Chapter 3

Memory Management

Agenda

Memory Management

- Why?
- What?
- How?

General Memory Problem

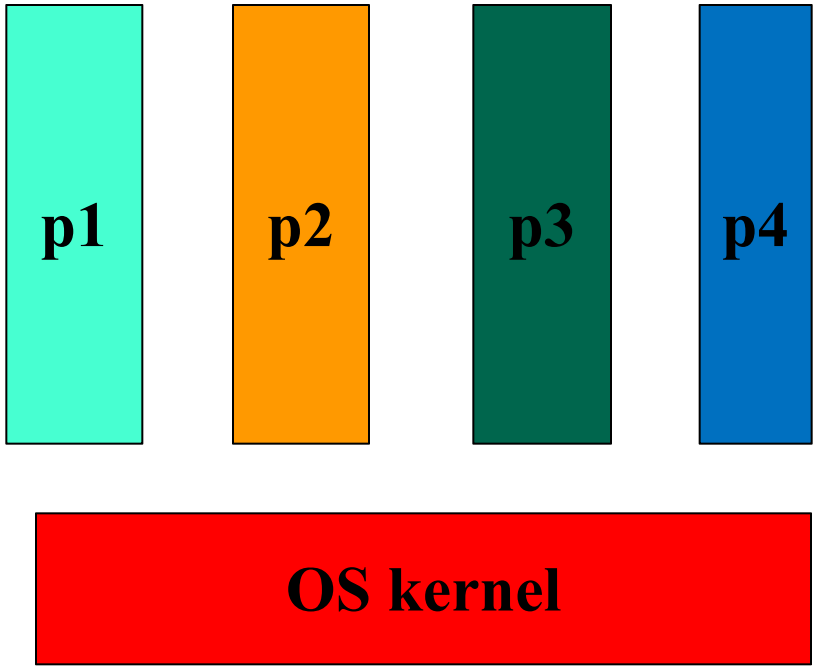
- We have a limited (expensive) physical resource: main memory
- We want to use it as efficiently as possible
- We have an abundant, but slower, resource: disk

Lots of Variants

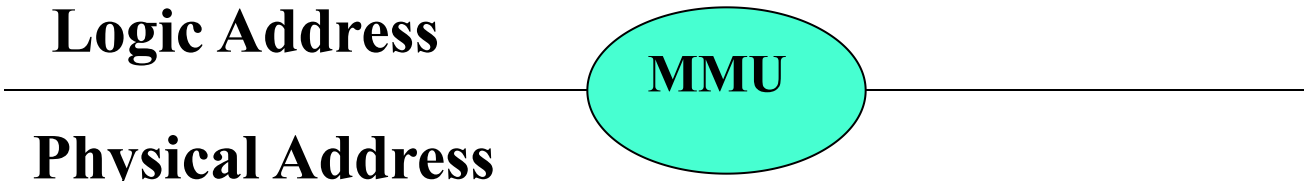
- Many processes, total size less than memory
  - Technically possible to pack them together
  - Will programs know about each other's existence?
- One process, size exceeds memory
  - Can you only keep part of the program in memory?
- Lots of processes, total size exceeds memory
  - What programs are in memory, and how to decide?

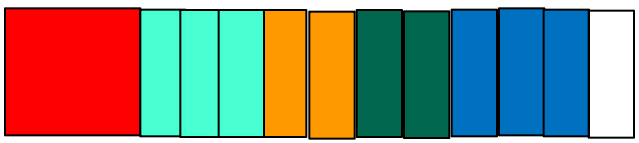
Memory Manager

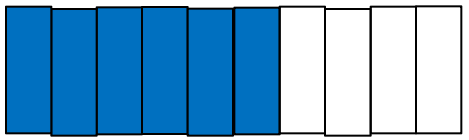
- It's task: Manage memory hierarchy
  - Track used and free memory
  - Allocate memory to processes
  - Reclaim (De-allocate) memory
  - **Swapping** between main memory and disk



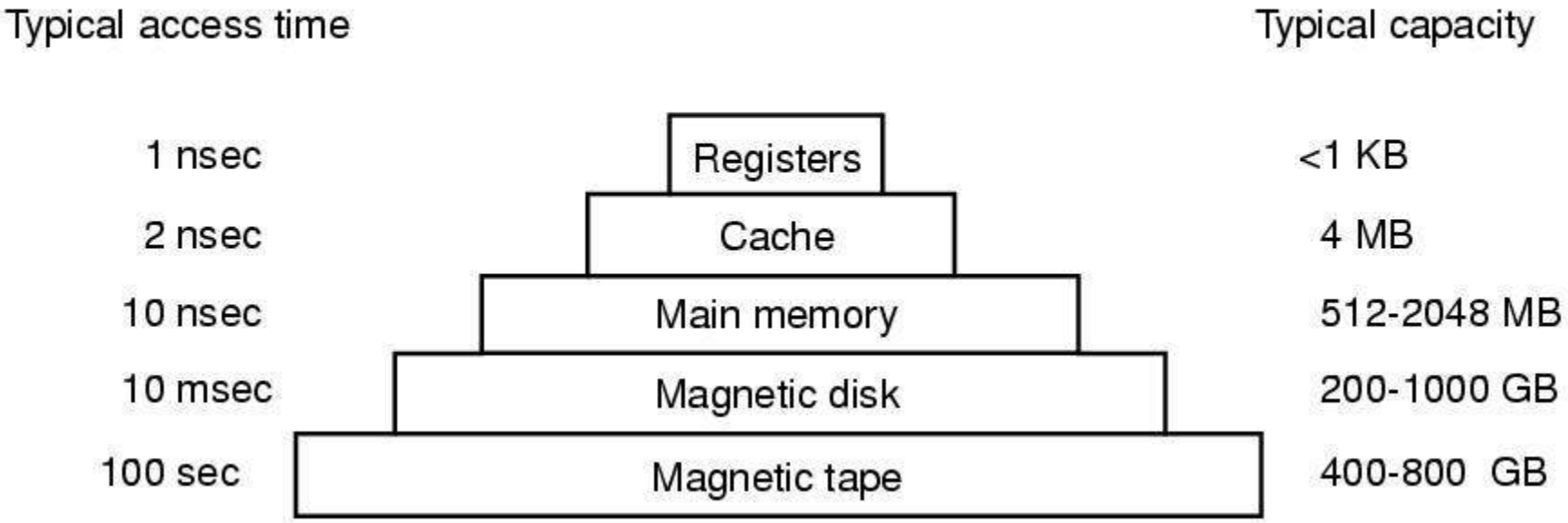
- Abstraction
- Protection
- Share
- Virtualization







Memory Hierarchy



A typical memory hierarchy.
The numbers are very rough approximations.

Memory Cache

Questions when dealing with cache:

- When to put a new item into the cache.
- Which cache line to put the new item in.
- Which item to remove from the cache when a slot is needed.
- Where to put a newly evicted item in the larger memory.

Types of Memory

- **Primary Memory** (a.k.a. RAM)
  - Holds data and programs used by a process that is executing
  - Only type of memory that a CPU deals with
- **Secondary Memory** (i.e. hard disk)
  - Non-volatile memory used to store data when a process is not executing

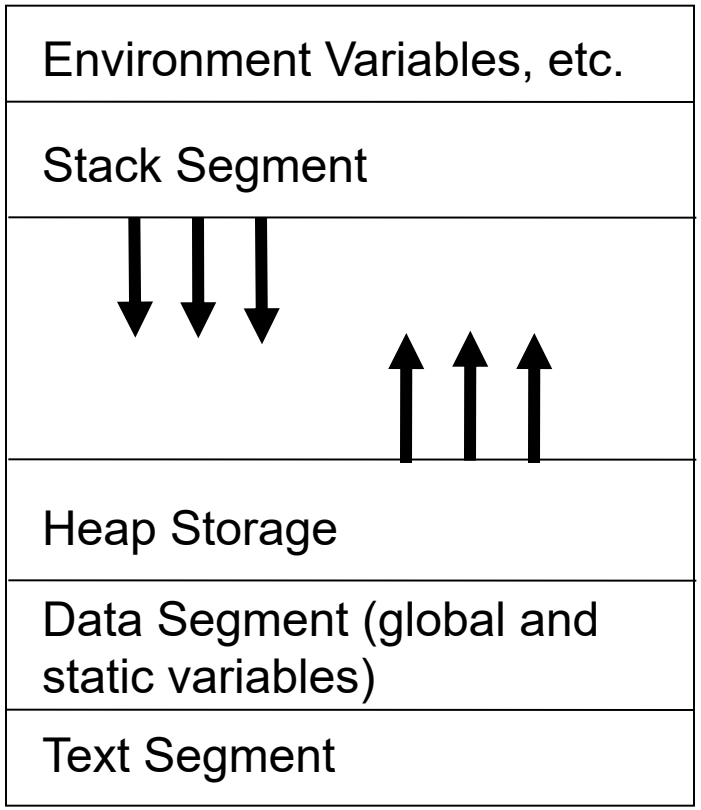
The Memory Manager (MM)

- **Purpose:** to manage the use of primary and secondary memory.
- **Responsible for:**
  - Allocating primary memory to processes
  - Moving processes between primary and secondary memory
  - Optimizing memory usage

Process Memory

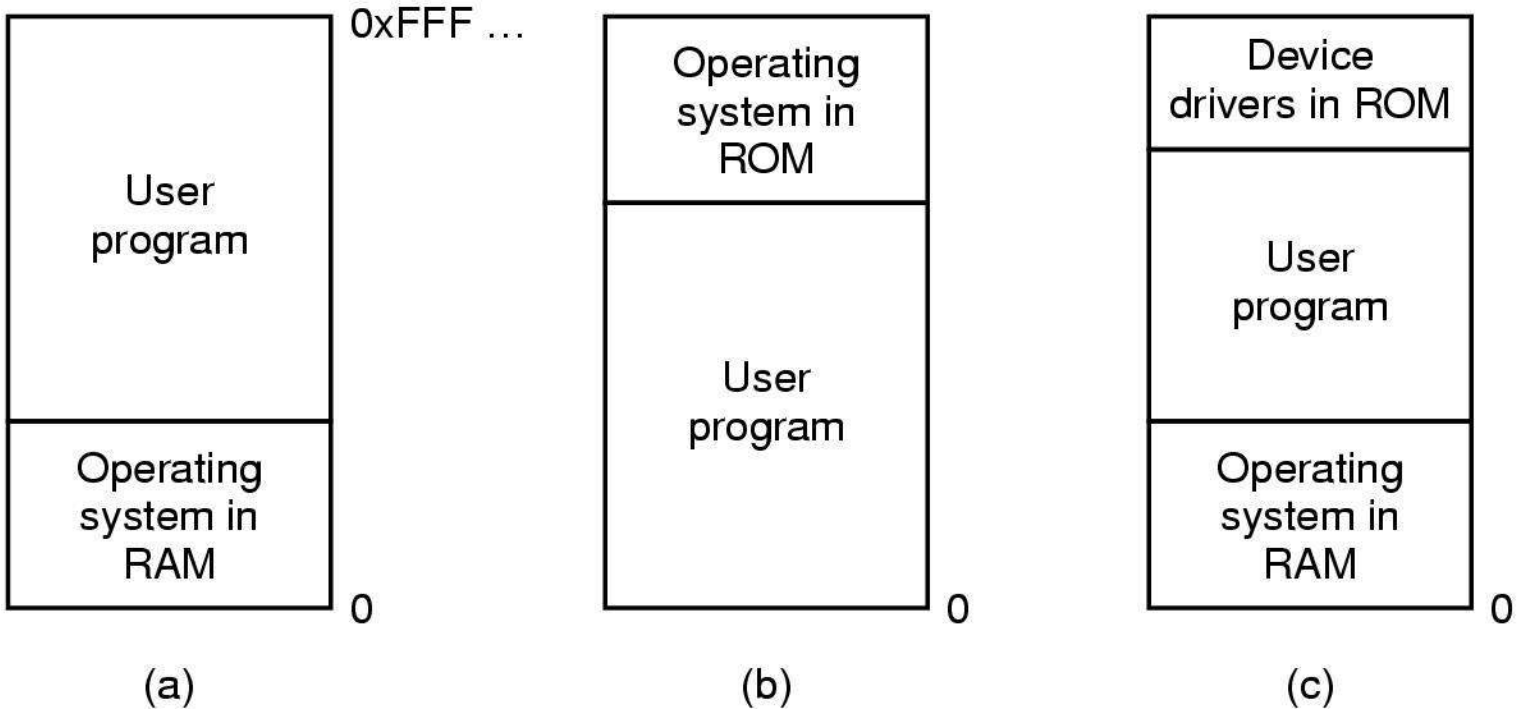
- There are two types of memory that can
- be used within a process:
- **Stack:** used for local/automatic variables and for passing parameters to function calls
- **Heap:** used for dynamic memory allocation

Process Memory Layout



- Allocates more memory than needed at first
- Heap grows towards stack for dynamic memory allocation
- Stack grows towards heap when automatic variables are created

Single-Partition Strategies

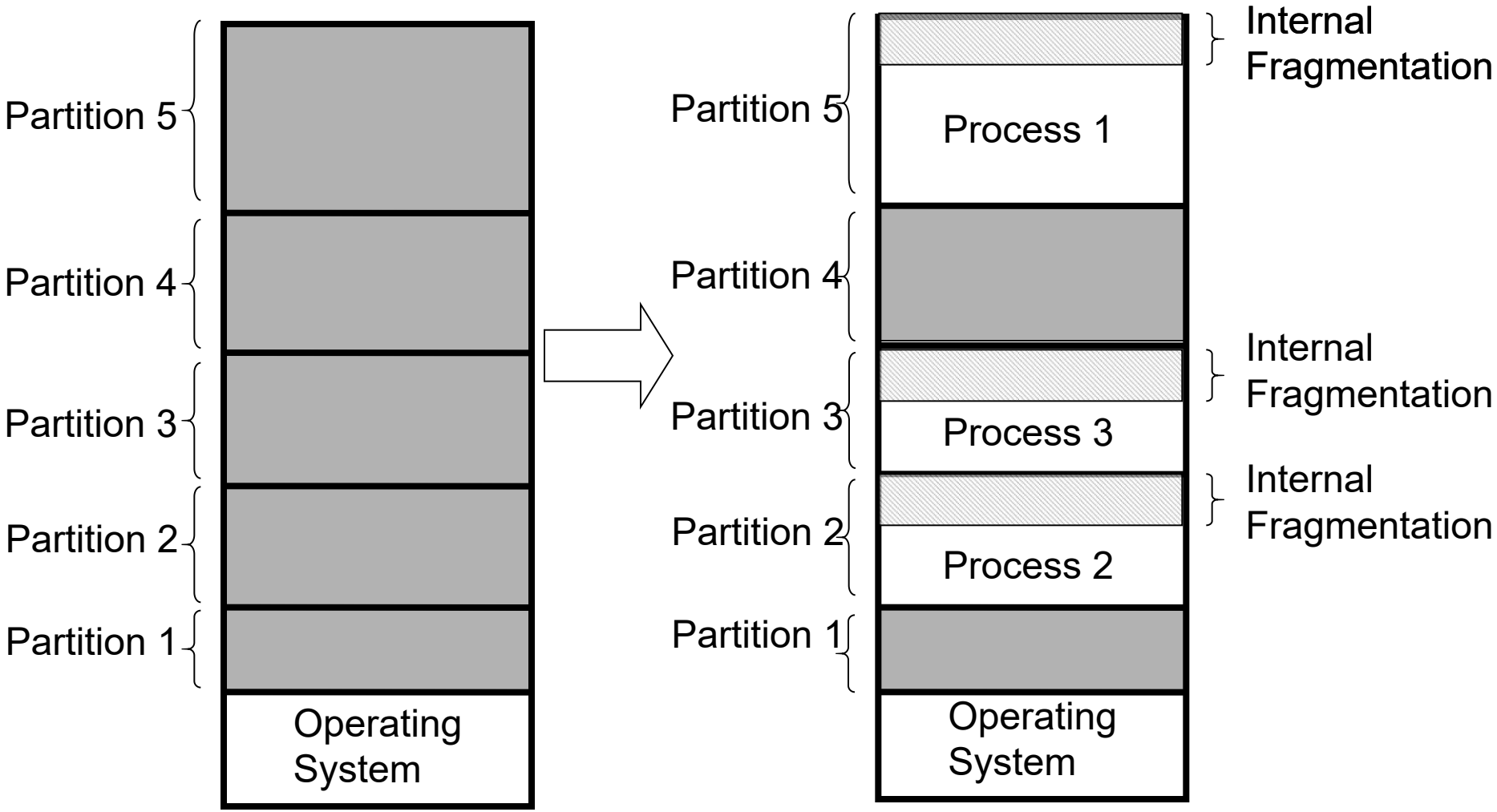


Three simple ways of organizing memory with an operating system and one user process.

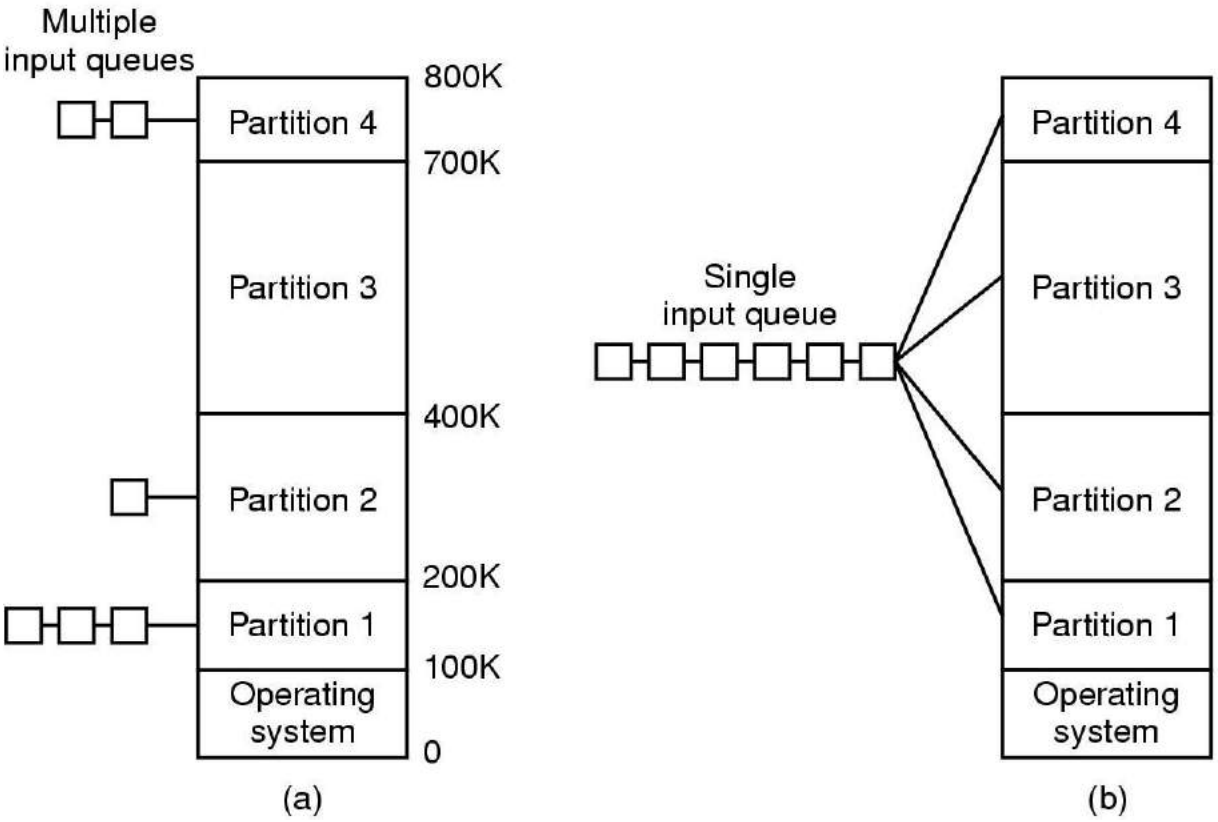
Fixed-Partition Strategies

- Memory divided into *fixed-size* regions
- Size of each region usually not equal
- MM will allocate a region to a process that *best fits* it
- Unused memory within an allocated partition called *internal fragmentation*

Fixed-Partition Example



Multiprogramming with Fixed Partitions

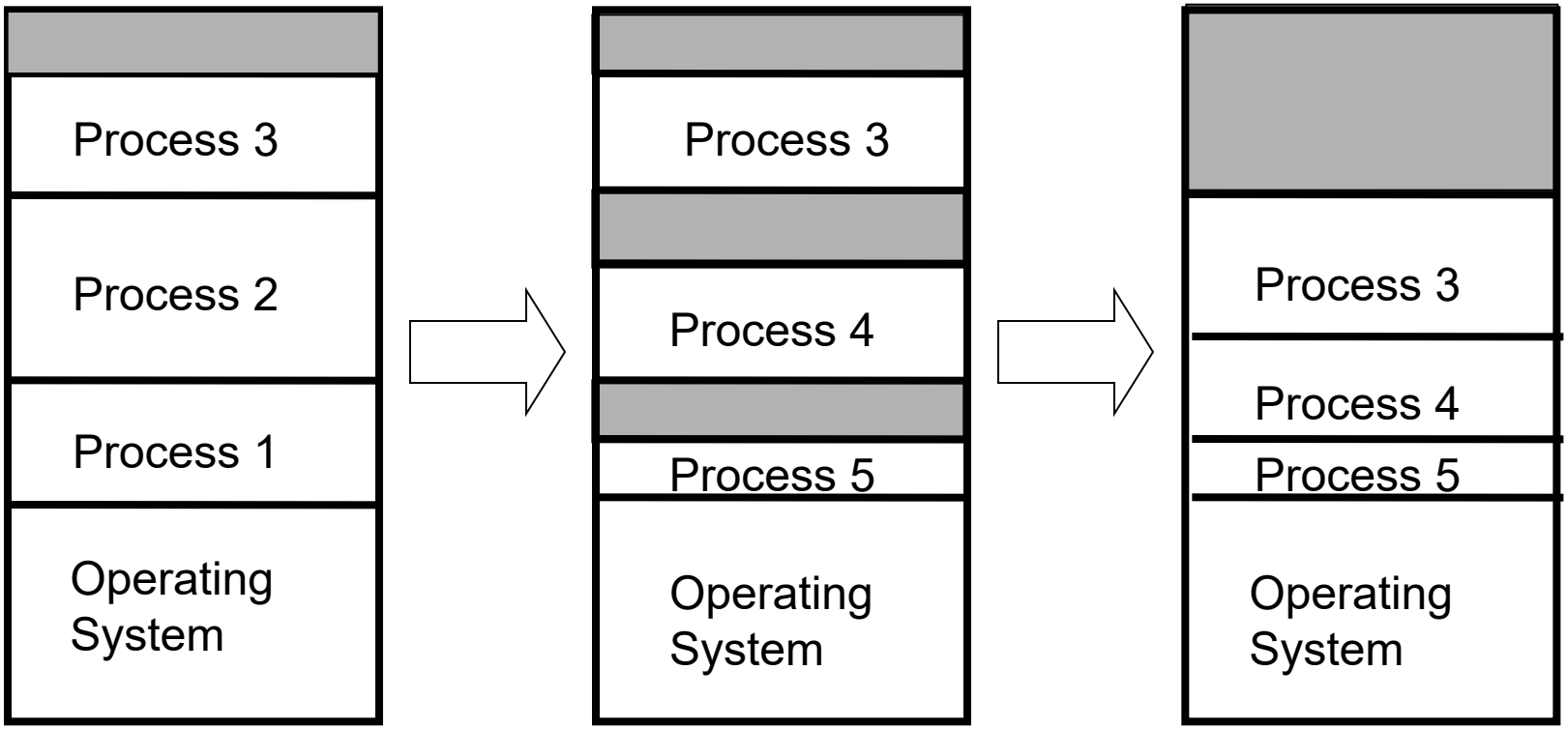


- Fixed memory partitions
  - separate input queues for each partition
  - single input queue

Variable-Partition Strategies

- MM allocates regions equal to the memory requirements of a process at any given time
- As processes die, *holes* develop in the memory
- MM inserts new processes into holes

Variable-Partition Example



More…

- Results in *External Fragmentation*
- After a while, only small processes will be able to run due to too much external fragmentation
- MM must *compact* the memory to make more space available for larger processes

Relocation

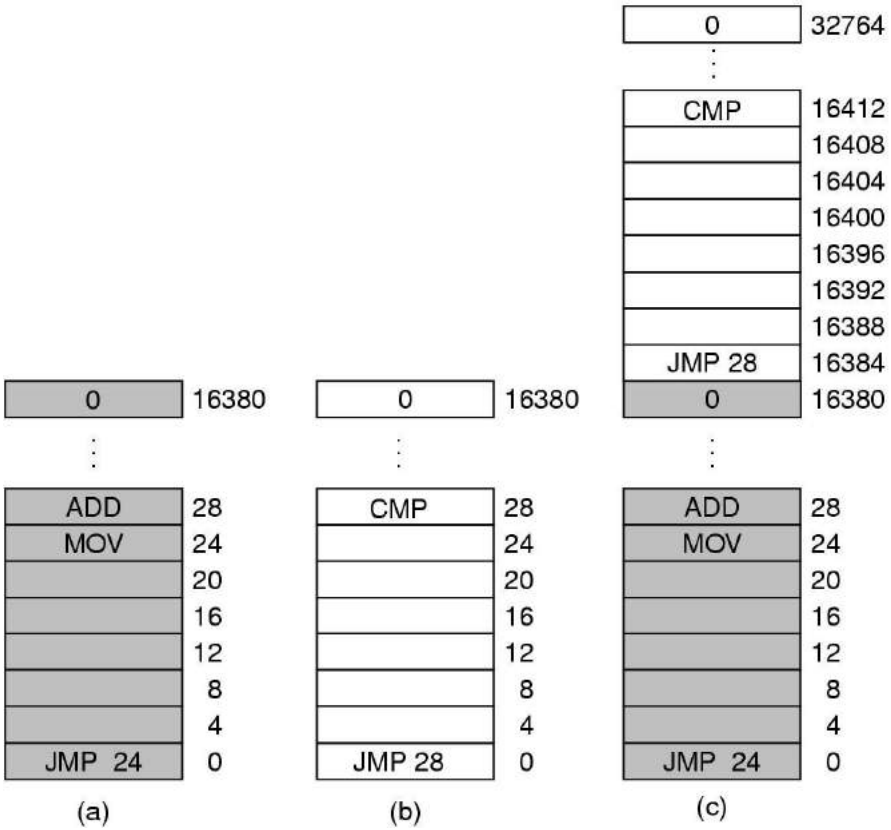
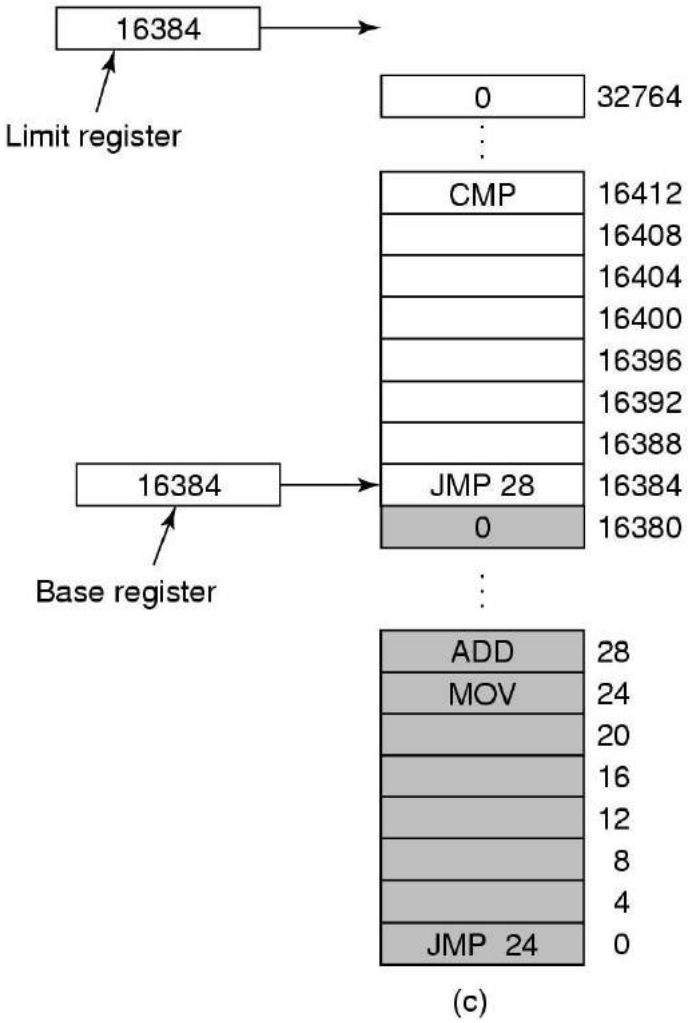


Illustration of the relocation problem.

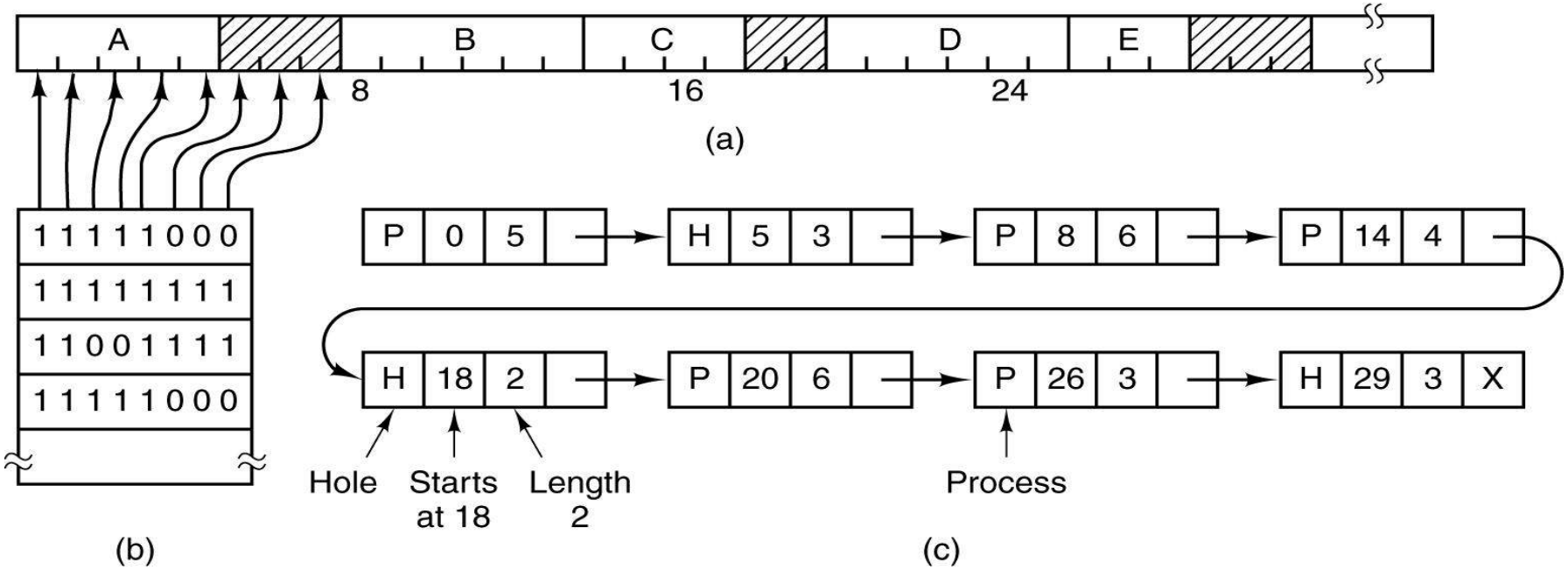
(a) (b) logical address (c) physical address

Base and Limit Registers



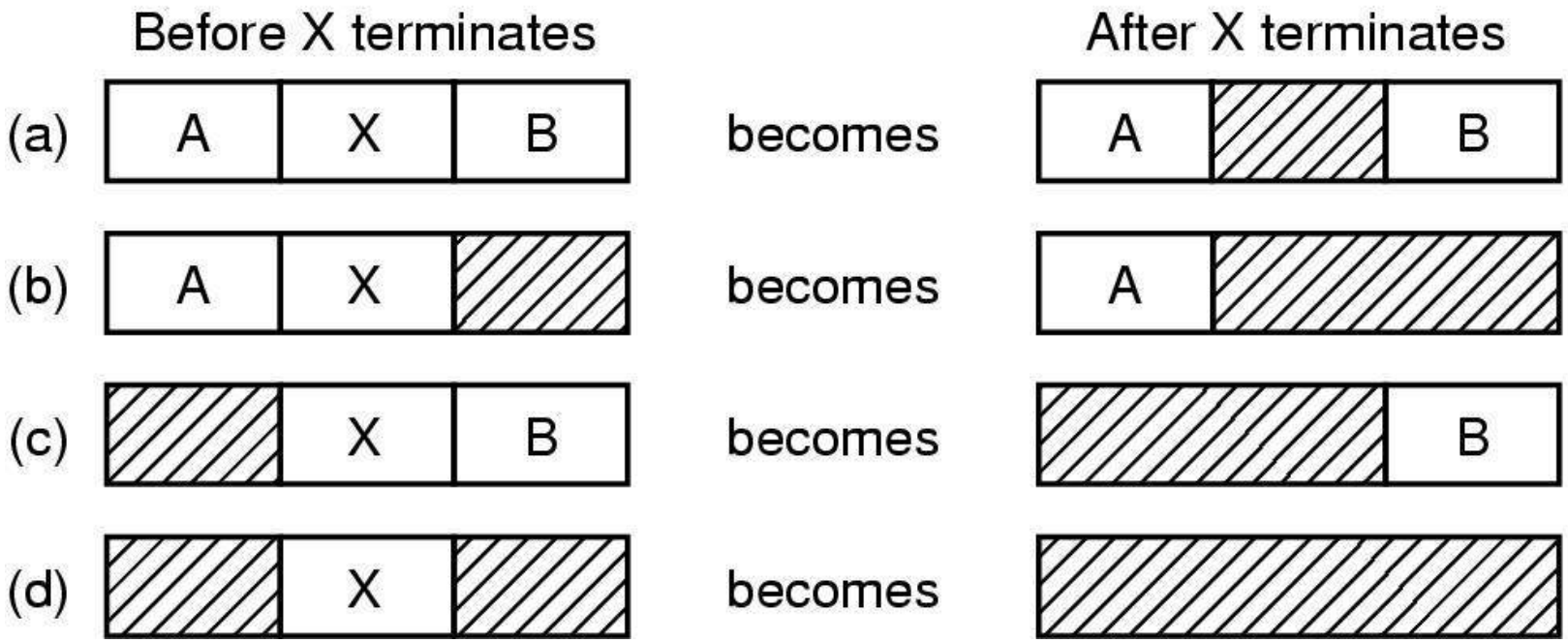
Base and limit registers can be used to give each process a separate address space.

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

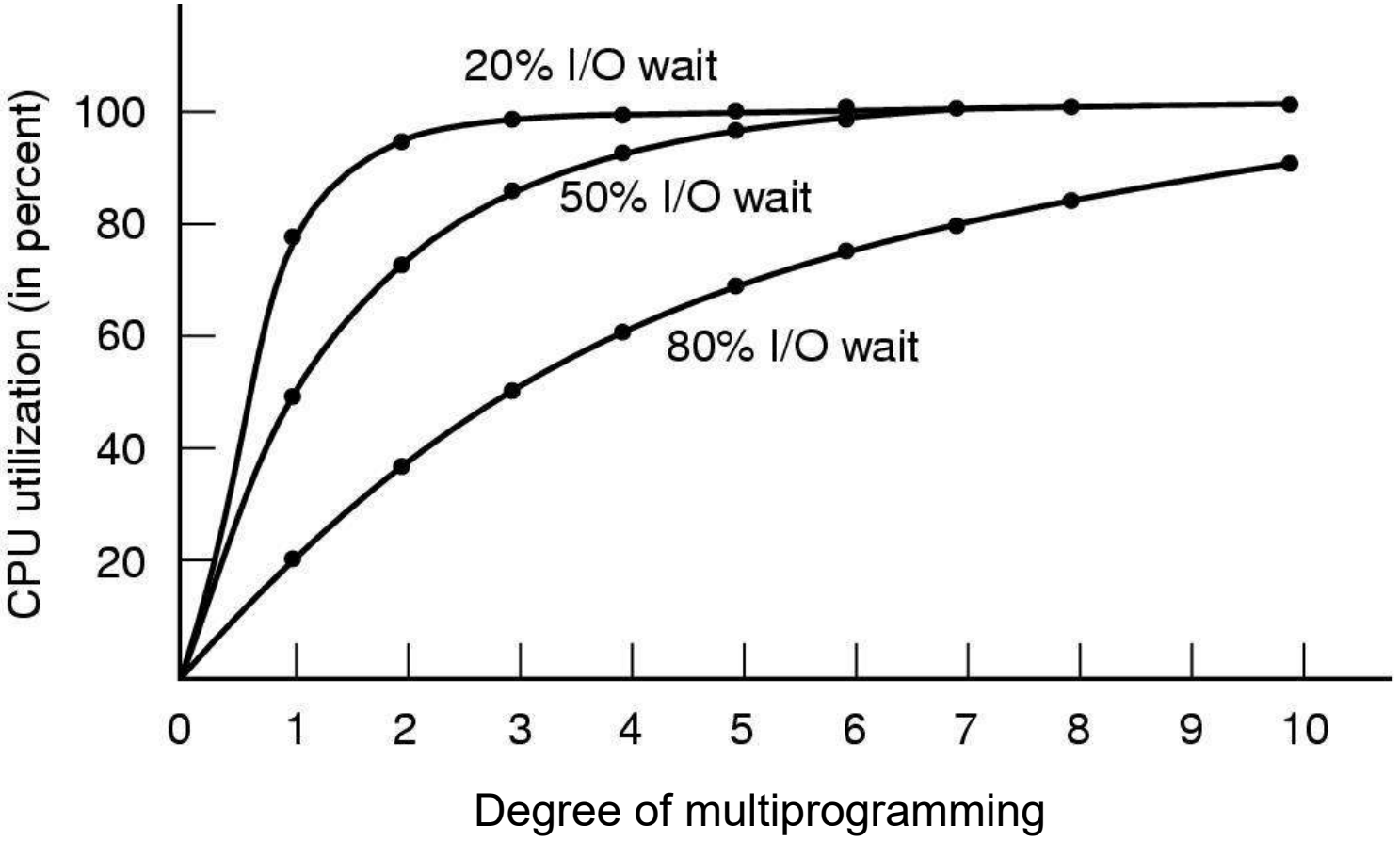


Four neighbor combinations
for the terminating process, X.

Storage Placement Strategies

- **First Fit**
  - Scan; use the first available hole whose size is sufficient to meet the need.
  - Problem: Creates average size holes; more fragementation closer to scan start.
- **Next Fit**
  - Minor variation of first fit: keep track of where last search ended, restart from there.
  - Problem: slightly worse performance than first fit.
- **Best Fit**
  - Use the hole whose size is equal to the need, or if none is equal, the smallest hole that is large enough.
  - Problem: Creates small holes that can't be used.
- **Worst Fit**
  - Use the largest available hole.
  - Problem: Gets rid of large holes, making it difficult to run large processes.
- **Quick Fit**
  - maintains separate lists for some of the more common sizes requested.
  - When a request comes for placement, it finds the closest fit.
  - This is a very fast scheme, but a merge is expensive. If merge is not done, memory will quickly fragment into a large number of holes.

Modeling Multiprogramming



CPU utilization as a function of number of processes in memory

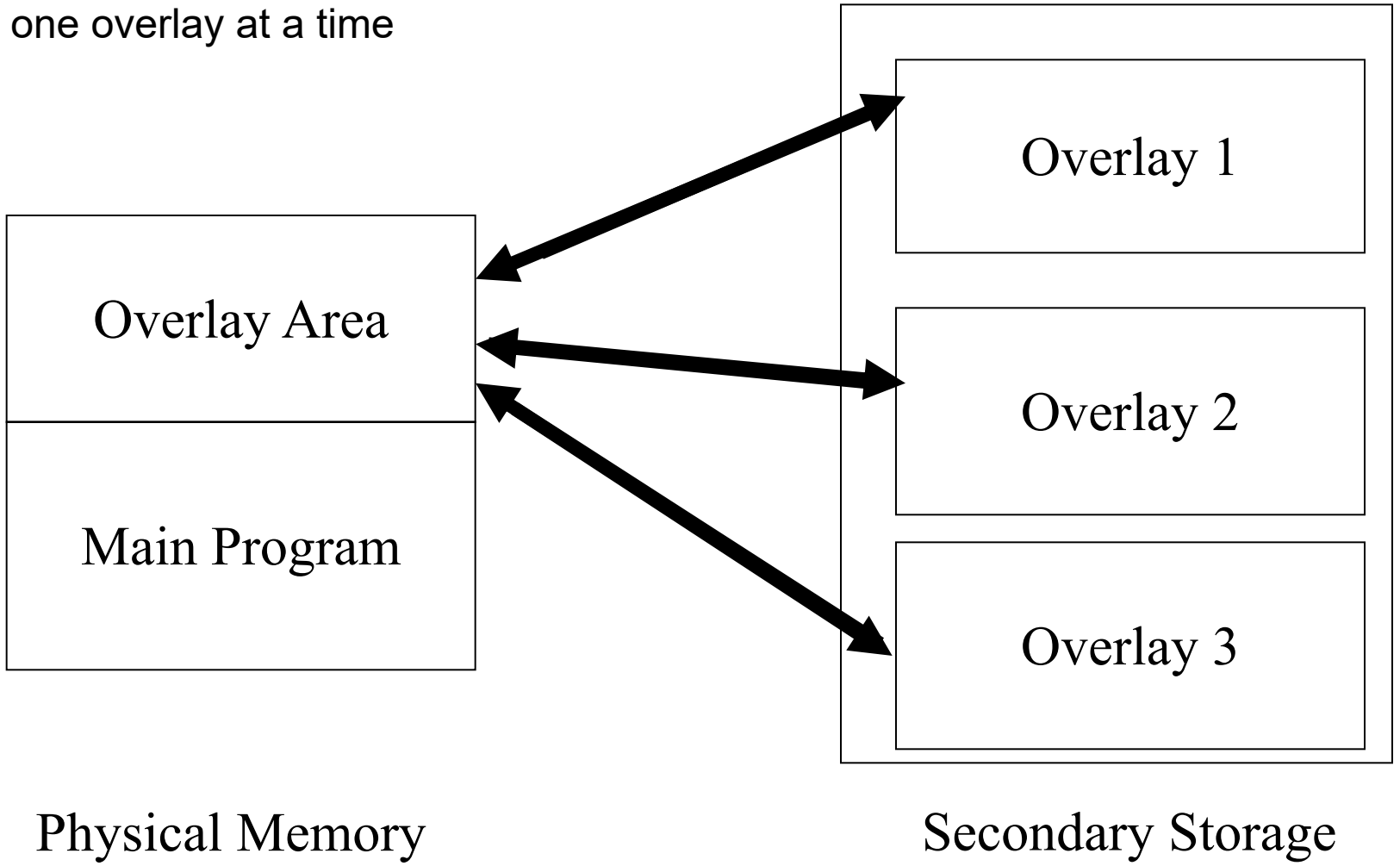
Chapter 3

Memory Management

Virtual Memory

Overlaying

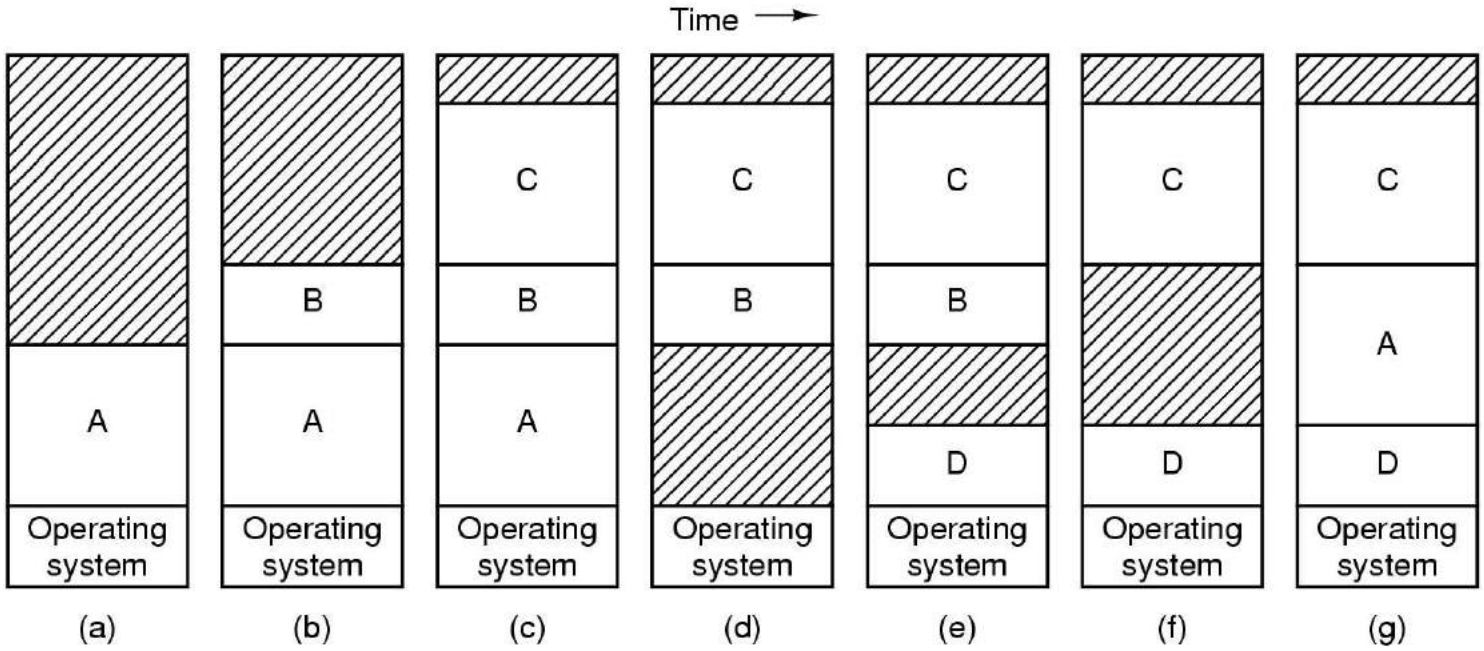
Used when process memory requirement exceeds the physical memory space
Split process space into multiple, sequentially runnable parts
Load one overlay at a time



Swapping（1）

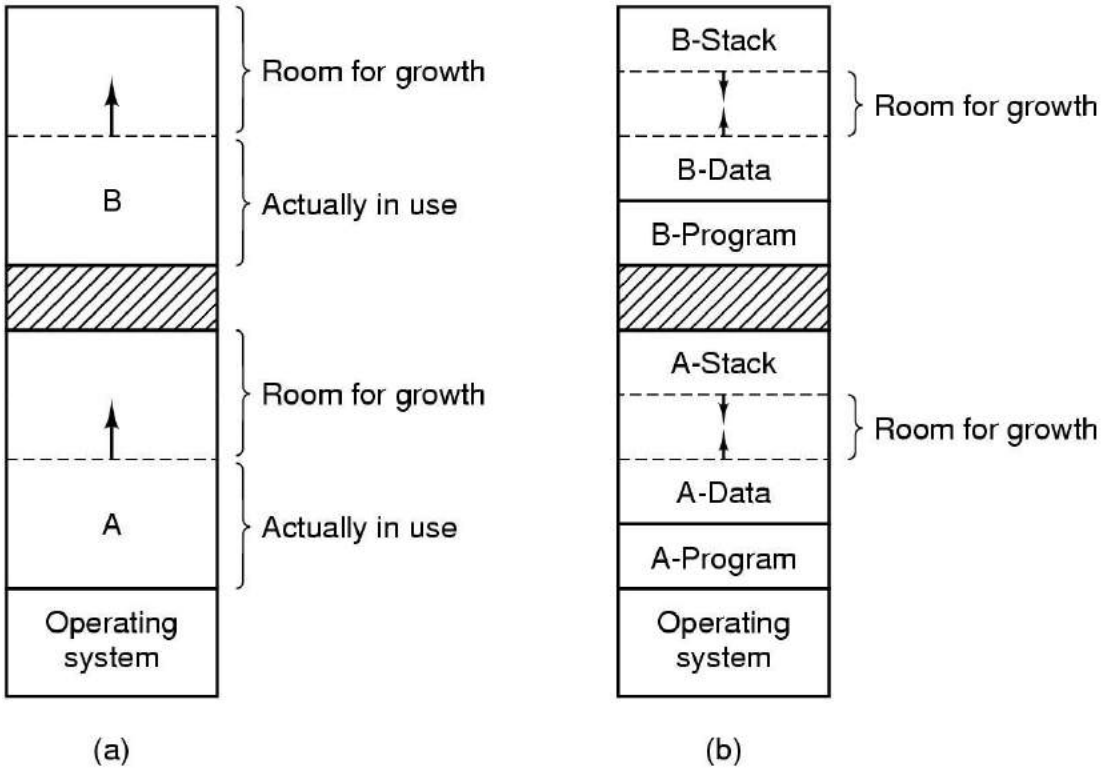
- Comes from the basis that when a process is blocked, it does not need to be in memory
- Thus, it is possible to save a process' entire address space to disk
- Saving to a "swap file" or a "swap partition"

Swapping (2)



- Memory allocation changes as
  - processes come into memory
  - leave memory
- Shaded regions are unused memory

Swapping (3)



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

Compaction (Similar to Garbage Collection)

- Assumes programs are all relocatable (how supported?)
- Processes must be suspended during compaction
- Needed only when fragmentation gets very bad

Memory Management Problems

- Fixed partitions suffer from internal fragmentation
- Variable partitions suffer from external fragmentation
- Compaction suffers from overhead
- Overlays are painful to program efficiently
- Swapping requires writing to disk sectors

Alternative Approach: Virtual Memory

- Provide user with virtual memory that is as big as user needs
- Store virtual memory on disk
- Store in real memory those parts of virtual memory currently under use
- Load and store cached virtual memory without user program intervention (“transparently”)

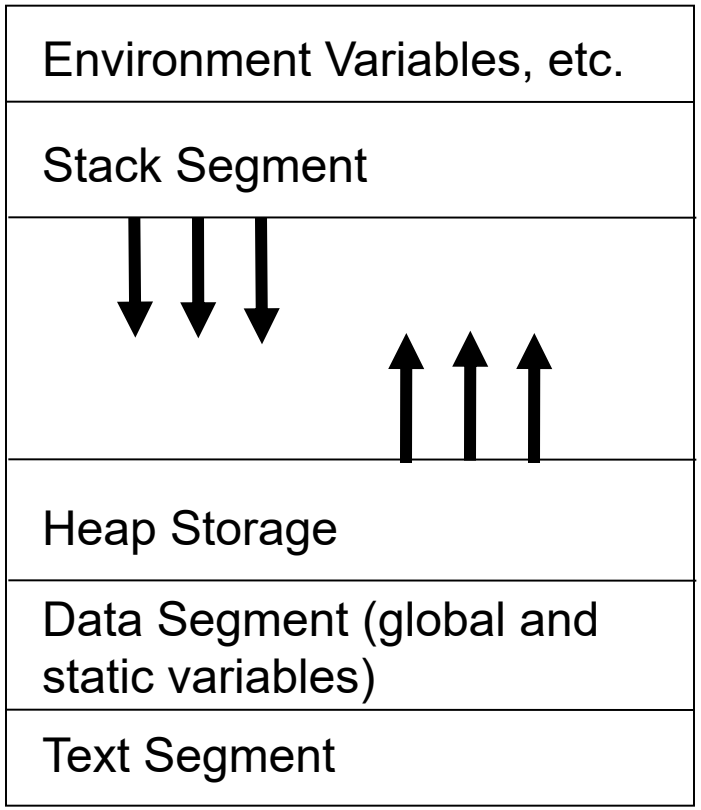
Virtual Memory

- Comes from the basis that all of a process' address space is not needed at once
- Thus, chop up the address space into smaller parts and only load the parts that are needed
- These parts need not be contiguous in memory!

