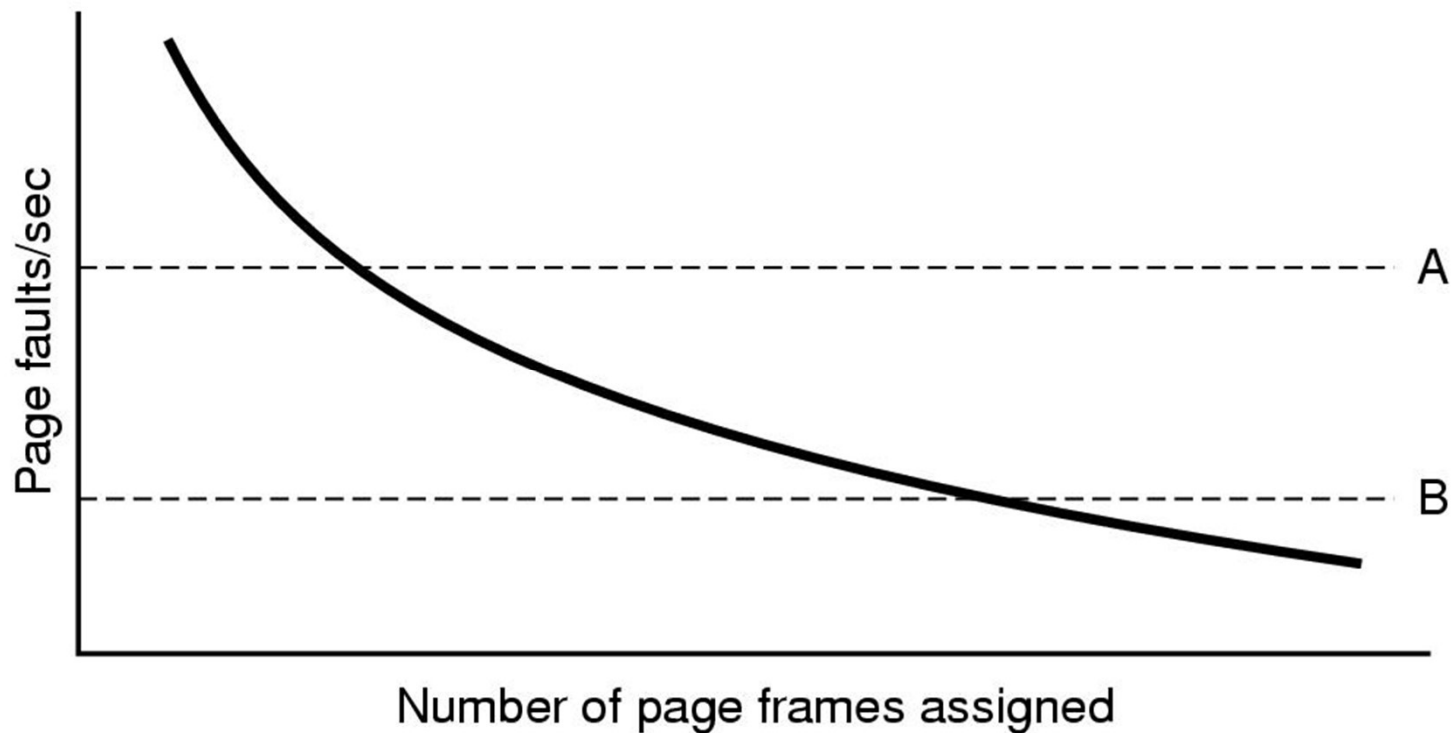
Design Issues for Paging Systems

- Local versus Global Allocation Policies
 - Original configuration
 - Local page replacement
 - Global page replacement



Design Issues for Paging Systems

- Local versus Global Allocation Policies
 - Page fault rate as a function of the number of page frames assigned





Design Issues for Paging Systems

- 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

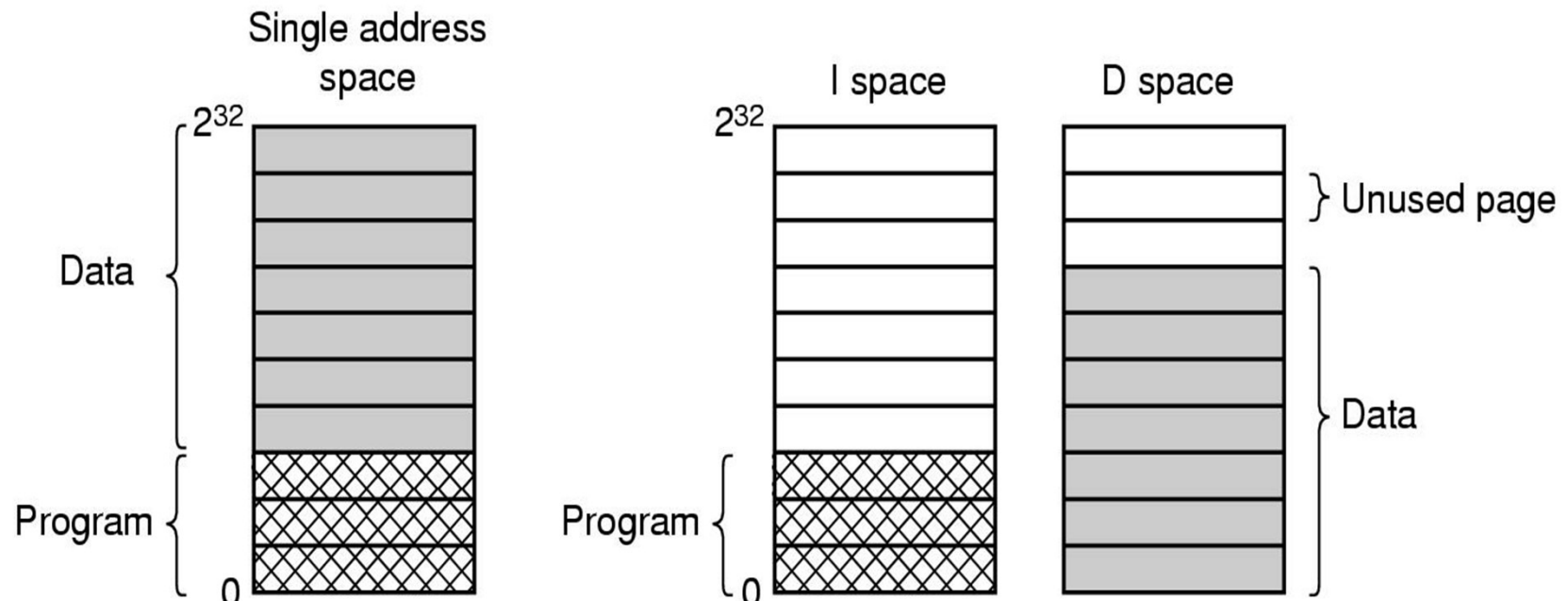


Design Issues for Paging Systems

- Page size
 - 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
 - use up much valuable space in the TLB

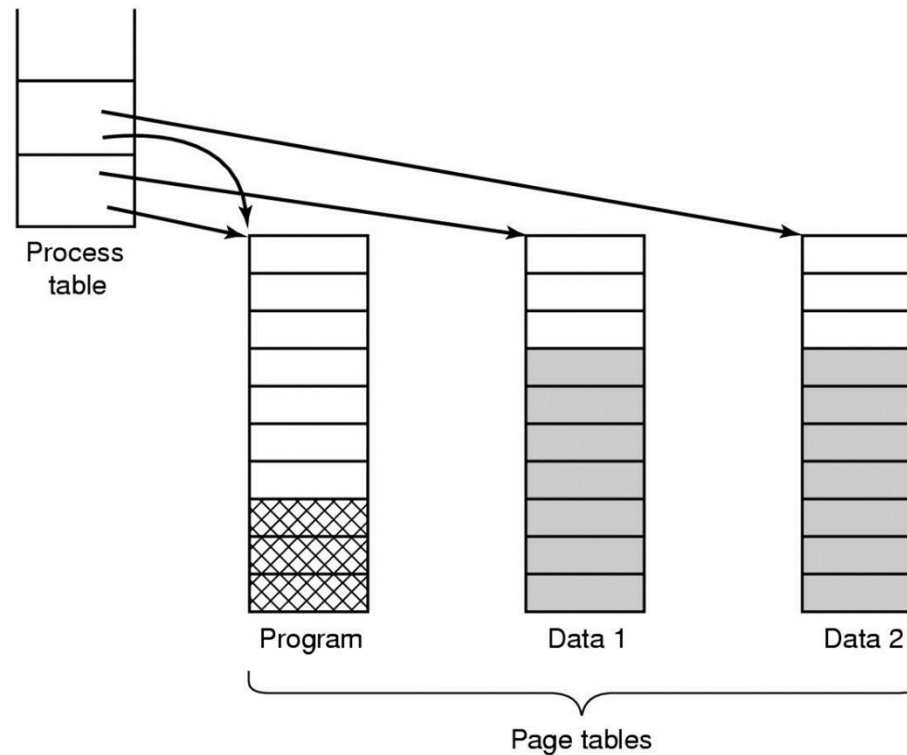
Design Issues for Paging Systems

- Separate Instruction and Data Spaces
 - One address space
 - Separate I and D spaces



Design Issues for Paging Systems

- Shared pages
 - Two processes sharing same program sharing its page table






Design Issues for Paging Systems

- 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
 - Process creation
 - determine program size
 - create page table
 - allocated space in the swap area
 - initialize with program text and data
 - 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
 - Considering shared pages




Operating System Involvement with Paging

- Page Fault Handling
 1. Hardware traps to kernel
 - saving the program counter on the stack
 2. General registers saved
 - assembly-code routine
 - calls the operating system as a procedure
 3. OS determines which virtual page needed
 - hardware register
 - software simulation of current instruction
 4. OS
 - Checks validity of address
 - Checks protection consistency
 - Run page replacement algorithm



Operating System Involvement with Paging

- Page Fault Handling
 5. If selected frame is dirty
 - Schedule page for transfer to disk
 - Context switch takes place
 - frame is marked as busy
 6. OS schedules to bring new page in from disk
 - faulting process is still suspended
 7. Page tables updated
 - frame is marked as being in the normal state
 8. Faulting instruction backed up to when it began
 - program counter is reset to point to that instruction

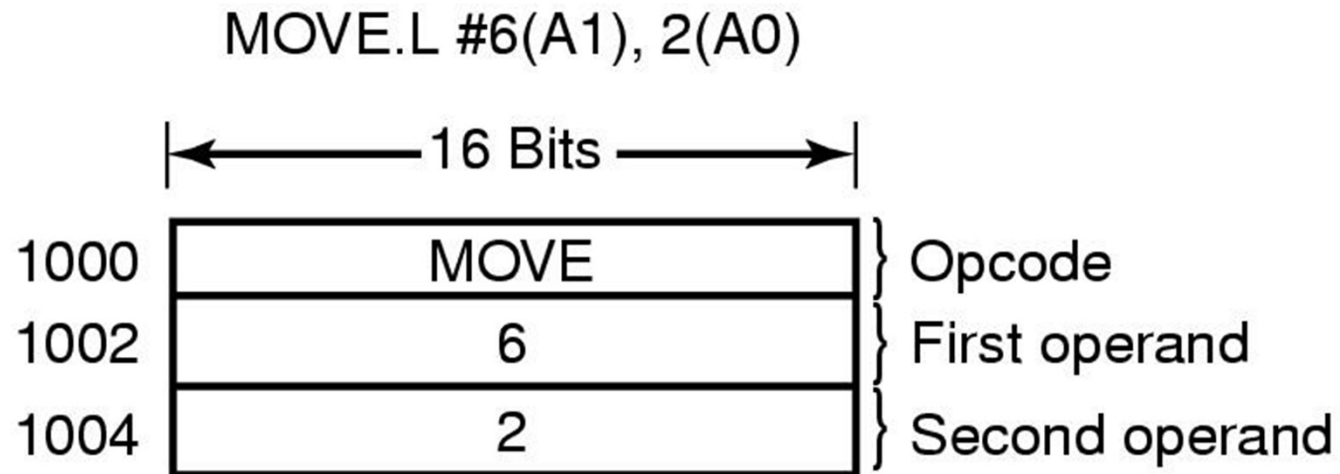


Operating System Involvement with Paging

- Page Fault Handling
 9. Faulting process scheduled
 - operating system returns to the assembly-language routine that called it
 10. routine
 - reloads the registers and other state information
 - returns to user space
 11. Program continues as if no fault had occurred