Benefits of Virtual Memory

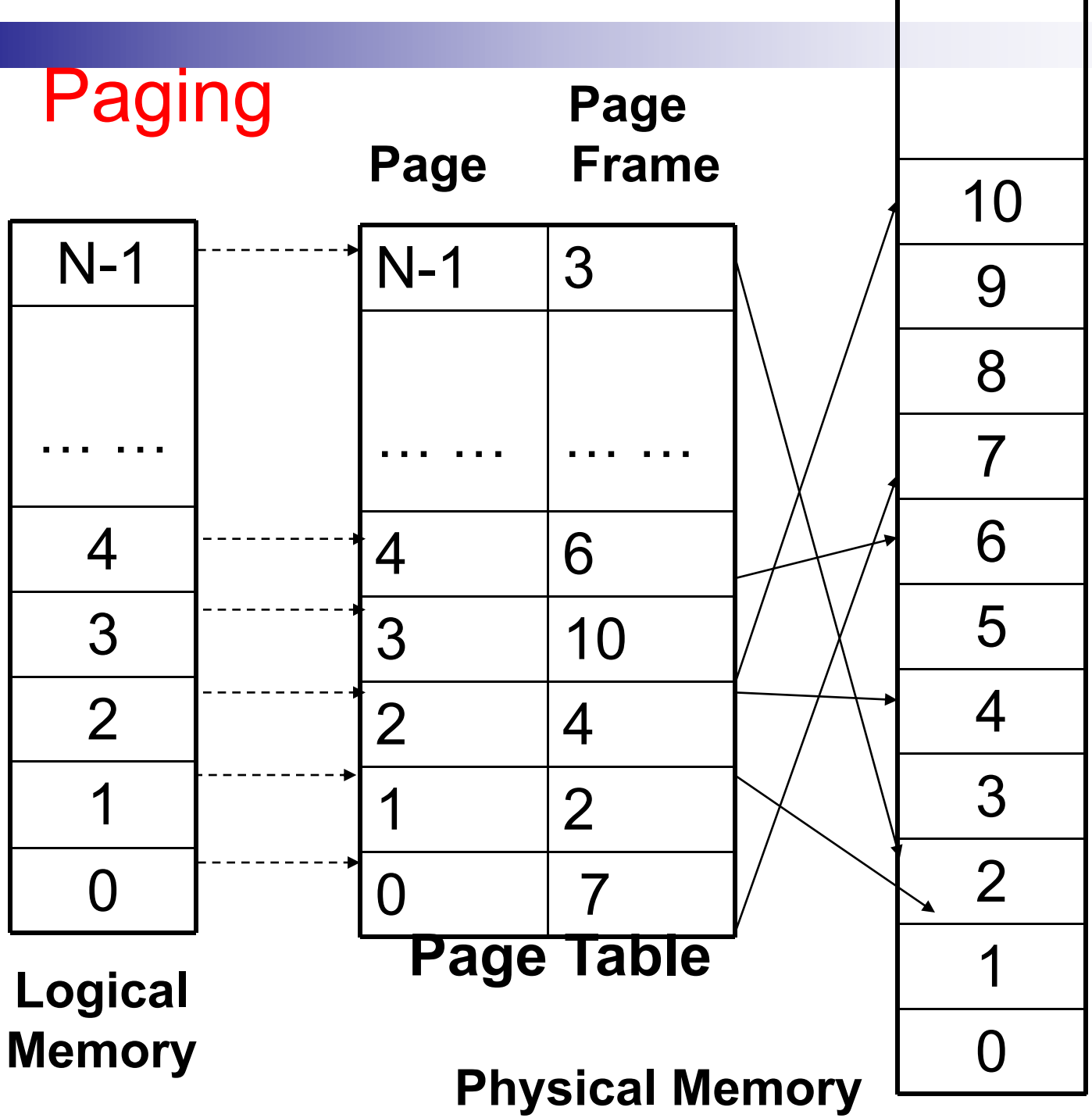
- Use secondary storage($)
  - Extend DRAM($$$) with reasonable performance
- Protection
  - Processes do not step on each other
- Convenience
  - Flat address space
  - Processes have the same view of the world
  - Load and store cached virtual memory without user program intervention
- Reduce fragmentation:
  - make cacheable units all the same size (**page**=allocation unit)
- Remove memory deadlock possibilities:
  - permit pre-emption of real memory

Process Memory Layout



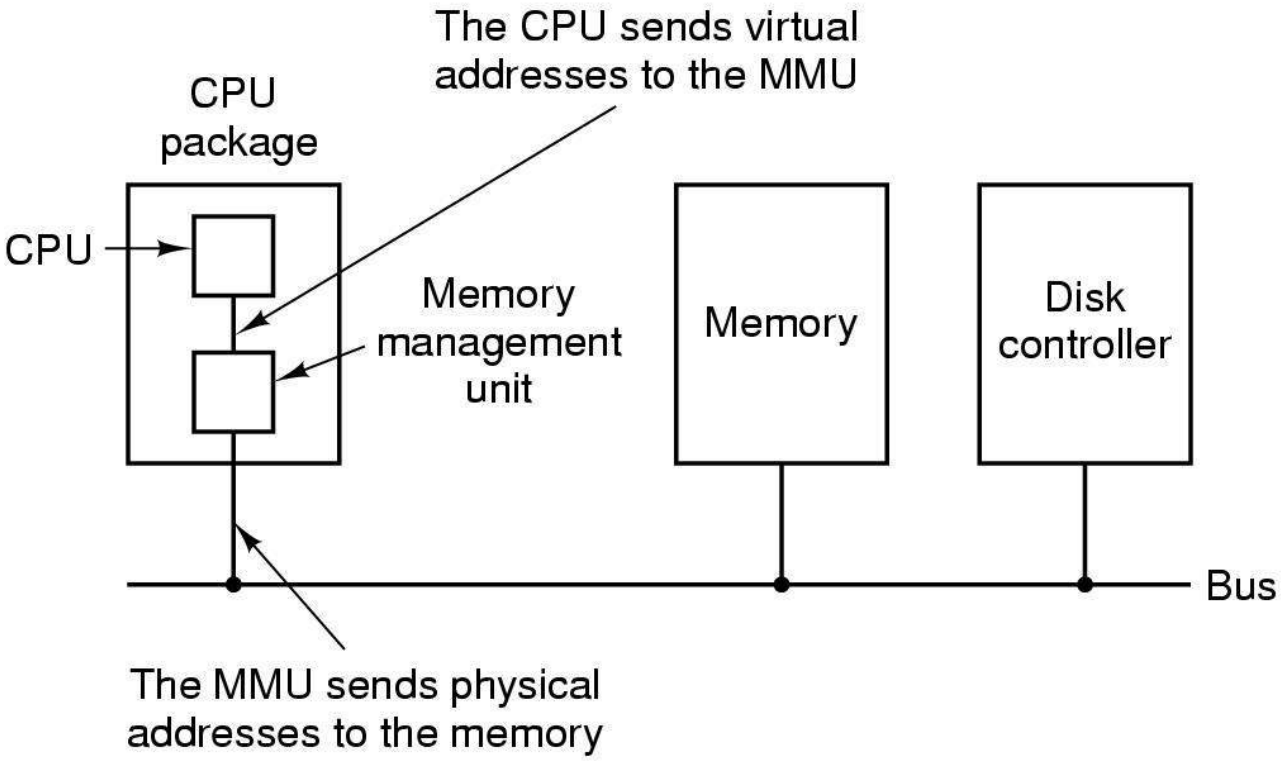
- Allocates more memory than needed at first
- Heap grows towards stack for dynamic memory allocation
- Stack grows towards heap when automatic variables are created

Paging


Page
Page Frame
N-1
N-1
3
... ...
... ...
... ...
4
4
6
3
3
10
2
2
4
1
1
2
0
0
7
Page Table
Logical Memory
10
9
8
7
6
5
4
3
2
1
0
Physical Memory

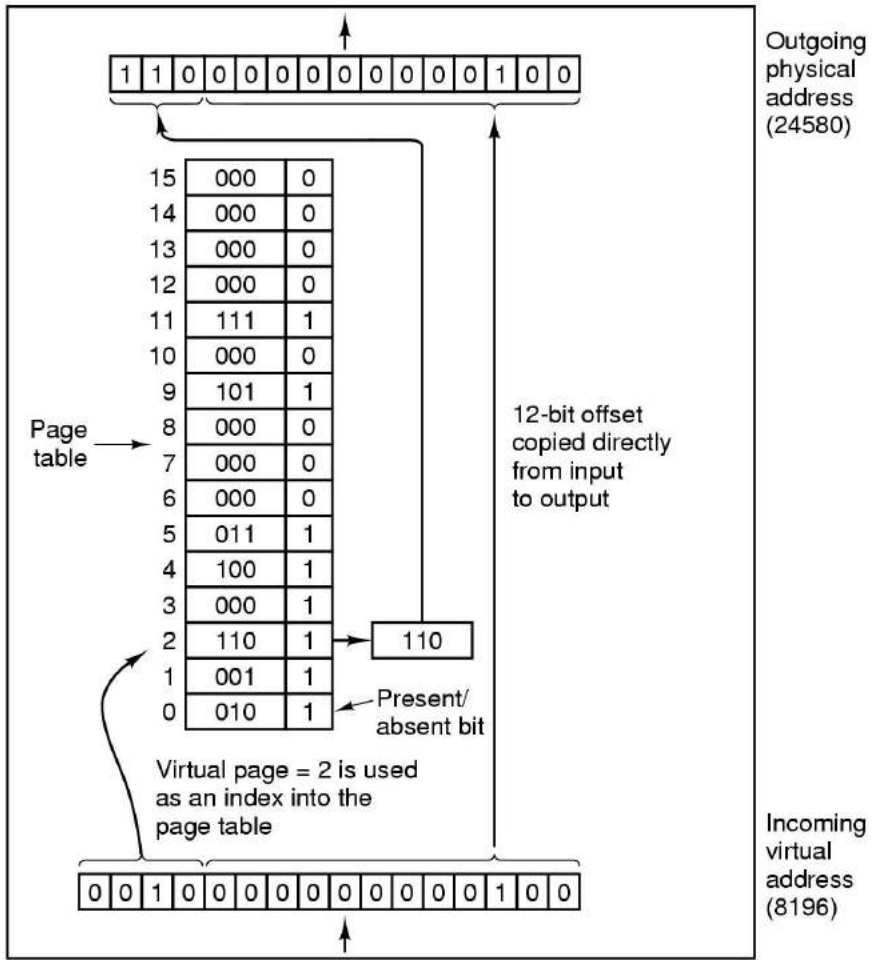
Virtual Memory
Paging

Move REG, 1000



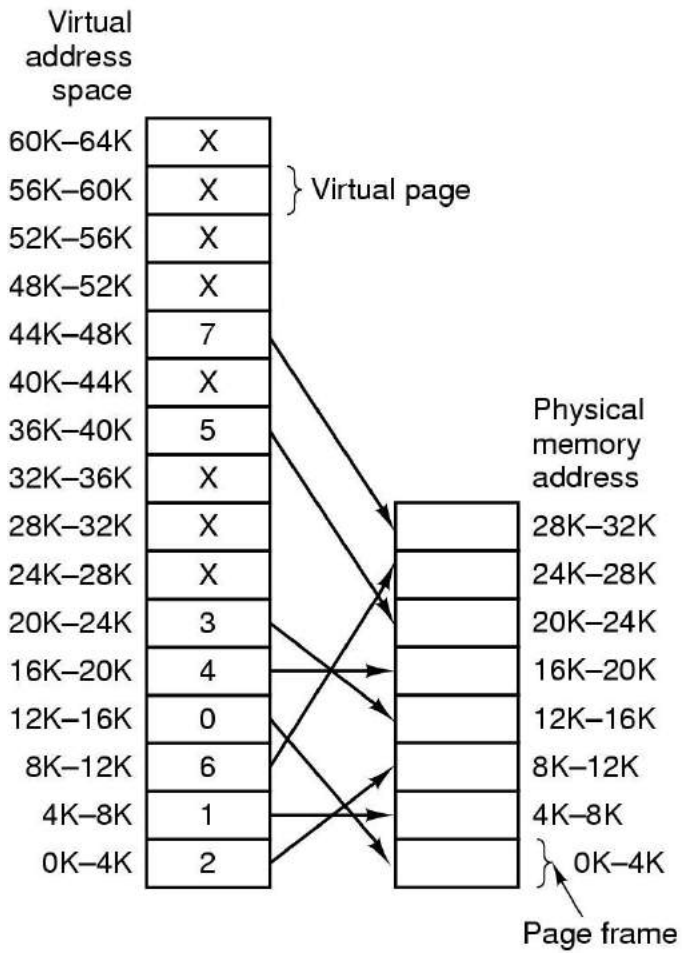
The position and function of the MMU

Paging (cont)



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

Paging



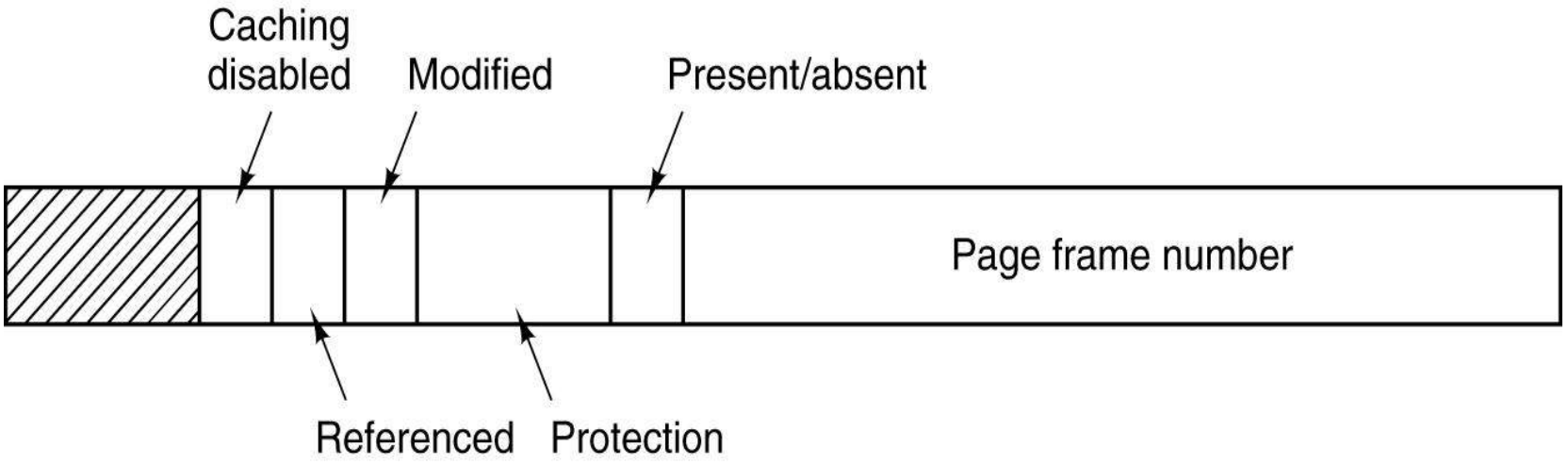
Move REG, 0

Move REG, 8192

Move REG, 20500 ?

Relation between virtual addresses and physical memory addresses given by page table.

Structure of Page Table Entry



A typical page table entry.

Speeding Up Paging

Paging implementation issues:

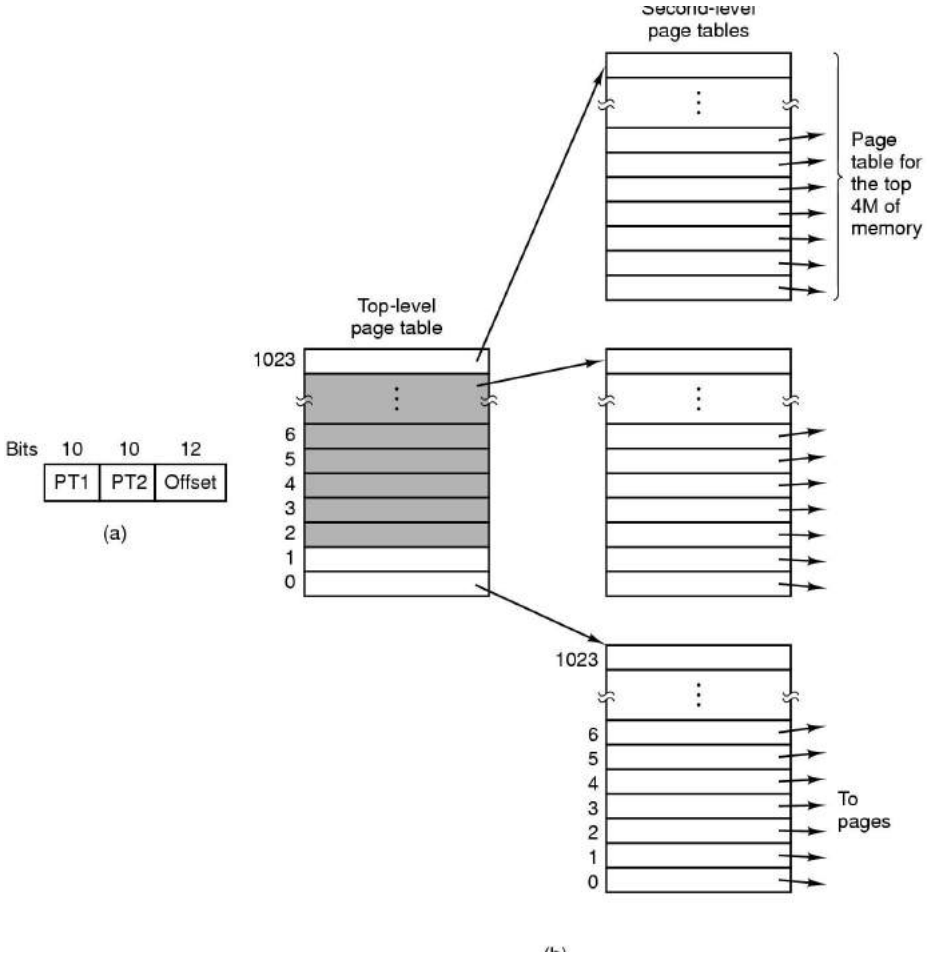
- The mapping from virtual address to physical address must be fast.
- If the virtual address space is large, the page table will be large.

Translation Lookaside Buffers

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

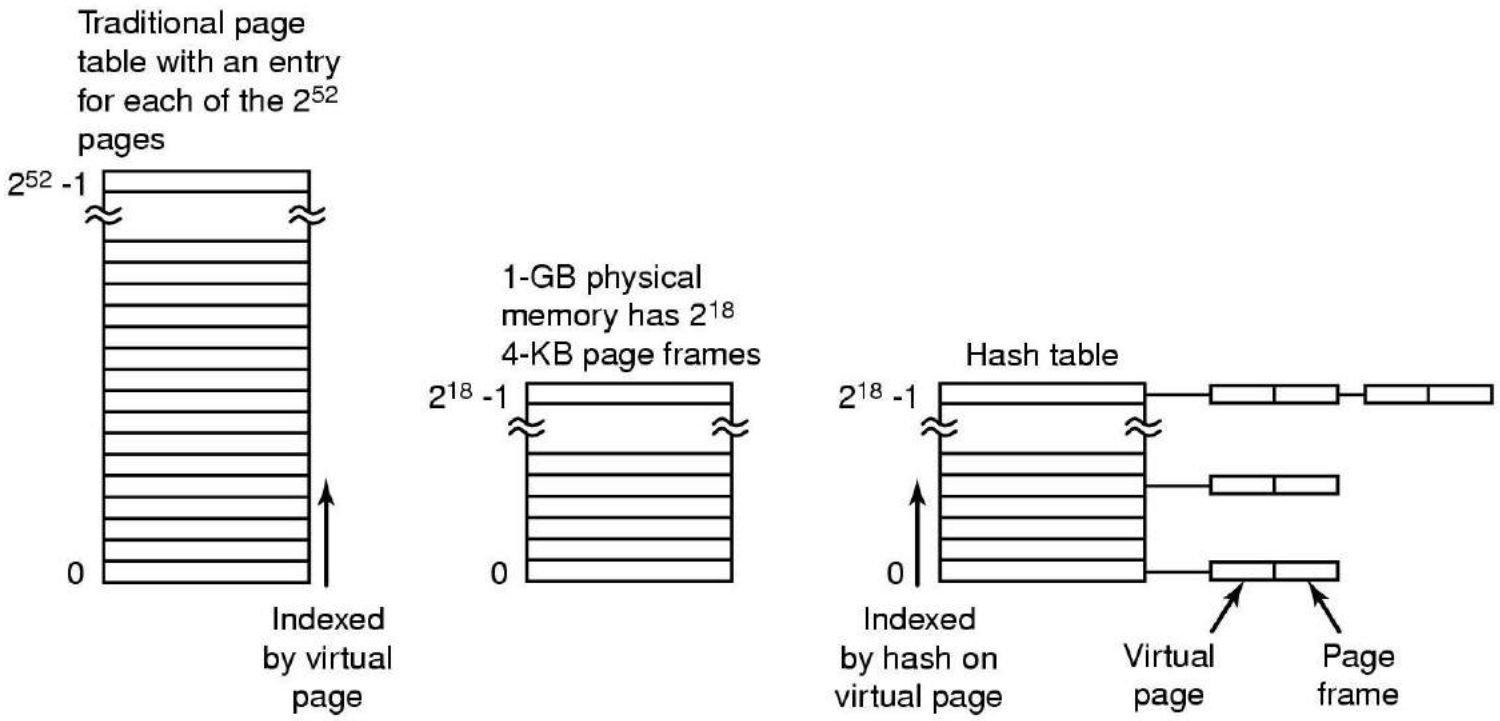
A TLB to speed up paging.

Multilevel Page Tables



(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.

Virtual Memory Usage

- Virtual memory is used in most modern operating systems:
  - Windows NT/2000/XP uses one or more “page files” to swap pages
  - Linux uses a hard disk partition (“swap partition”) to swap to

Pros/Cons

- Since only the necessary parts of the process are loaded, processes load faster and it allows much better memory utilization
- Needs lots of extra hardware to accomplish the job (efficiently)
- In some cases too much paging (i.e. “thrashing”) can occur, which is *very* slow

Page Fault Handling (1)

- The hardware traps to the kernel, saving the program counter on the stack.
- An assembly code routine is started to save the general registers and other volatile information.
- The operating system discovers that a page fault has occurred, and tries to discover which virtual page is needed.
- 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)

- If the page frame selected is dirty, the page is scheduled for transfer to the disk, and a context switch takes place.
- 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.
- When disk interrupt indicates page has arrived, page tables updated to reflect position, frame marked as being in normal state.

Page Fault Handling (3)

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

作业：

- P139　2, 3, 4, 5

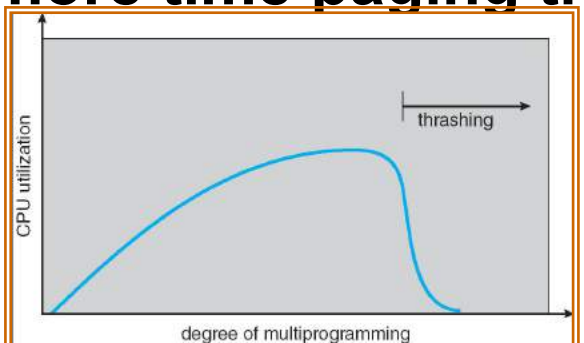
Chapter 3

Memory Management

Page Replacement

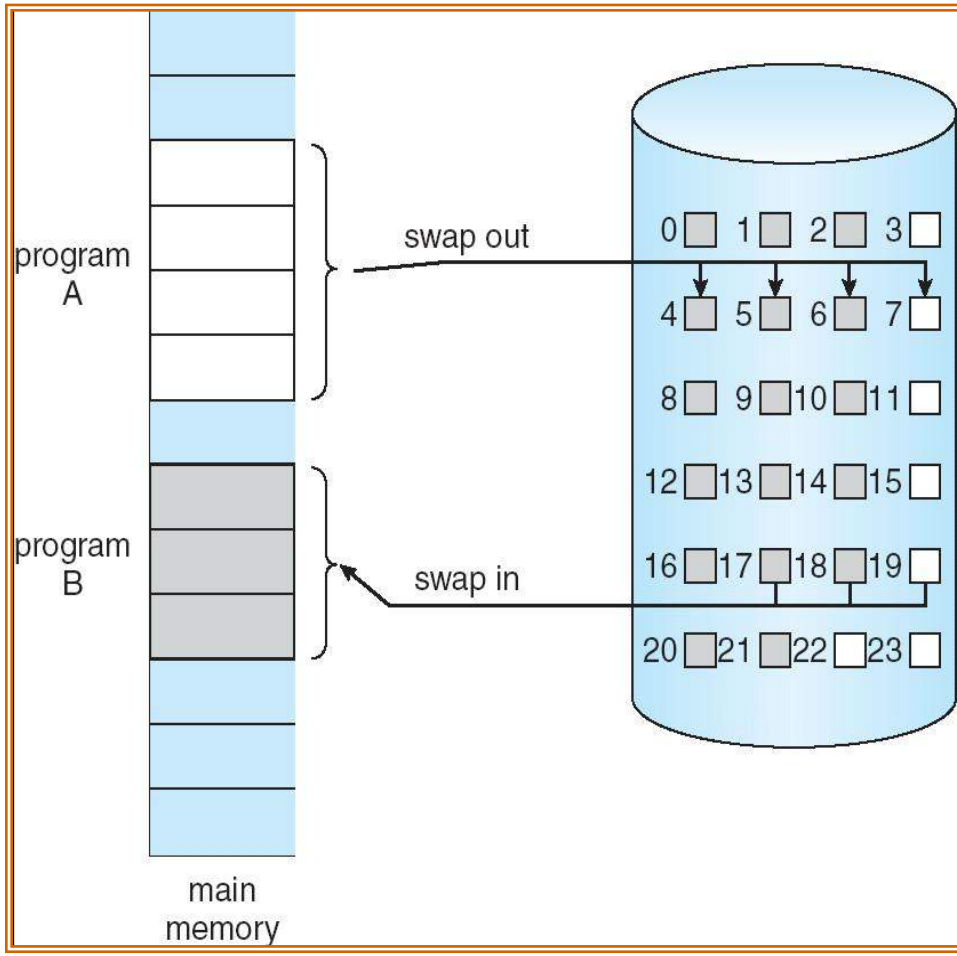
Thrashing

- **If a process does not have "enough" pages, the page-fault rate is very high. This leads to:**
  - **low CPU utilization**
  - **OS scheduler thinks that it needs to increase the degree of multiprogramming → new processes are added → even less pages for each process …**
- **Thrashing ≡ a process is busy swapping pages in and out**
  - **→ Spending more time paging than executing.**



Demand Paging

- **Demand paging: pages are only loaded into memory when they are demanded during execution**
  - **Less I/O needed**
  - **Less memory needed**
  - **Higher degree of multiprogramming**
  - **Faster response**
- **Pager (lazy swapper) never swaps a page into memory unless that page will be needed.**
- **An extreme case: Pure demand paging starts a process with no pages in memory …**



Transfer of a Paged Memory to Contiguous Disk Space

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 over written
- Better not to choose an often used page
  - will probably need to be brought back in soon

Page Replacement Algorithms

- Random page replacement
- 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

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

Optimal Example

12 references,
7 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	B	A
A	no	D	B	A
B	no	D	B	A
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Not Recently Used Page Replacement Algorithm（NRU）

- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified

  0：not referenced, not modified

  1：not referenced, modified

  2：referenced, not modified

  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

FIFO

12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A		
B	yes	B	A	
C	yes	C	B	A
D	yes	D	C	B
A	yes	A	D	C
B	yes	B	A	D
E	yes	E	B	A
A	no	E	B	A
B	no	E	B	A
C	yes	C	E	B
D	yes	D	C	E
E	no	D	C	E

Belady's Anomaly (for FIFO)

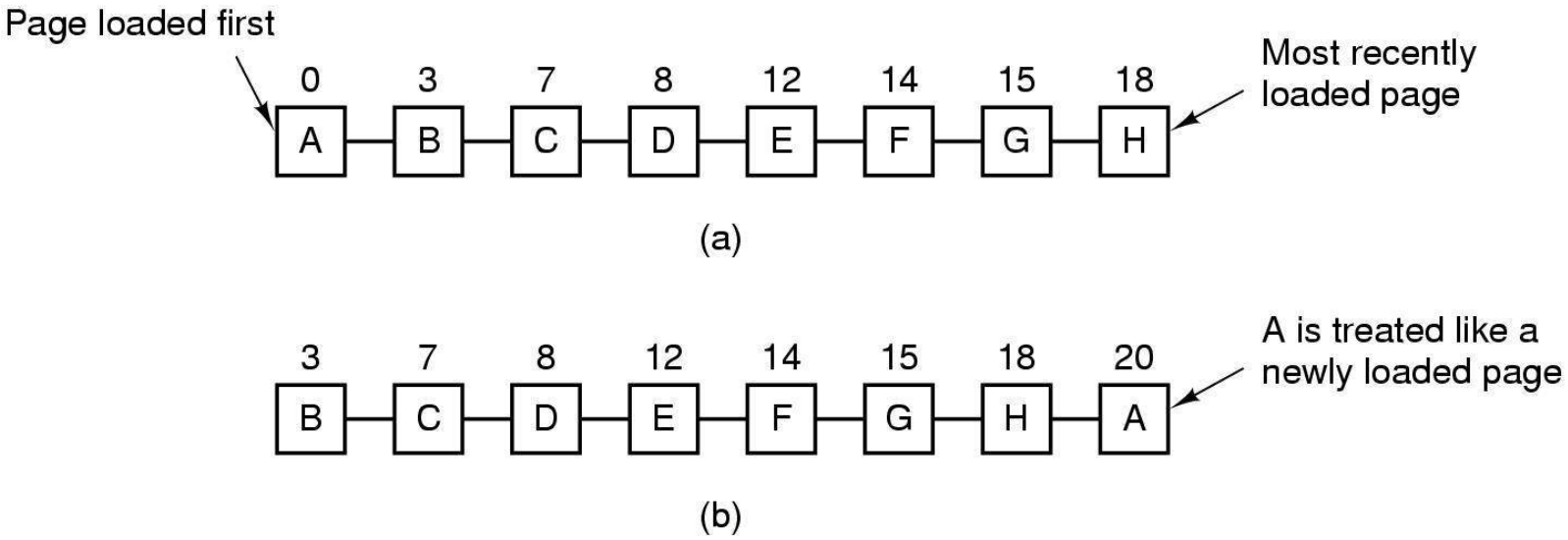
Same reference string as with 3 frames (9 page faults).

12 references,
10 faults

Page Refs	4 Page Frames Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	D	C	B	A
B	no	D	C	B	A
E	yes	E	D	C	B
A	yes	A	E	D	C
B	yes	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

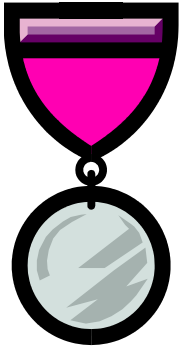
Belady's Anomaly for FIFO: (Sometimes) as the number of page frames increase, so does the fault rate.

Second Chance Algorithm



Operation of second chance.
(a) Pages sorted in FIFO order.
(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.

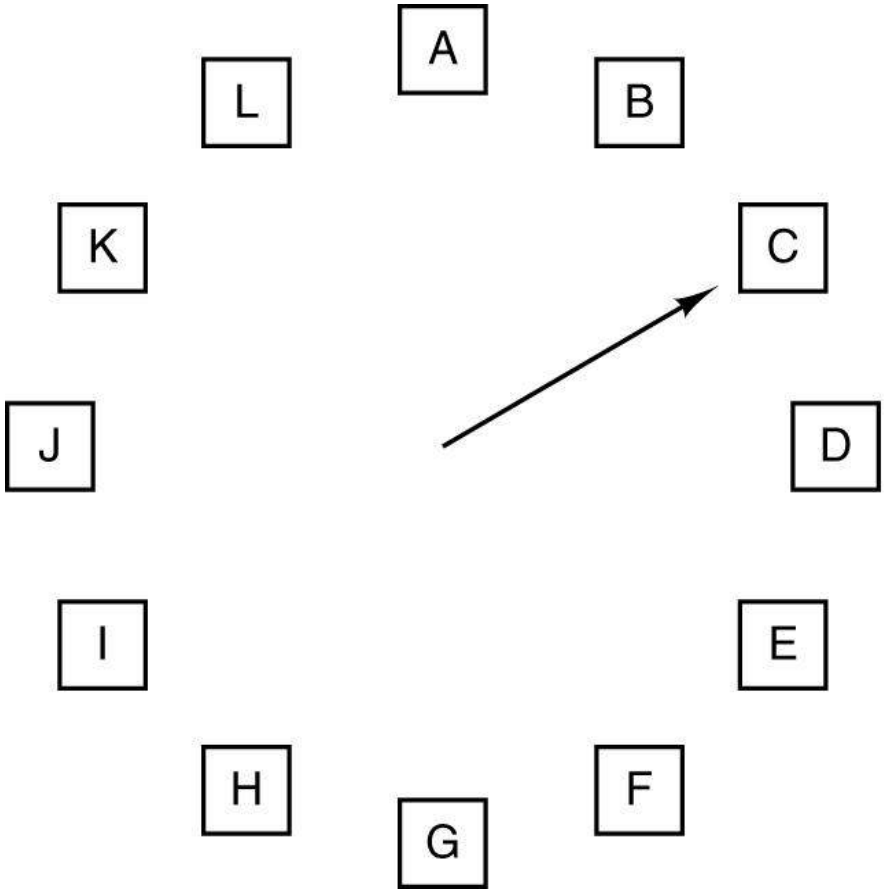
Second Chance Example



12 references,
9 faults

Page Refs	3 Page Frames			
	Fault?	Page Contents		
A	yes	A*		
B	yes	B*	A*	
C	yes	C*	B*	A*
D	yes	D*	C	B
A	yes	A*	D*	C
B	yes	B*	A*	D*
E	yes	E*	B	A
A	no	E*	B	A*
B	no	E*	B*	A*
C	yes	C*	E	B
D	yes	D*	C*	E
E	no	D*	C*	E*

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

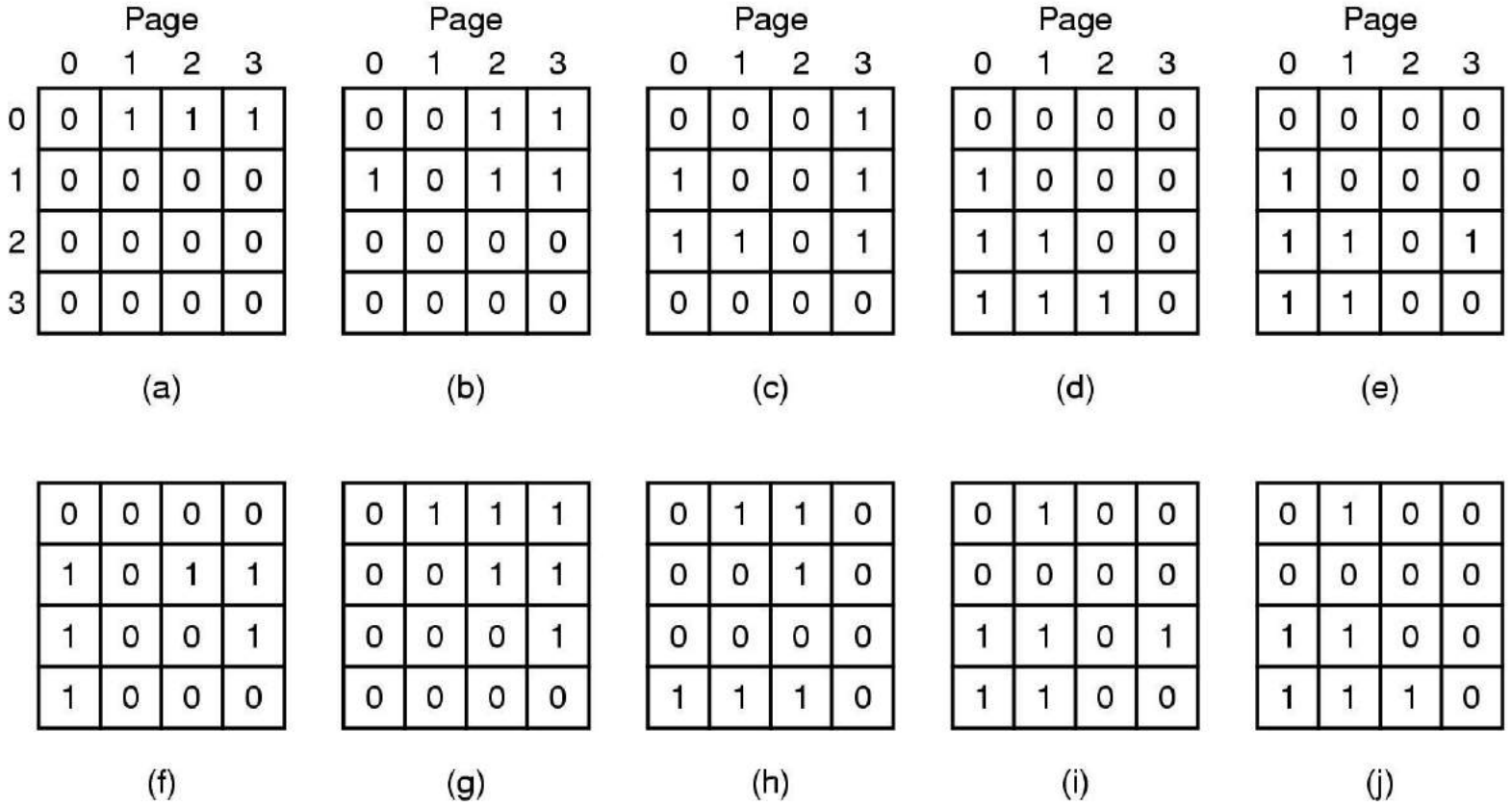
R = 1: Clear R and advance hand

The clock page replacement algorithm.

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
  - 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
  - periodically zero the counter

LRU Page Replacement Algorithm

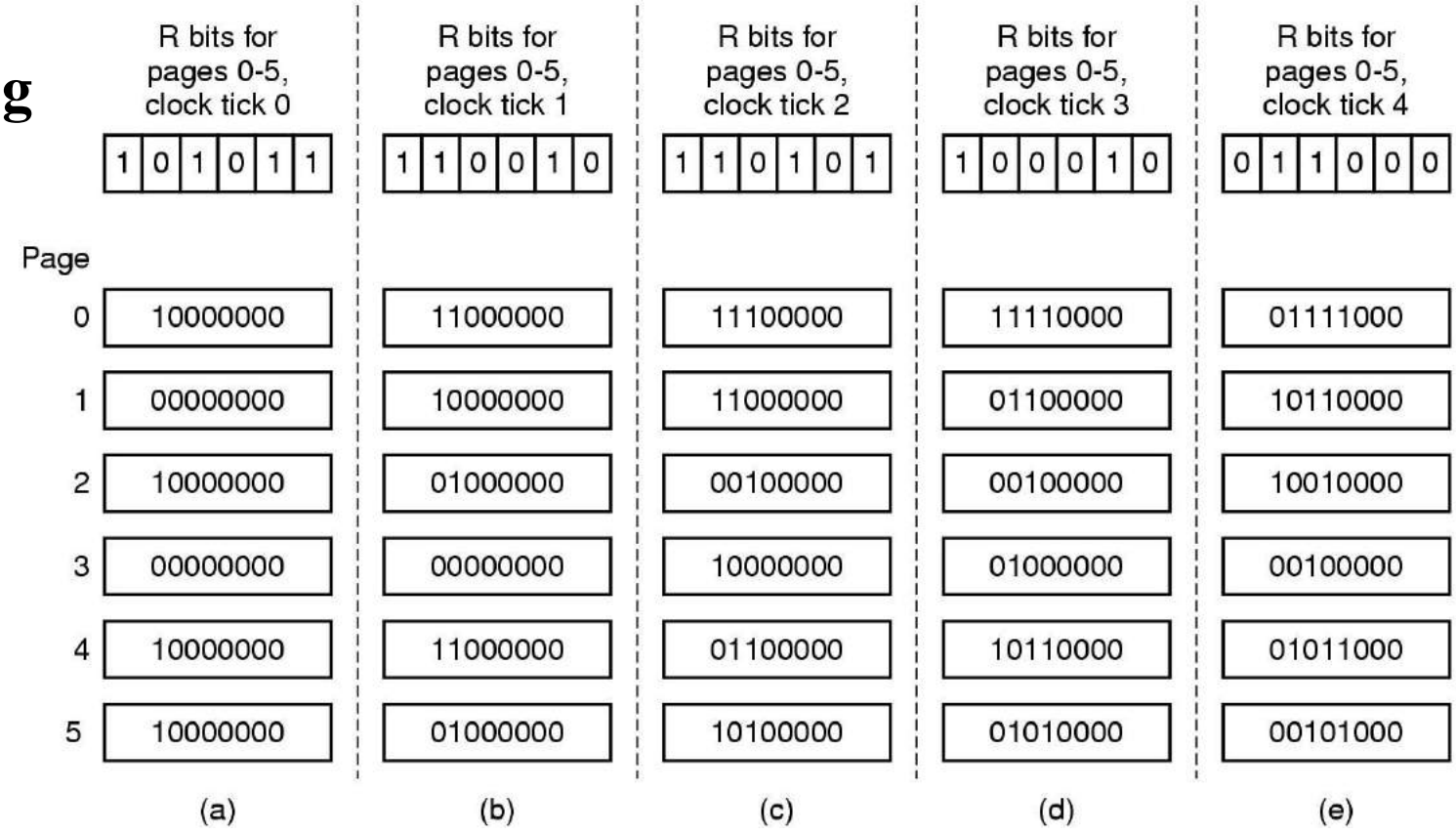


LRU using a matrix when pages are referenced in the order 0, 1, 2, 3, 2, 1, 0, 3, 2, 3.