Instruction Backup

- An instruction causing a page fault



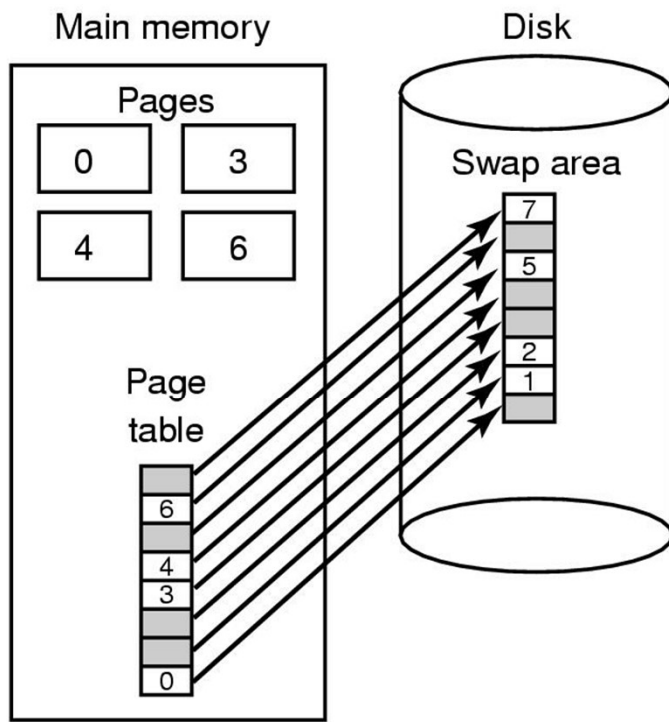


Locking Pages in Memory

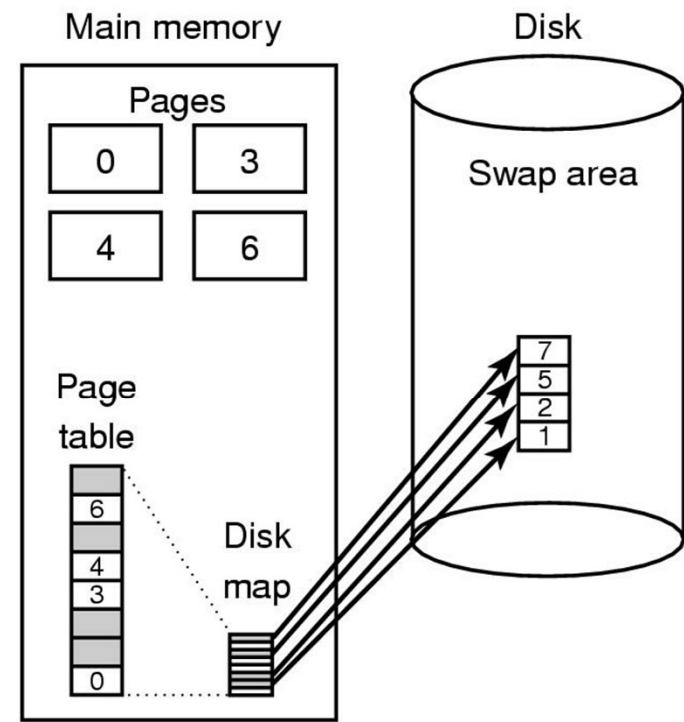
- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
 - while waiting for I/O, another processes starts up
 - has a page fault
 - buffer for the first proc may be chosen to be paged out
- Need pinning: specify some pages locked
 - exempted from being target pages

Backing Store

- (a) Paging to static swap area
- (b) Backing up pages dynamically



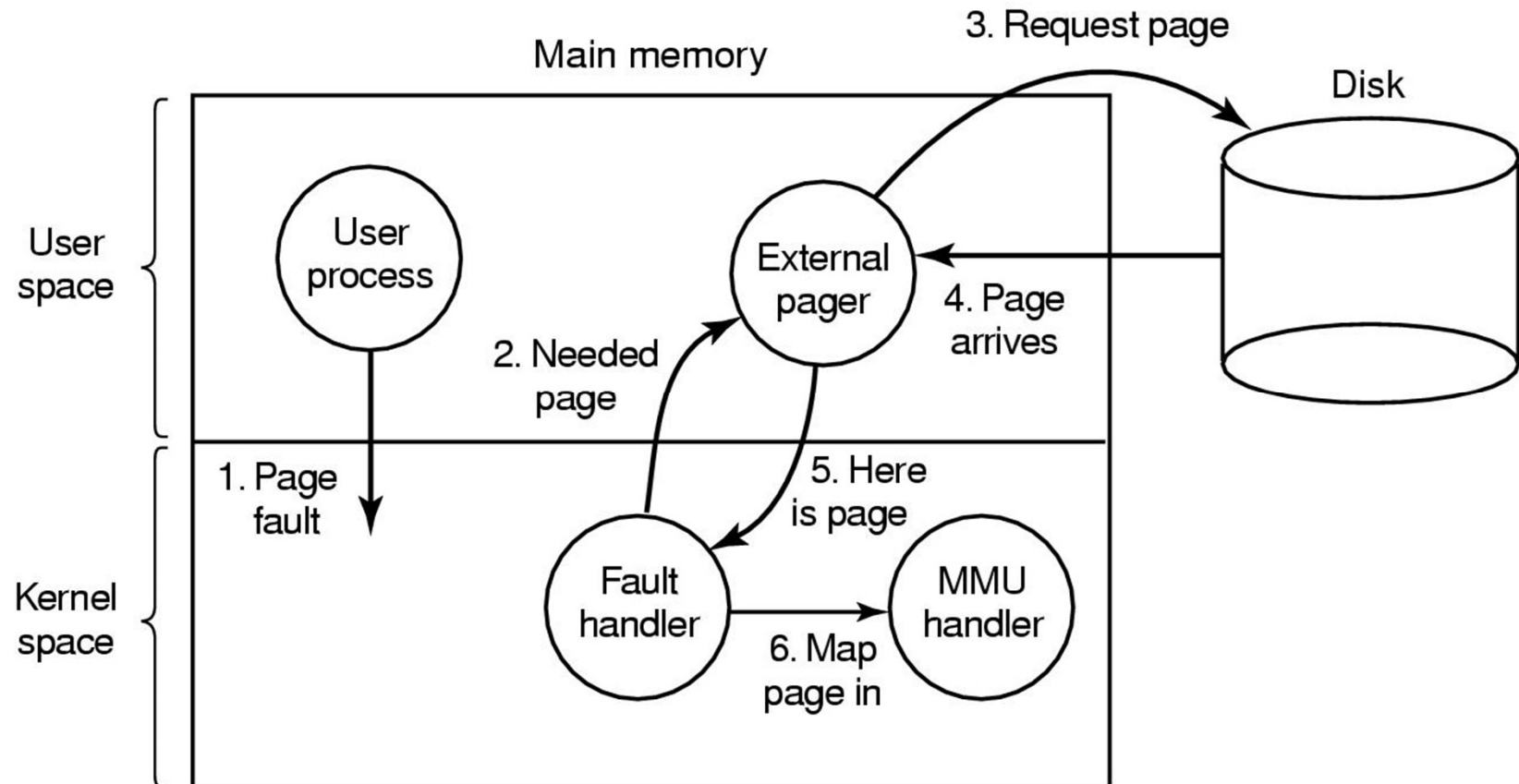
(a)



(b)