Simulating LRU in Software

NFU

Aging



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).

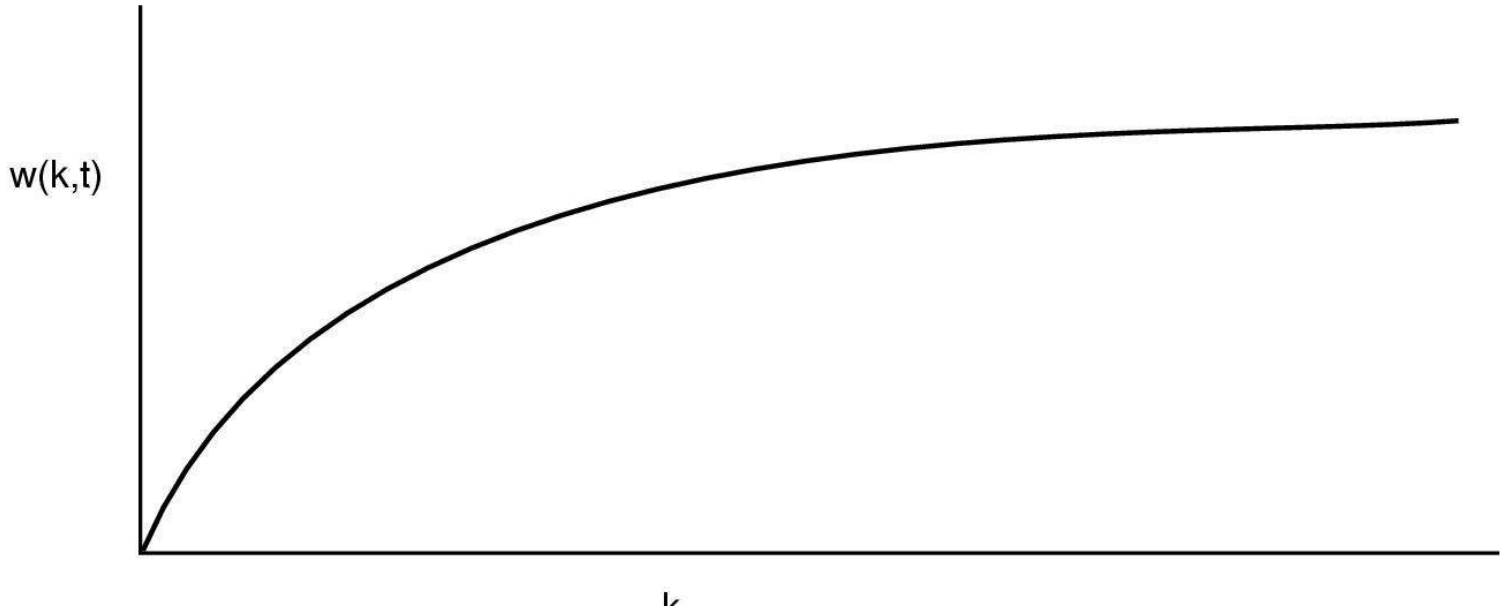
LRU and Anomalies

Anomalies cannot occur.

12 references, 8 faults

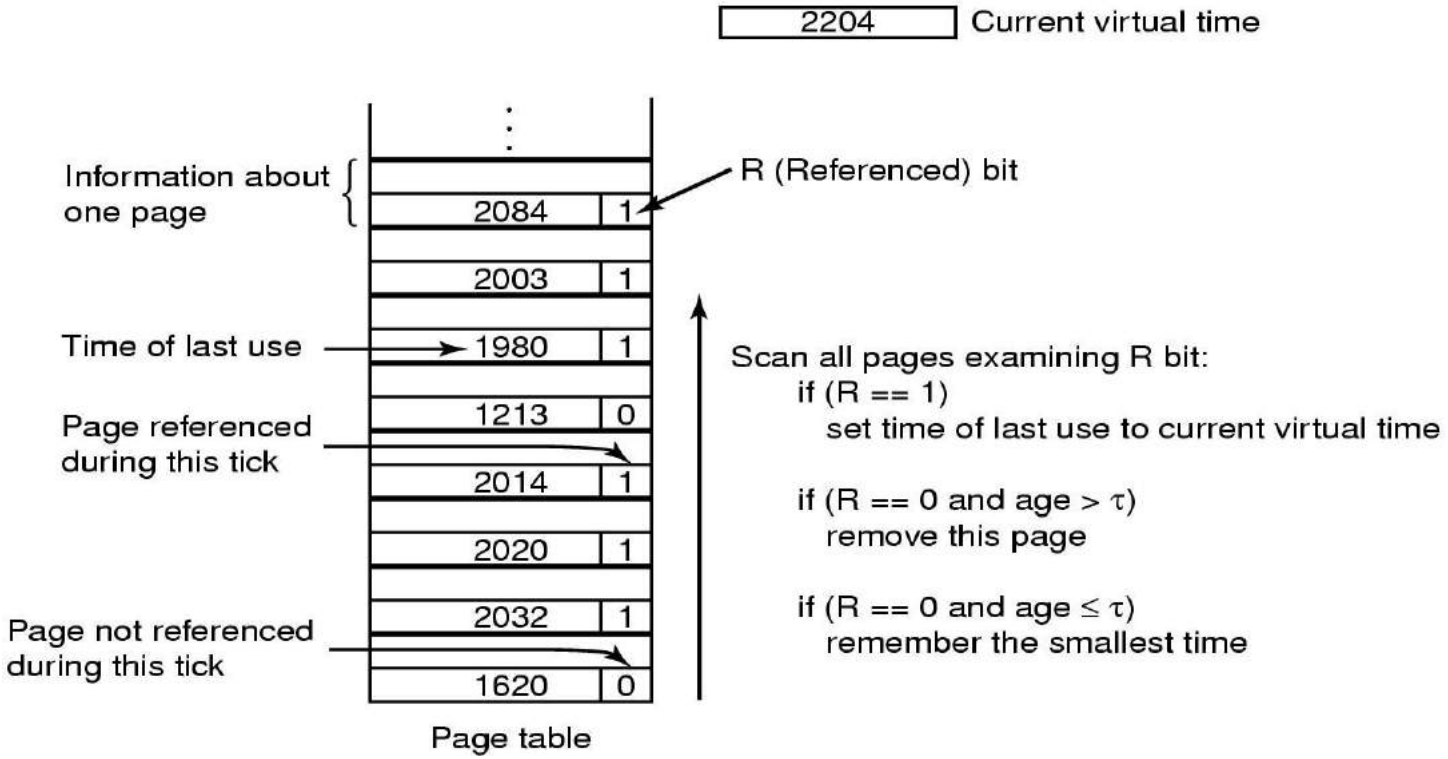
Page Refs	4 Page Frames				
	Fault?	Page Contents			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
A	no	A	D	C	B
B	no	B	A	D	C
E	yes	E	B	A	D
A	no	A	E	B	D
B	no	B	A	E	D
C	yes	C	B	A	E
D	yes	D	C	B	A
E	yes	E	D	C	B

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 Replacement (2)



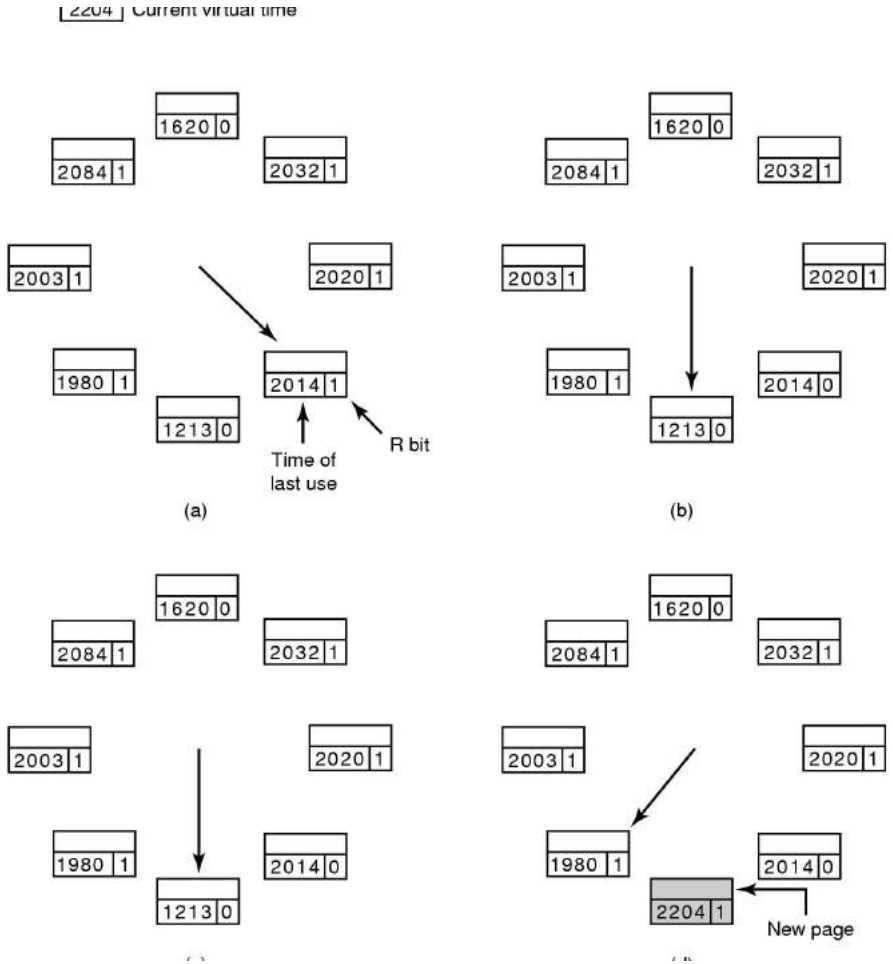
The working set algorithm.

The WSClock Page Replacement Algorithm (1)

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

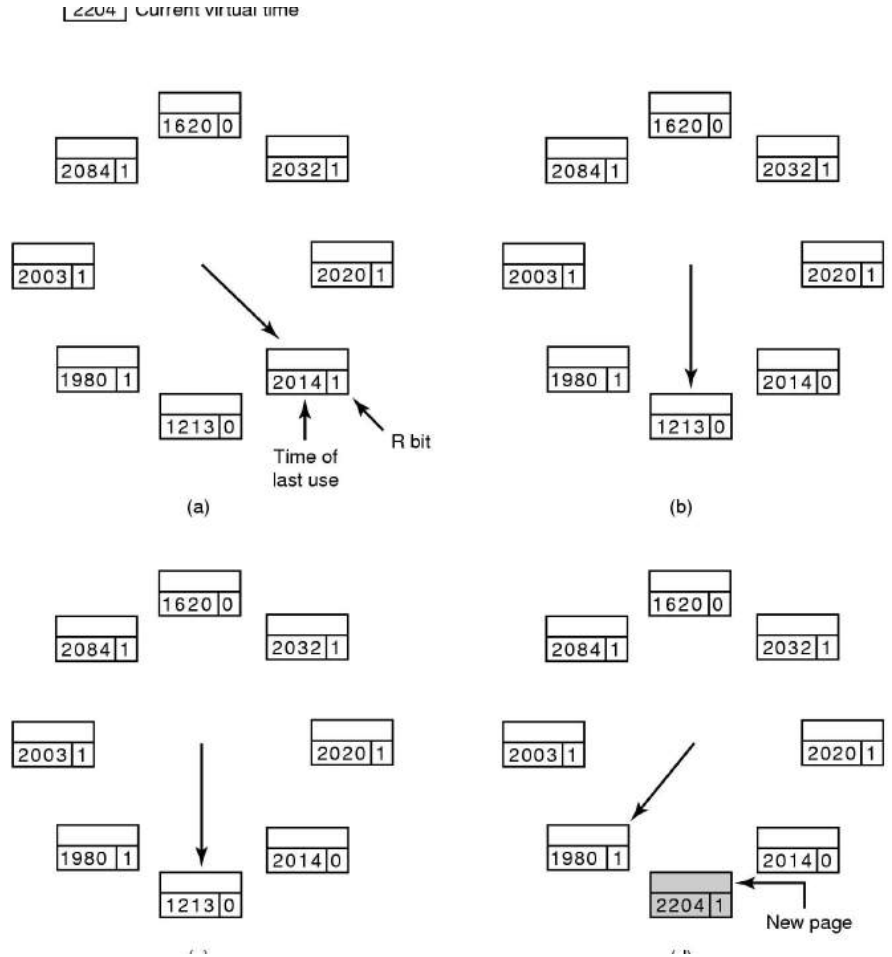
- At least one write has been scheduled.
- No writes have been scheduled.

The WSClock Page Replacement Algorithm (2)



Operation of the WSClock algorithm. (a) and (b) give an example of what happens when R = 1.

The WSClock Page Replacement Algorithm (3)



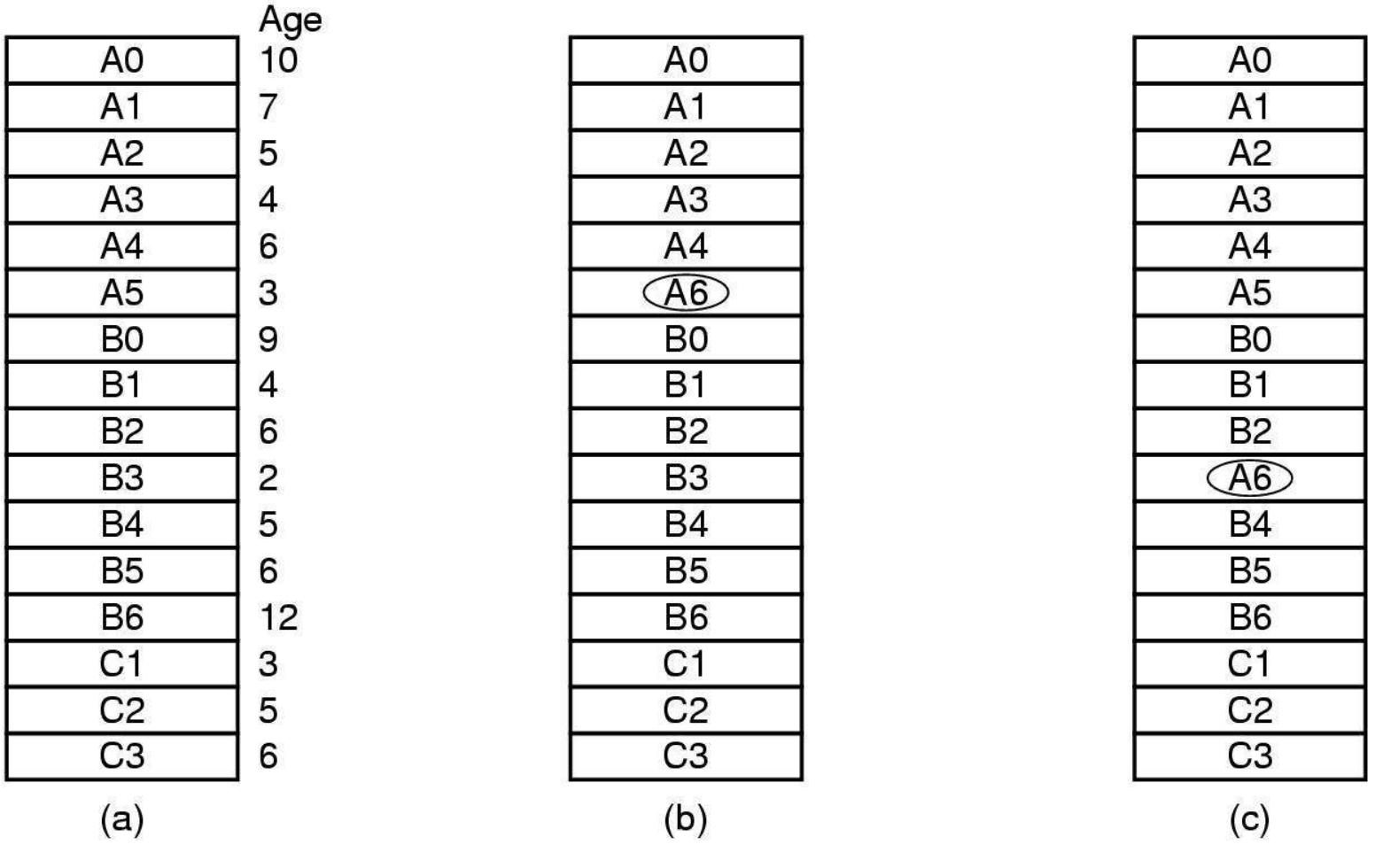
Operation of the WSClock algorithm.
(c) and (d) give an example of R = 0.

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

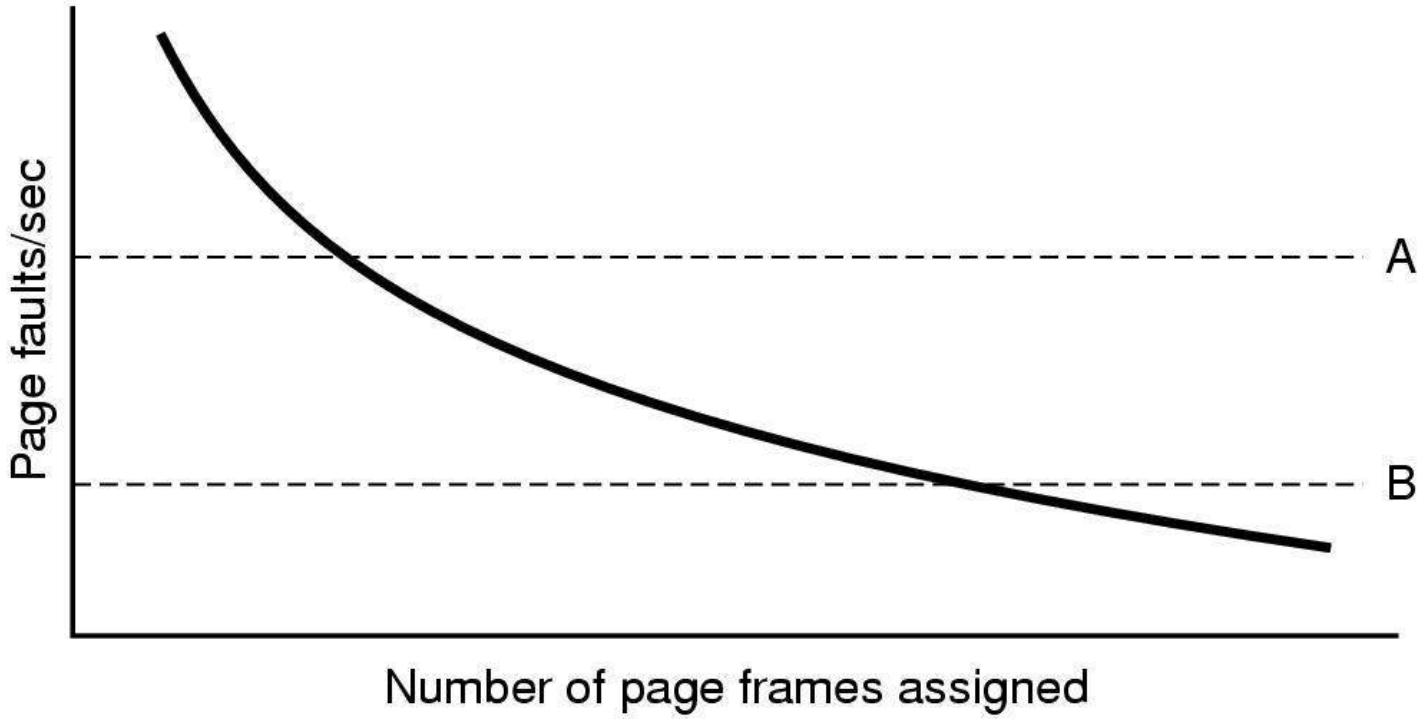
Page replacement algorithms discussed in the text.

Local versus Global Allocation Policies (1)



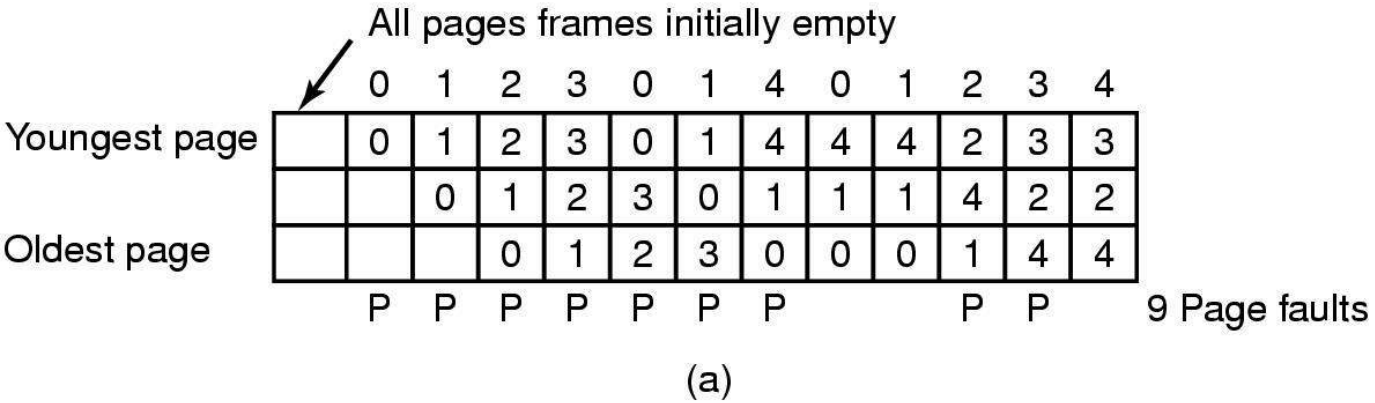
(a) Original configuration. (b) Local page replacement. (c) Global page replacement.

Local versus Global Allocation Policies (2)

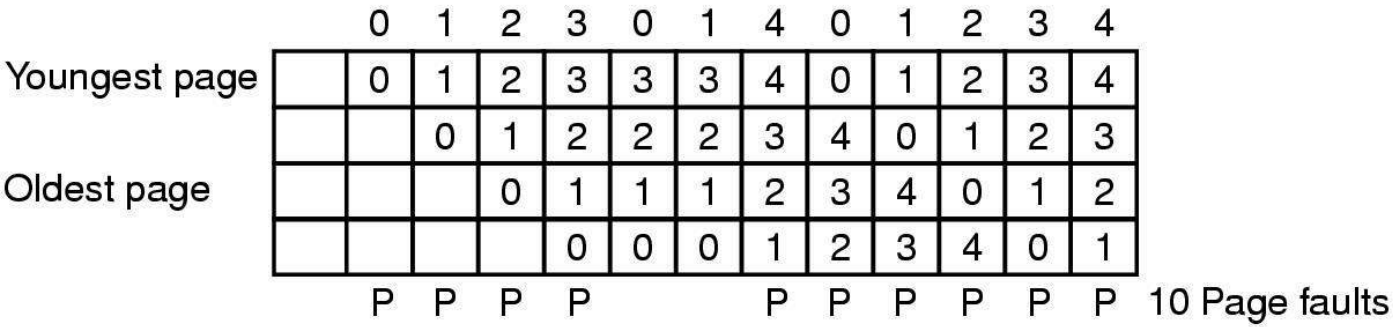


Page fault rate as a function
of the number of page frames assigned.

Modeling Page Replacement Algorithms
Belady's Anomaly



(a)



(b)

FIFO with 3 page frames

FIFO with 4 page frames

P's show which page references show page faults

Stack Algorithms

Reference string	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Page faults	P	P	P	P	P	P	P		P					P			P						P	
Distance string	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	2	3	5	3

State of memory array, *M*, after each item in reference string is processed

Load Control

Despite good designs, system may still thrash

When

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

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**

Page Size (2)

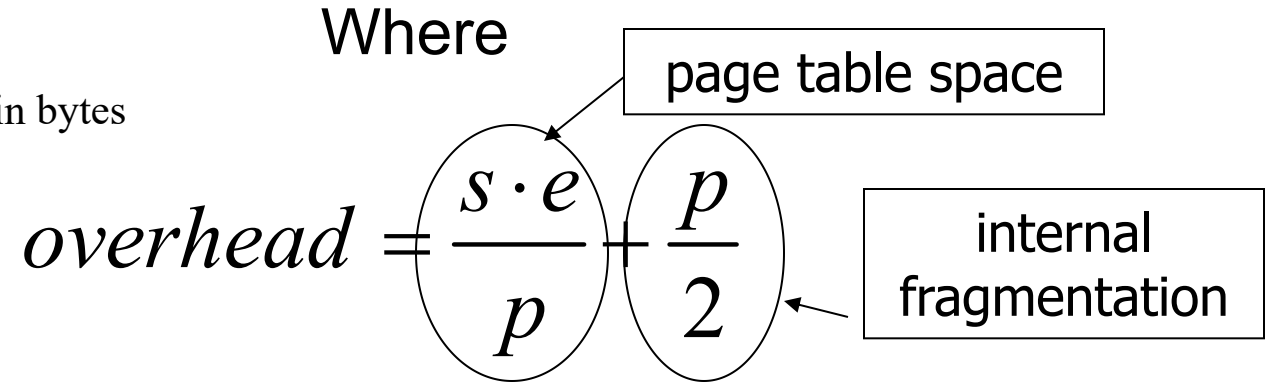
Overhead due to page table and internal fragmentation

s = average process size in bytes

p = page size in bytes

e = page entry

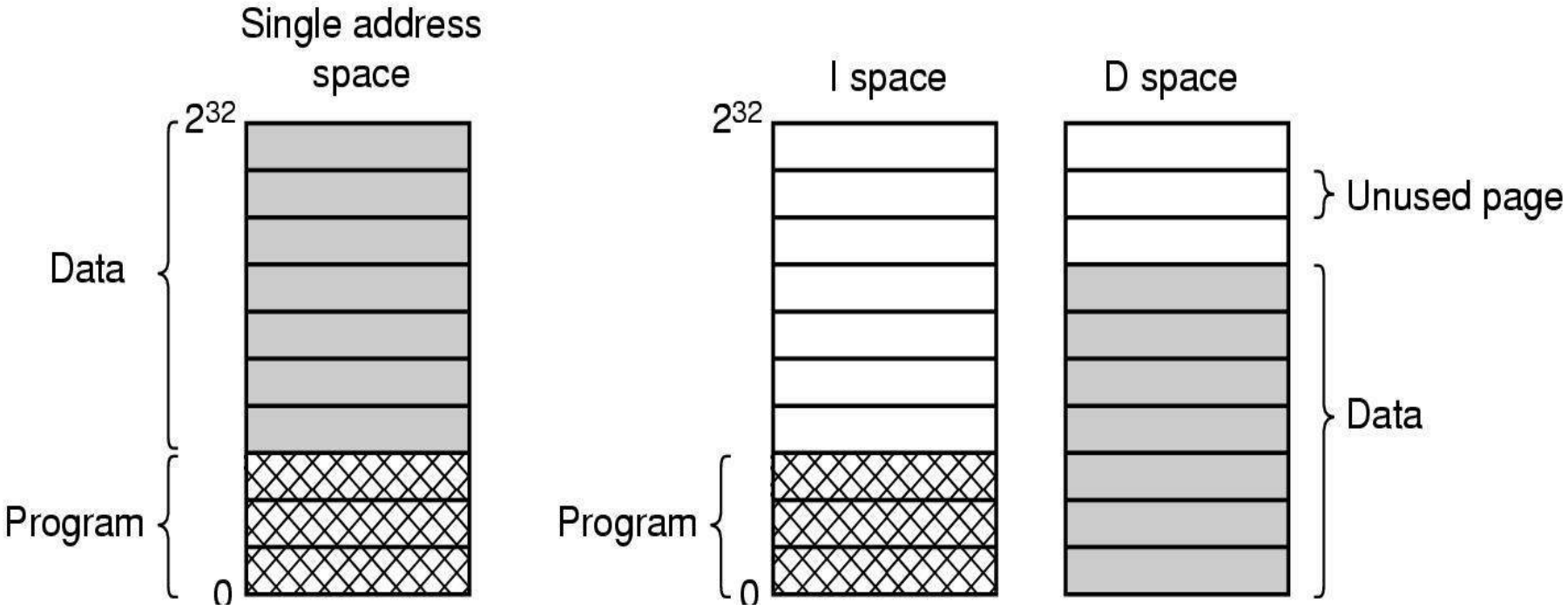
Where



Optimized when

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

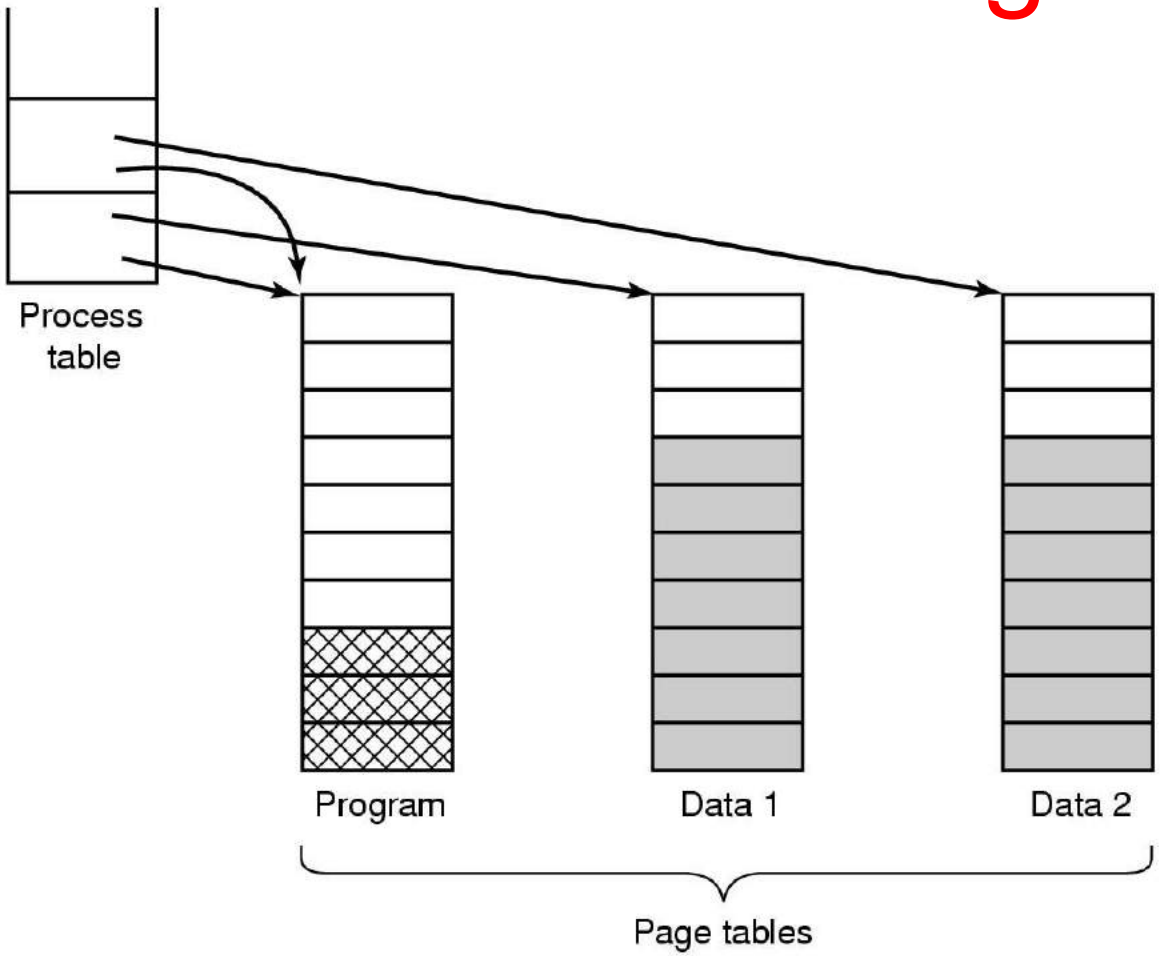
Separate Instruction and Data Spaces



One address space

Separate I and D spaces

Shared Pages



Two processes sharing same program sharing its page table

Implementation Issues

Operating System Involvement with Paging

Four times when OS involved with paging

Process creation

- determine program size
- create page table

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

Page Fault Handling (1)

Hardware traps to kernel

General registers saved

OS determines which virtual page needed

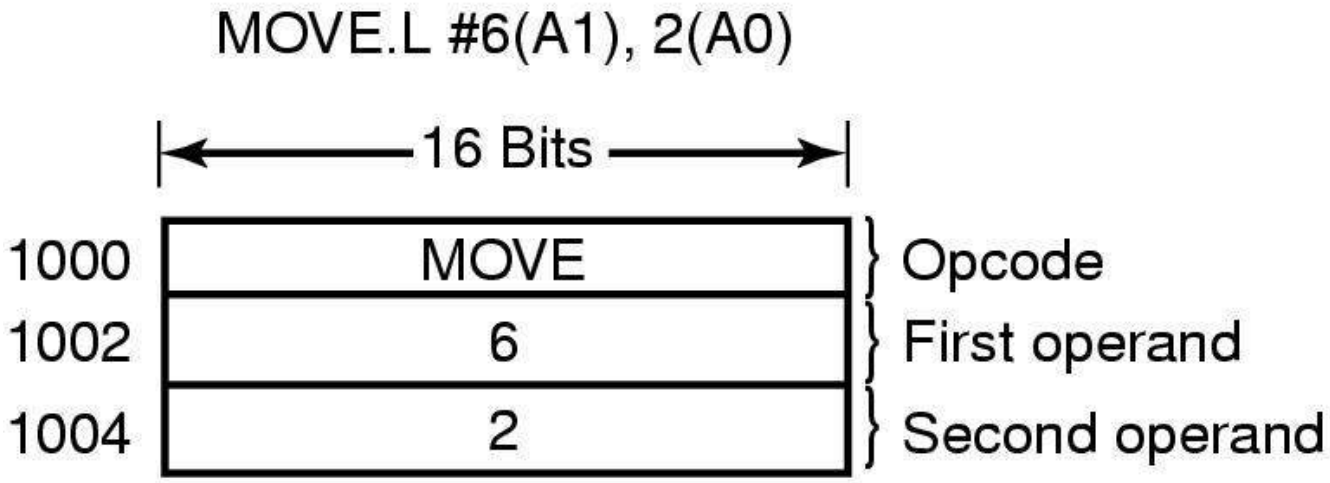
OS checks validity of address, seeks page frame

If selected frame is dirty, write it to disk

Page Fault Handling (2)

- 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

Instruction Backup



An instruction causing a page fault

Locking Pages in Memory

Virtual memory and I/O occasionally interact

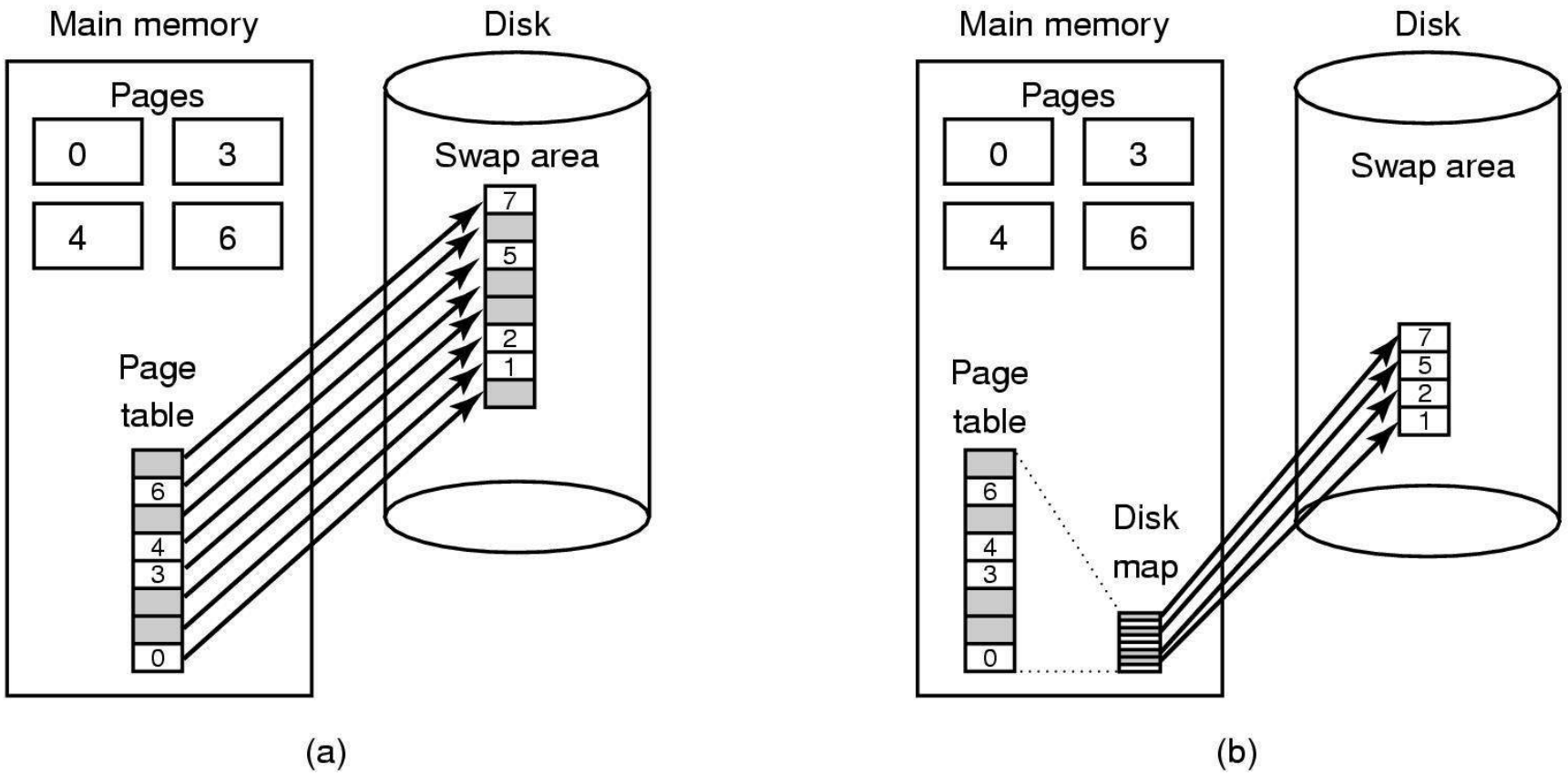
Proc issues call for read from device into buffer

- 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 to specify some pages locked

- exempted from being target pages

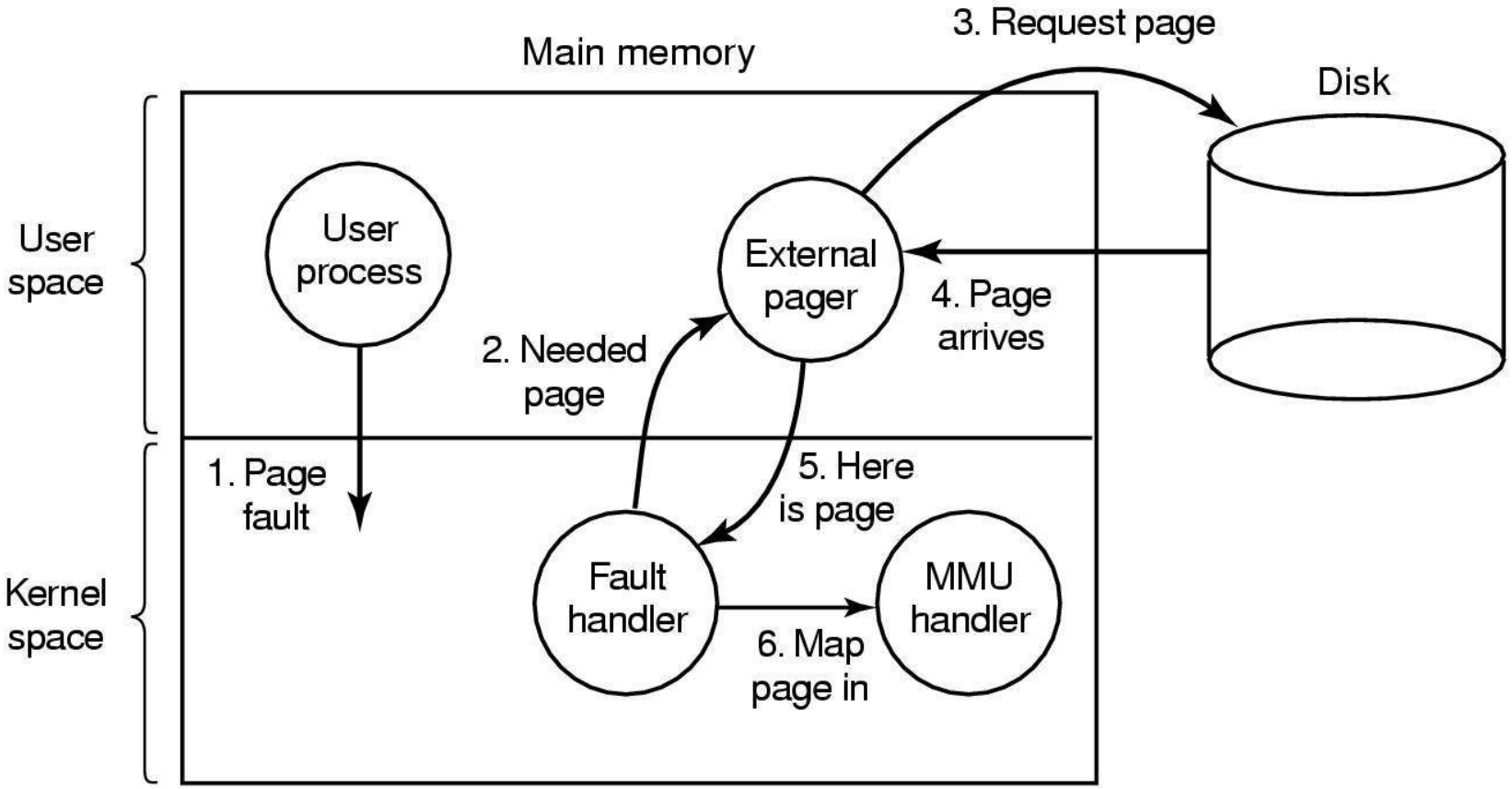
Backing Store



(a) Paging to static swap area

(b) Backing up pages dynamically

Separation of Policy and Mechanism



Page fault handling with an external pager

作业

P 139 页

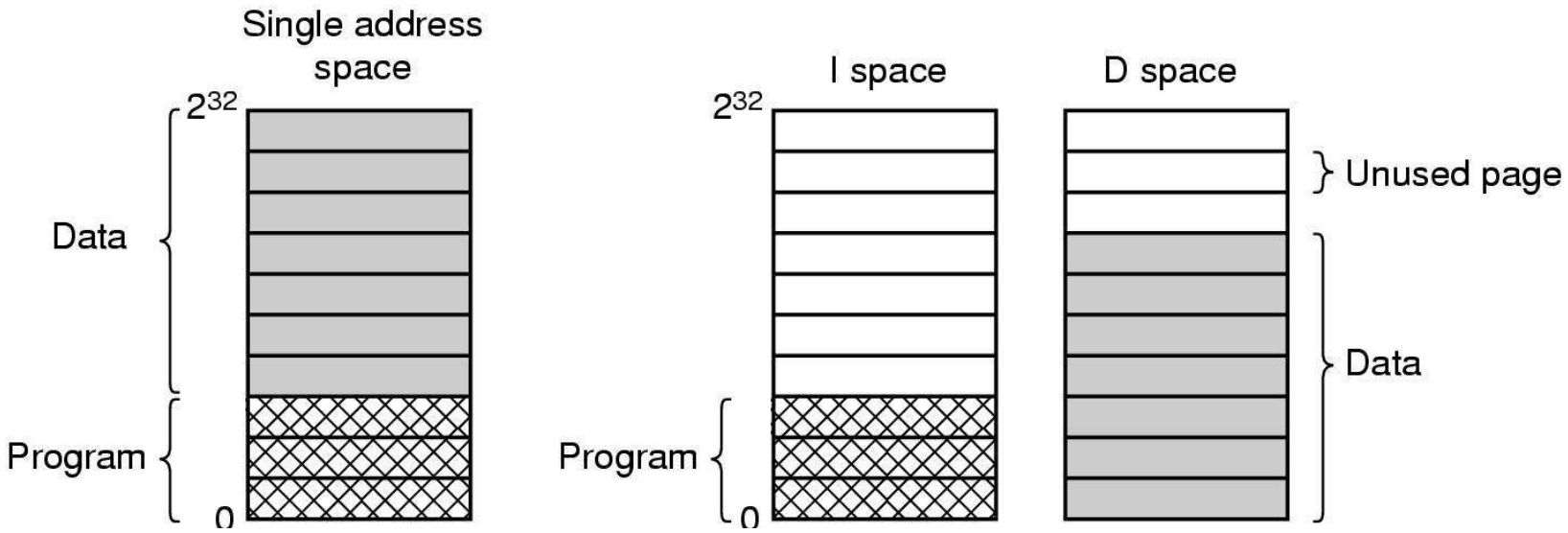
12，22，24，26，27，28

Chapter 3

Memory Management

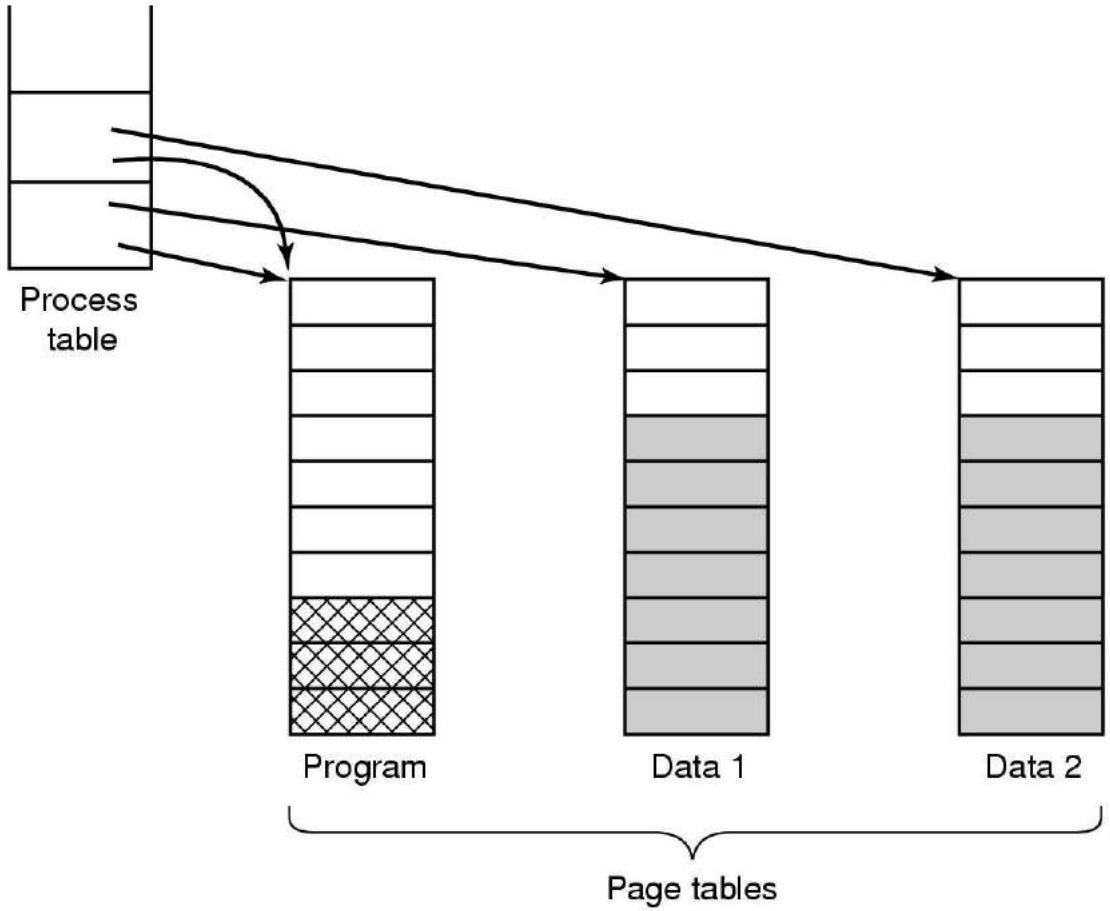
Segmentation

Separate Instruction and Data Spaces



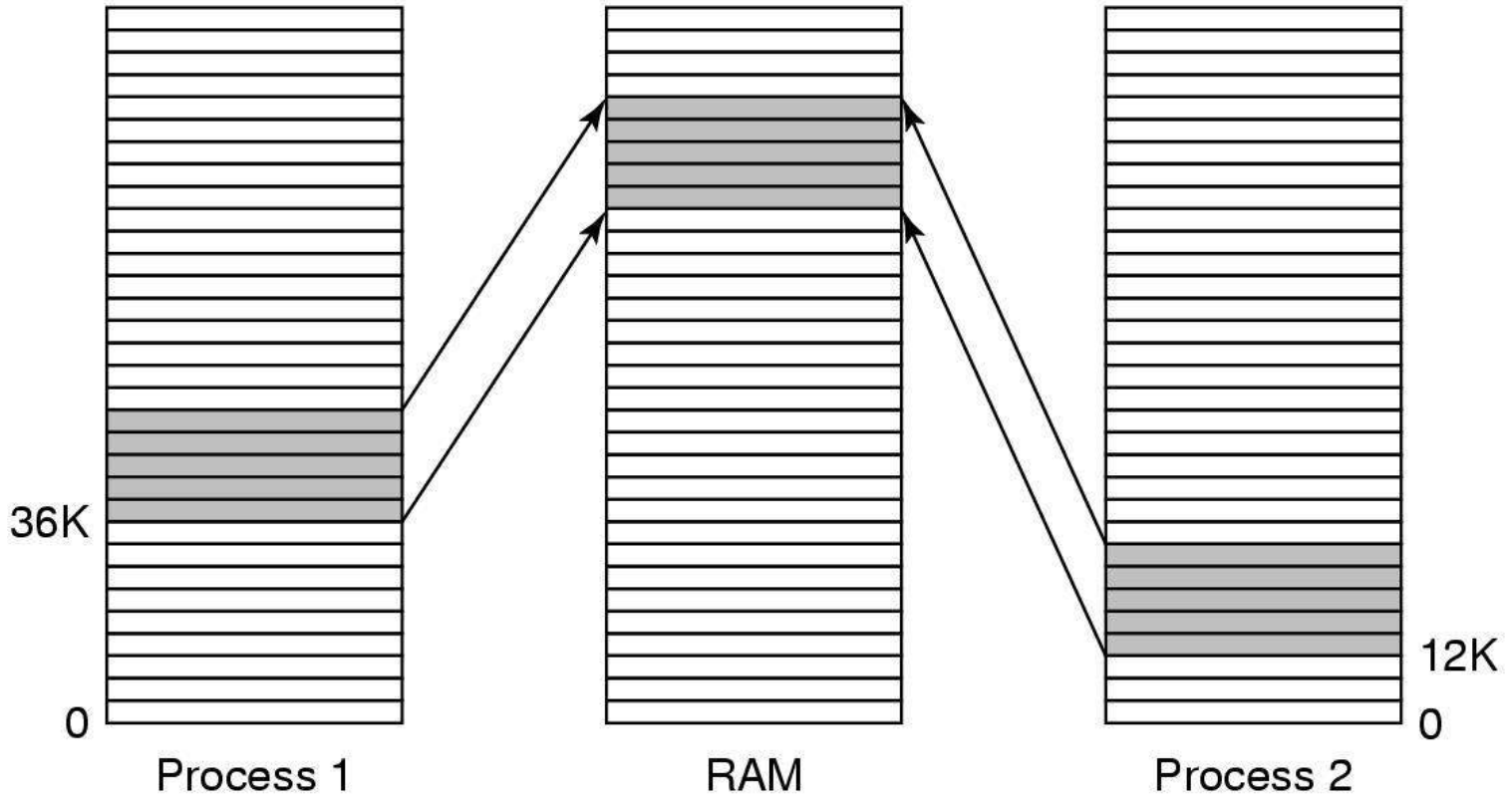
(a) One address space.
(b) Separate I and D spaces.