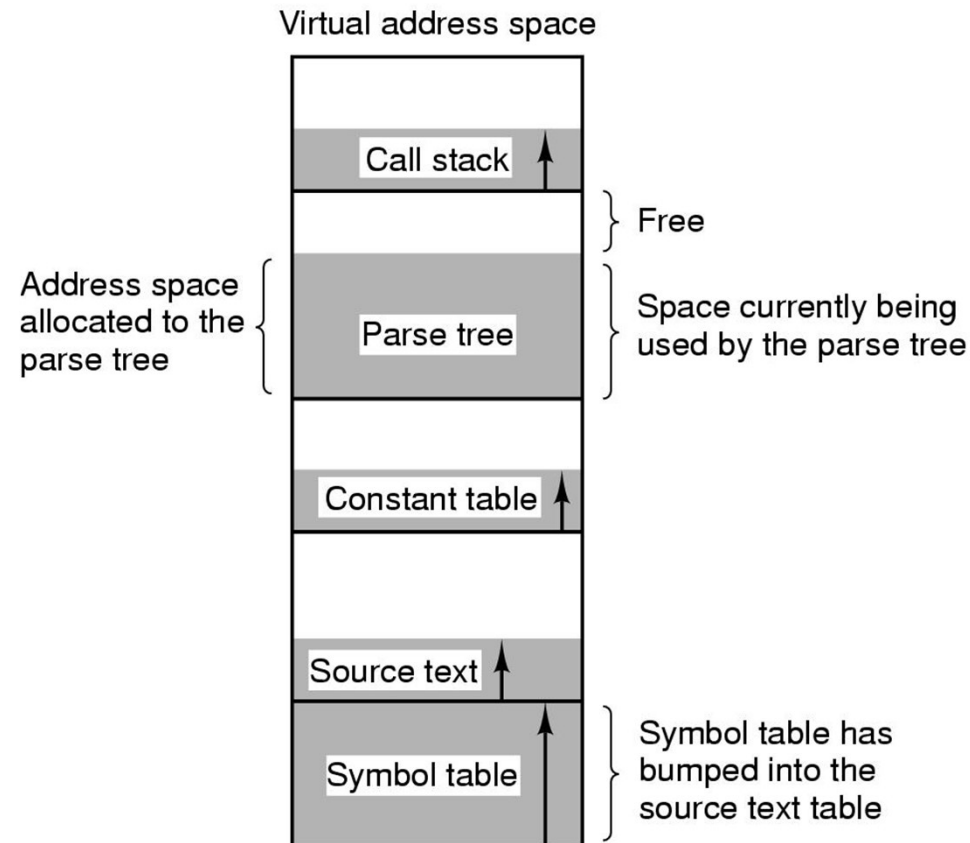
Separation of Policy and Mechanism

- Page fault handling with an external pager



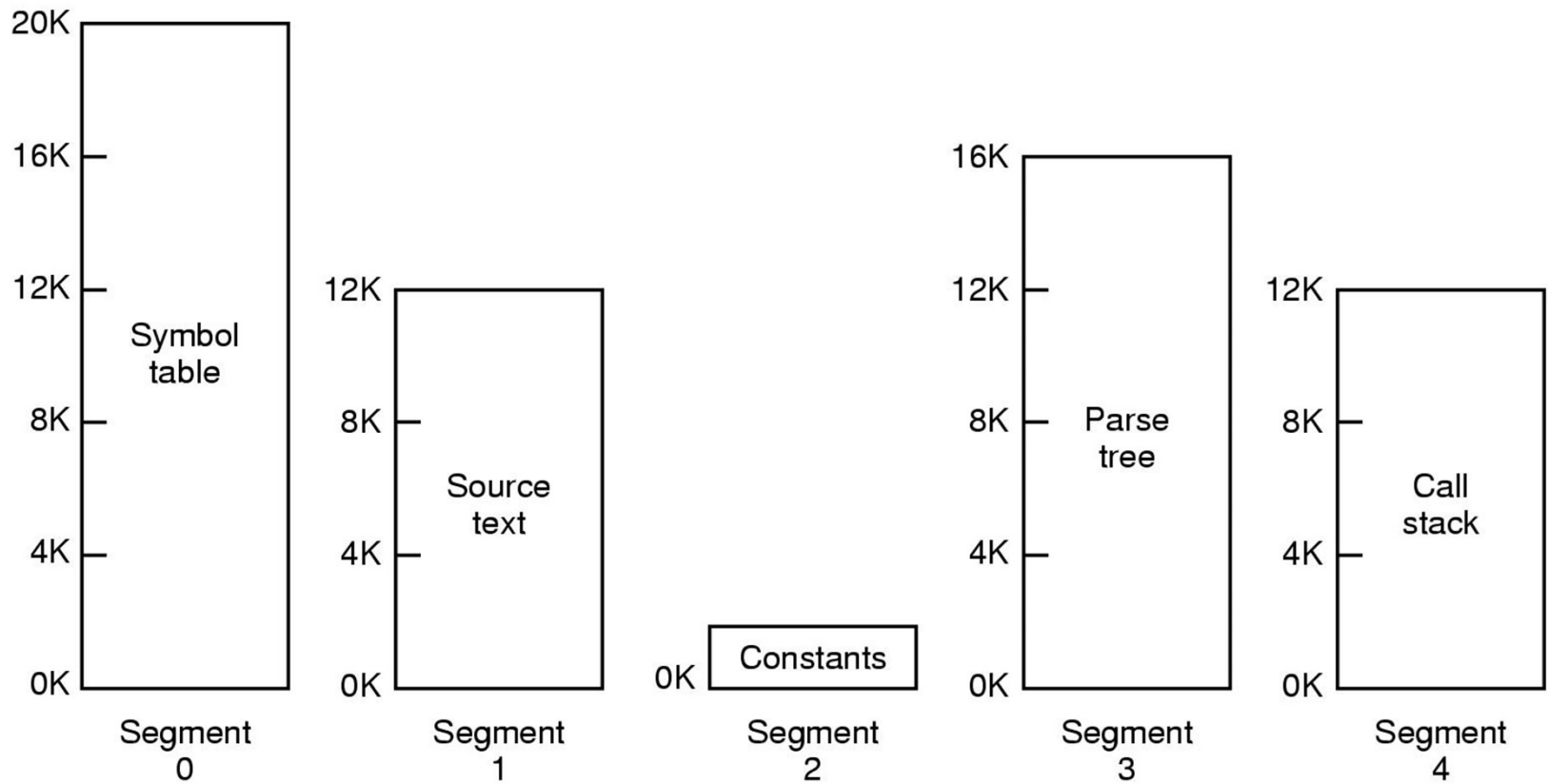
Segmentation (1)

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



Segmentation (2)

- Allows each table to grow or shrink, independently



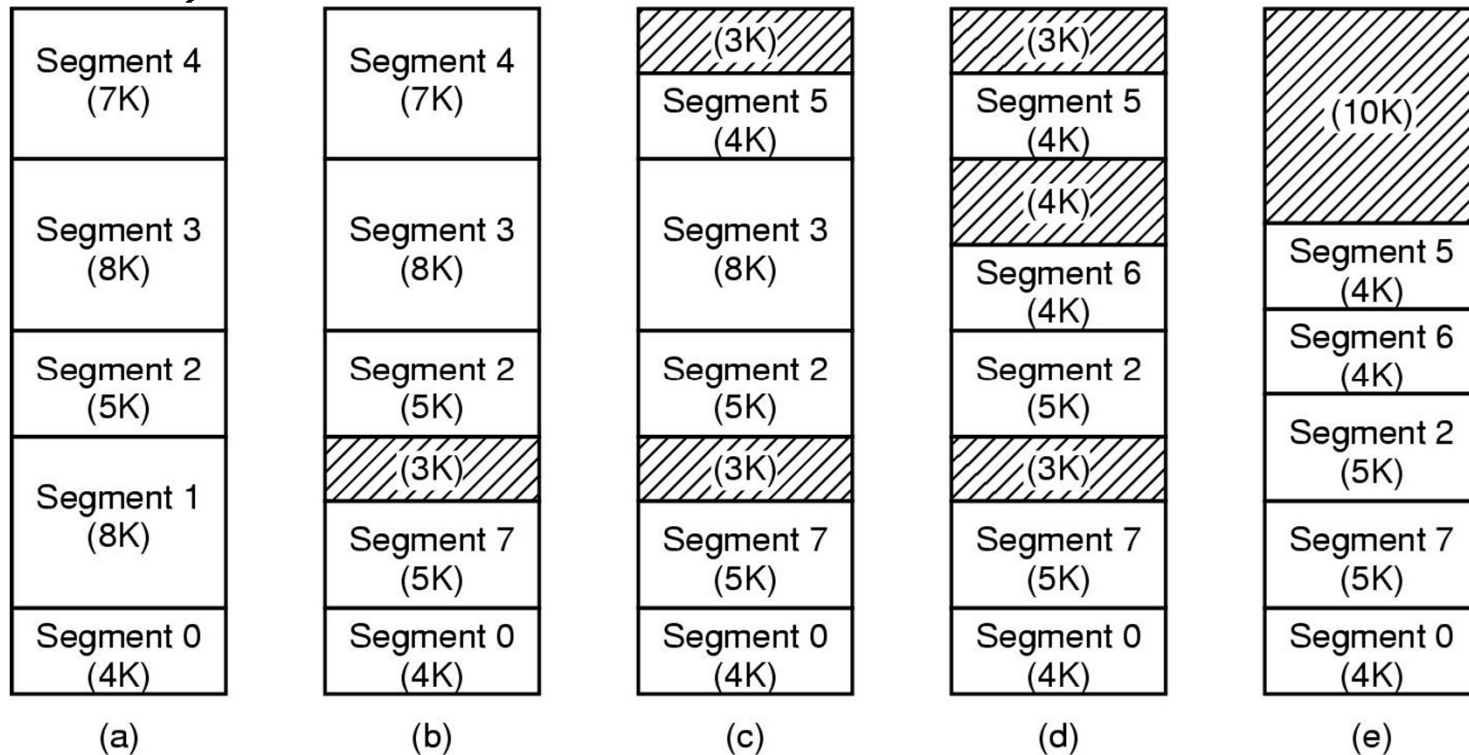
Segmentation (3)