Shared Pages



Two processes sharing the same program sharing its page table.

Shared Libraries



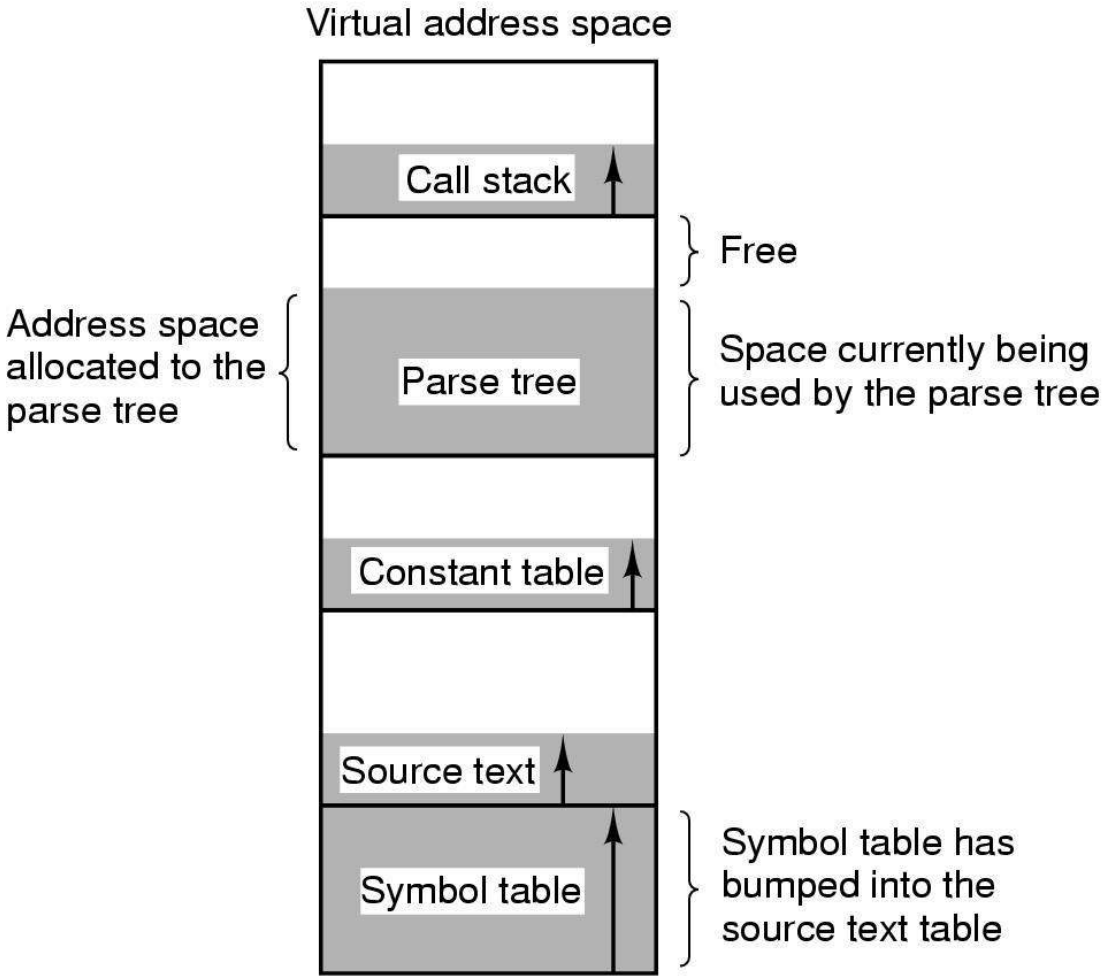
A shared library being used by two processes.

Segmentation (1)

A compiler has many tables that are built up as compilation proceeds, possibly including:

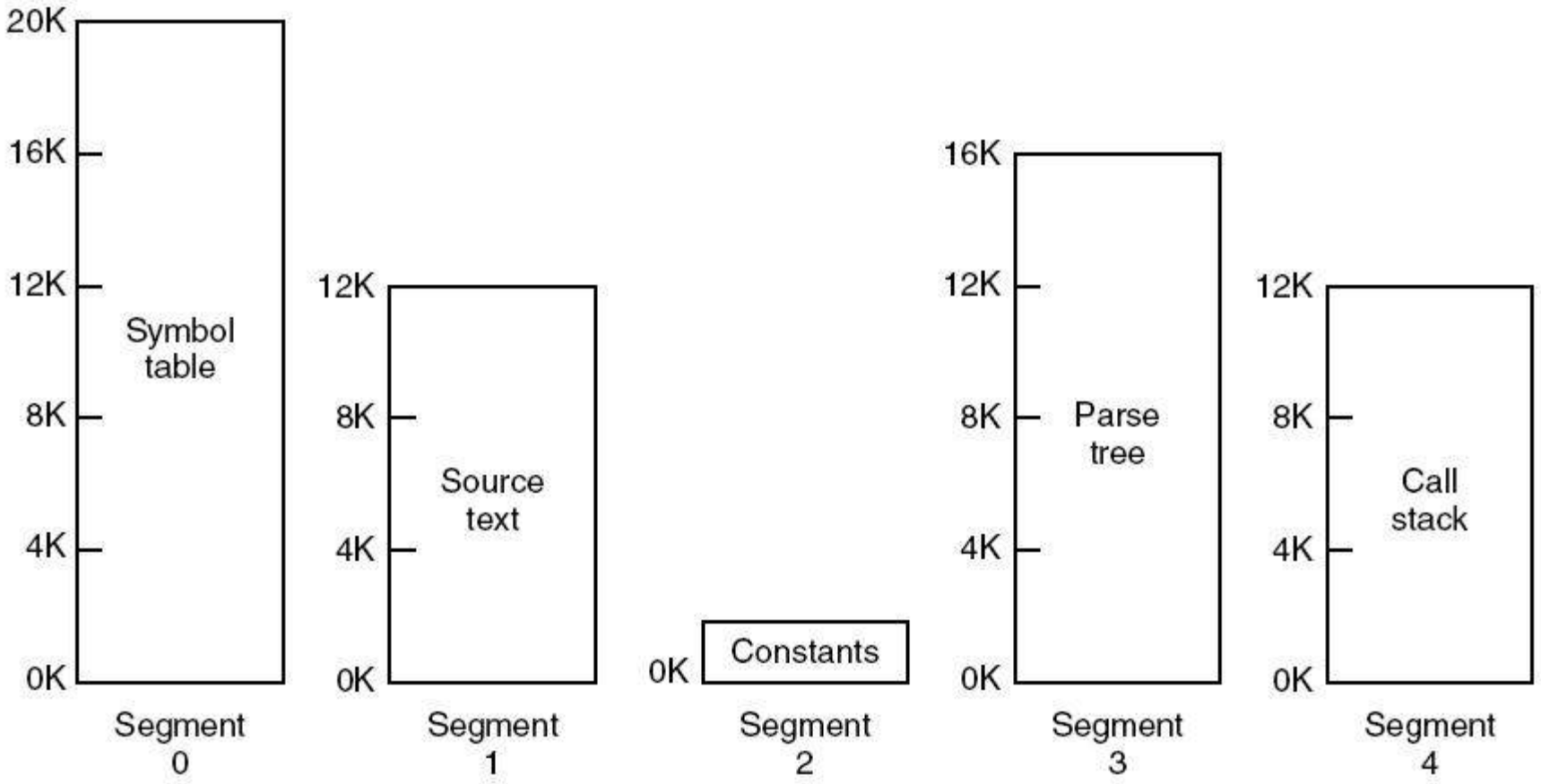
- The source text being saved for the printed listing (on batch systems).
- The symbol table – the names and attributes of variables.
- The table containing integer, floating-point constants used.
- The parse tree, the syntactic analysis of the program.
- The stack used for procedure calls within the compiler.

Segmentation (2)



In a one-dimensional address space with growing tables, one table may bump into another.

Segmentation (3)



A segmented memory allows each table to grow or shrink independently of the other tables.

Implementation of Pure 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

Comparison of paging and segmentation.

Segmentation

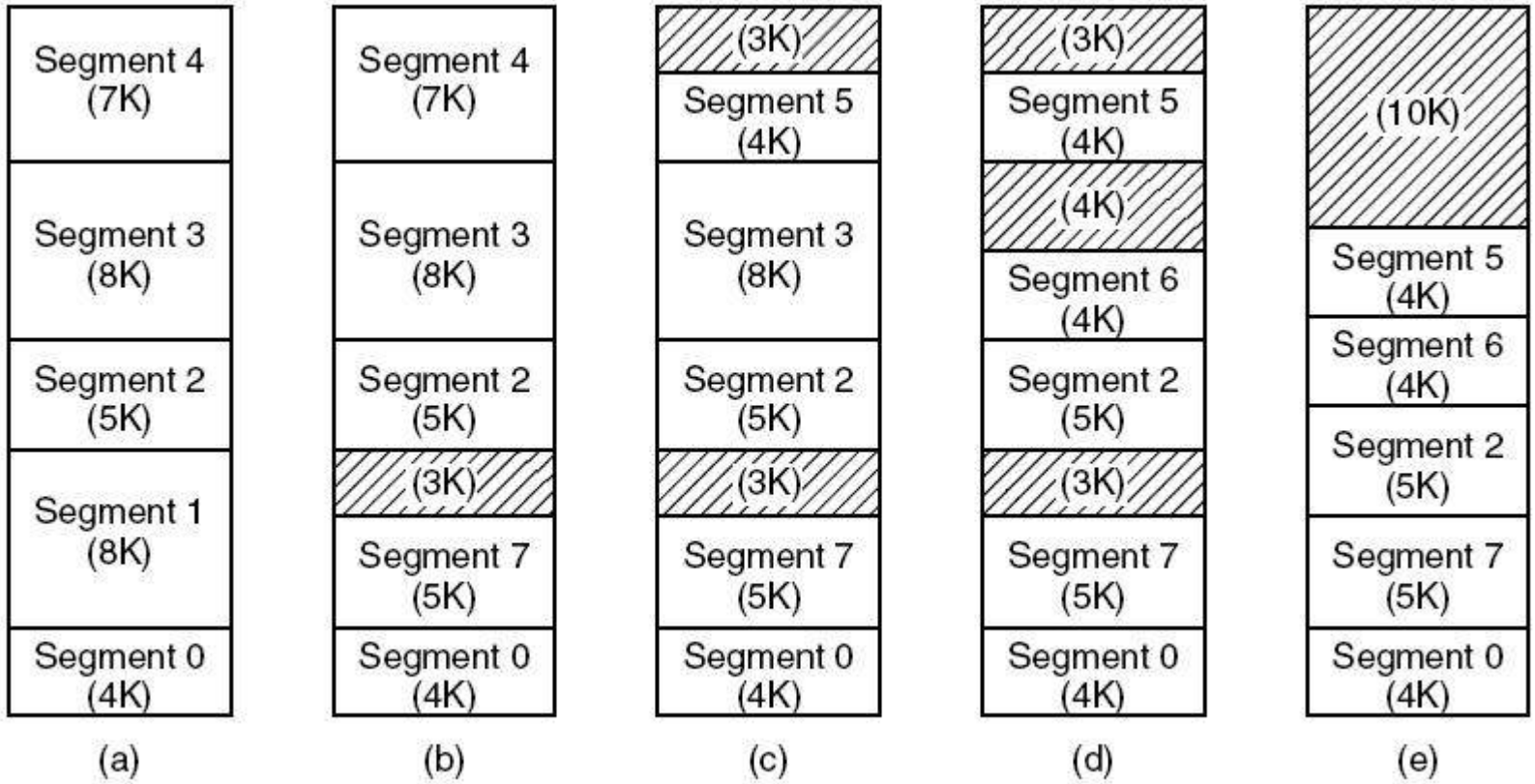


Figure 3-34. (a)-(d) Development of checkerboarding. (e) Removal of the checkerboarding by compaction.

Segmentation with Paging: MULTICS (1)

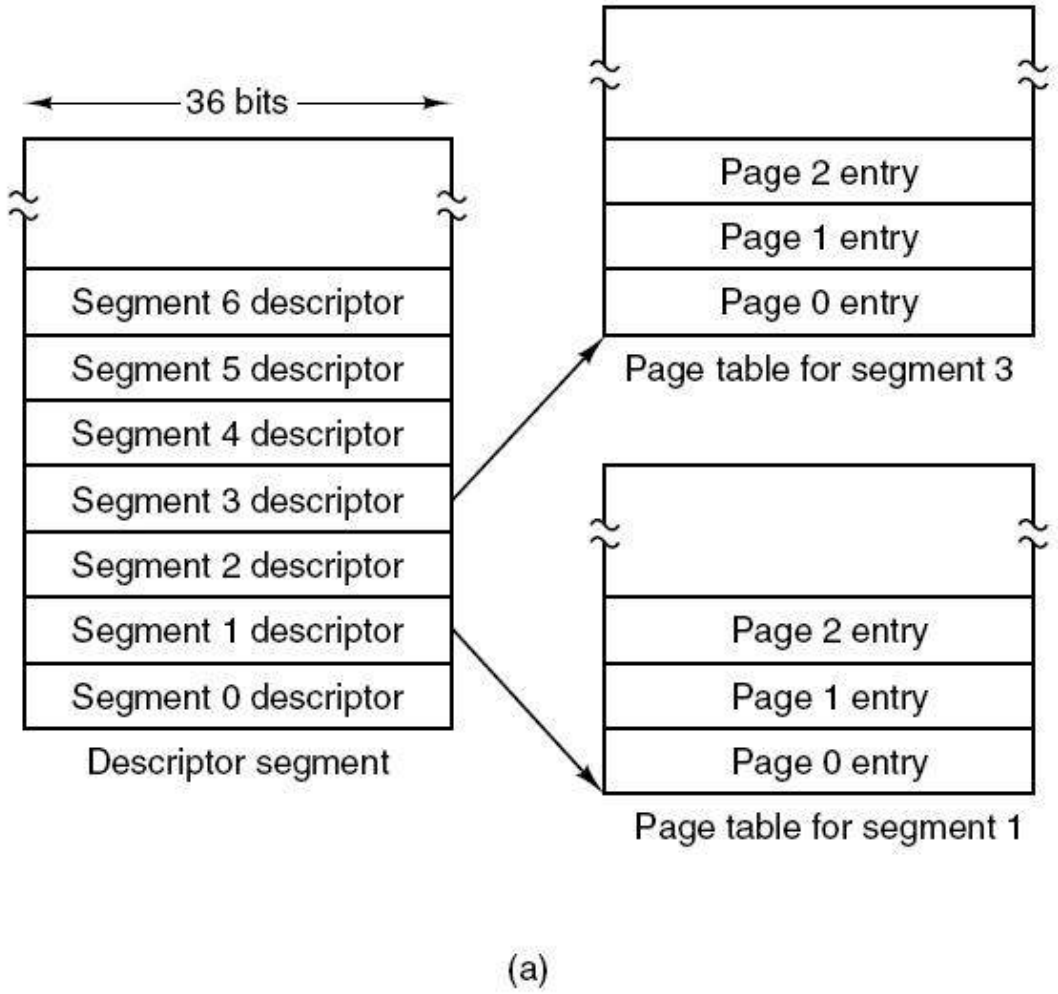


Figure 3-35. The MULTICS virtual memory. (a) The descriptor segment points to the page tables.

Segmentation with Paging: MULTICS (2)

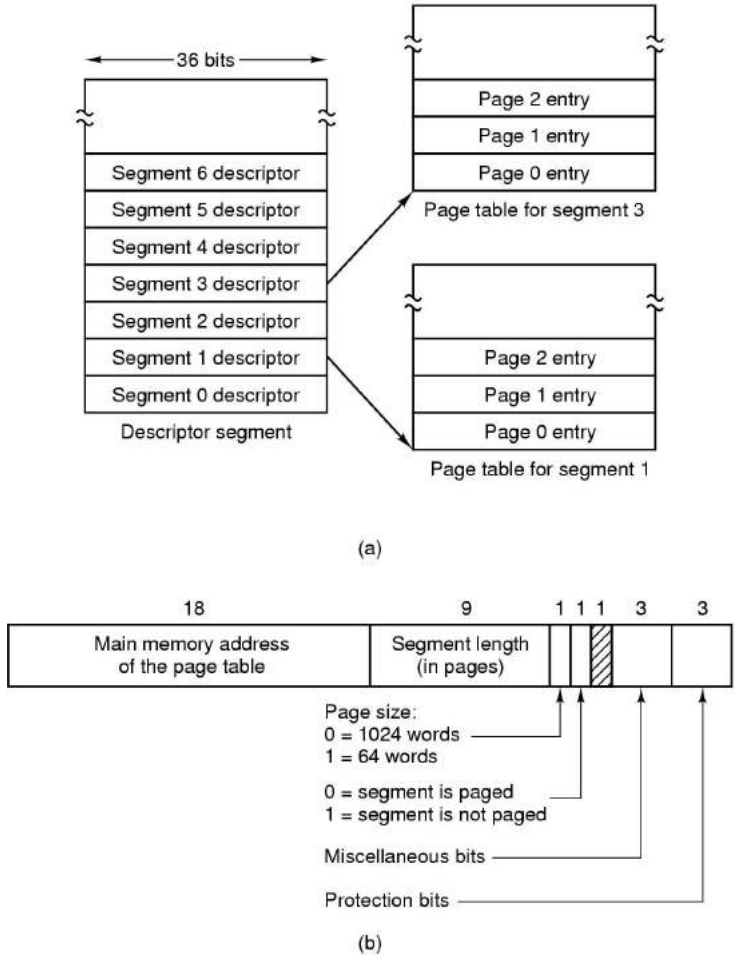


Figure 3-35. The MULTICS virtual memory. (b) A segment descriptor. The numbers are the field lengths.

Segmentation with Paging: MULTICS (3)

When a memory reference occurs, the following algorithm is carried out:

- The segment number used to find segment descriptor.
- Check is made to see if the segment's page table is in memory.
  - If not, segment fault occurs.
  - If there is a protection violation, a fault (trap) occurs.

Segmentation with Paging: MULTICS (4)

- Page table entry for the requested virtual page examined.
  - If the page itself is not in memory, a page fault is triggered.
  - If it is in memory, the main memory address of the start of the page is extracted from the page table entry
- The offset is added to the page origin to give the main memory address where the word is located.
- The read or store finally takes place.

Segmentation with Paging: MULTICS (5)

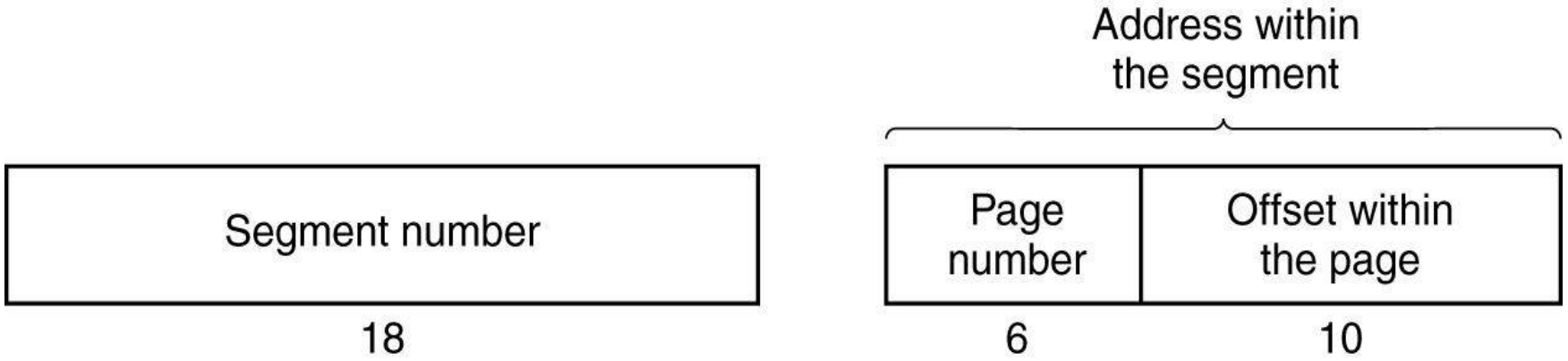


Figure 3-36. A 34-bit MULTICS virtual address.

Segmentation with Paging: MULTICS (6)

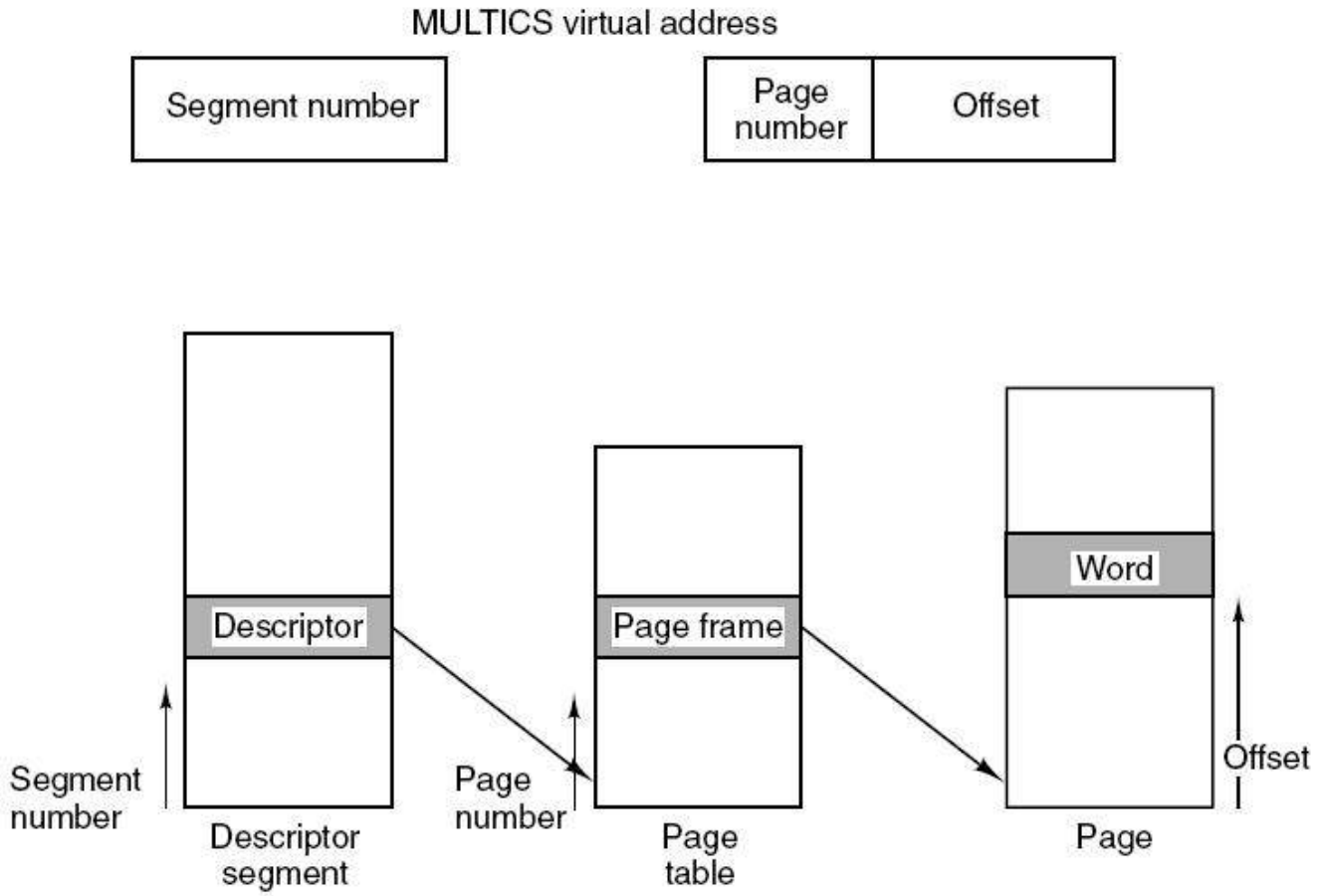
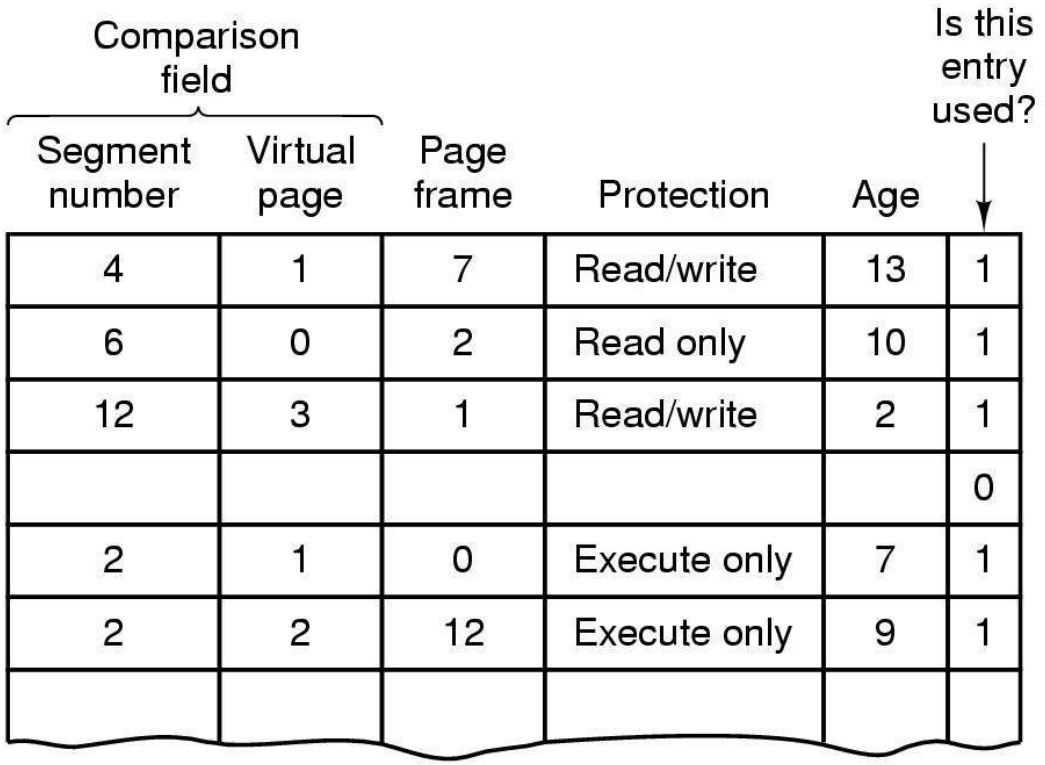


Figure 3-37. Conversion of a two-part MULTICS address into a main memory address.

Segmentation with Paging: MULTICS (7)



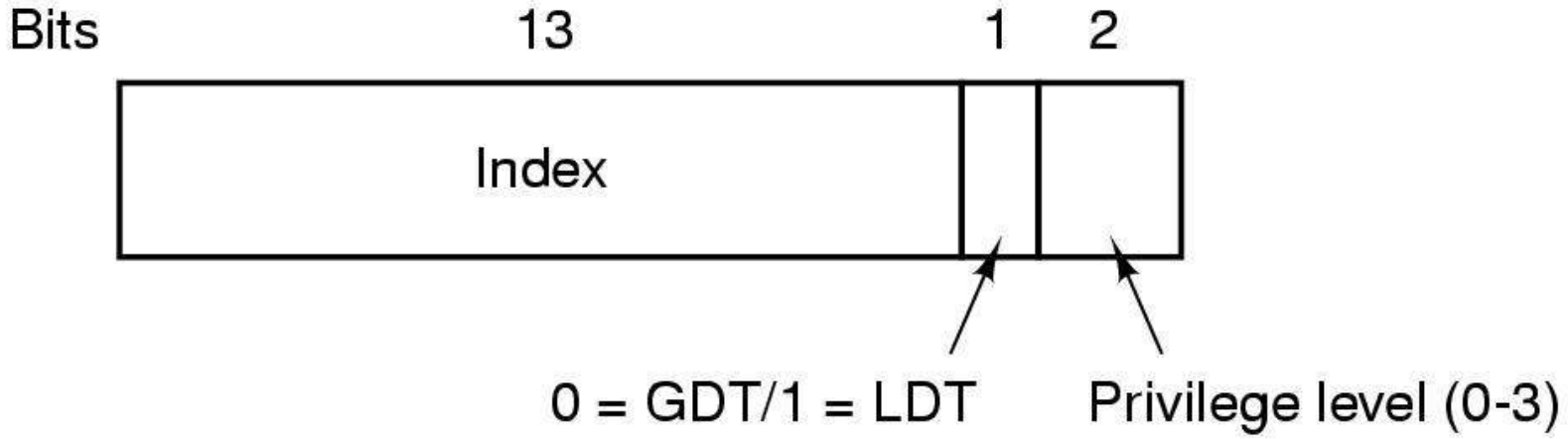
Comparison field: Segment number	Comparison field: Virtual page	Page frame	Protection	Age	Is this entry used?
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

Figure 3-38. A simplified version of the MULTICS TLB. The existence of two page sizes makes the actual TLB more complicated.

Summary

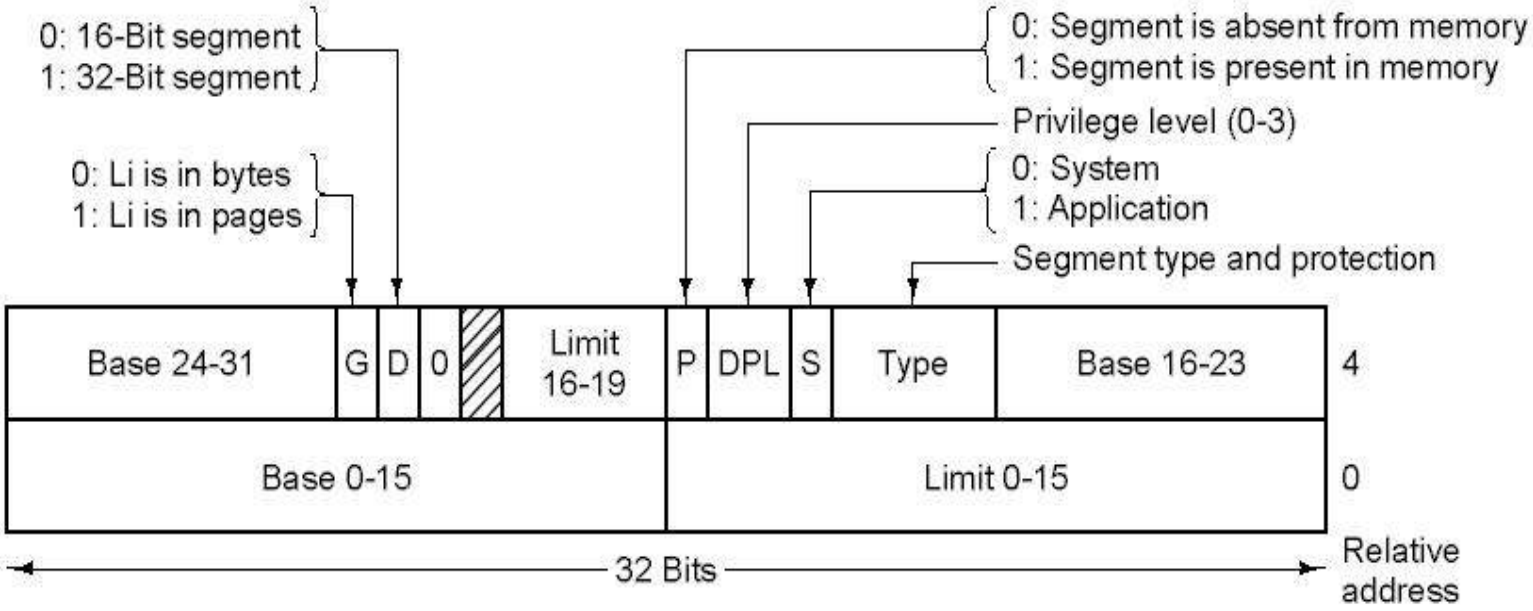
	Fixed Partition	Variable Partition	Paging	Segmentation	Segmentation with Paging
Fragmen-tation	Internal	External	Internal	External	Internal
Continuity	Whole process	Whole process	Page	Segment	Page
Swapping	Whole process	Whole process	Page	Segment	Segment
Relocation	Base register	Base register	Page table	Segment table	Segment table with page tables

Segmentation with Paging: Pentium (1)



A Pentium selector

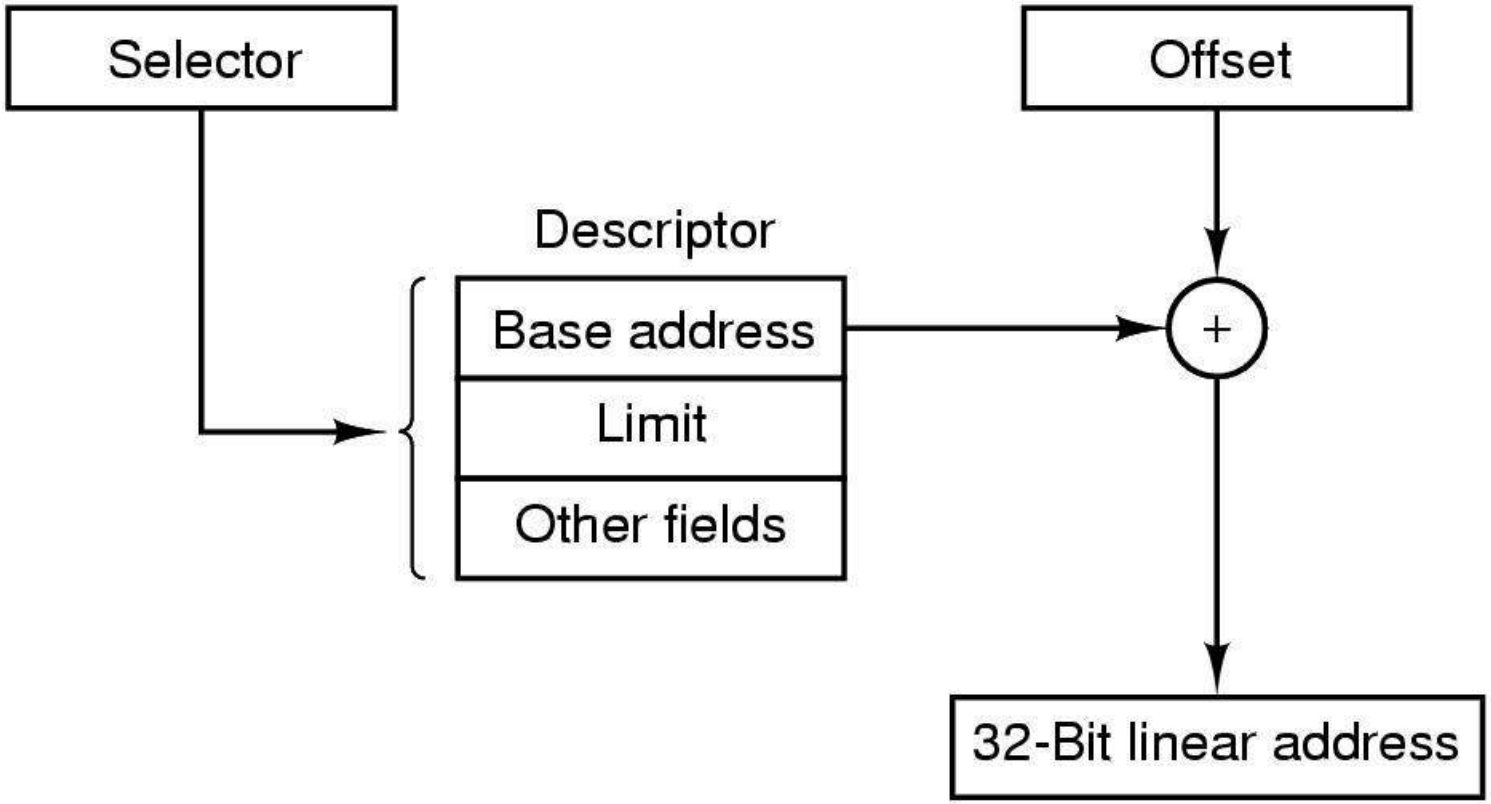
Segmentation with Paging: Pentium (2)



Pentium code segment descriptor

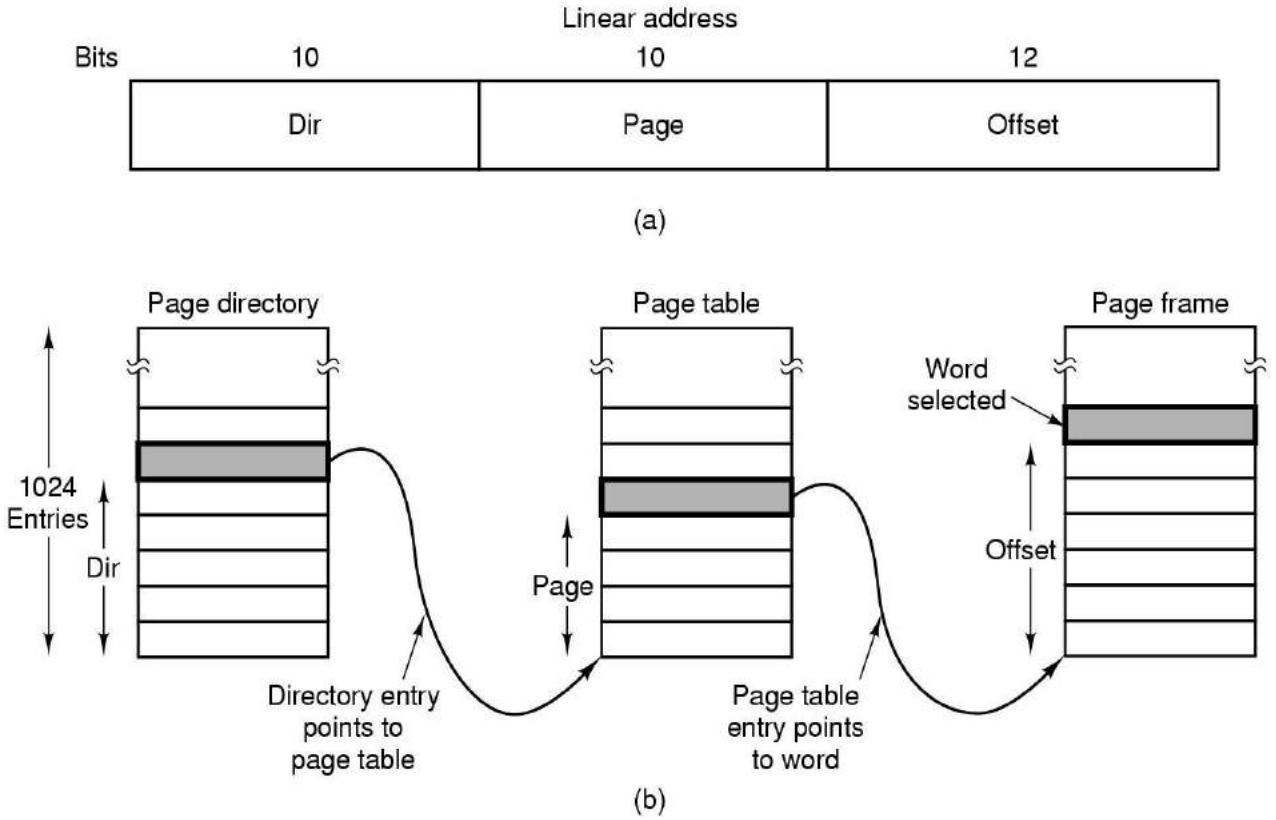
Data segments differ slightly

Segmentation with Paging: Pentium (3)



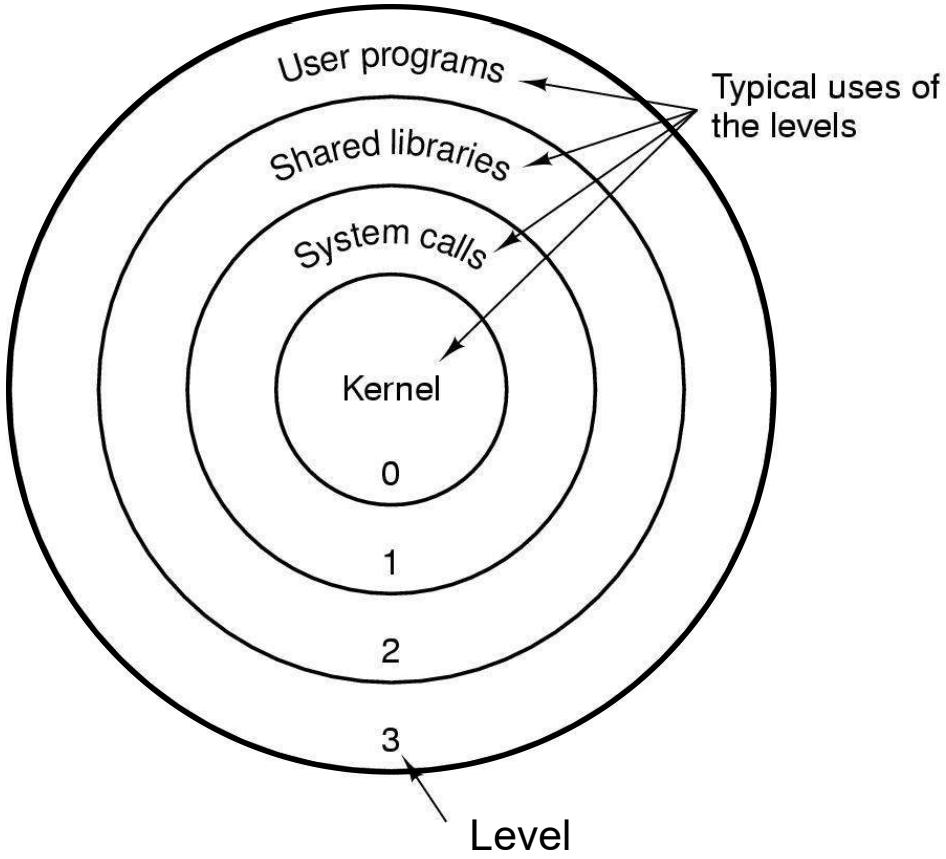
Conversion of a (selector, offset) pair to a linear address

Segmentation with Paging: Pentium (4)



Mapping of a linear address onto a physical address

Segmentation with Paging: Pentium (5)



Protection on the Pentium

Chapter 5 File Management

File Overview

File Systems (1)

Essential requirements for long-term information storage:

- It must be possible to store a very large amount of information.
- The information must survive the termination of the process using it.
- Multiple processes must be able to access the information concurrently.