- Comparison of paging and segmentation

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

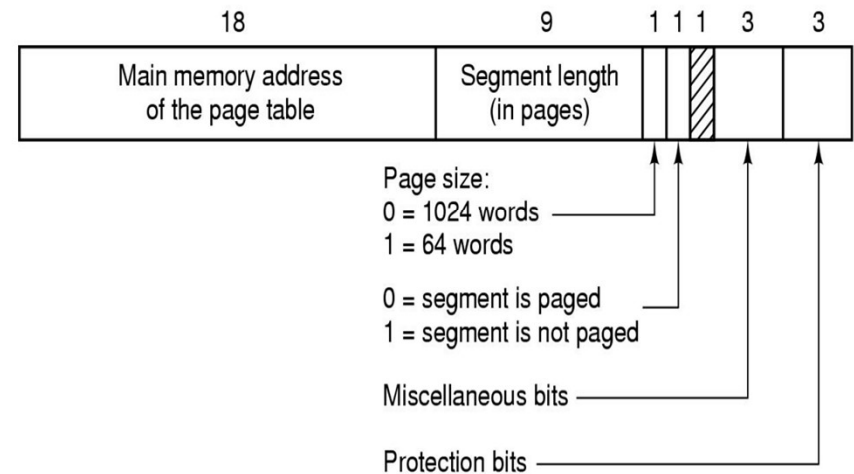
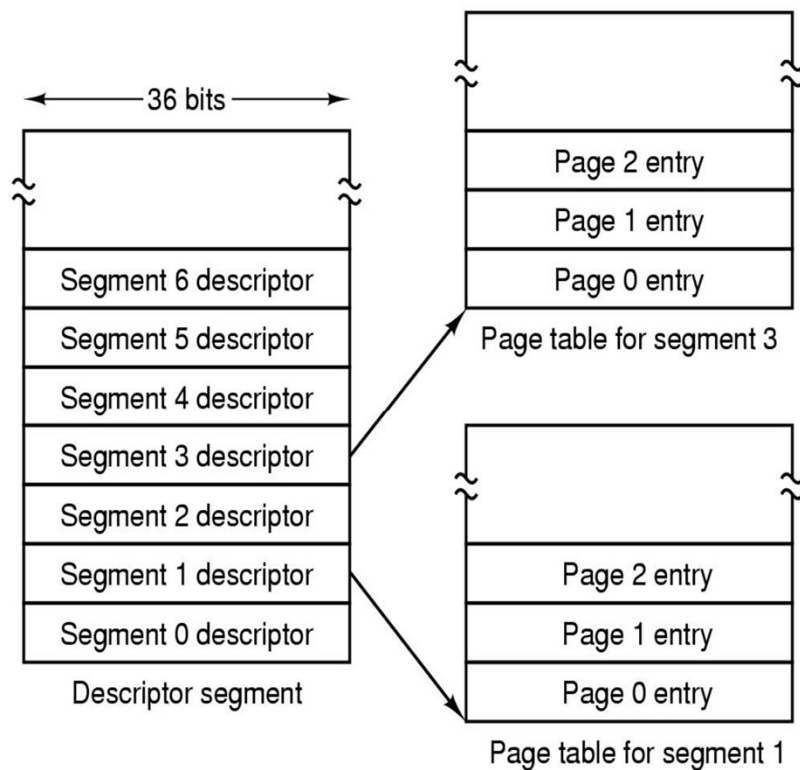
Implementation of Pure Segmentation

- (a)-(d) Development of checkerboarding
- (e) Removal of the checkerboarding by compaction



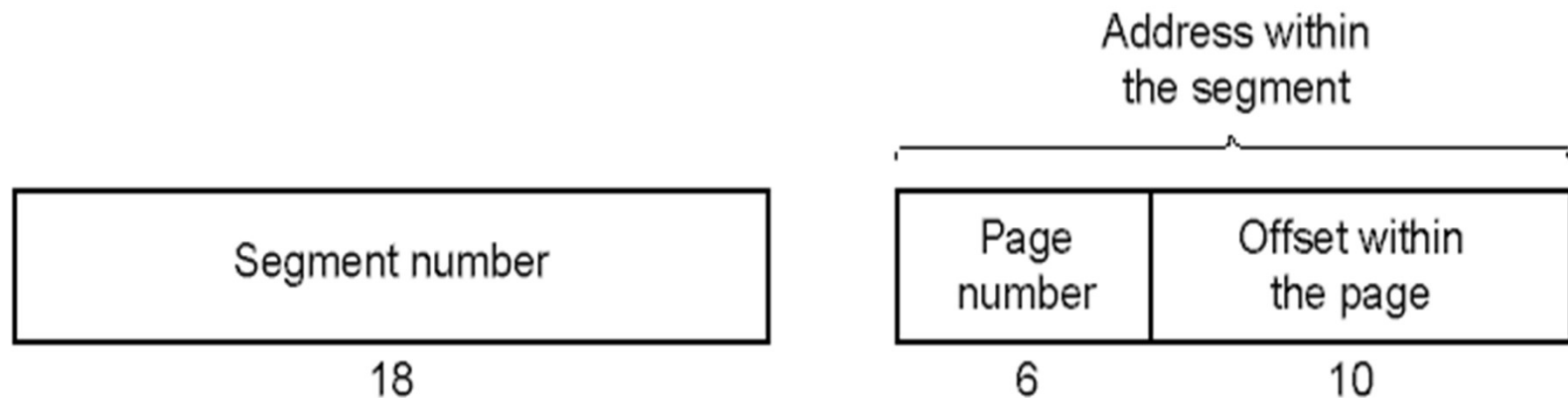
Segmentation with Paging: MULTICS (1)