File & File Structure

- File: a named collection of related information that is recorded on secondary storage
  - contiguous logical address space
  - Types:
    - Data: numeric, character, binary
    - Program
- File Structure: depends on its type
  - None - sequence of words, bytes
  - Complex Structures
  - Simple record structure
    - Lines
    - Fixed length
    - Variable length
    - Formatted document
    - etc.

File Systems (2)

Think of a disk as a linear sequence of fixed-size blocks and supporting reading and writing of blocks. Questions that quickly arise:

- How do you find information?
- How do you keep one user from reading another's data?
- How do you know which blocks are free?

File Structure

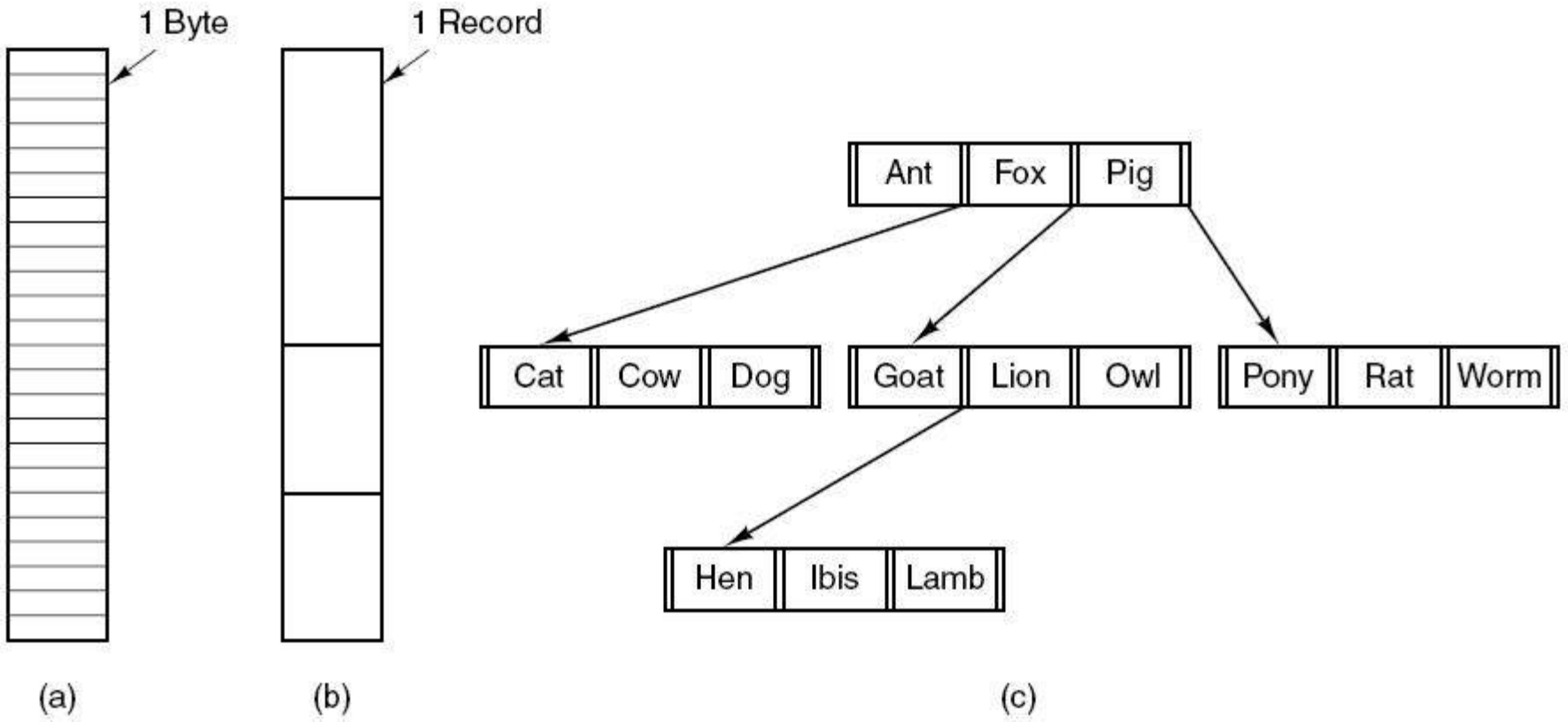


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

Linux Filenames

- Should be descriptive of the content
- Should use only alphanumeric characters:
  UPPERCASE, lowercase, number, @, _
- Should not include embedded blanks
- Should not contain shell metacharacters:
  *** ? > < / ; & ! | \ ` ' " [ ] ( ) { }**
- Should not begin with + or - sign
- Are case sensitive
- Are hidden if the first character is a **.** (period)
- Can have a maximum of 255 characters

File Naming

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Figure 4-1. Some typical file extensions.

File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4a. Some possible file attributes.

File Types

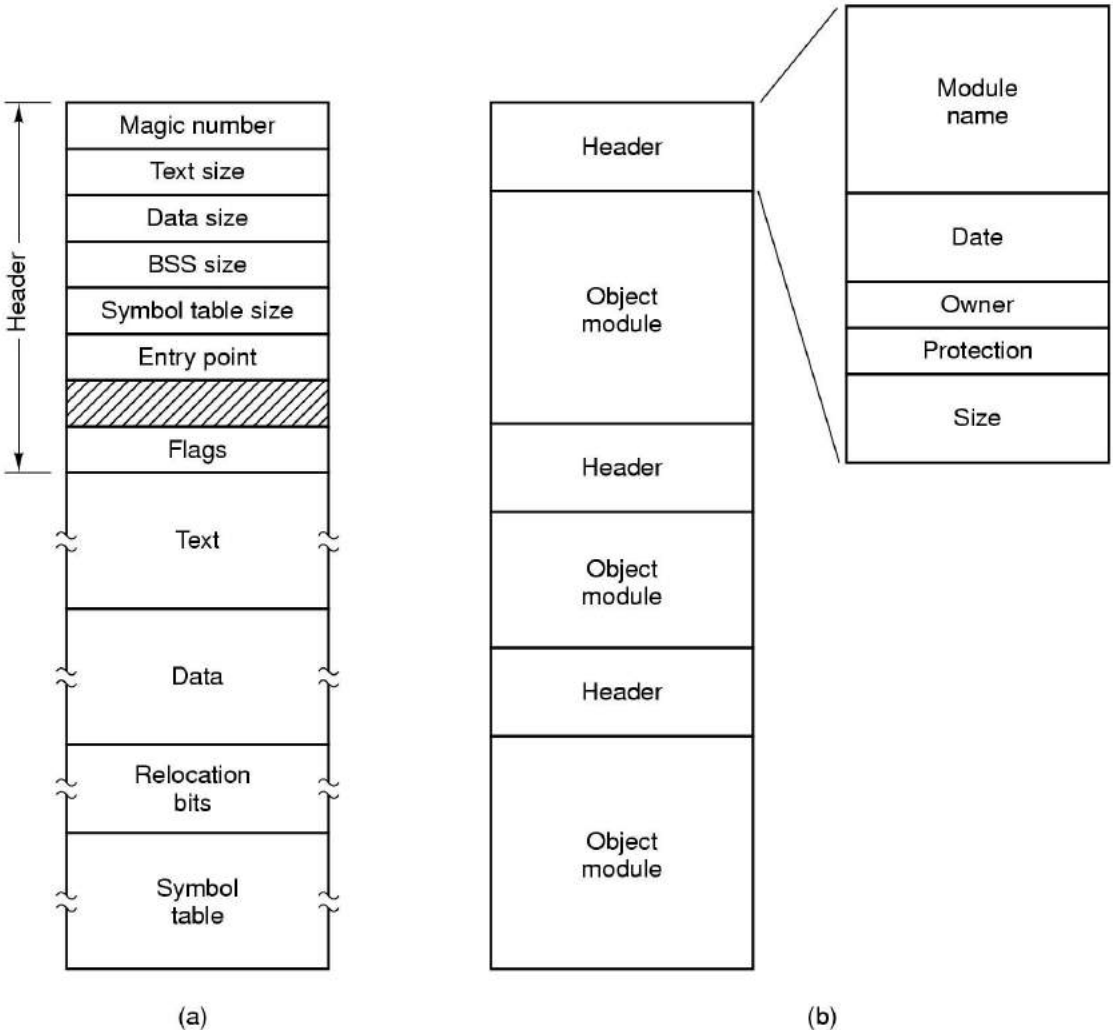


Figure 4-3. (a) An executable file. (b) An archive.

File Operations

The most common system calls relating to files:

- Create
- Delete
- Open
- Close
- Read
- Write
- Append
- Seek
- Get Attributes
- Set Attributes
- Rename

Example Program Using File System Calls (1)

```
/* File copy program. Error checking and reporting is minimal. */

#include <sys/types.h>                  /* include necessary header files */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI prototype */

#define BUF_SIZE 4096                   /* use a buffer size of 4096 bytes */
#define OUTPUT_MODE 0700                /* protection bits for output file */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* syntax error if argc is not 3 */

    /* Open the input file and create the output file */
    in_fd = open(argv[1], O_RDONLY);    /* open the source file */
    if (in_fd < 0) exit(2);             /* if it cannot be opened, exit */
    out_fd = creat(argv[2], OUTPUT_MODE); /* create the destination file */
    if (out_fd < 0) exit(3);            /* if it cannot be created, exit */
```

. . .

Figure 4-5. A simple program to copy a file.

Example Program Using File System Calls (2)

```
        /* Copy loop */
        while (TRUE) {
                rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
        if (rd_count <= 0) break;                         /* if end of file or error, exit loop */
                wt_count = write(out_fd, buffer, rd_count); /* write data */
                if (wt_count <= 0) exit(4);               /* wt_count <= 0 is an error */
        }

        /* Close the files */
        close(in_fd);
        close(out_fd);
        if (rd_count == 0)                                /* no error on last read */
                exit(0);
        else
                exit(5);                                  /* error on last read */
}
```

Figure 4-5. A simple program to copy a file.

Hierarchical Directory Systems (1)

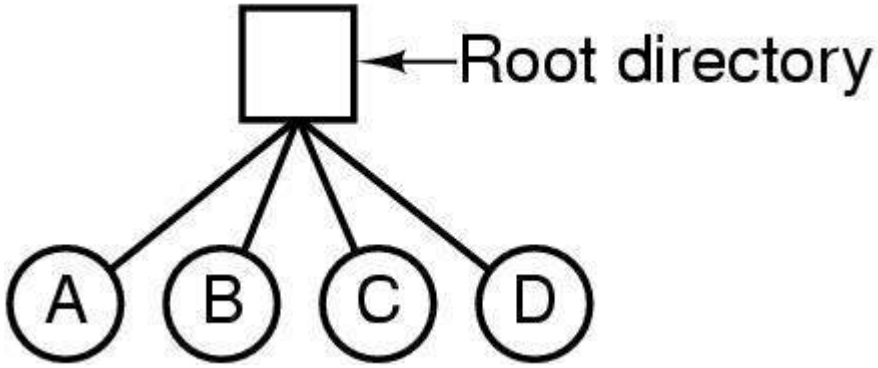


Figure 4-6. A single-level directory system containing four files.

Hierarchical Directory Systems (2)

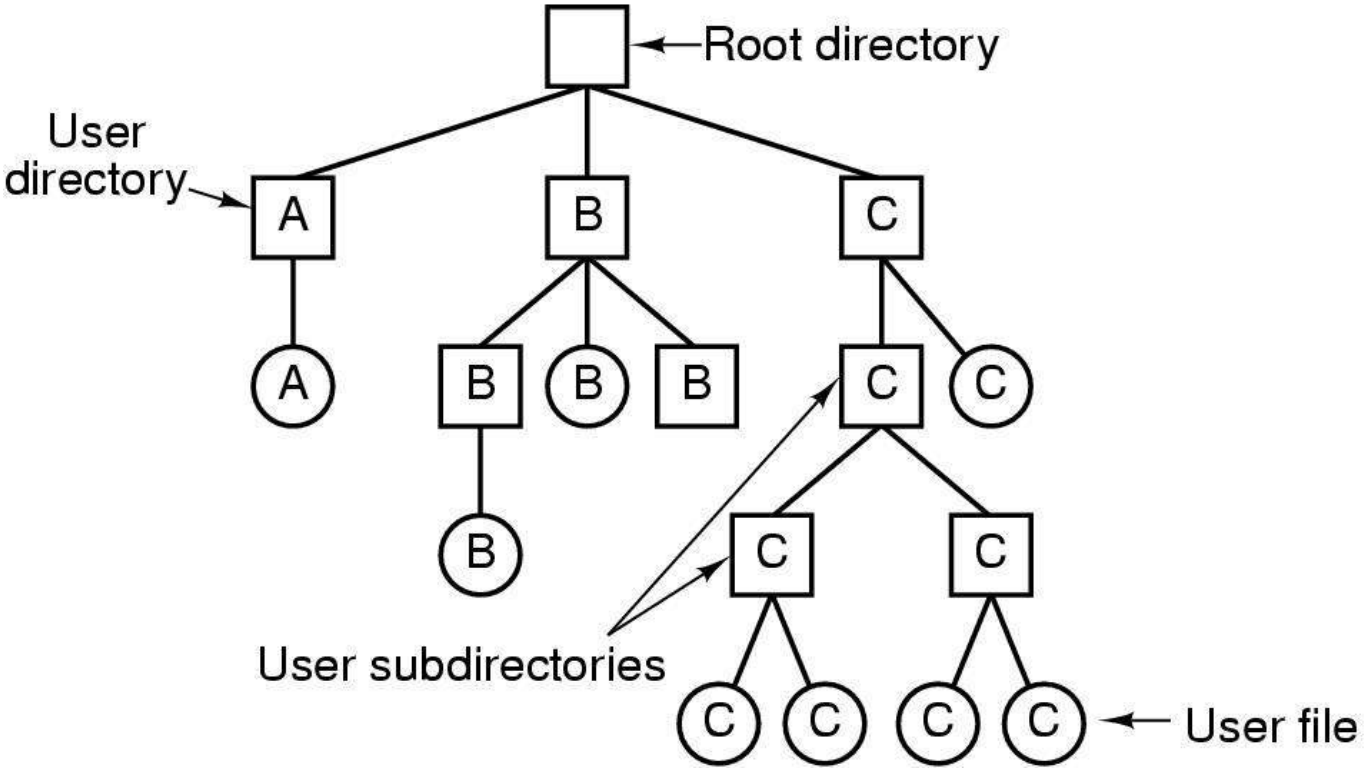


Figure 4-7. A hierarchical directory system.

Path Names

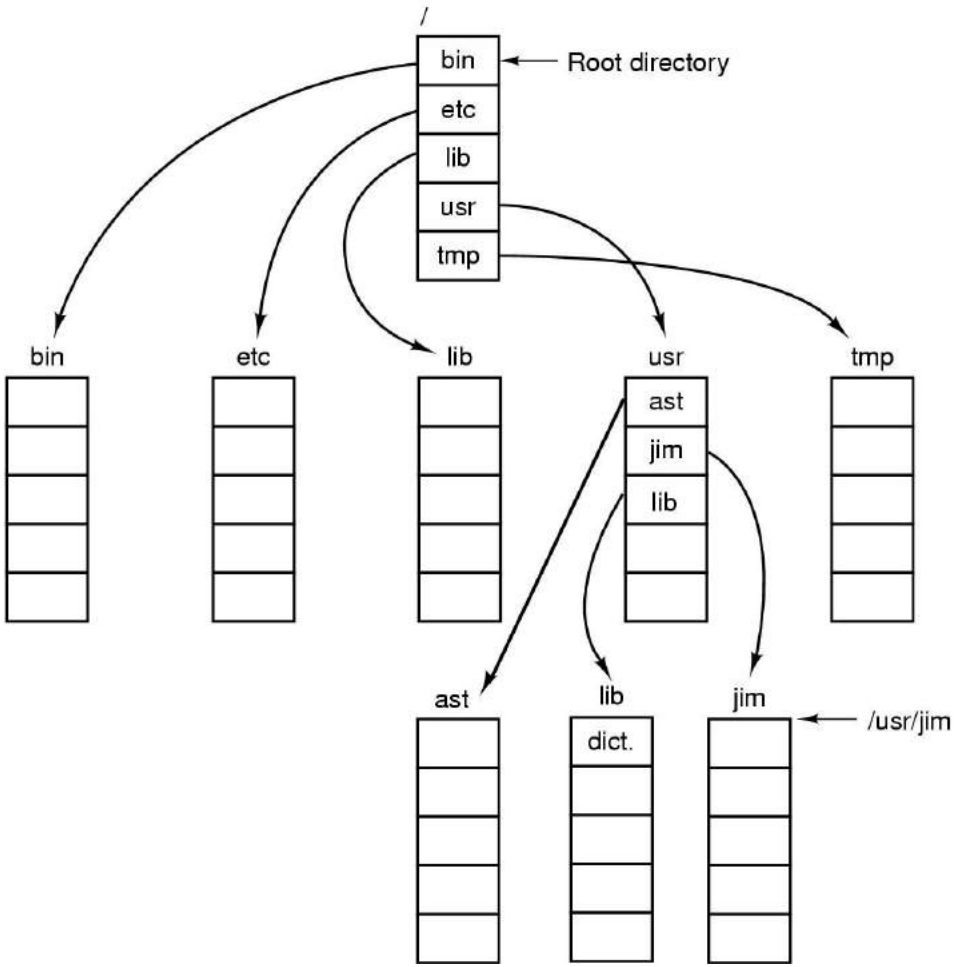
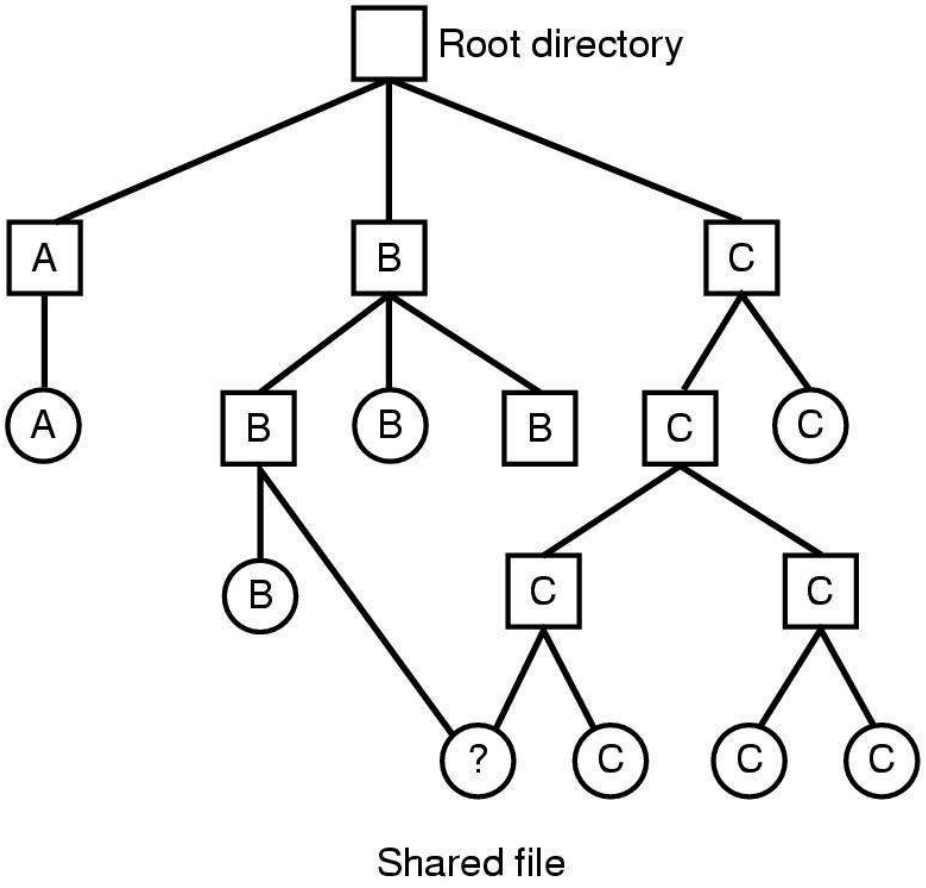


Figure 4-8. A UNIX directory tree.

Shared Files



File system containing a shared file.

Directory Operations

System calls for managing directories:

- Create
- Delete
- Opendir
- Closedir
- Readdir
- Rename
- Link
- Uplink

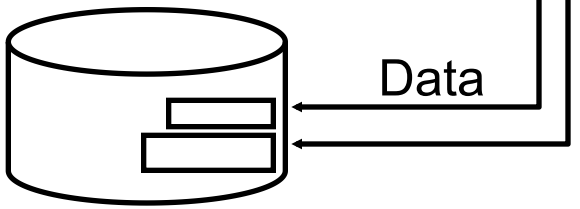
Directory Contents

Directory

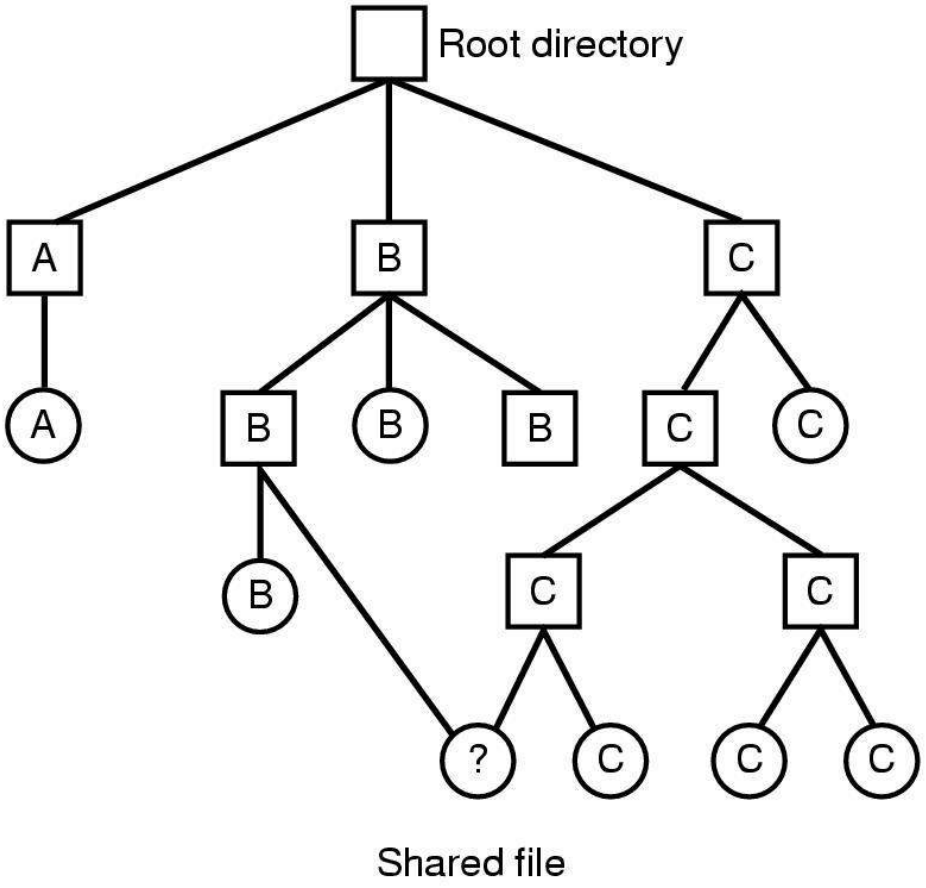
i-node Table

name	i-node
subdir1	4
myfile	10

#	type	mode	links	user	group	date	size	loc
4	dir	755	2	team01	staff	July 10 10:15	512	
10	file	644	1	team01	staff	July 11 11:00	96	



Shared Files



File system containing a shared file.

Shared Files (2)

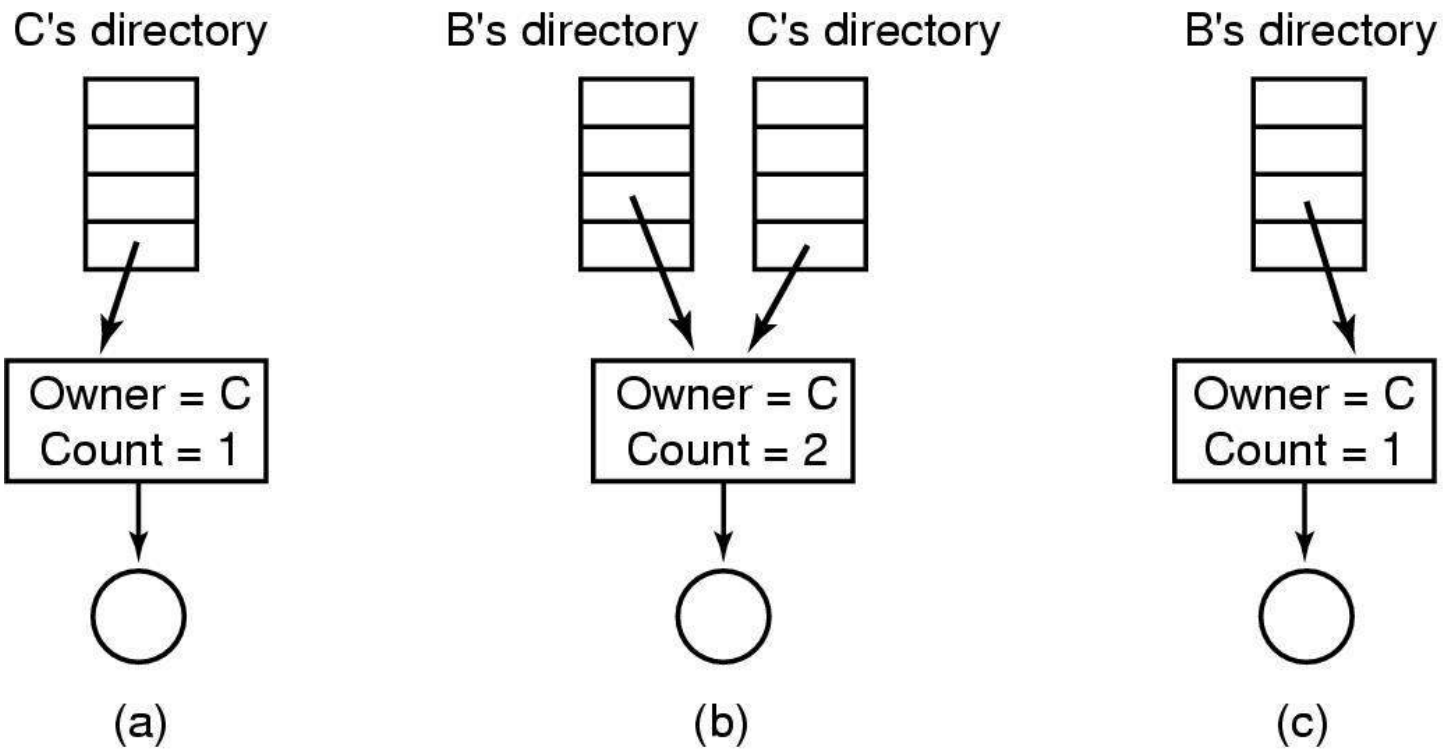


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

Linking Files

```
ln source_file target_file
```

- Allows files to have more than one name in the directory structure
- Both files reference the same i-node
- Cannot be used with directories, cannot span file systems

```
$ ls -li
63 -rw-r--r--    1 team01   staff    1910 Nov 21 14:19 man_files

$ ln man_files manuals

$ ls -li
63 -rw-r--r--    2 team01   staff    1910 Nov 21 14:19 man_files
63 -rw-r--r--    2 team01   staff    1910 Nov 21 14:19 manuals

$
```

Linking Files (cont.)

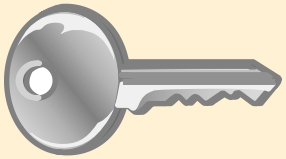
`ln -s source_file target_file`

- Creates an indirect reference to a file (symbolic link)
- Name references the original file's name and path
- Can be used with directories and span file systems

```
$ ls -li
63 -rw-r--r--    1 team01   staff    1910 Nov 21 14:19 man_files

$ ln -s man_files manuals

$ ls -li
63 -rw-r--r--    1 team01   staff    1910 Nov 21 14:19 man_files
66 lrwxrwxrwx    1 team01   staff    1910 Nov 21 14:19 manuals -> man_files

$
```

Permissions

File permissions are assigned to:



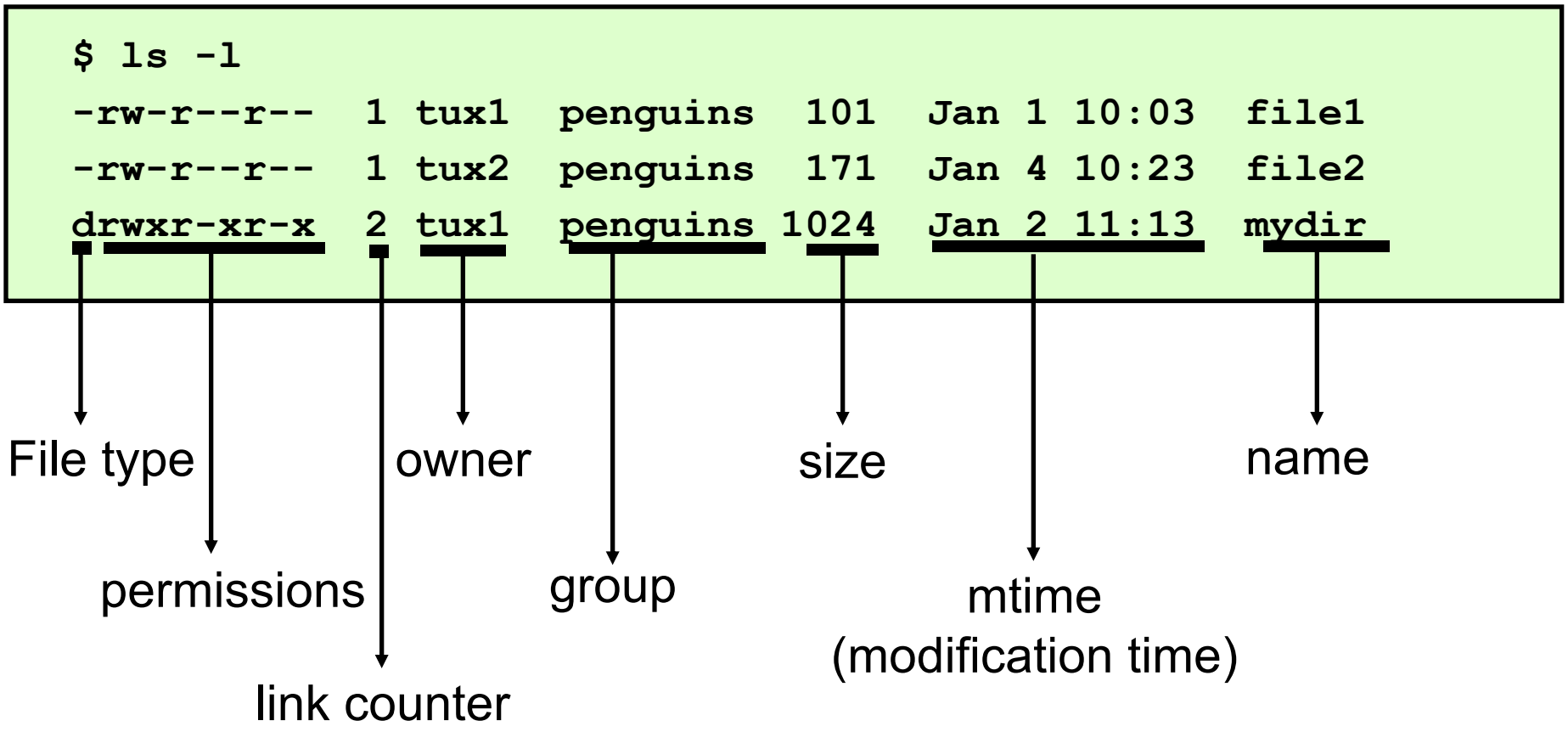
1. The **owner** of a file
2. The **members** of the group the file is assigned to
3. All **other users**

Permissions can only be changed by the owner and **root**!

Viewing Permissions

To show the permissions of a file, use the **ls** command with the **-l** option.

```
$ ls -l
-rw-r--r--  1 tux1  penguins  101  Jan 1 10:03  file1
-rw-r--r--  1 tux2  penguins  171  Jan 4 10:23  file2
drwxr-xr-x  2 tux1  penguins 1024  Jan 2 11:13  mydir
```



Permissions Notation

rwxrwxrwx

r	read
w	write
x	execute

owner group other

Regular files:

r file is readable

w file is writeable

x file is executable (if in an executable format)

Directories:

r contents of directory can be listed (**ls**)

w contents can be modified (add/delete files)

x change into directory is possible (**cd**)

Chapter 5 File Management

File System Implementation

Filesystems supported

- Traditional: ext2
- Second generation: ext3, ReiserFS, IBM JFS, xfs
- FAT-12, FAT-16, FAT-32, VFAT, NTFS (read-only)
- CD-ROM (ISO 9660)
- UMSDOS (UNIX-like FS on MS-DOS)
- NFS (Network File System)
- SMBFS (Windows share), NCPFS (Novell Netware share)
- /proc (for kernel and process information)
- SHMFS (Shared Memory Filesystem)

What is a filesystem?

- Place to store files and refer to them
- Hierarchical structure through use of directories
- A filesystem can be stored on any block device
  - Floppy disk
  - Hard disk
  - Partition
  - RAID, LVM volume
  - File (for use with a loop device)
  - RAM disk

Implementation of File System ——File System Layout

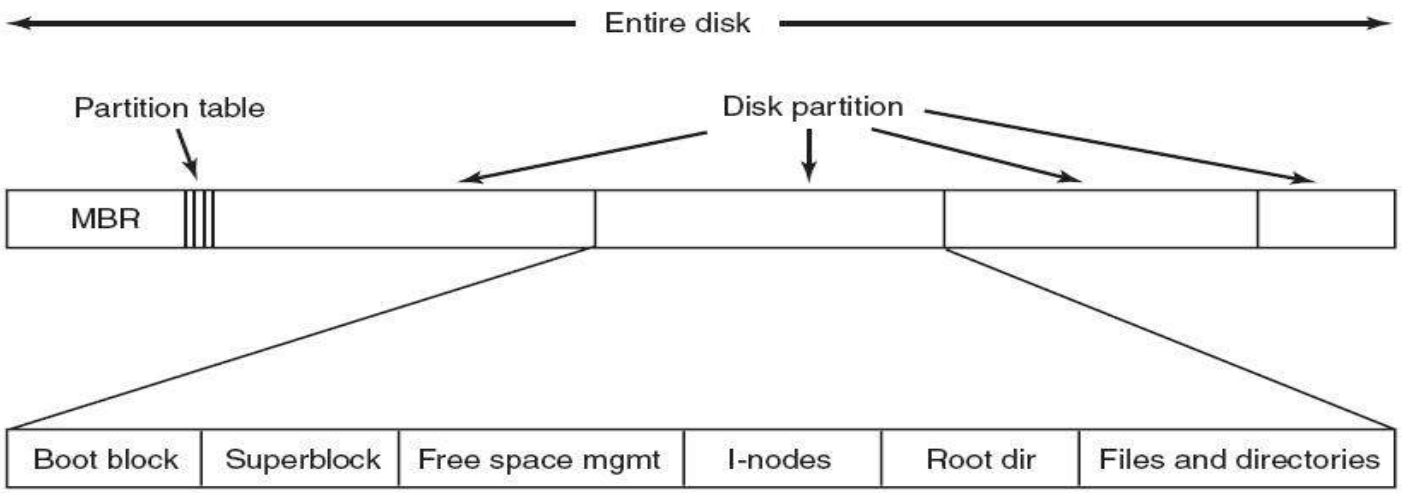


Figure 4-9. A possible file system layout.

Implementation of File System ——Implementing Files

Contiguous Allocation

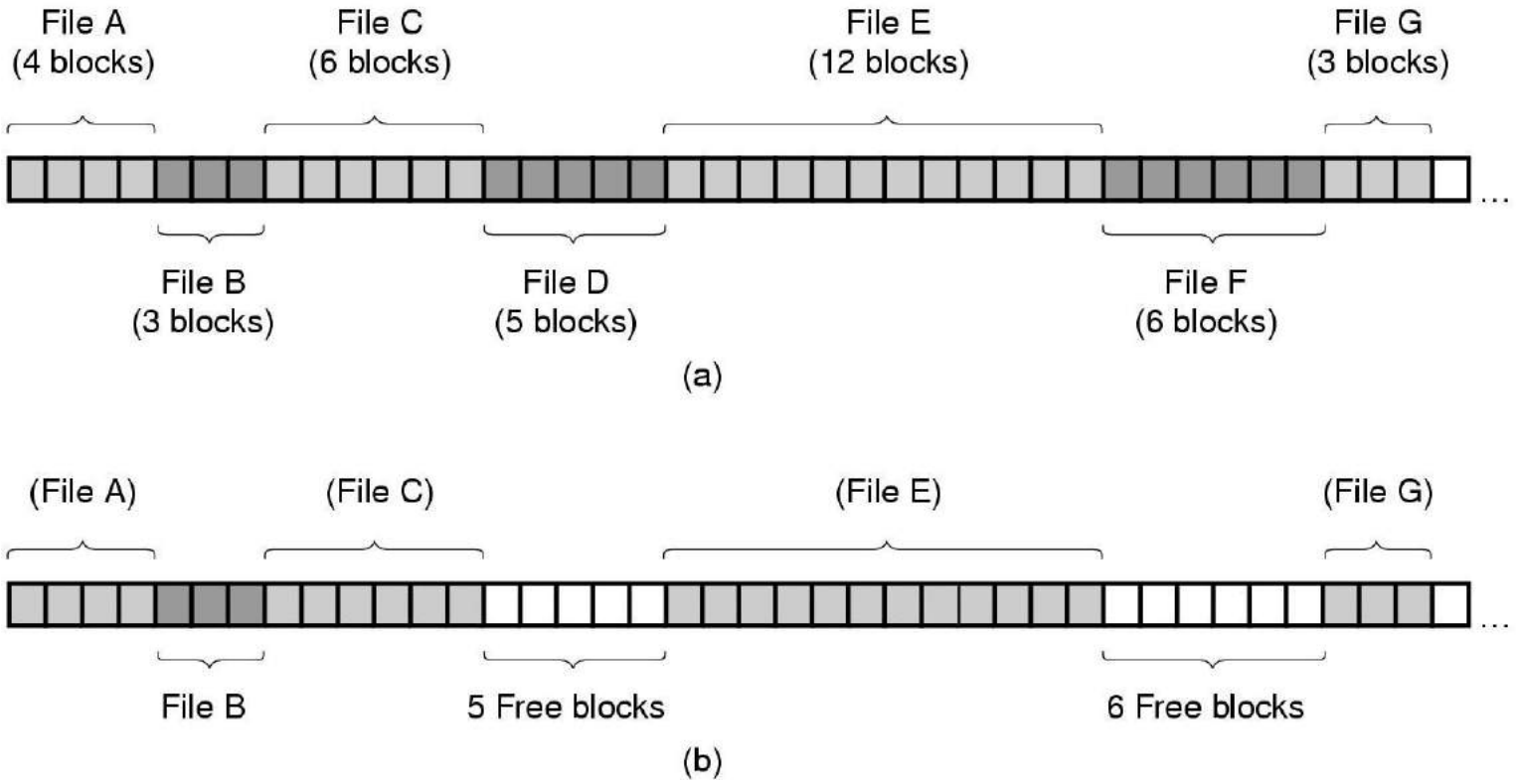


Figure 4-10. (a) Contiguous allocation of disk space for 7 files. (b) The state of the disk after files D and F have been removed.

Linked List Allocation

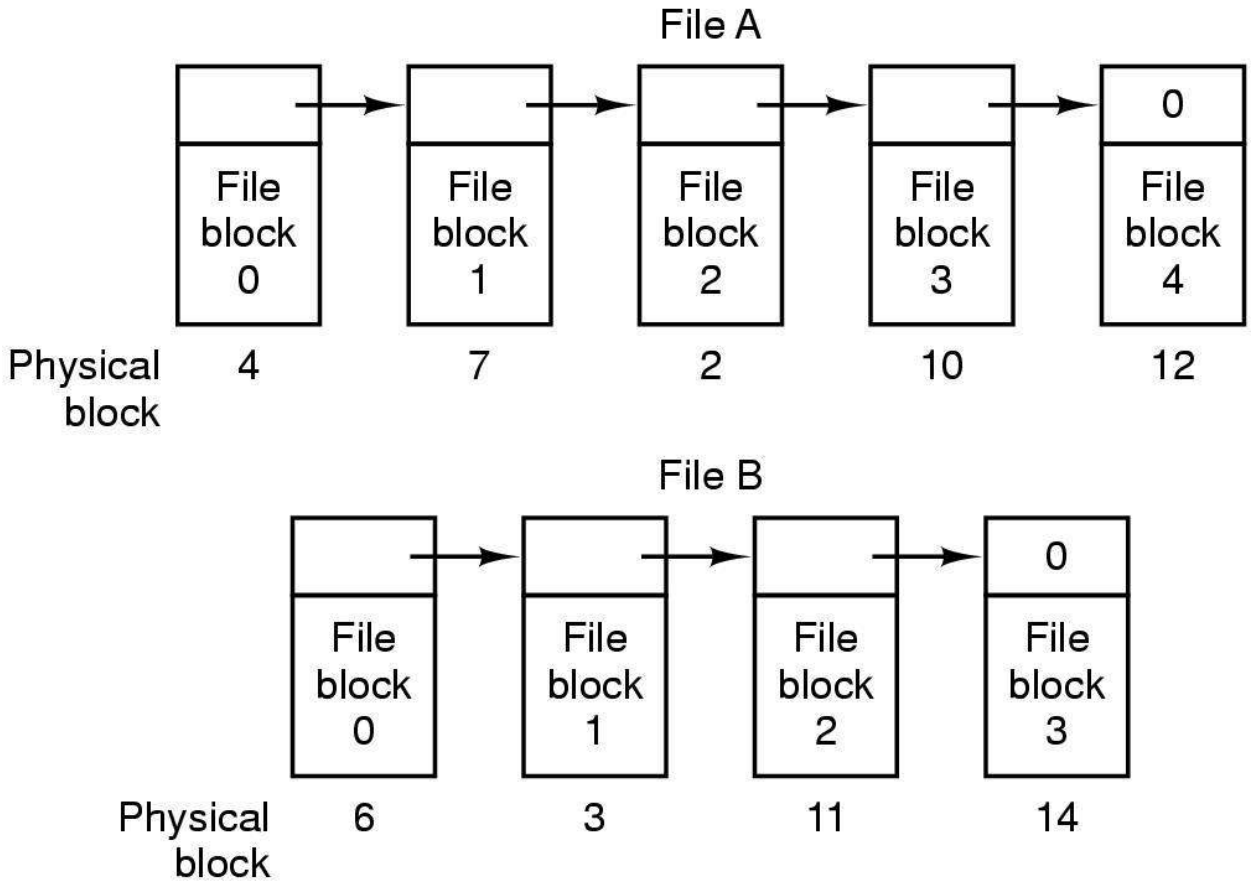


Figure 4-11. Storing a file as a linked list of disk blocks.