- Descriptor segment points to page tables
- Numbers are field lengths



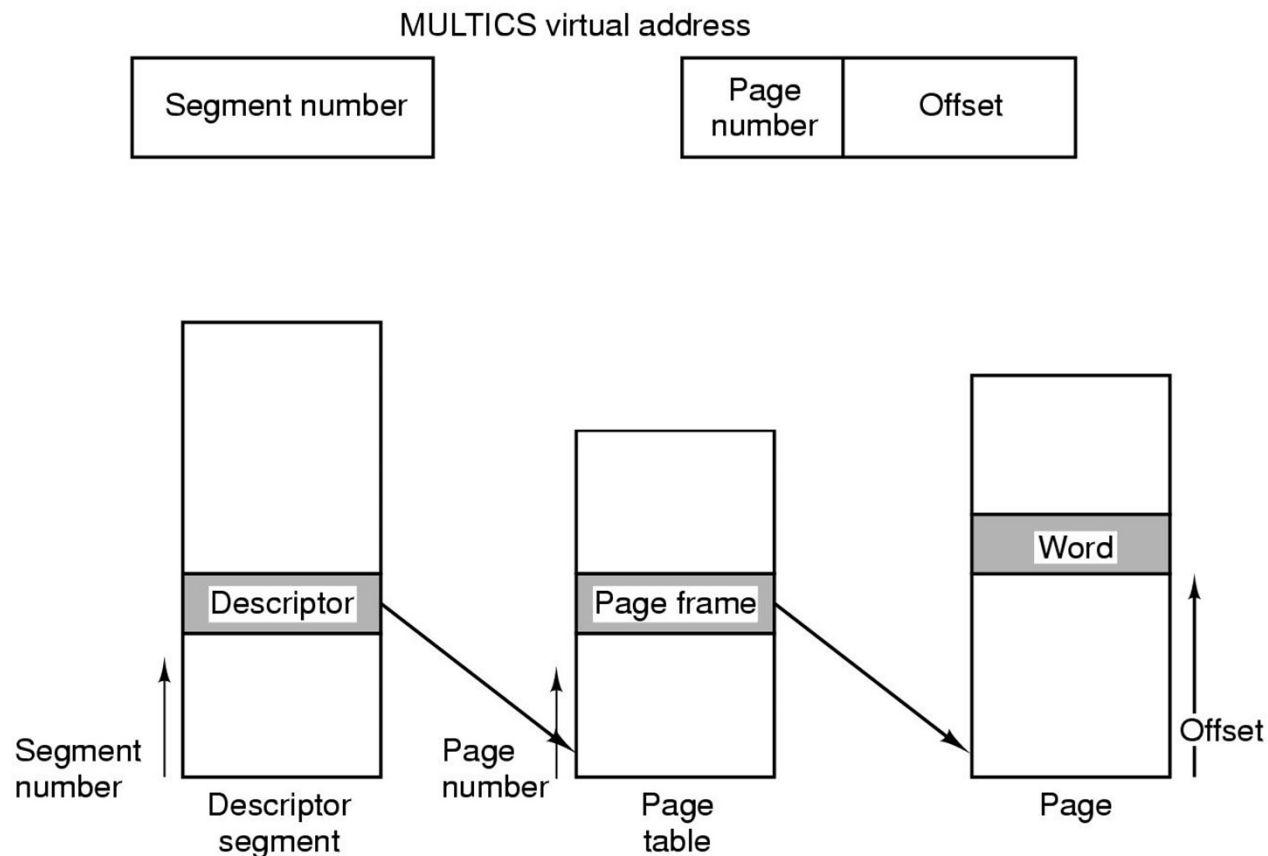
Segmentation with Paging: MULTICS (2)

- A 34-bit MULTICS virtual address



Segmentation with Paging: MULTICS (3)