Linked List Allocation Using a Table in Memory

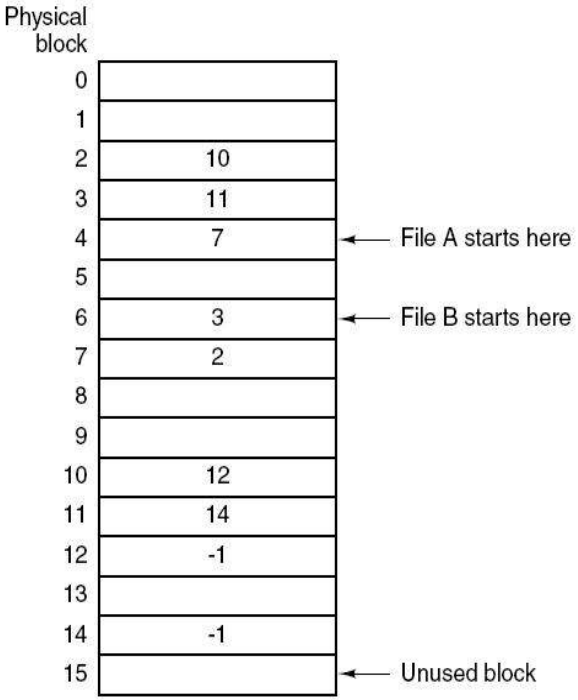


Figure 4-12. Linked list allocation using a file allocation table in main memory.

I-nodes

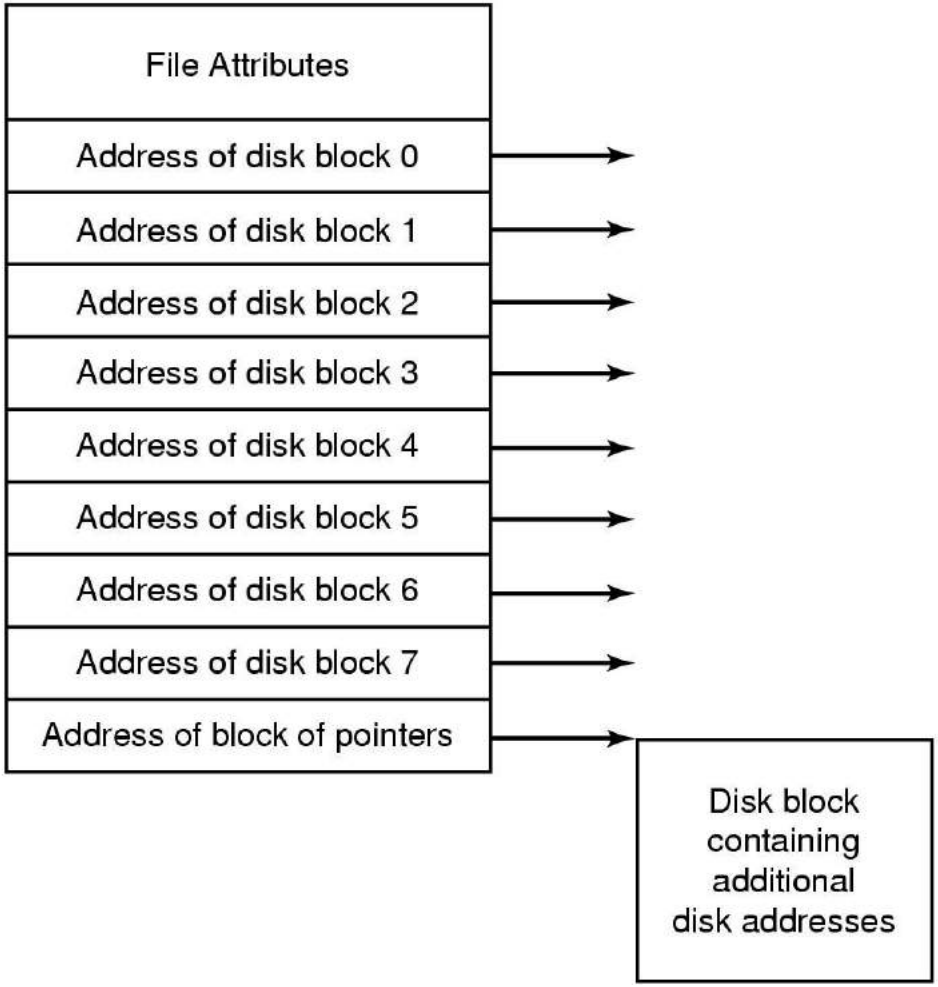
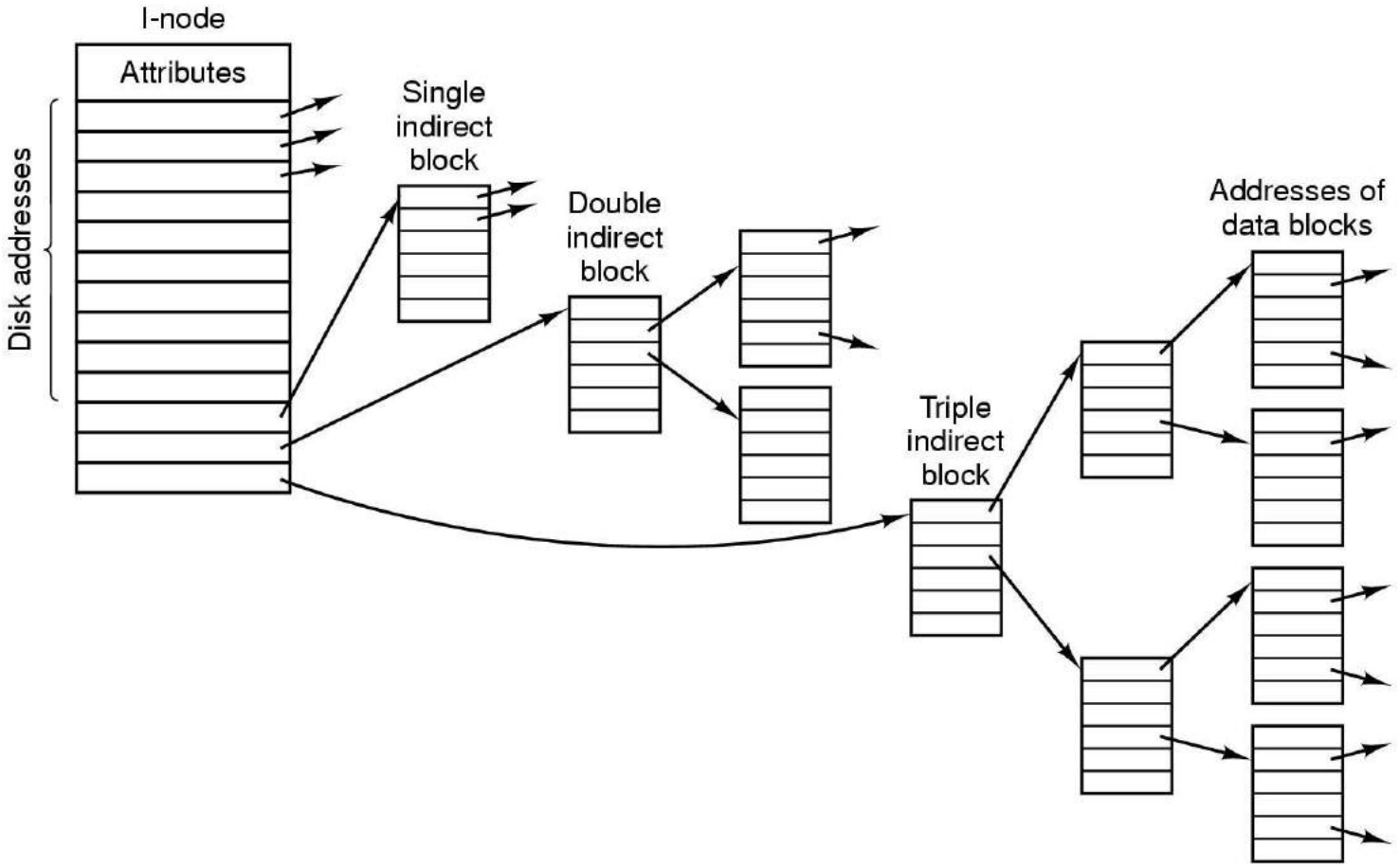


Figure 4-13. An example i-node.

i-node



A UNIX i-node.

Implementation of File System—— Implementing Directories (1)

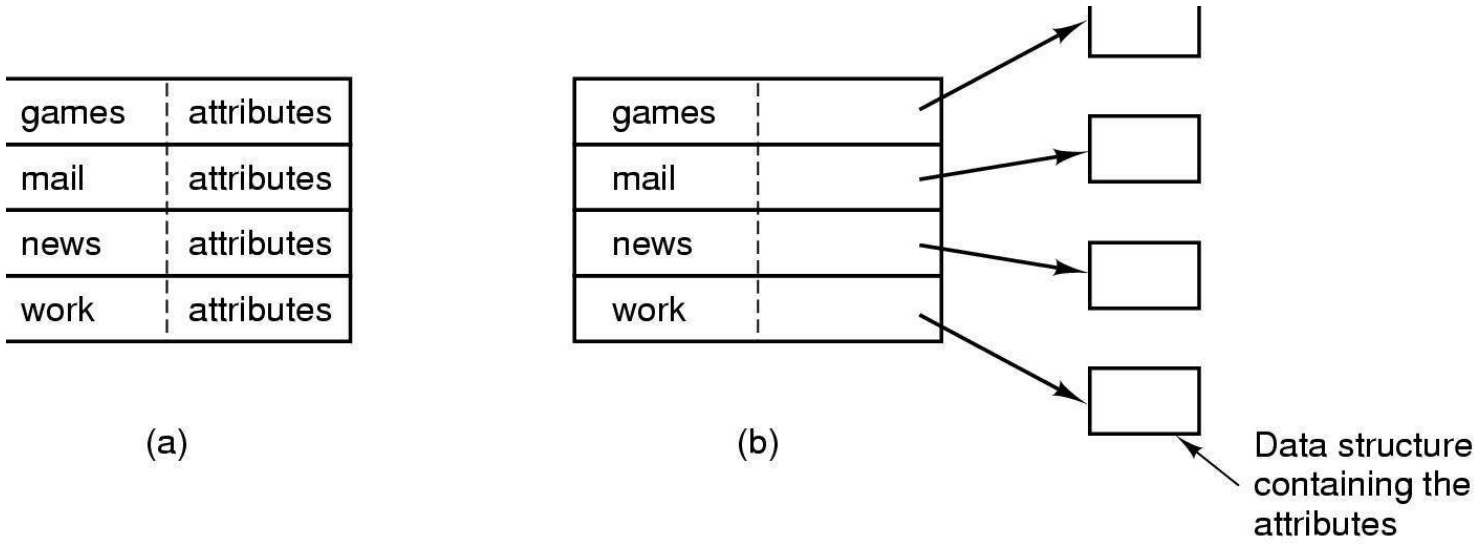


Figure 4-14. (a) A simple directory containing fixed-size entries with the disk addresses and attributes in the directory entry. (b) A directory in which each entry just refers to an i-node.

Implementing Directories (2)

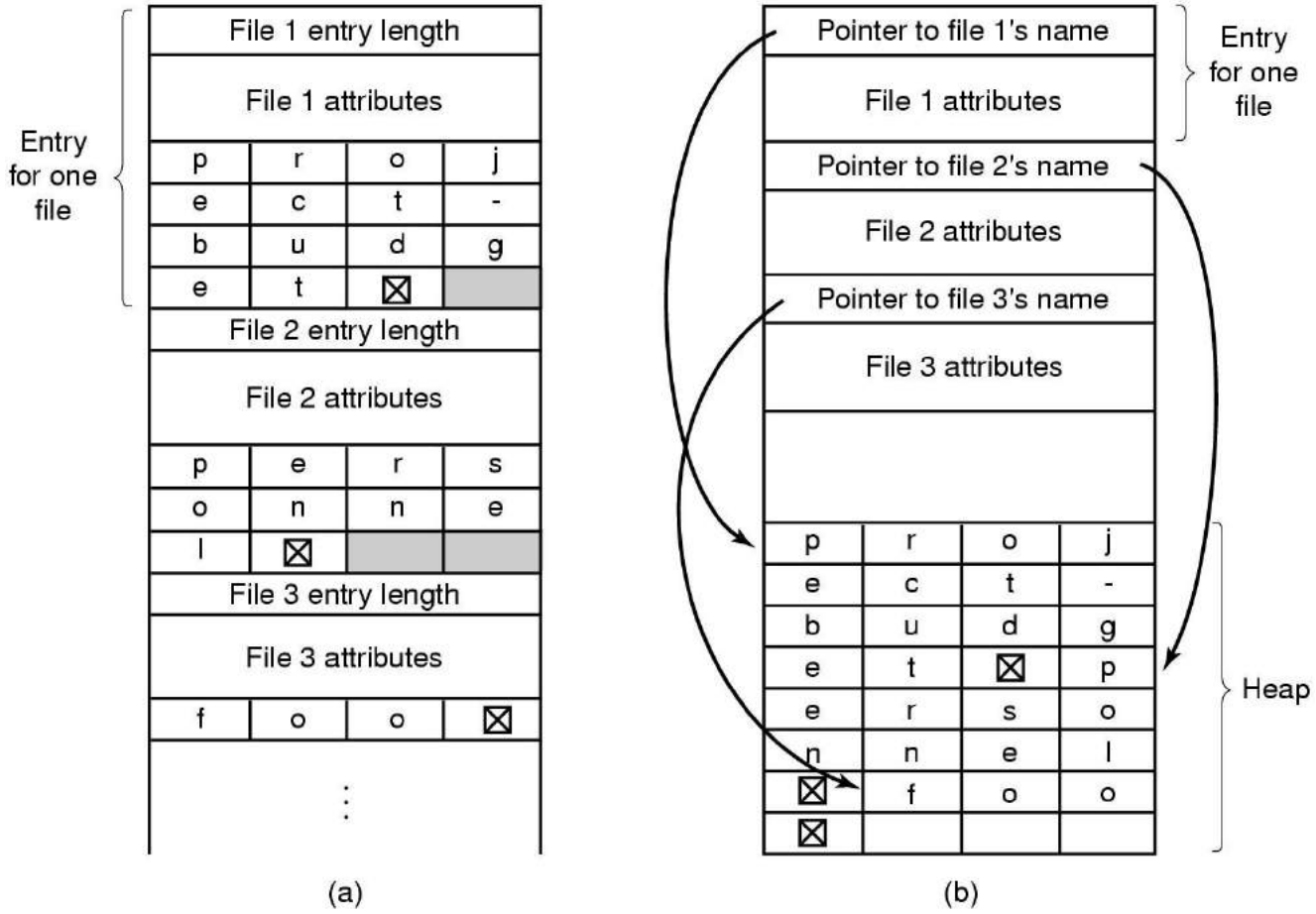


Figure 4-15. Two ways of handling long file names in a directory. (a) In-line. (b) In a heap.

Shared Files (1)

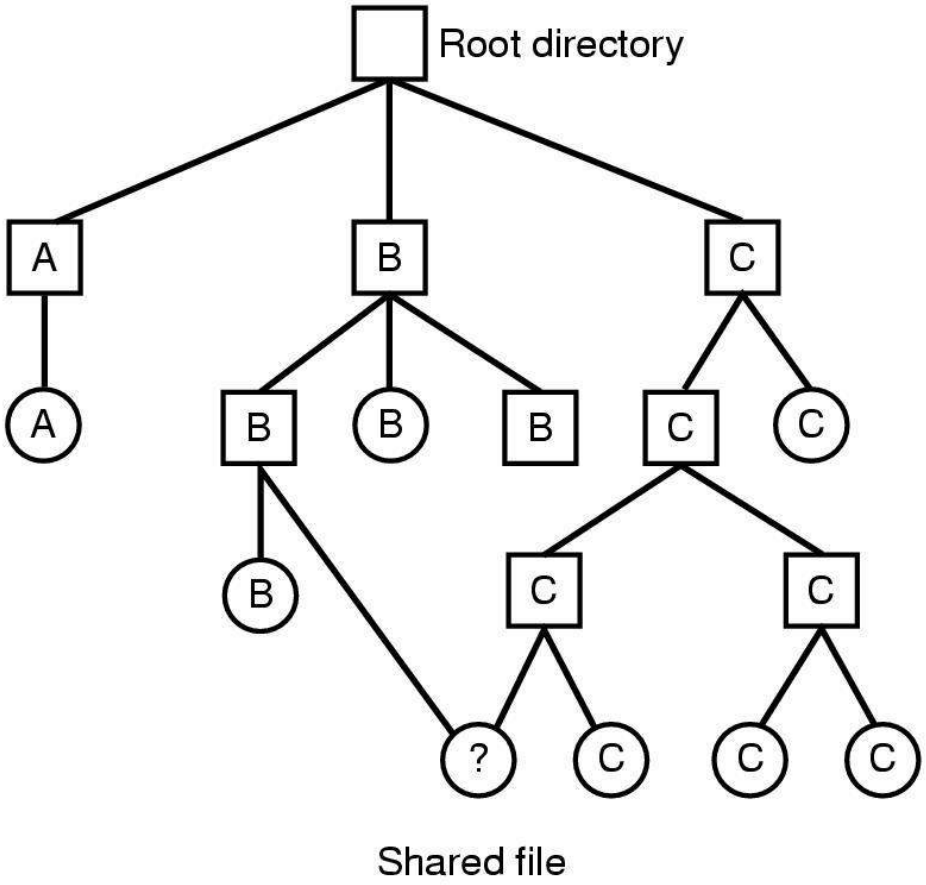


Figure 4-16. File system containing a shared file.

Shared Files (2)

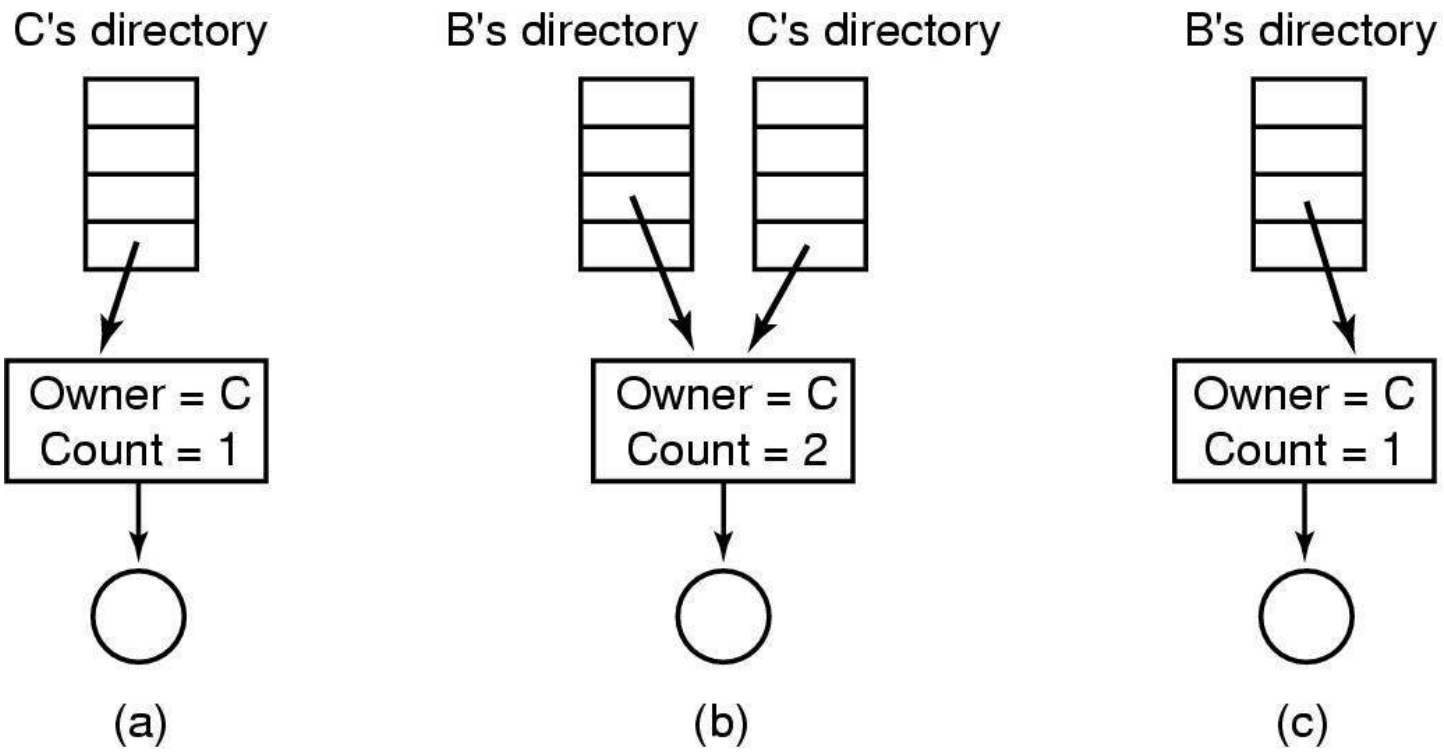
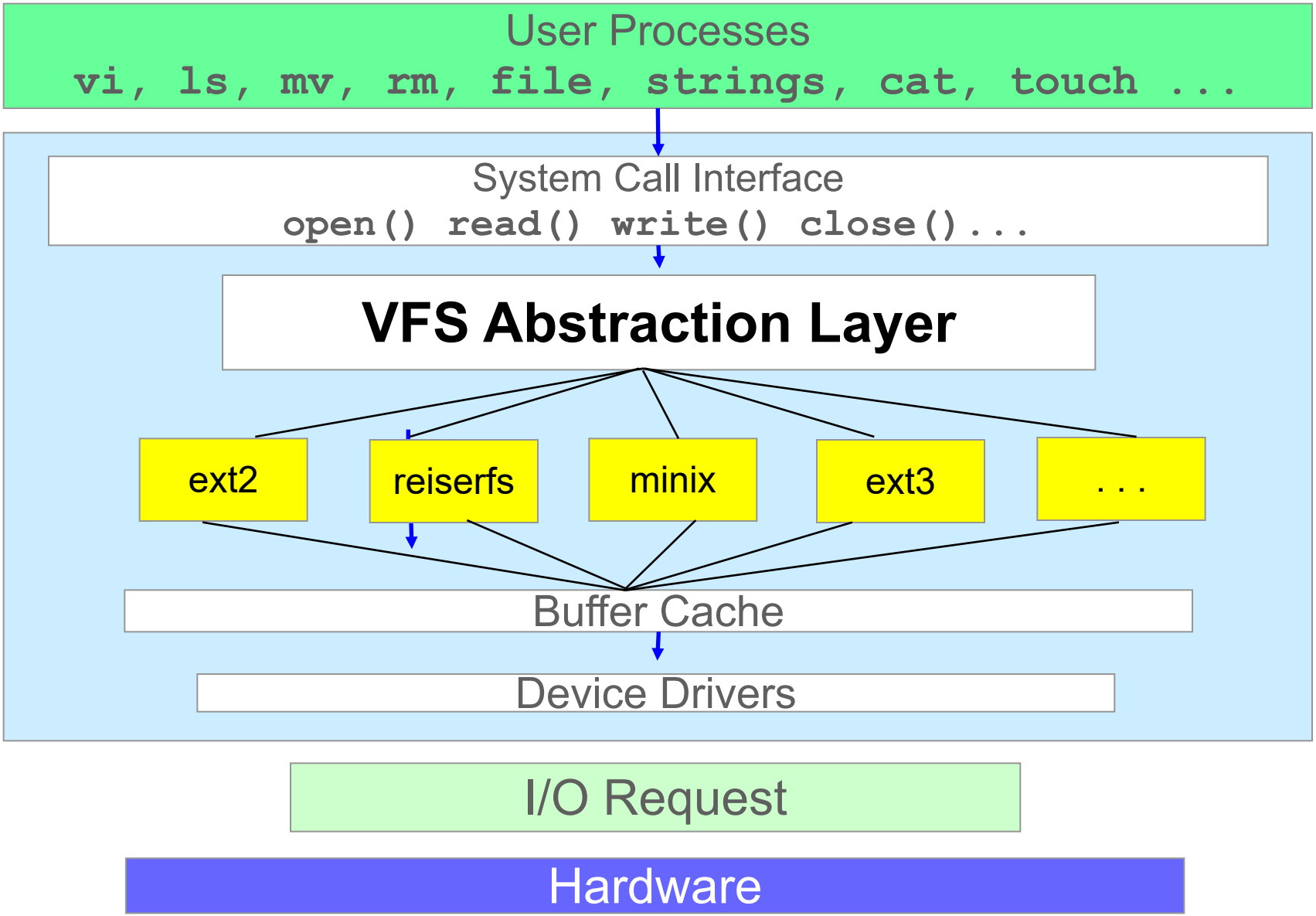


Figure 4-17. (a) Situation prior to linking. (b) After the link is created. (c) After the original owner removes the file.

The virtual file systems (1)



Virtual File Systems (2)

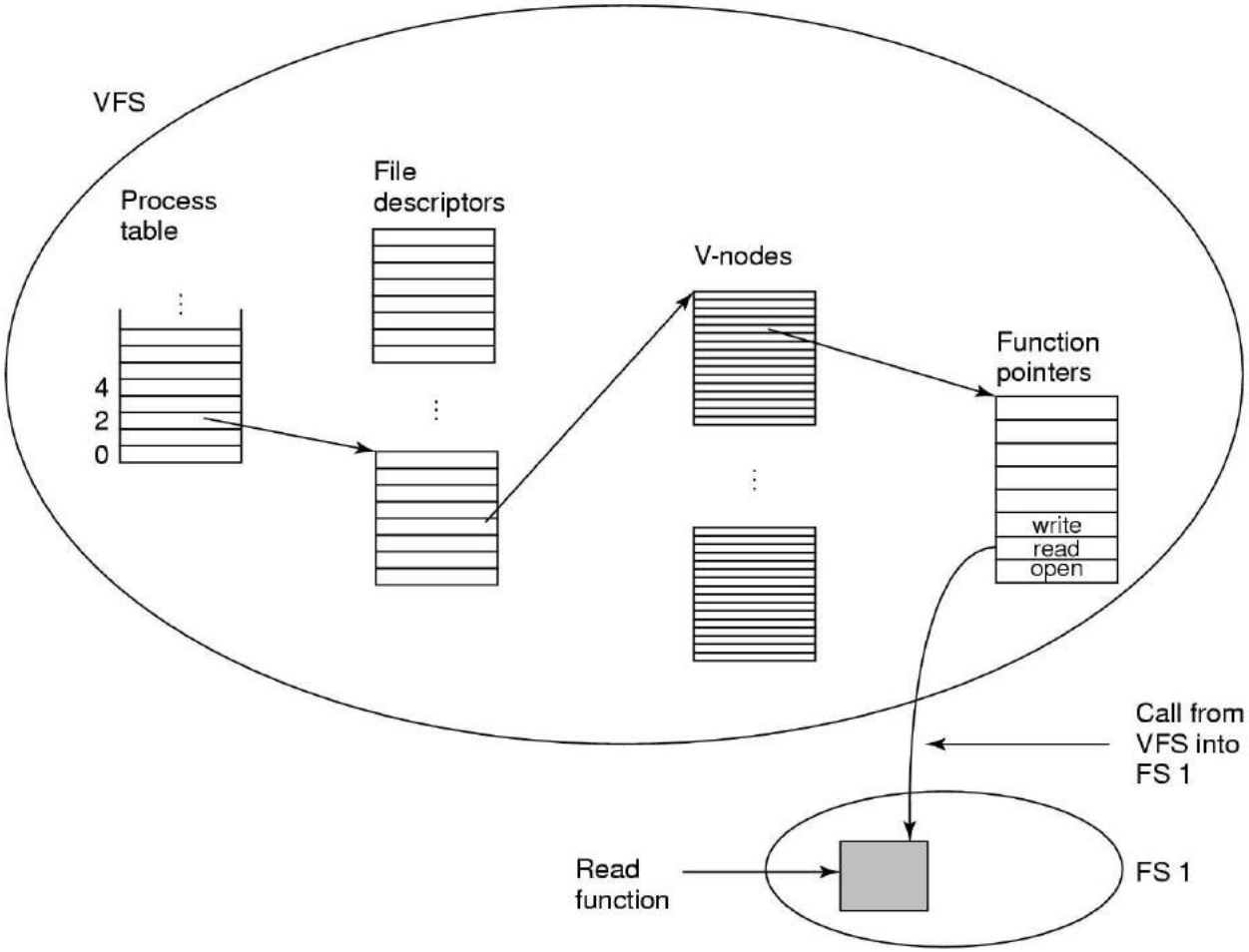


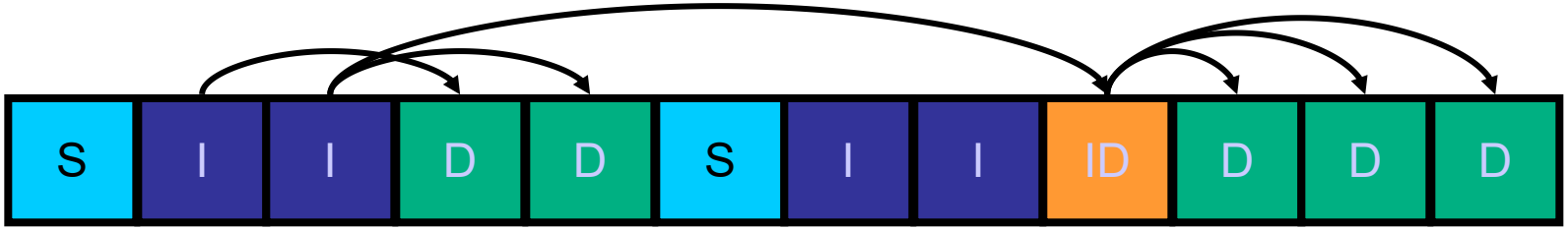
Figure 4-19. A simplified view of the data structures and code used by the VFS and concrete file system to do a read.

Filesystems supported

- Traditional: ext2
- Second generation: ext3, ReiserFS, IBM JFS, xfs
- FAT-12, FAT-16, FAT-32, VFAT, NTFS (read-only)
- CD-ROM (ISO 9660)
- UMSDOS (UNIX-like FS on MS-DOS)
- NFS (Network File System)
- SMBFS (Windows share), NCPFS (Novell Netware share)
- /proc (for kernel and process information)
- SHMFS (Shared Memory Filesystem)

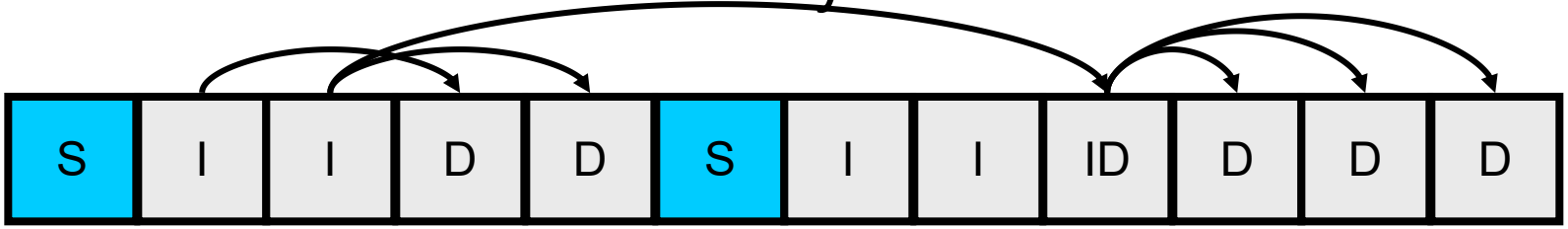
Filesystem example: ext2

- Partition divided into blocks of 1024, 2048 or 4096 bytes
  - Blocksize depends on size of filesystem and expected usage
- Blocks can have different usage:
  - Superblock
  - Index node (inode) block
  - Indirect block (double, triple)
  - Data block



Superblock

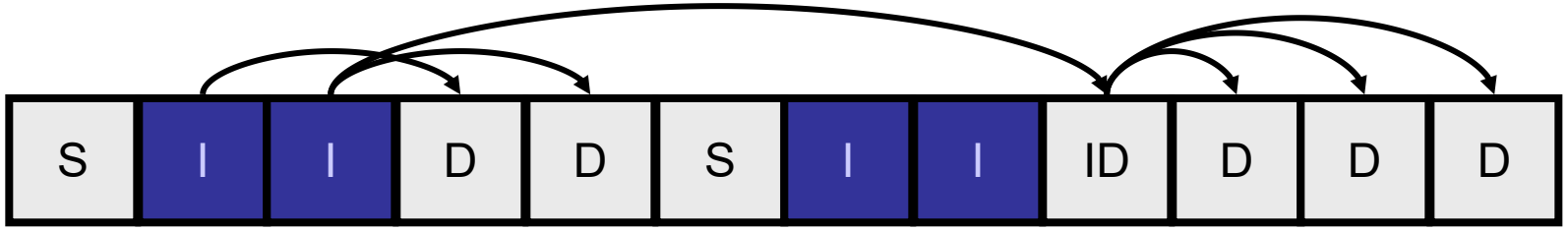
- **First block of filesystem, several copies (at 8193, 16385, ...)**
- **Contains general info on filesystem**
  - Last mounted time/place
  - Block size
  - Pointers to free inodes
  - Pointers to free blocks
  - Pointer to root of filesystem



Inodes

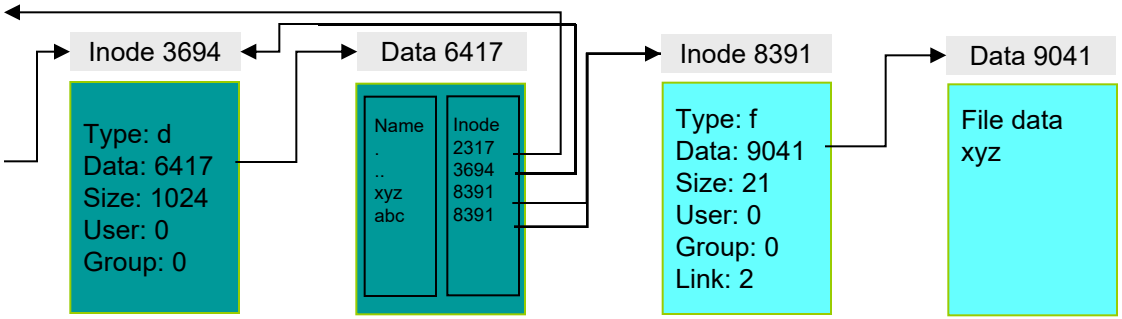
- 128 bytes （8 inodes per 1024 –Byte block ）
- Contains information about a file: owner, group, type, size, permissions, ctime, atime, mtime, ...
- Contains pointers to data blocks
- Contains pointers to an indirect block, a double indirect block, and a triple indirect block

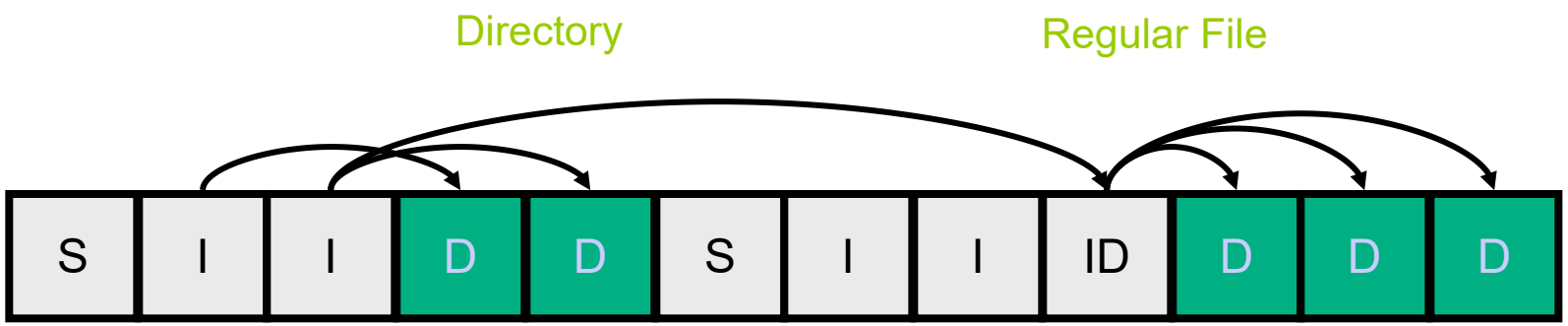
Owner / Group
File Type
File Size
File Permissions
Time Stamps: create time access time modification time
Link Counter
Additional Flags: (ACL, EXT2,_FLAGS)
Pointers to Block Data



Data blocks

- Contain file data
- File may be a directory, in which case the data is the list of file names and inodes in that directory
- Multiple file names may point to the same inode! (Or files may have multiple names)





Ext2fs summary

- The most important components of a filesystem are the inodes and the data blocks
- The filesystem is full if:
  - No more inodes are available
  - No more data blocks are available
- So tune your filesystem according to the number of bytes per file:
  - Blocksize (1024, 2048, or 4096 possible)
  - Bytes-per-inode (4096 default)

Other filesystem features

Filesystems can have other features that can be useful:

- Access Control Lists (ACL)
  - Allow more extended permissions, not just rwxrwxrwx
  - Not yet supported by VFS abstraction layer
- Journaling
  - Keeps a journal of operations that are going to take place and operations that were successfully committed
  - Should make recovery from a crash faster
  - Slight performance decrease
- Extended file attributes
  - Examples: immutable, auto compression, undeletable
- Labels
  - Allow mounting based on label instead of device name
- Performance optimizations

作业

- P182　9，11，19，20

Management and Optimization of File System

Disk Space Management Block Size (1)

Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69

Length	VU 1984	VU 2005	Web
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Figure 4-20. Percentage of files smaller than a given size (in bytes).

Disk Space Management Block Size (2)

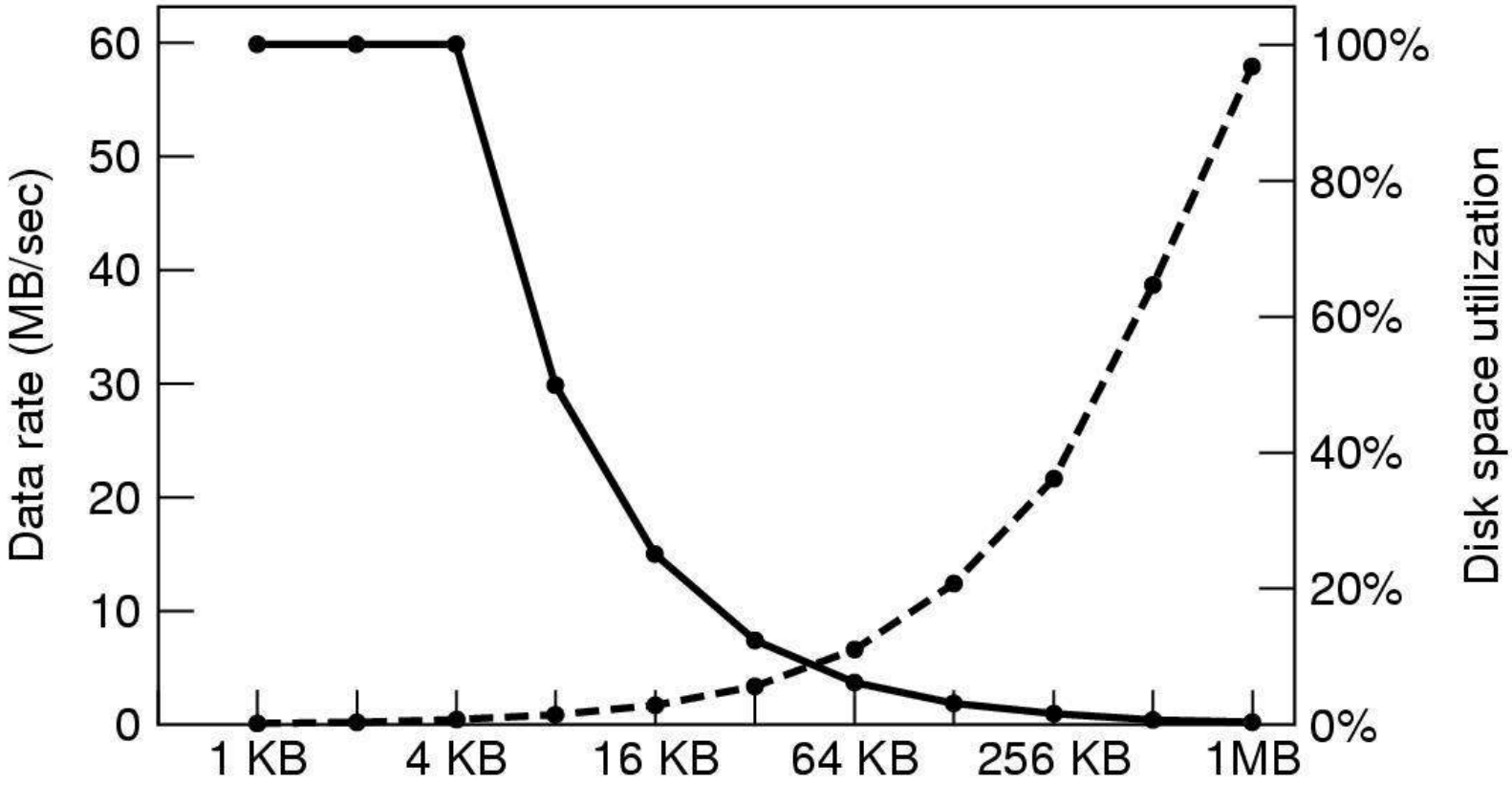
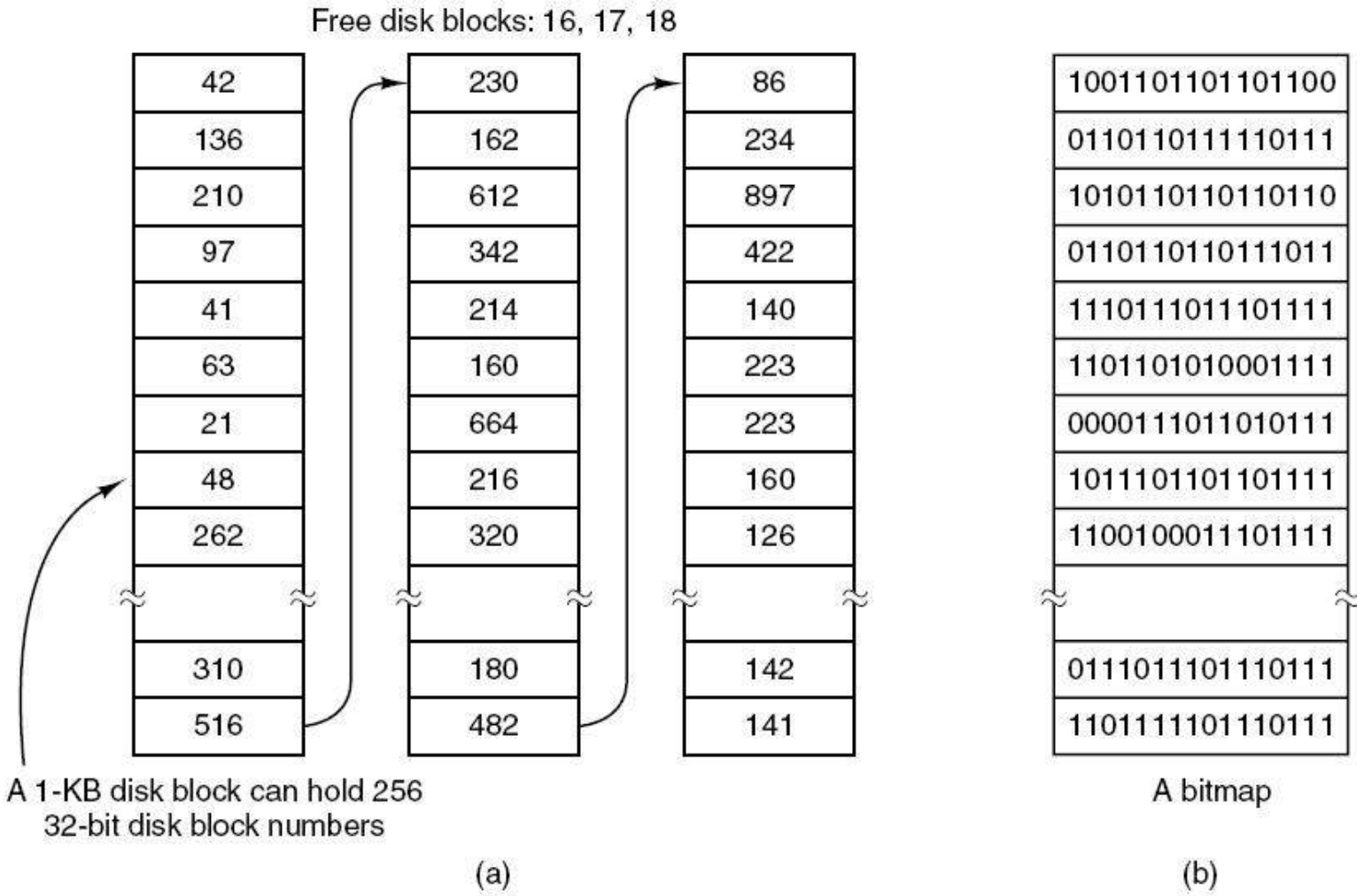


Figure 4-21. The dashed curve (left-hand scale) gives the data rate of a disk. The solid curve (right-hand scale) gives the disk space efficiency. All files are 4 KB.

Keeping Track of Free Blocks



(a) Storing the free list on a linked list. (b) A bitmap.

Creating a filesystem

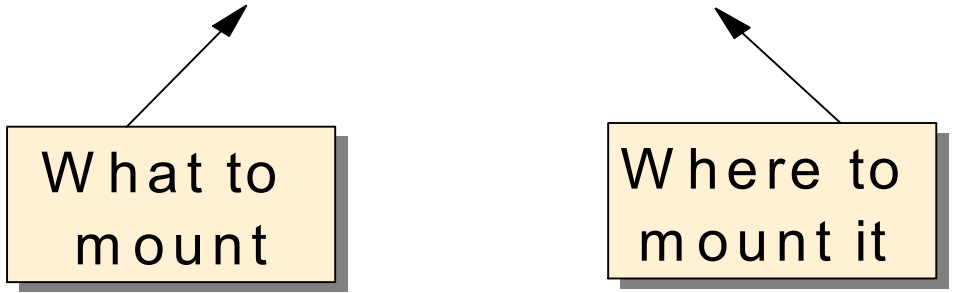
- Creating a filesystem is done with an **mkfs** variant
  - **mke2fs**, **mke2fs -j**
  - **mkreiserfs**
  - **mkjfs**
- Typical options:
  - **b** blocksize sets blocksize
  - **i** bytes-per-inode sets number of inodes
  - **c** checks disk for bad blocks
- Example:

```
# mke2fs -b 1024 -i 4096 -c /dev/sda6
...
Writing inode tables: done
Writing superblocks and filesystem accounting
info: done
...
```

Mount

- **`mount`** is the glue that logically connects file systems to the directory hierarchy
- File systems are associated with devices represented by special files in **/dev** (the logical volume)
- When a file system is mounted, the logical volume and its contents are connected to a directory in the hierarchical tree structure

```
# mount  /dev/lv00  /home/patsie
```



Comparing filesystems

Journaled Filesystems used by Linux:

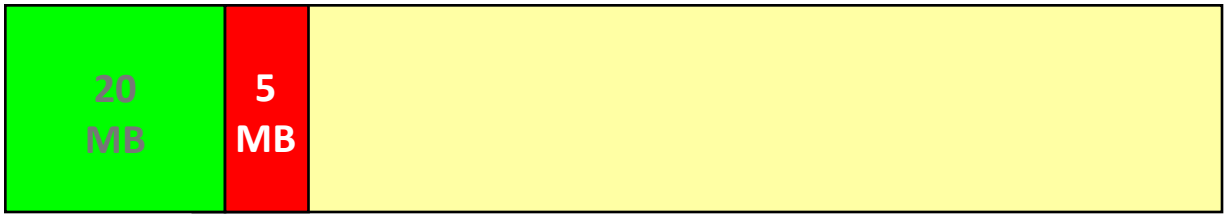
	ext2	ext3	jfs	reiser	xfs
Journal	no	yes (10 MB default)	yes (auto resized)	yes (32 MB default)	yes
resizeable	yes, but only when unmounted	yes, but only when unmounted	yes	yes	yes, but only when mounted
maximum size	File: 2 TB FS: 16 TB	File: 2 TB FS: 16 TB	File: 4 PB FS: 32 PB	File: 16 TB FS: 1 EB	File: 2 TB FS: 8 EB
type	inodes (completely block oriented)	inodes (completely block oriented)	inodes (allocated in a b-tree)	b-Tree	inodes (allocated in a b-tree)

SHMFS-specific information

- SHMFS: POSIX compliant Shared Memory Filesystem
- Filesystem stored in memory, expands when used to required size
- Not persistent across reboot
- Typically mounted on /dev/shm
- Required by certain applications

Quota concepts

- Quotas limit the amount of data a user/group is allowed to store
- Defined on a per-filesystem basis
- Based on block and/or inode usage per user or group
- Two limits per quota: Soft and hard
  - User exceeds soft limit ⇨ warning only
  - User exceeds hard limit ➡ error
- Grace period identifies how long the soft limit may be exceeded
  - After that period, a user gets errors instead of warnings



Filesystem: 300 MB

Each user may consume only 20 MB permanently and 25 MB temporarily

Disk Quotas

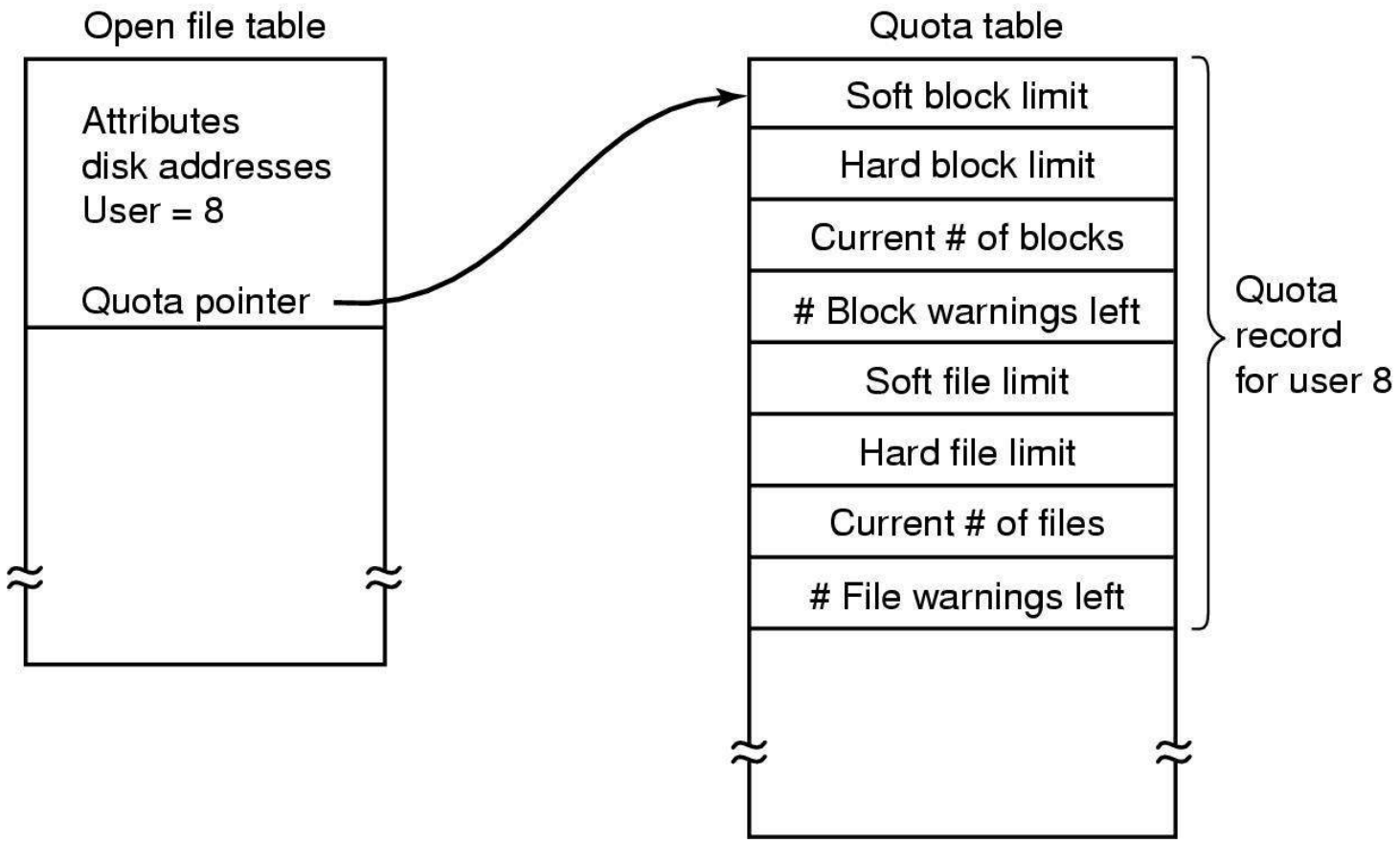


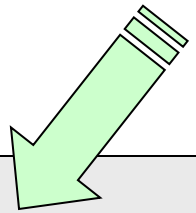
Figure 4-24. Quotas are kept track of on a per-user basis in a quota table.

Quota implementation on Linux

- Quota support compiled into the kernel
  - No daemon necessary
- Implemented on a per-filesystem basis
  - A user can have different quota on different filesystems
  - Stored in aquota.user and aquota.groups in the root of the filesystem
- Quota checking should be enabled when mounting the filesystem
  - Mount options: **usrquota**, **grpquota**
  - Can be specified in /etc/fstab
- Quota checking should be turned on after mounting with the **quotaon** command
  - Automatically executed from bootscript after **mount -a**

Enabling quota

- Modify /etc/fstab



```
/dev/sda2 /           ext3    defaults                    1 1
/dev/sda4 /home       ext3    defaults,usrquota,grpquota  1 2
/dev/sdb  /mnt/cdrom  iso9660 noauto,owner,ro             0 0
/dev/sda3 swap        swap    defaults                    0 0
/dev/fd0  /mnt/floppy msdos   noauto,owner                0 0
none      /proc       proc    defaults                    0 0
none      /dev/pts    devpts  gid=5,mode=620              0 0
```

- Create aquota.user and aquota.group in the filesystem's root directory
- Remount the partition
- Calculate current usage and turn on quota checking

```
# touch /home/aquota.user /home/aquota.group
# mount -o remount /home
# quotacheck /home
# quotaon /home
```

Configuring quota

- Done with the **edquota** command
  - Starts **$EDITOR** (default: **vi**) in a subshell
  - Only edit the block/inode soft/hard quota numbers
- User quota: **edquota -u *username***

```
Disk quotas for user tux1 (uid 501):
Filesystem     blocks    soft    hard     inodes    soft    hard
/dev/sda4       10700   20000   25000        407       0       0
/dev/sda9         320     300     350         23      30      50
~
~
~
"/tmp/Edp.a9fSEQK" 3L, 213C
```

- Group quota: **edquota -g *groupname***
- Grace period: **edquota -t**
- Copy quota: **edquota -p tux1 -u tux2 tux3 tux4**

Quota information

- **quota** command
  - Reports on the quota of one user
  - Can be executed by anyone
  - A regular user can only view his own quota

```
tux1$ quota
Disk quotas for user tux1 (uid 501):
Filesystem  blocks  quota  limit  grace  files   quota   limit   grace
/dev/sda4    10700  20000  25000            407       0       0
```

- **repquota** command
  - Reports on the quota of all users and groups
  - Can only be executed by root

```
root# repquota /dev/hda4
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
root      --  848804       0       0          56892     0     0
.
tux1      ++    1500    1000    1500  7days      112   112   115   none
tux2      --     176    1000    1500                44     0     0
```

File System Backups (1)

Backups to tape are generally made to handle one of two potential problems:

- Recover from disaster.
- Recover from stupidity.

Physical Backup VS Logical Backup

Backup schemes

- Full backup
  - Preserves the whole system
- System backup
  - Preserves system directories and files
  - Must include backup/restore tools
  - Usually on bootable media (floppy, optical)
- Data backup
  - Preserves user data
- Incremental or differential backup
  - Only backup files that changed
  - Very fast, but takes more time to restore
  - Must be used carefully
  - Needs more media

File System Backups (2)

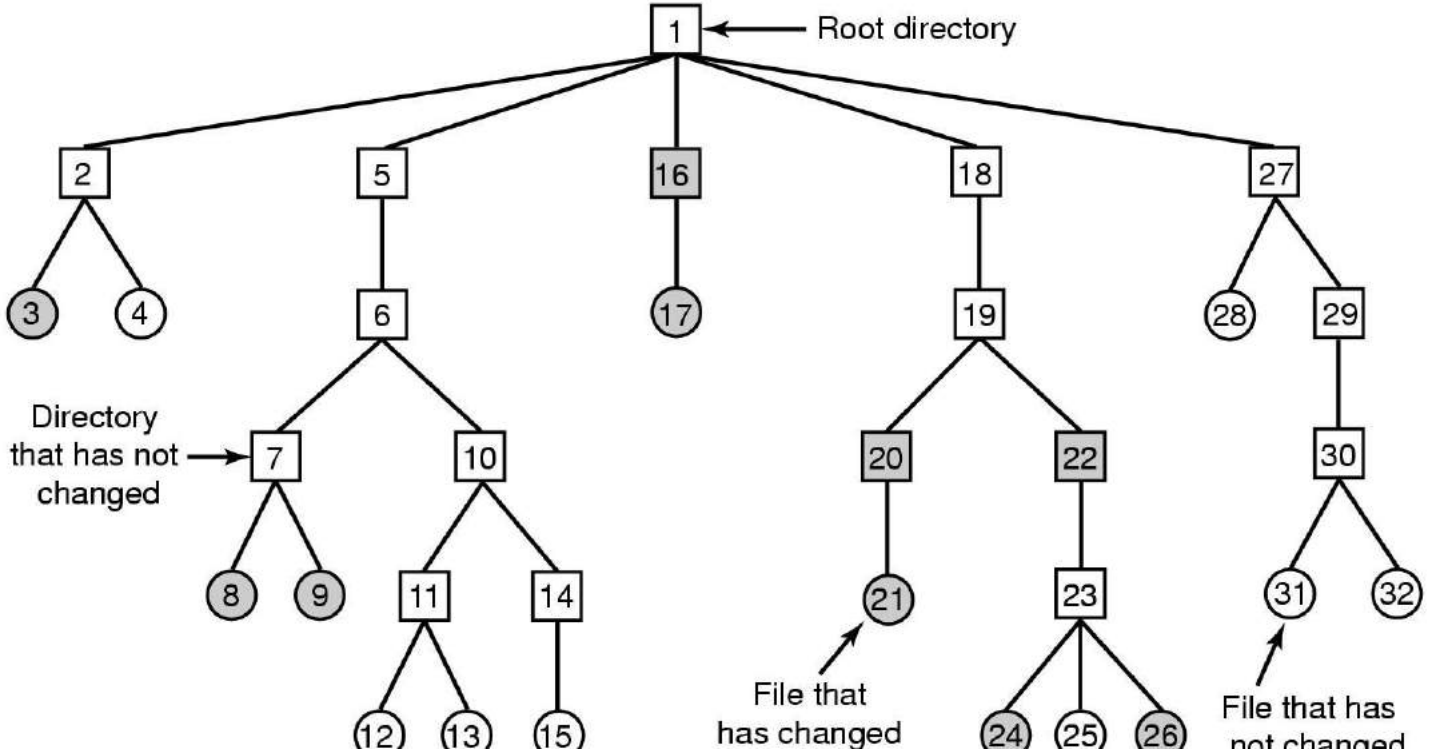


Figure 4-25. A file system to be dumped. Squares are directories, circles are files. Shaded items have been modified since last dump. Each directory and file is labeled by its i-node number.

File System Backups (3)

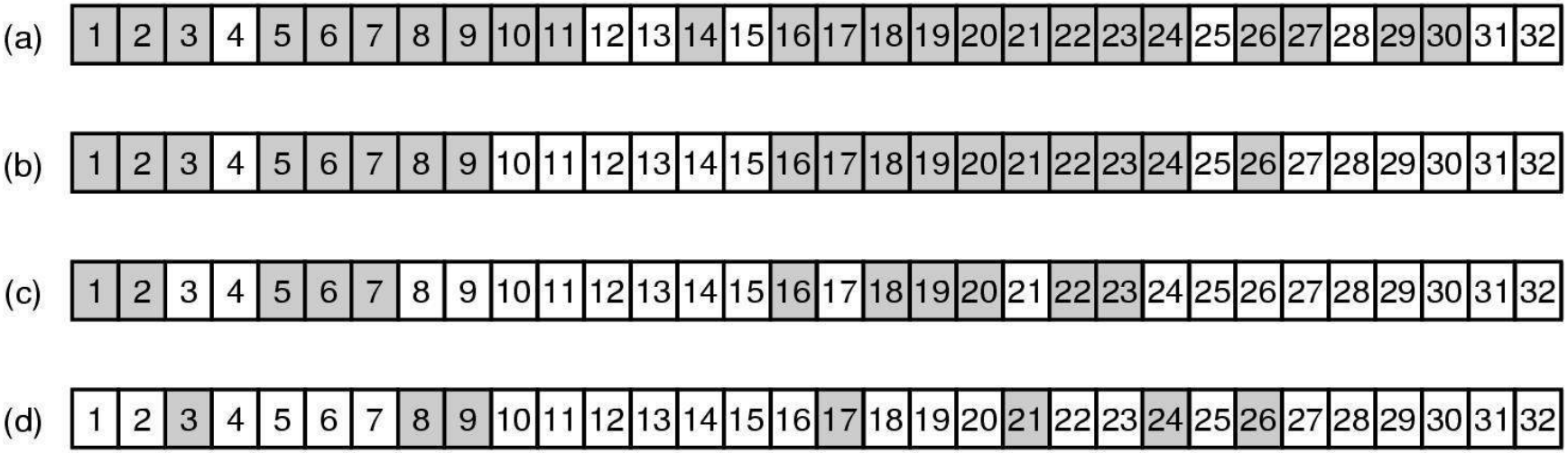
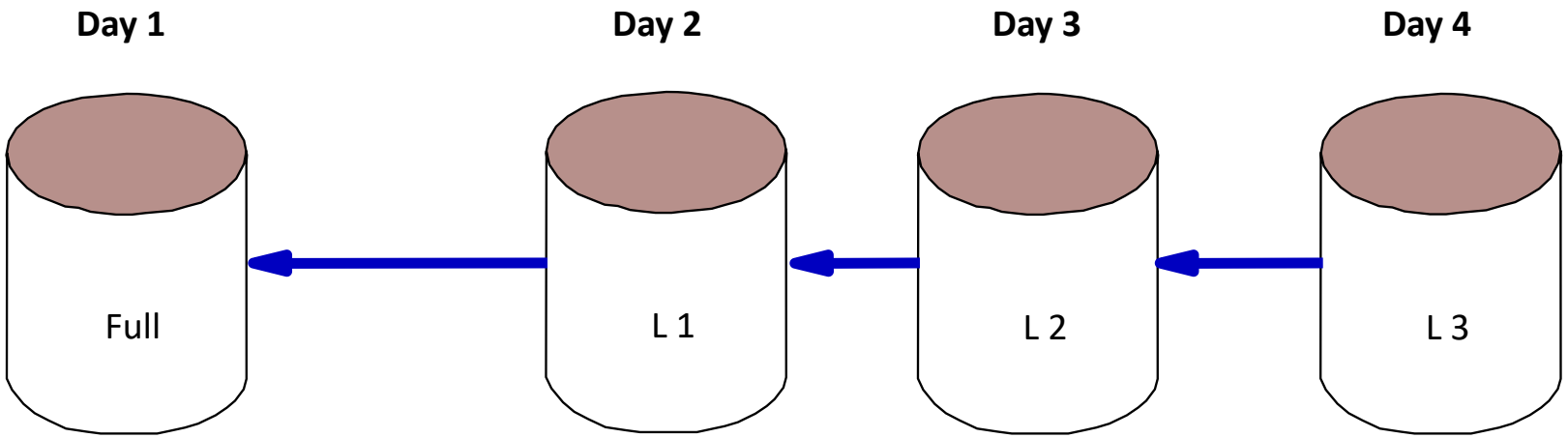
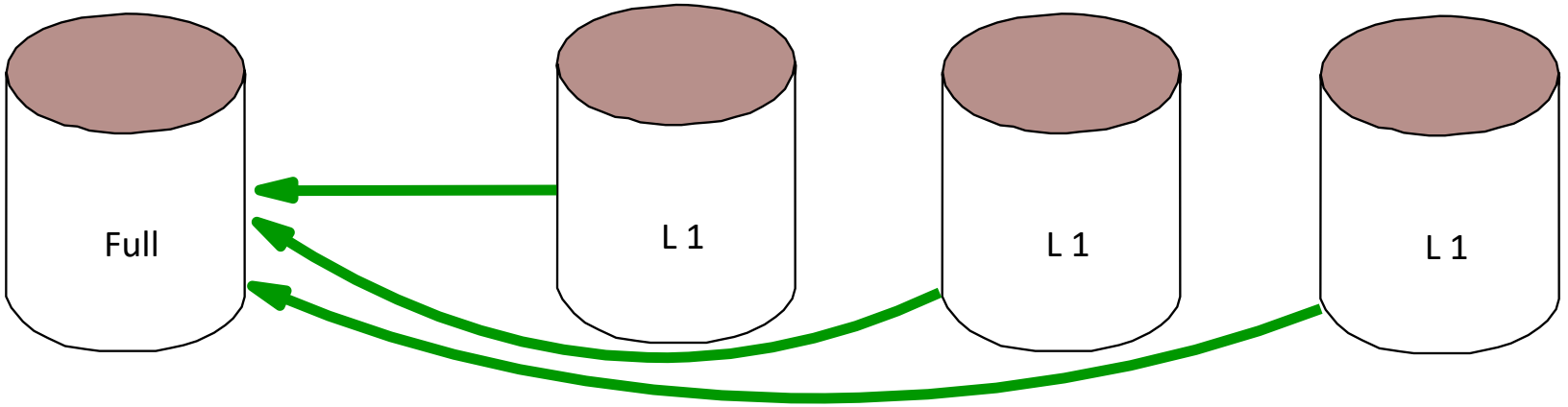


Figure 4-26. Bitmaps used by the logical dumping algorithm.

Incremental versus differential backup



Incremental backup



Differential backup

Sample monthly backup scheme

Su	Mo	Tue	We	Thu	Fr	Sa
					1 *Level 0*	2
3	4 *Level 2*	5 *Level 3*	6 *Level 4*	7 *Level 5*	8 *Level 1*	9
10	11 *Level 2*	12 *Level 3*	13 *Level 4*	14 *Level 5*	15 *Level 1*	16
17	18 *Level 2*	19 *Level 3*	20 *Level 4*	21 *Level 5*	22 *Level 1*	23
24	25 *Level 2*	26 *Level 3*	27 *Level 4*	28 *Level 5*	29 *Level 0*	30
31						

Backup devices

- Tape drive
  - Large capacity, fast
  - Requires new tapes regularly
- CD-R, CD-RW, DVD
  - Cheap but relatively slow
- (Removable) Hard disk
  - Fast but expensive
- Diskette drive
  - Often available but cumbersome for large backups
- Network
  - Useful in large installations; usually requires commercial software (for instance, Tivoli Storage Manager)

Default backup tools

- **`tar`**
  - Backs up individual files
  - Widely available
  - Excellent for transferring data between platforms
- **`cpio`**
  - Backs up individual files
  - Widely available
  - Difficulties with many symbolic links
- **`dump`**
  - Backs up whole filesystems
  - Can handle incremental backups (nine levels)
- **`dd`**
  - Useful for making bit-for-bit dumps of disks and filesystems

File System Consistency

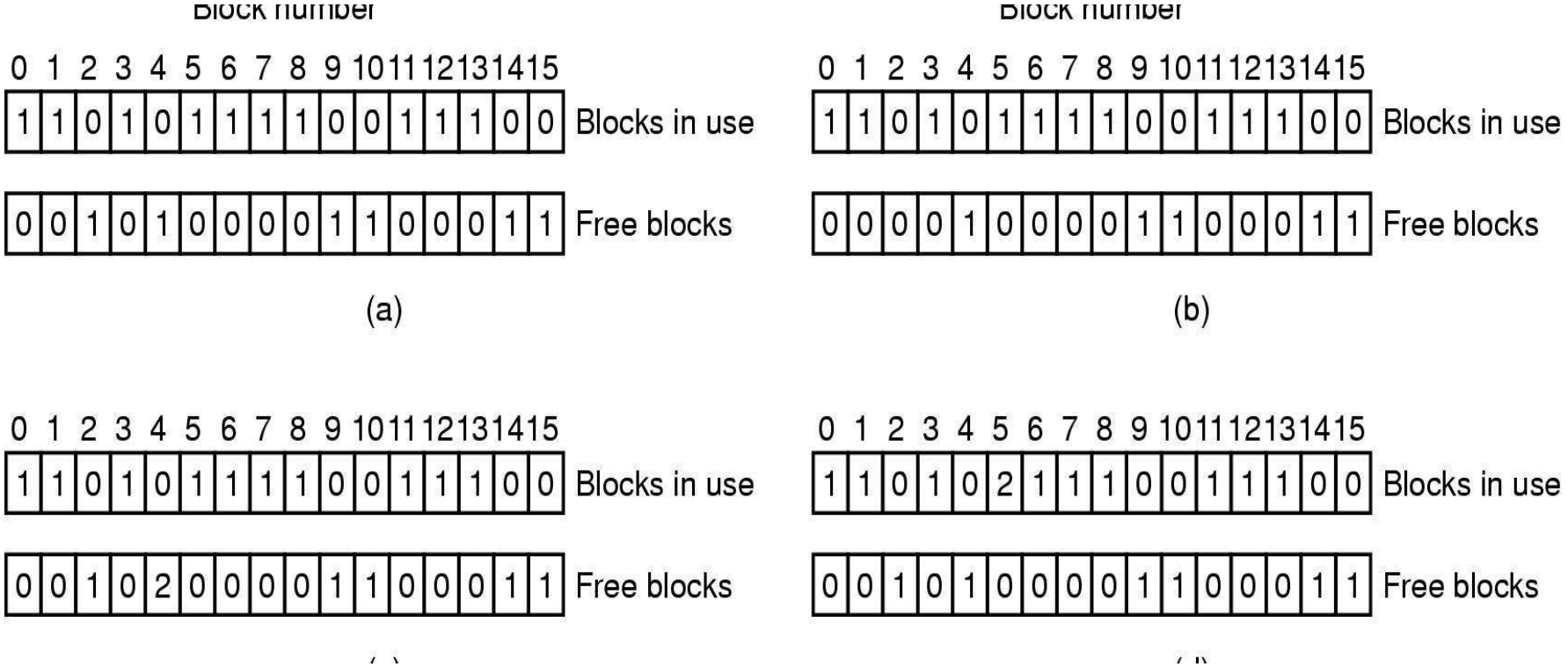


Figure 4-27. File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block.

Caching (1)

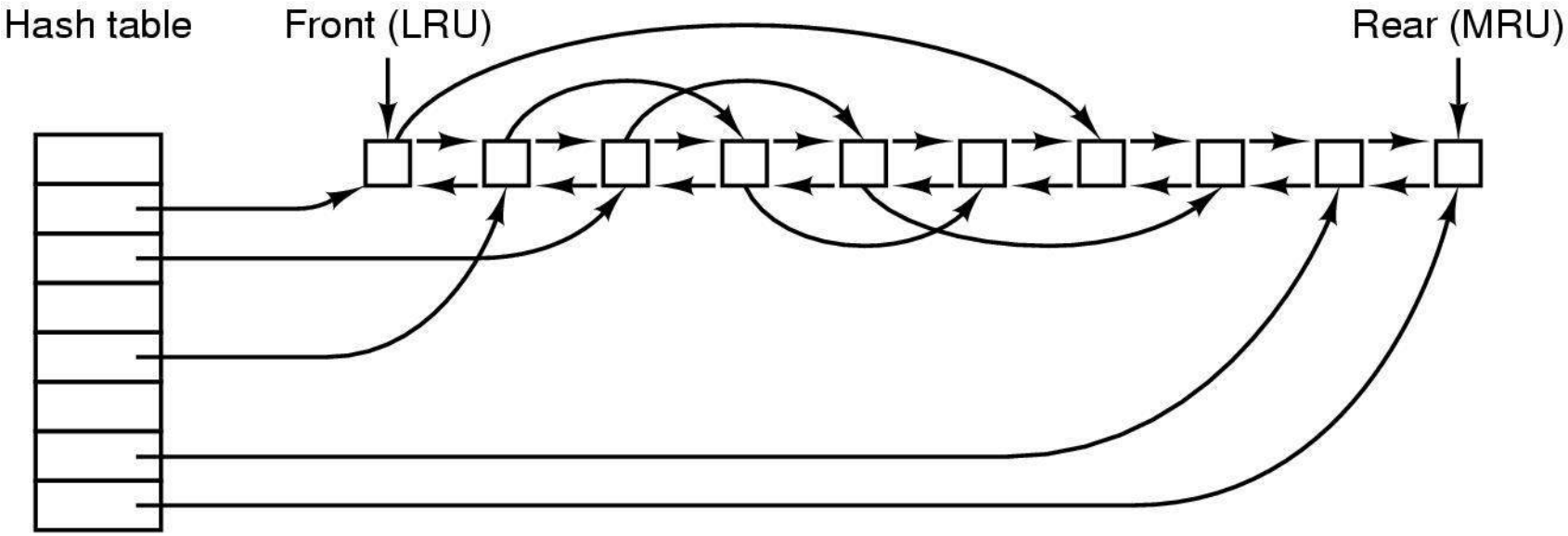


Figure 4-28. The buffer cache data structures.

Caching (2)

- Some blocks, such as i-node blocks, are rarely referenced two times within a short interval.
- Consider a modified LRU scheme, taking two factors into account:
  - Is the block likely to be needed again soon?
  - Is the block essential to the consistency of the file system?

Reducing Disk Arm Motion

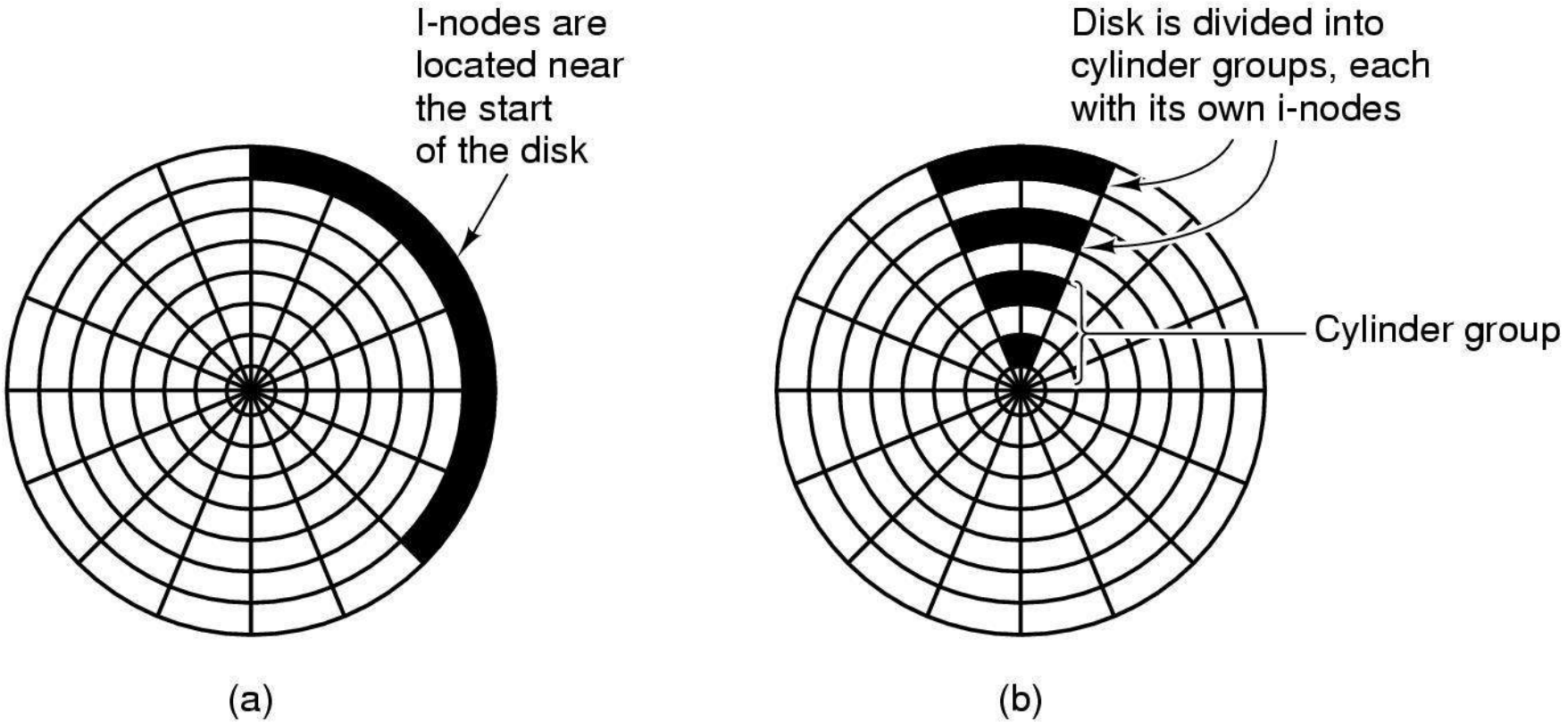


Figure 4-29. (a) I-nodes placed at the start of the disk. (b) Disk divided into cylinder groups, each with its own blocks and i-nodes.

The ISO 9660 File System

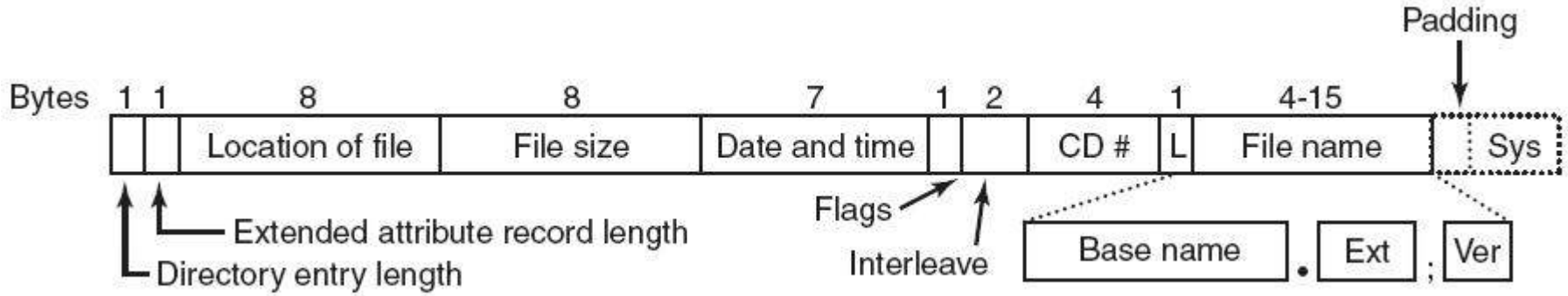


Figure 4-30. The ISO 9660 directory entry.

Rock Ridge Extensions

Rock Ridge extension fields:

- PX - POSIX attributes.
- PN - Major and minor device numbers.
- SL - Symbolic link.
- NM - Alternative name.
- CL - Child location.
- PL - Parent location.
- RE - Relocation.
- TF - Time stamps.

Joliet Extensions

Joliet extension fields:

- Long file names.
- Unicode character set.
- Directory nesting deeper than eight levels.
- Directory names with extensions

The MS-DOS File System (1)

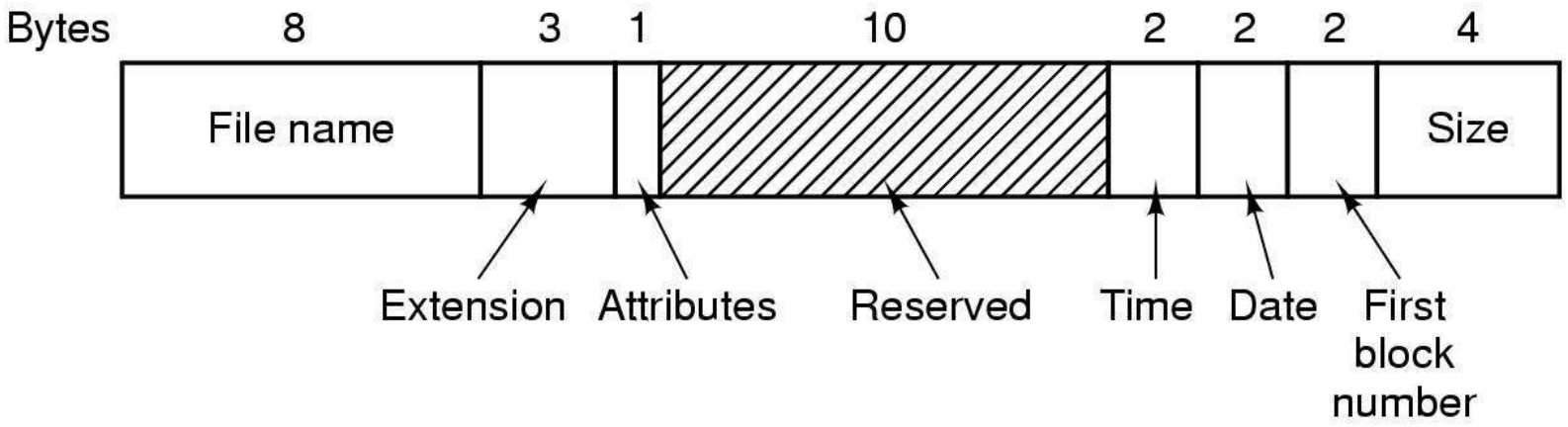


Figure 4-31. The MS-DOS directory entry.

The MS-DOS File System (2)

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Figure 4-32. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

The UNIX V7 File System (1)

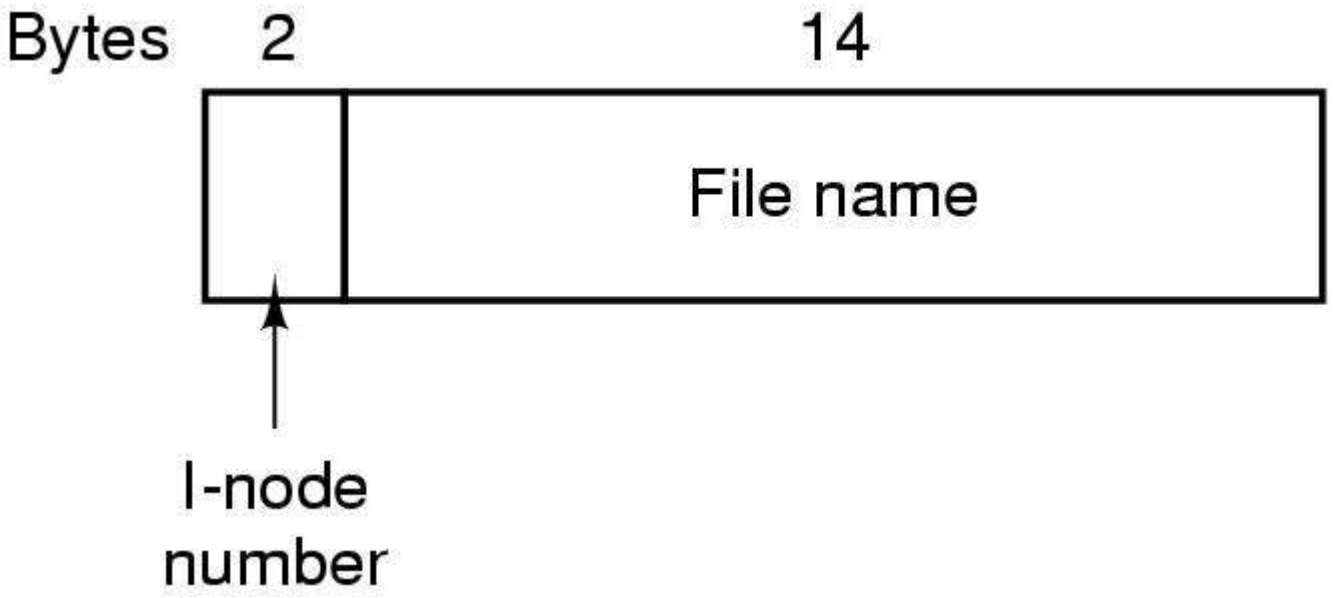


Figure 4-33. A UNIX V7 directory entry.

The UNIX V7 File System (2)

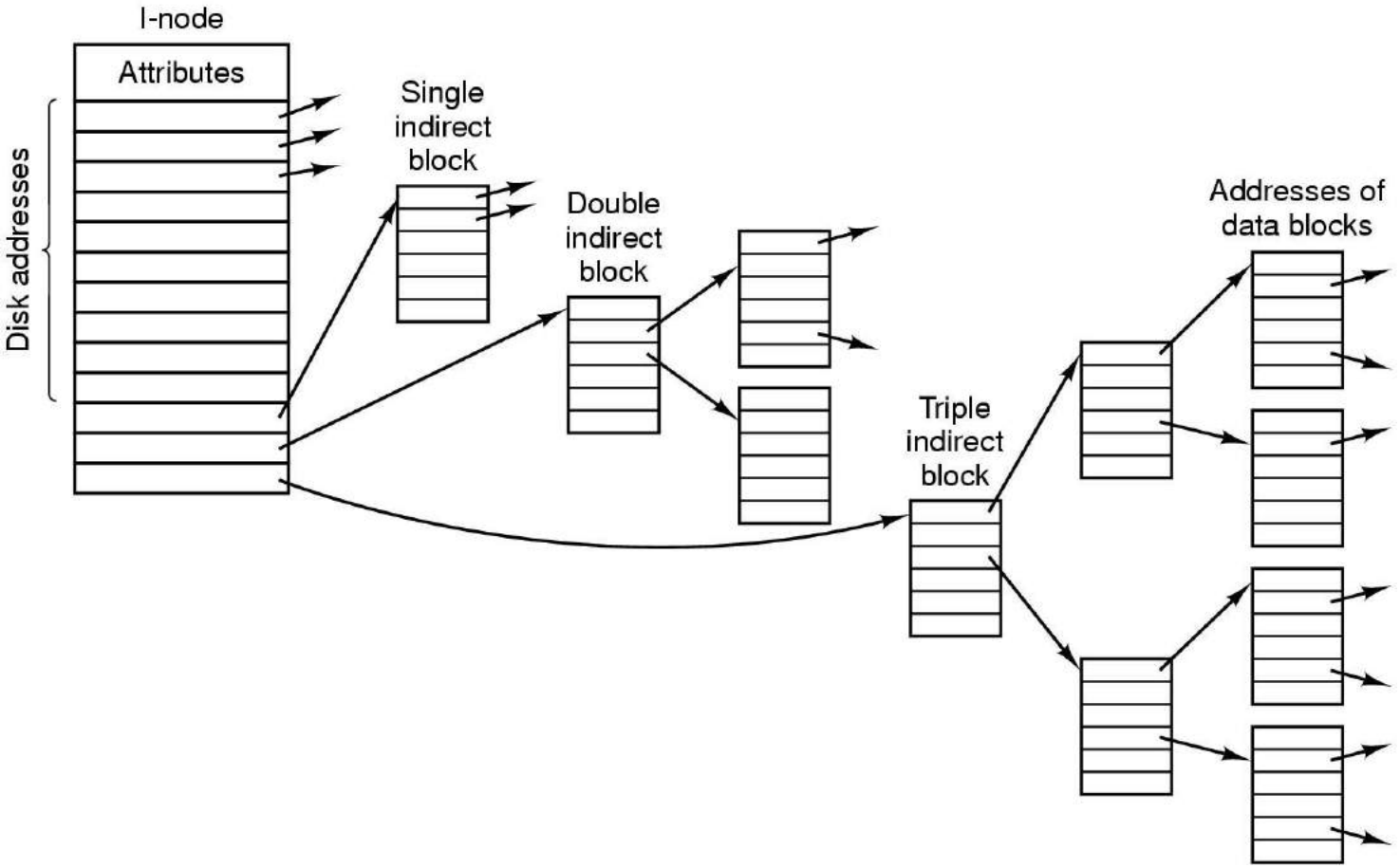


Figure 4-34. A UNIX i-node.

The UNIX V7 File System (3)

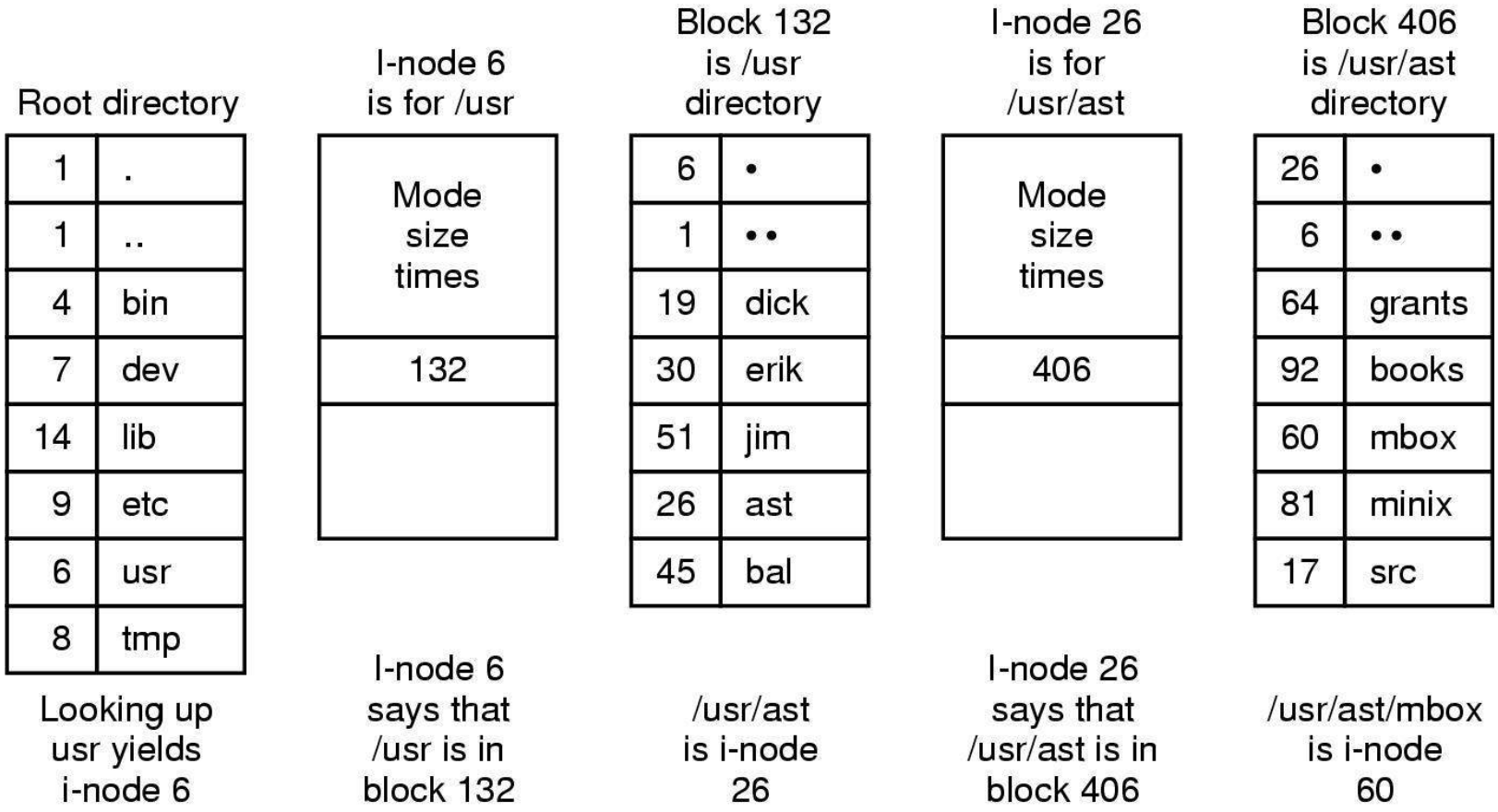


Figure 4-35. The steps in looking up */usr/ast/mbox*.