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



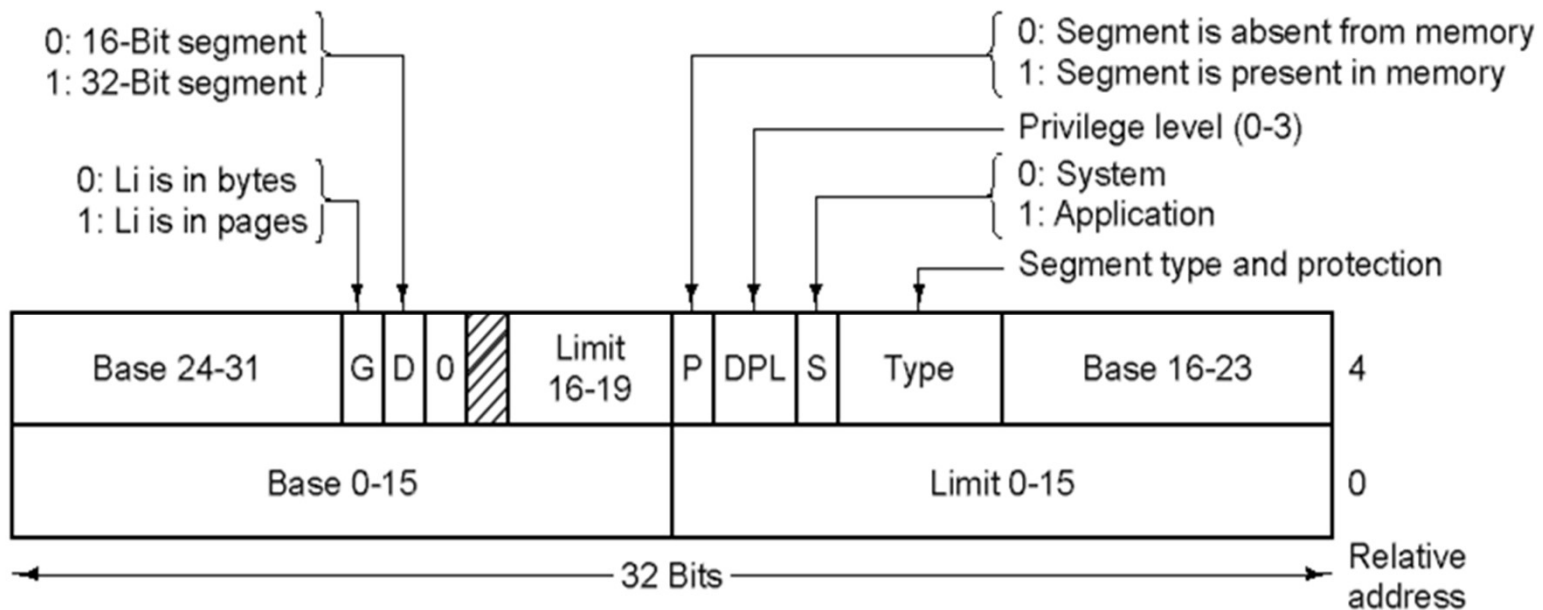
Segmentation with Paging: MULTICS (4)

- Simplified version of the MULTICS TLB

Comparison field		Page frame	Protection	Age	Is this entry used?
Segment number	Virtual page				↓
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

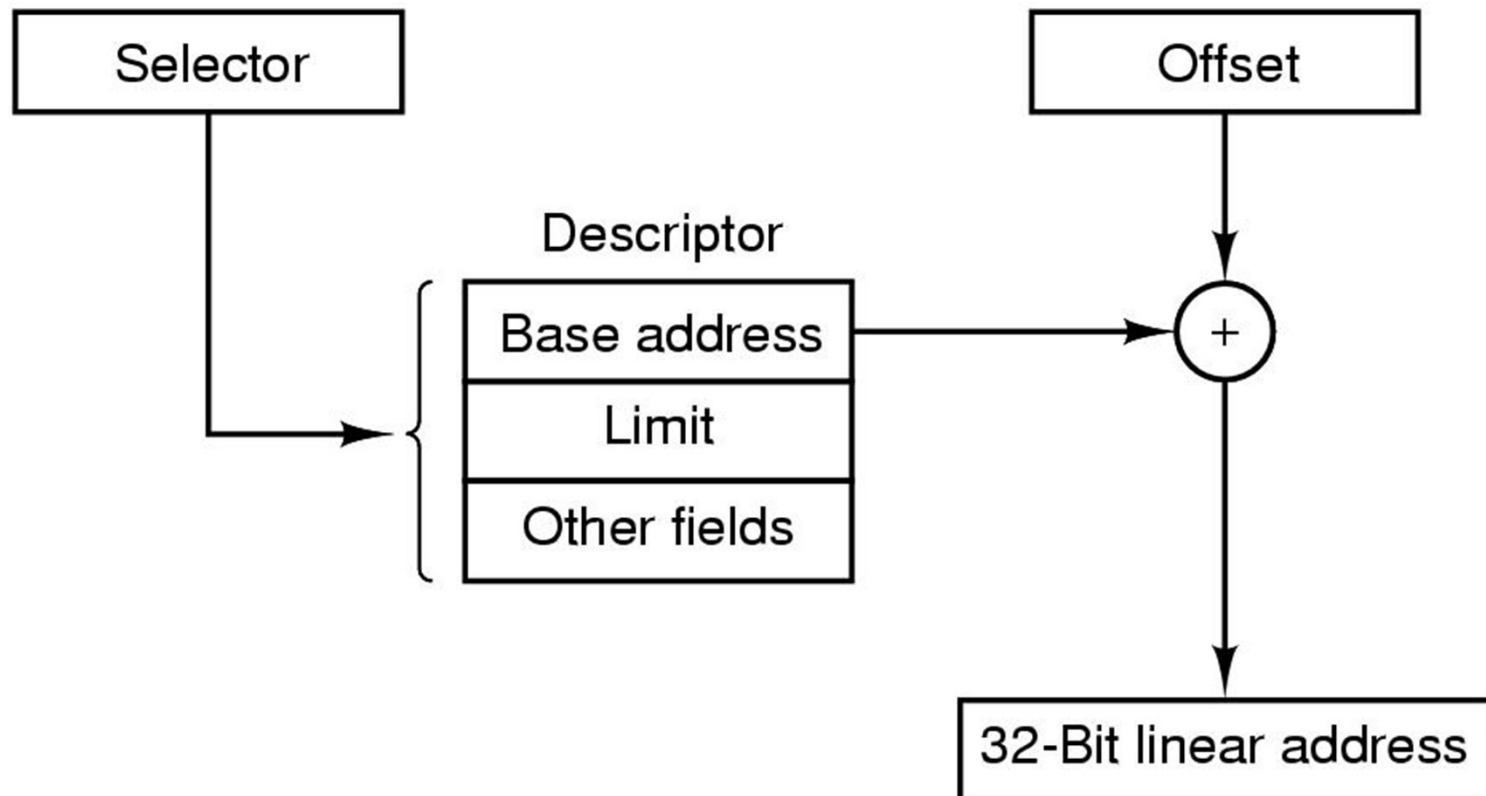
Segmentation with Paging: Intel x86

- x86 code segment descriptor
- Data segments differ slightly



Segmentation with Paging: Intel x86

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



Segmentation with Paging: Intel x86

- Mapping of a linear address onto a physical address