作业

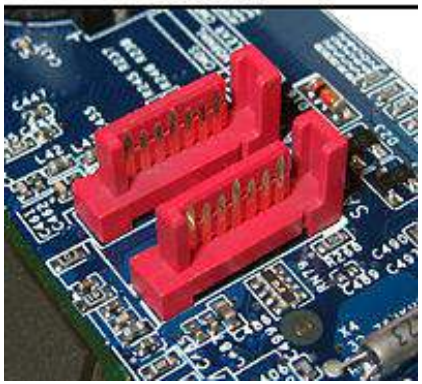
- P 183　27，28，29，32

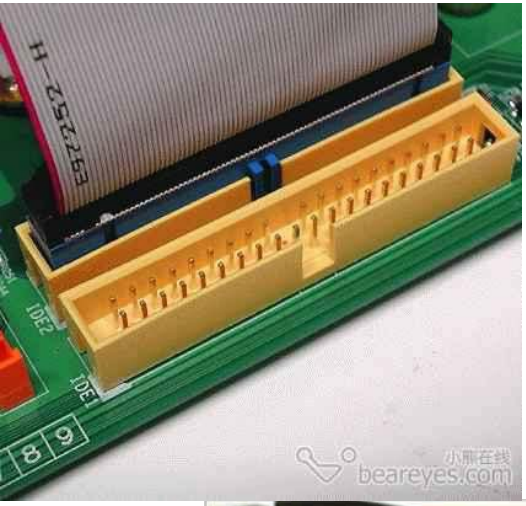
Chapter 5 File Management

Storage： Hard disks, LVM and RAID

Hard Disks

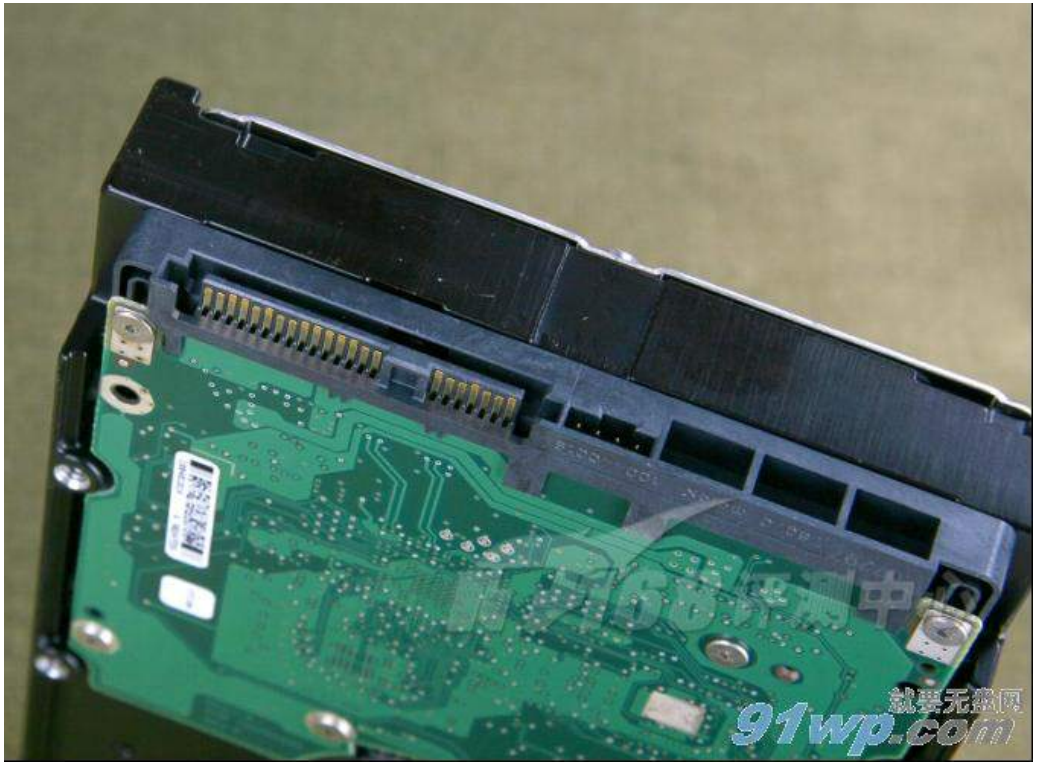
- IDE (Integrated Drive Electronics)
- ATA (Advanced Technology Attachment)
  - PATA (Parallel ATA or IDE)
  - SATA (Serial ATA)





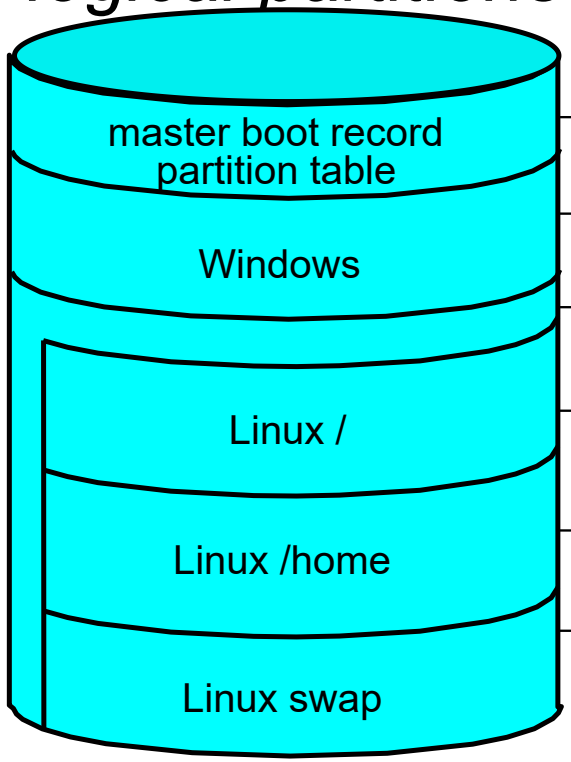
Hard Disks

- SCSI (Small Computer System Interface)
  - SAS (Serial Attached SCSI)
- SSD (Solid State Disk)



Hard disk partitions

- IDE and SCSI hard disks can be partitioned
- Maximum of four *primary partitions*
- One primary partition may be an *extended partition*
- An extended partition can hold an unlimited amount of *logical partitions* (Linux: max 59 for IDE, 11 for SCSI)



sda: The first sector of the disk contains the MBR and Partition Table

sda1: First primary partition holds a Windows filesystem

sda2: Second primary partition is an extended partition and holds three logical partitions

sda5: First logical partition holds a Linux filesystem that will be mounted as /

sda6: Second logical partition holds a Linux filesystem that will be mounted as /home

sda7: Third logical partition holds a Linux swap space

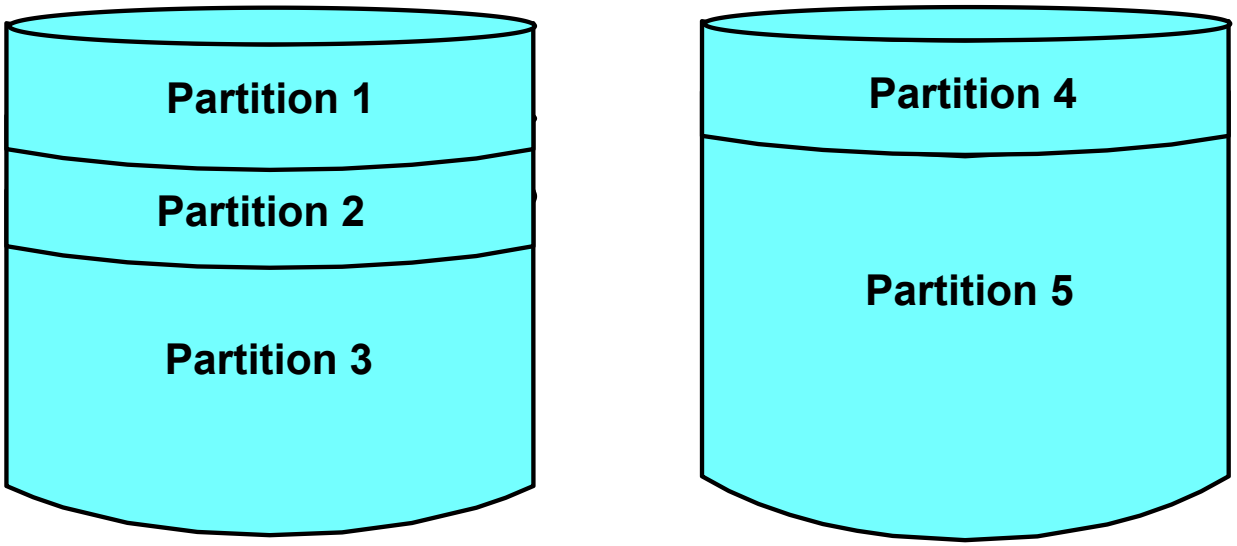
Partitioning tools

- **`fdisk`**
  - Virtually every PC OS comes with a tool **`fdisk`** to create partitions for that OS
    - Linux, Windows, and so forth
- **`parted`**
  - GPLed Linux program, available at www.gnu.org
  - Can create/resize/move/delete partitions
- GParted, QTParted
  - GUI utilities that use GNU Parted
  - Can create/resize/move/delete partitions
- Disk Druid and others
  - Partitioning program integrated in Linux install program

Components of storage

- Files
- Directories
- File systems
- Logical storage
- Physical storage
- Logical Volume Manager (LVM)

Traditional disk storage



PROBLEMS:

- Fixed partitions
- Expanding size of the partition
- Limitation on size of a file system and a file
- Contiguous data requirement
- Time and effort required in planning ahead

Benefits of the LVM

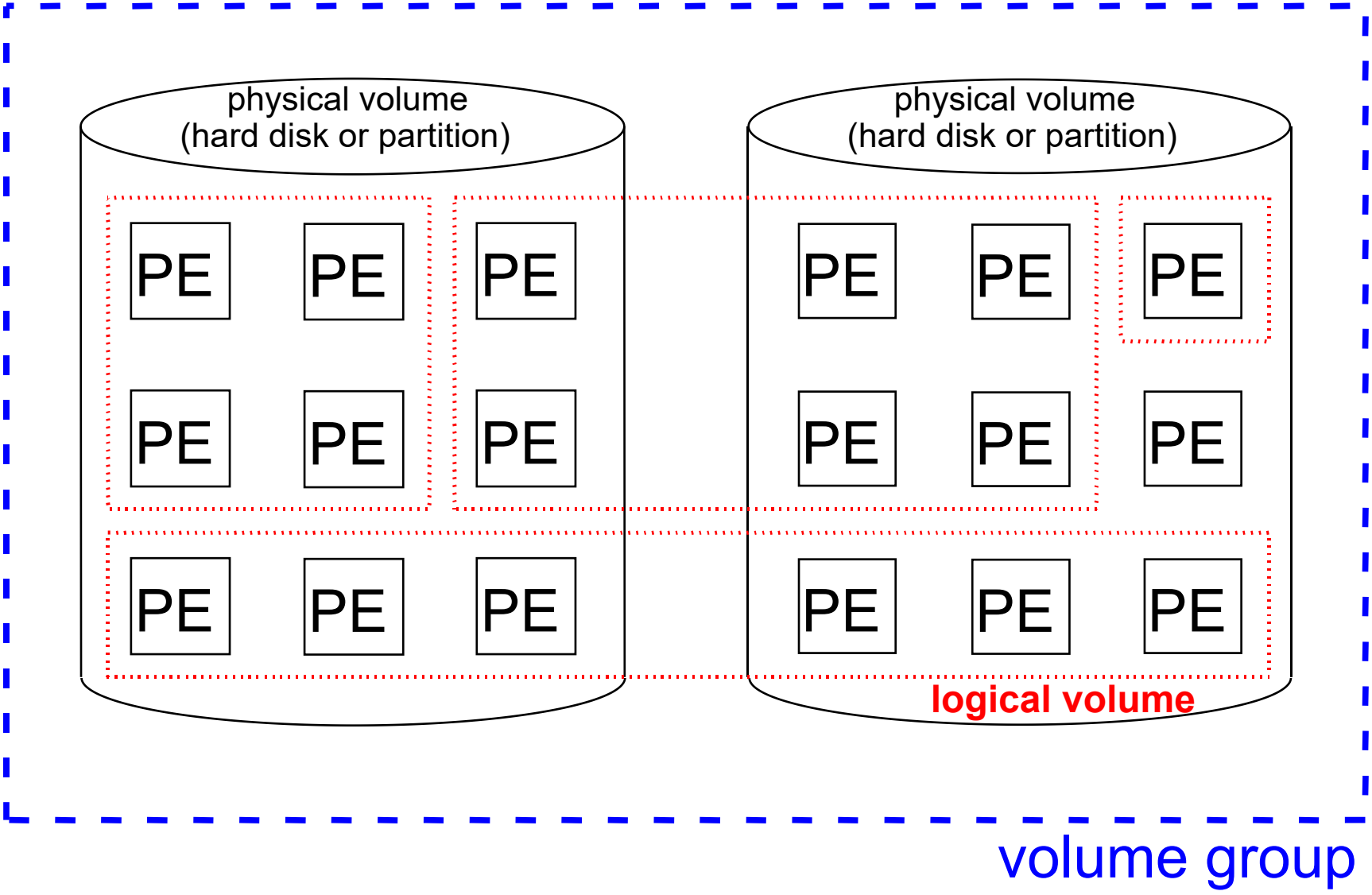
- Logical volumes solve noncontiguous space problems
- Logical volumes can span disks
- Logical volume sizes can be dynamically increased
- Logical volumes can be mirrored
- Physical volumes are easily added to the system
- Logical volumes can be relocated
- Volume group and logical volume statistics can be collected

These tasks can be performed dynamically!

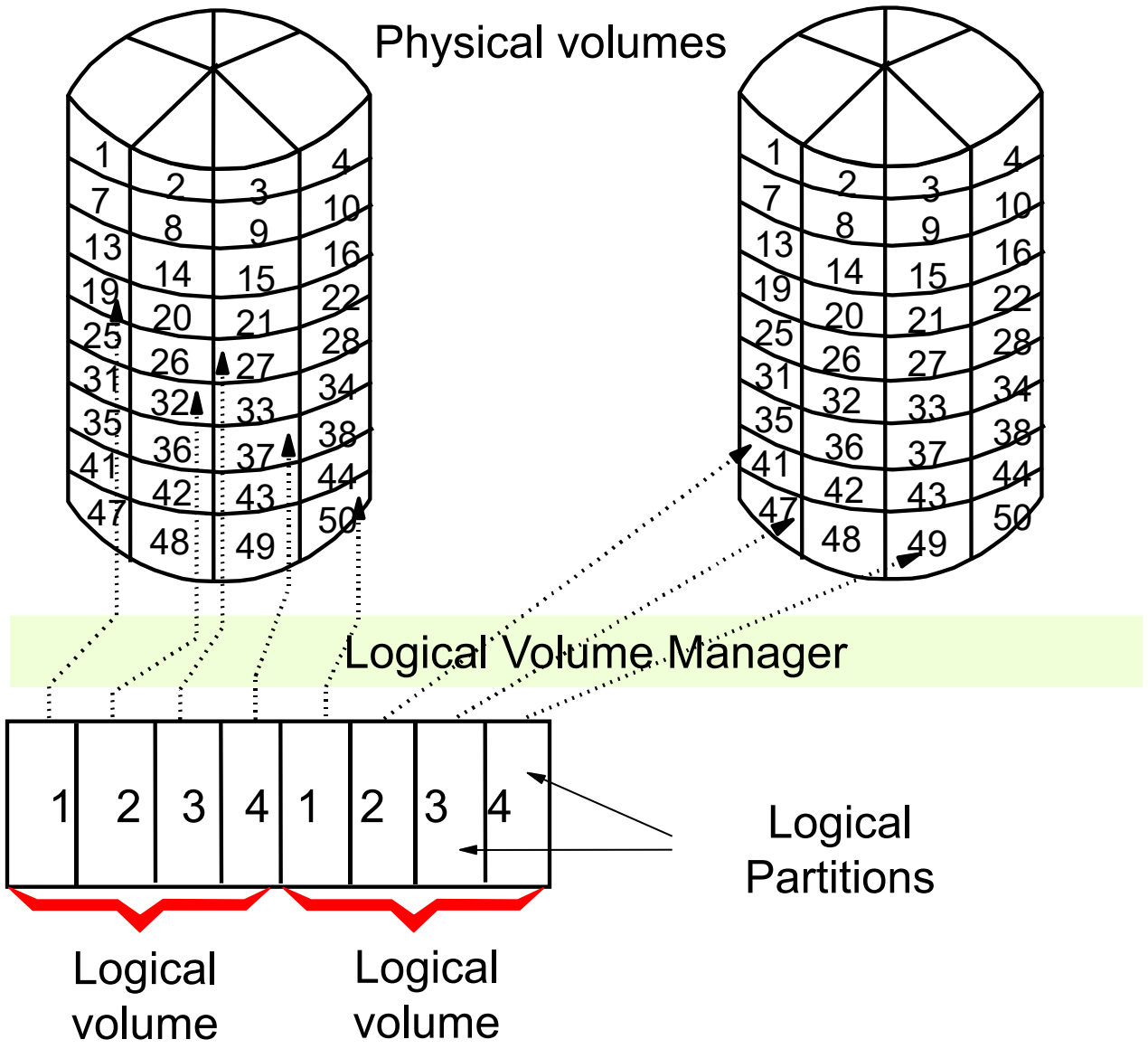
Logical volume management

- Logical volume management solves the disadvantages of traditional disk storage:
  - One or more physical volumes (hard disks, partitions) are assigned to a volume group (VG)
  - All physical volumes (PV) are split into physical extents (PE) of identical size (default 4 MB)
  - PEs in a VG can be combined into logical volumes (LV), which can be used like any block device
- An LV can span multiple disks
- To increase the size of an LV, add PEs
- To increase the size of a VG, add PVs

Logical volume management on Linux



Logical storage on AIX



LVM implementation overview

- Add hard disks and/or create partitions (type 0x8e) on existing hard disks
- Initialize physical volumes (disks or partitions)

```
# pvcreate /dev/sda3
# pvcreate /dev/sdb
```

- Create volume group **vg00** with physical volumes

```
# vgcreate vg00 /dev/sda3 /dev/sdb
```

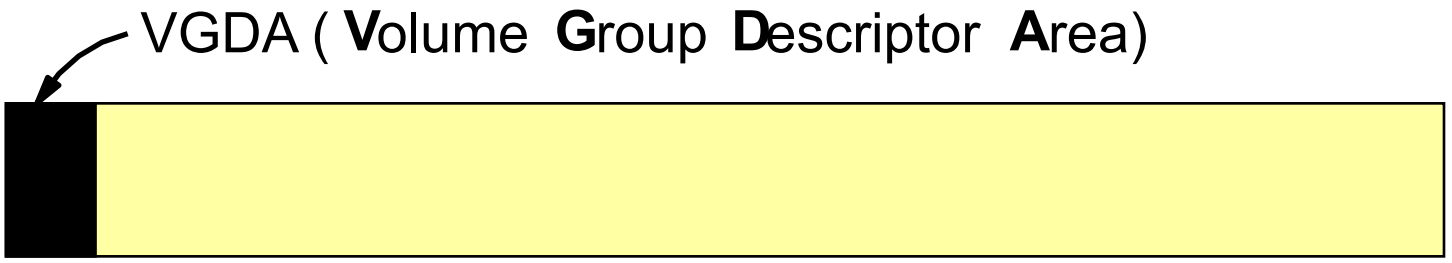
- Create logical volume **lv00** in volume group

```
# lvcreate -L 50M -n lv00 vg00
```

- Can now use /dev/vg00/lv00 as block device

Physical volume commands

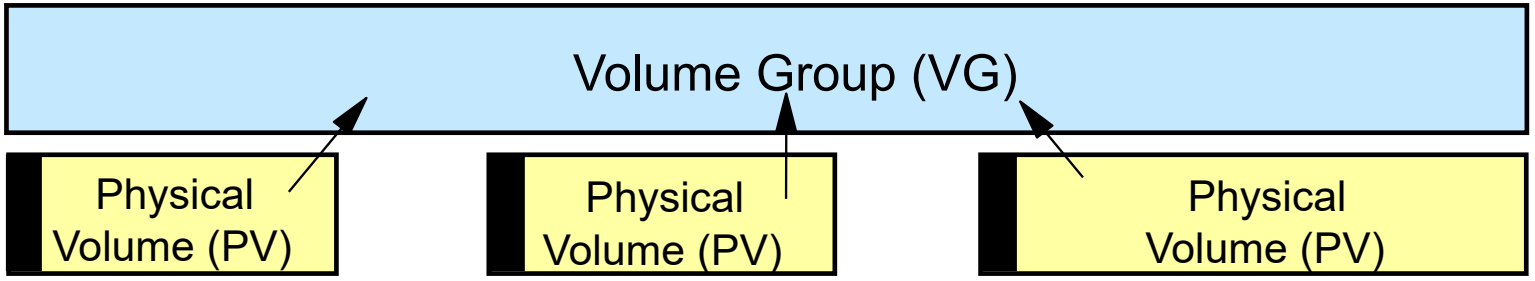
- **pvcreate <*pv*>**
  - Initializes a physical volume by putting an (empty) volume group descriptor area at the start of the PV



- **pvmove [-n <*lv*>] <*source pv*> [<*destination pv*>]**
  - Move PEs from one PV to another PV in the volume group
- **pvdisplay <*pv*>**
  - List information about a PV

Volume group commands

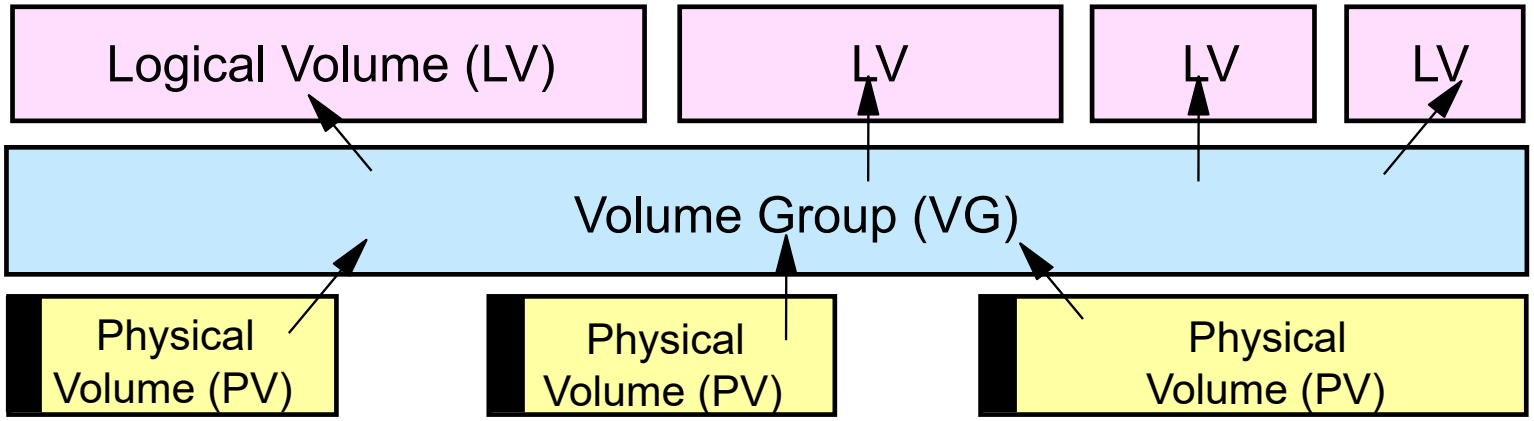
- **`vgcreate [-s <pe size>] <vg name> <pv> [<pv>...]`**
  - Create a volume group



- **`vgdisplay [<vg>]`**
  - Display information about a volume group
- **`vgremove <vg>`**
  - Delete a volume group

Logical volume commands

- **`lvcreate -L <size> [-n <lv name>] <vg> [<pv>...]`**
  - Create a logical volume in a volume group

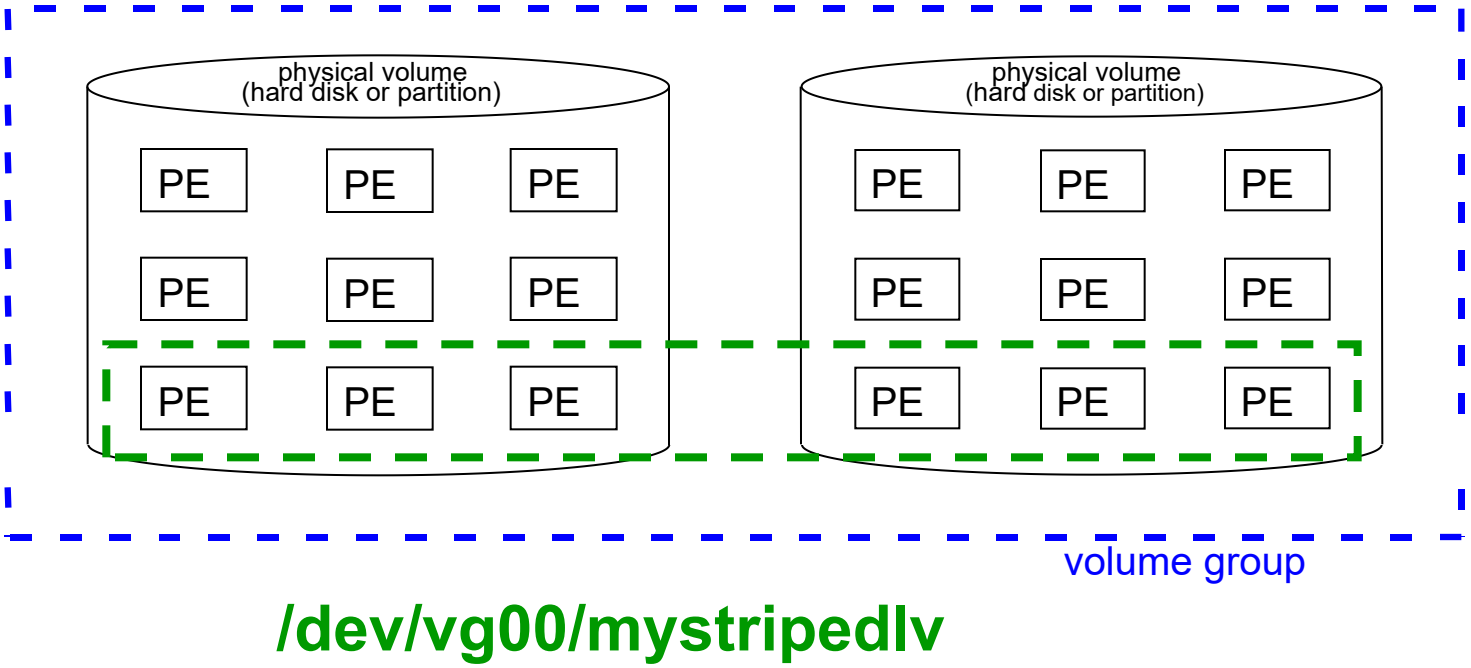


- **`lvdisplay <lv> [<lv>...]`**
  - Display information about a logical volume
- **`lvremove <lv> [<lv>...]`**
  - Remove a logical volume

Striping logical volumes

- A logical volume may be striped across two or more physical volumes during creation
- For large data transfers, this increases performance

```
# lvcreate -L 300M -i 2 -I 8 -n mystripedlv vg00
```



Extending/reducing a volume group

- To add or remove a physical volume to or from a volume group, use the **vgextend** and **vgreduce** commands
- To move physical extents from one physical volume to another, use **pvmove**

```
# vgextend vg00 /dev/sdb6
# vgreduce vg00 /dev/sda5
ERROR: can't reduce volume group "vg00" by used
physical volume "/dev/sda5"
# pvmove /dev/sda5 /dev/sdb6
# vgreduce vg00 /dev/sda5
```

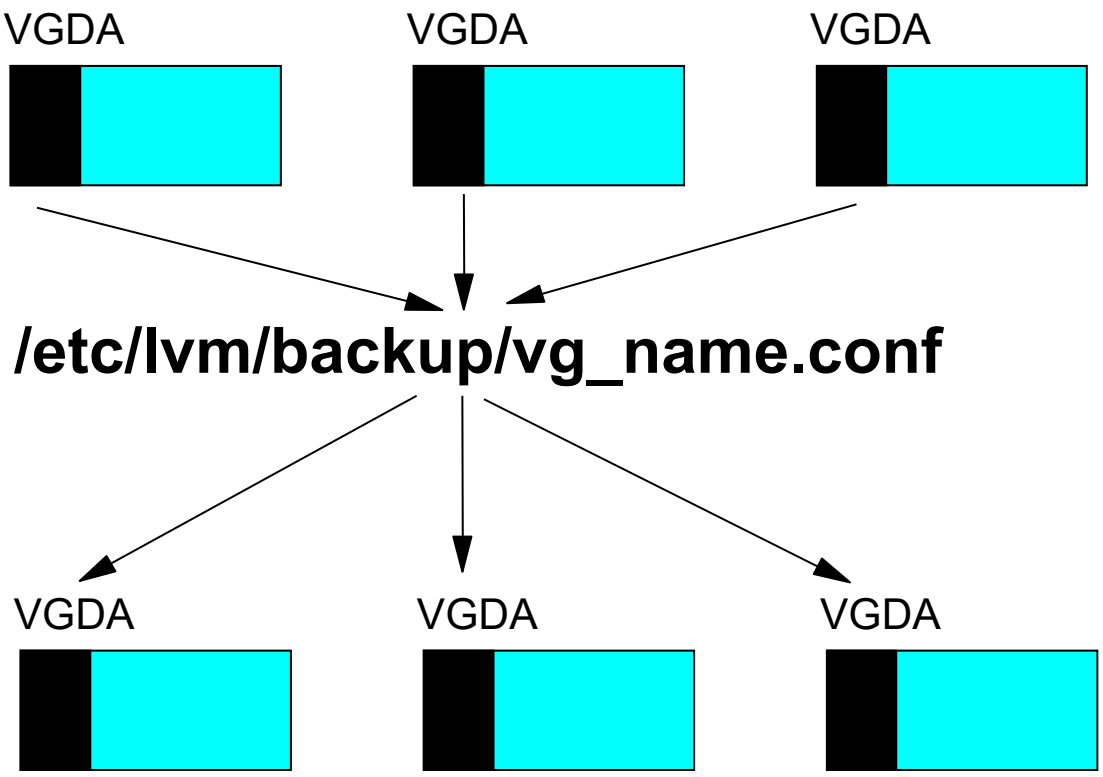
Extending/reducing a logical volume

- To extend/shrink a logical volume use the **`lvextend`**/**`lvreduce`** commands
  - Use **-L** option to specify size in bytes
  - Use **-l** option to specify size in PEs
- **`lvextend`**/**`lvreduce`** do *NOT* extend/shrink a filesystem in the LV automatically!
  (Extending/shrinking a filesystem will be covered later)

```
# lvextend -L +300M /dev/vg00/mylv
lvextend -- rounding relative size up to physical extent boundary
lvextend -- extending logical volume "/dev/vg00/mylv" to 380 MB
lvextend -- doing automatic backup of volume group "system"
lvextend -- logical volume "/dev/vg00/mylv" successfully extended
# lvreduce -l -12 /dev/system/mystripedlv
...
```

LVM backup and recovery

- It is very important to save the LVM metadata stored in the VGDA for recovering reasons.
  1. **`vgcfgbackup`**
  2. **`vgcfgrestore -n vg_name PV`**



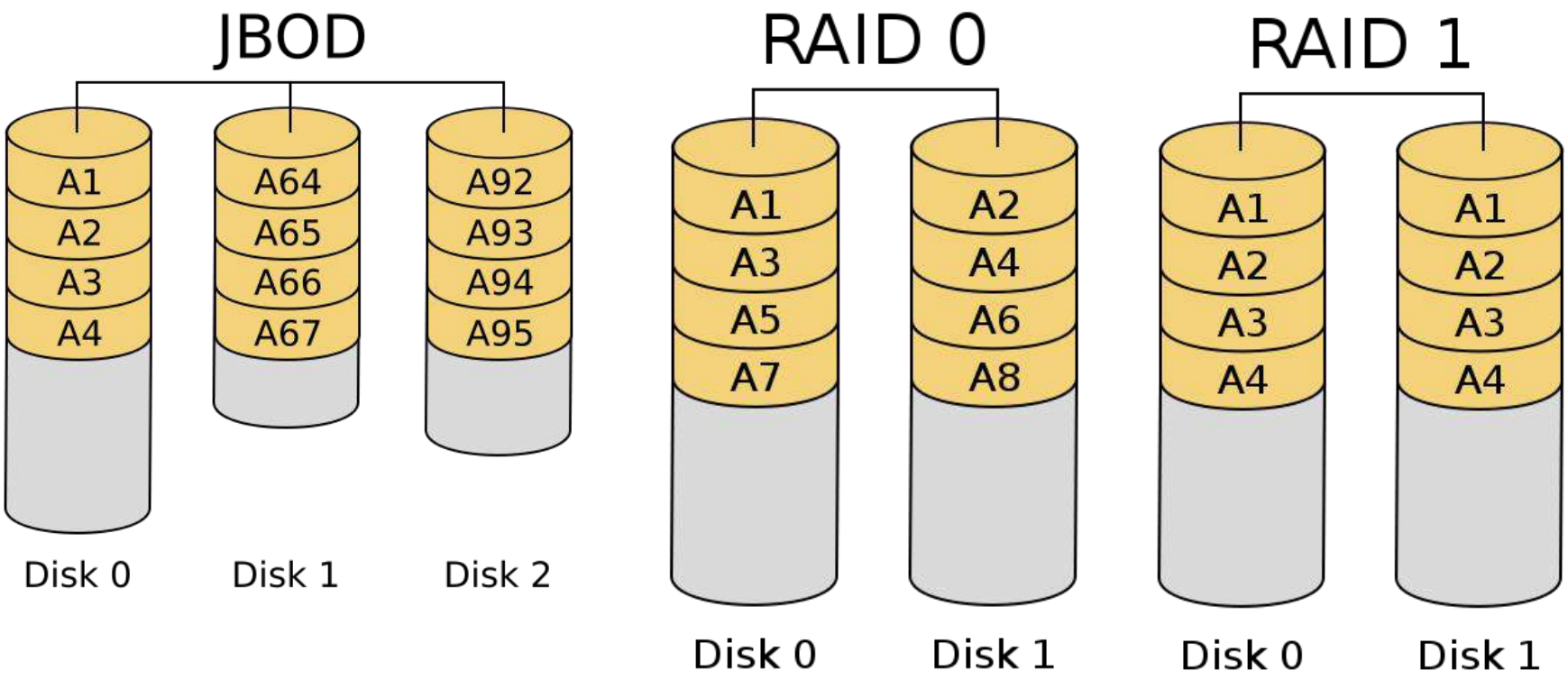
Additional LVM considerations

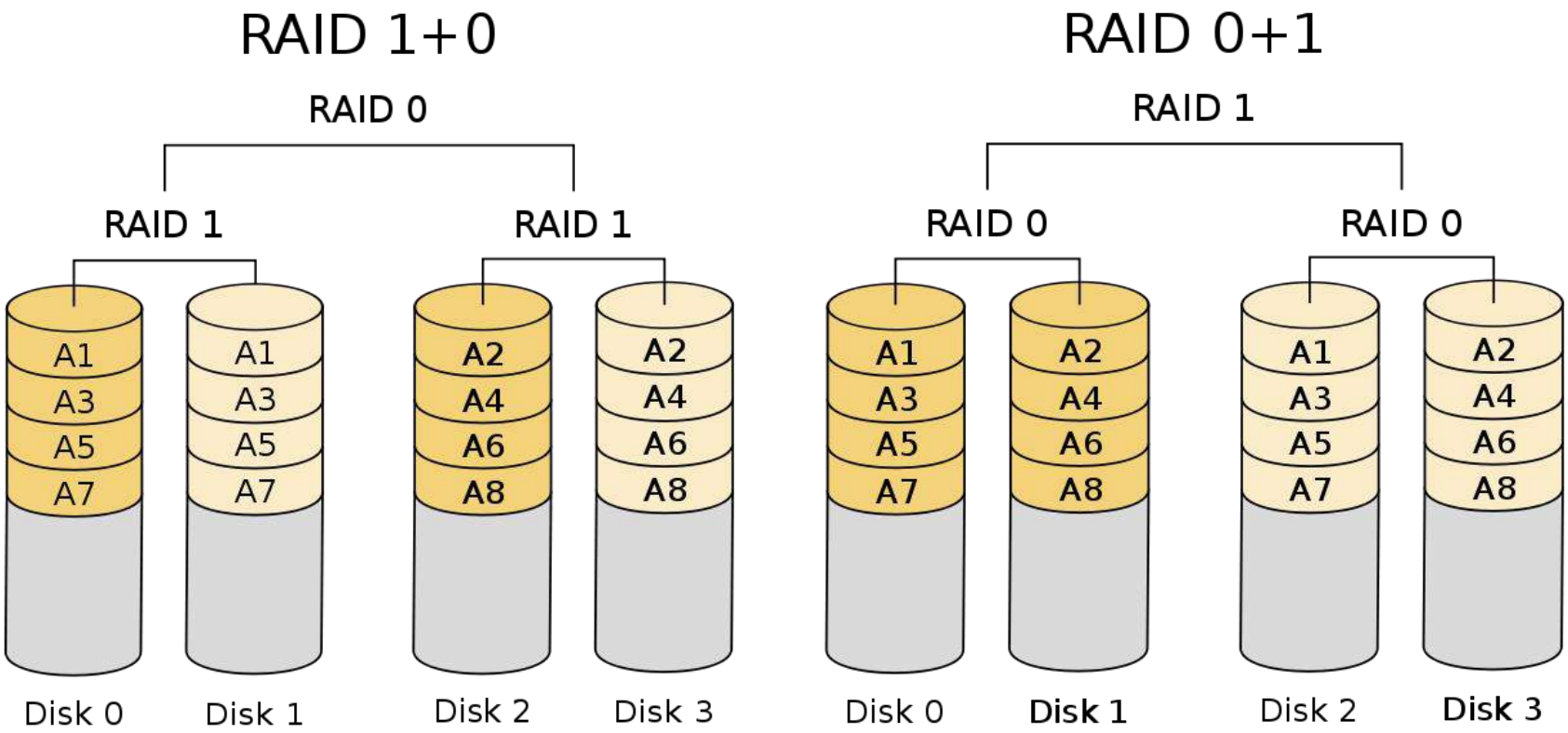
- Linux LVM implementation has "snapshot" capability
  - Can be useful for fast backups
- If LVM-based filesystems are listed in /etc/fstab, then LVM support needs to be included in the initial root disk (initrd)
- Mirroring is handled outside of the LVM structure
- LVM uses the device mapper kernel device driver
- LVM snapshots are read/write by default

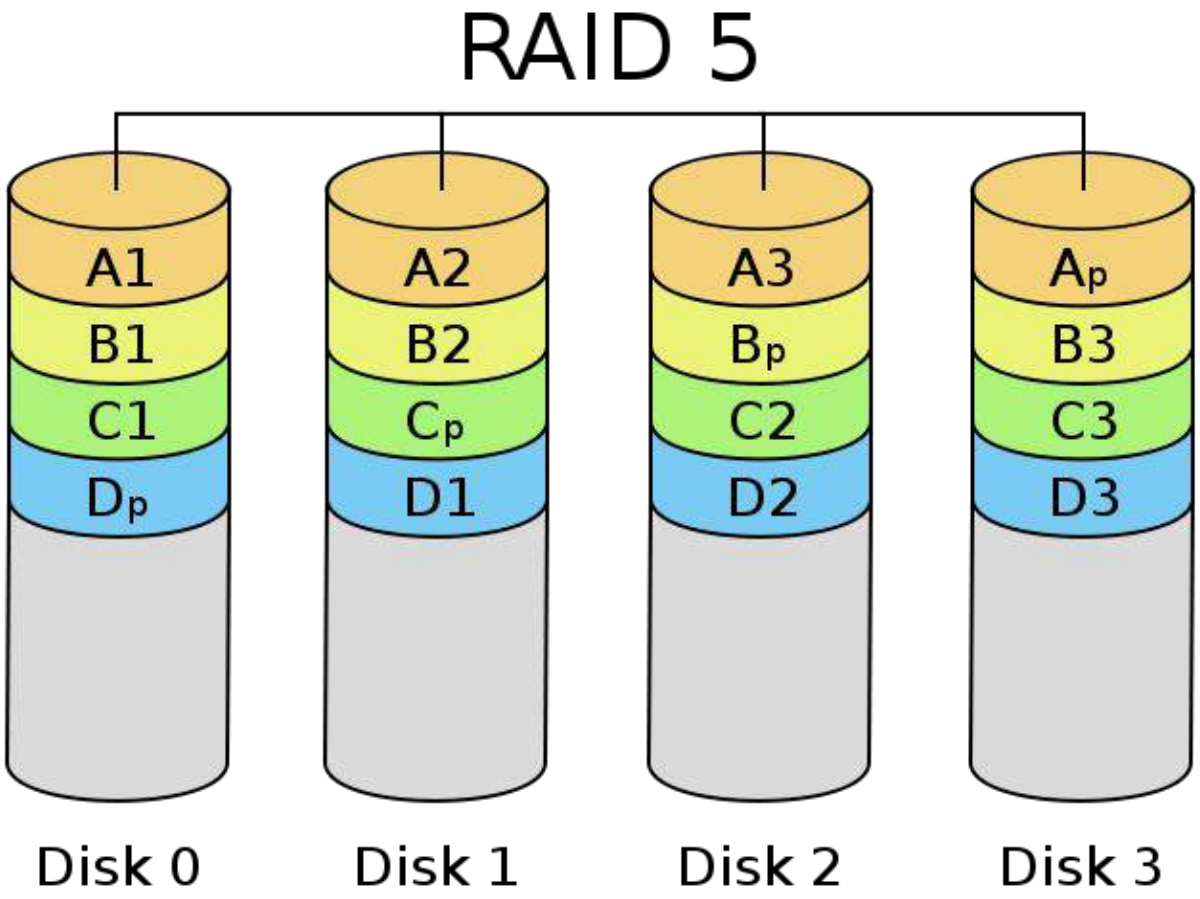
RAID

- Redundant Array of Independent Disks
- Typical PC hard disks, compared to expensive mainframe-quality hard disks, are:
  - Slower
  - Less reliable
  - Smaller
  - But less expensive
- RAID uses multiple hard disks in an array to create a logical device that is:
  - Faster
  - More reliable
  - Or larger
  - And still relatively inexpensive

RAID (Redundant Array of Independent Disk)




RAID 1+0
RAID 0
RAID 1
RAID 1
A1
A3
A5
A7
A1
A3
A5
A7
A2
A4
A6
A8
A2
A4
A6
A8
Disk 0
Disk 1
Disk 2
Disk 3
RAID 0+1
RAID 1
RAID 0
RAID 0
A1
A3
A5
A7
A2
A4
A6
A8
A1
A3
A5
A7
A2
A4
A6
A8
Disk 0
Disk 1
Disk 2
Disk 3


RAID 5
A1
B1
C1
Dp
A2
B2
Cp
D1
A3
Bp
C2
D2
Ap
B3
C3
D3
Disk 0
Disk 1
Disk 2
Disk 3

RAID (2)

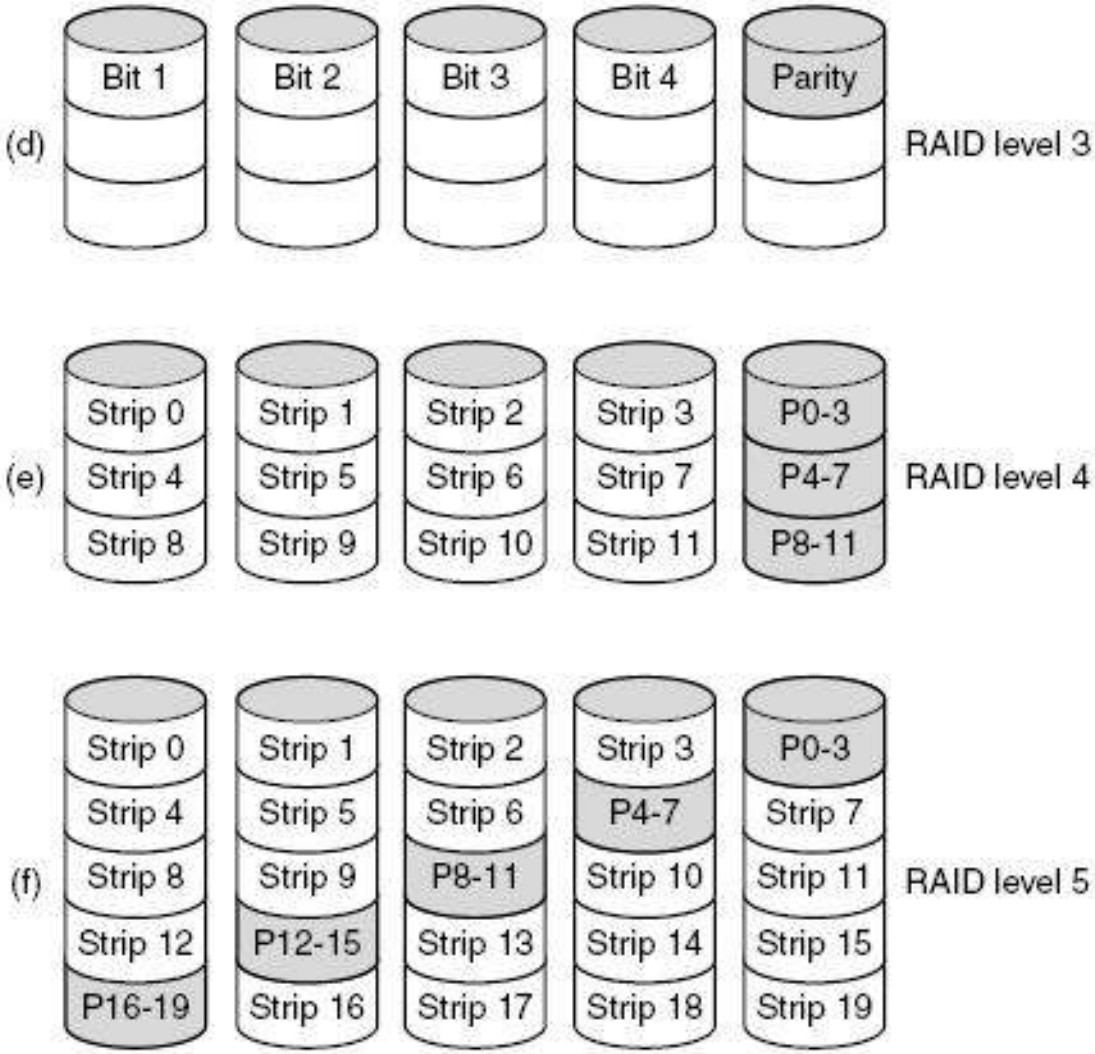


Figure 5-20. RAID levels 0 through 5.
Backup and parity drives are shown shaded.

RAID (1)

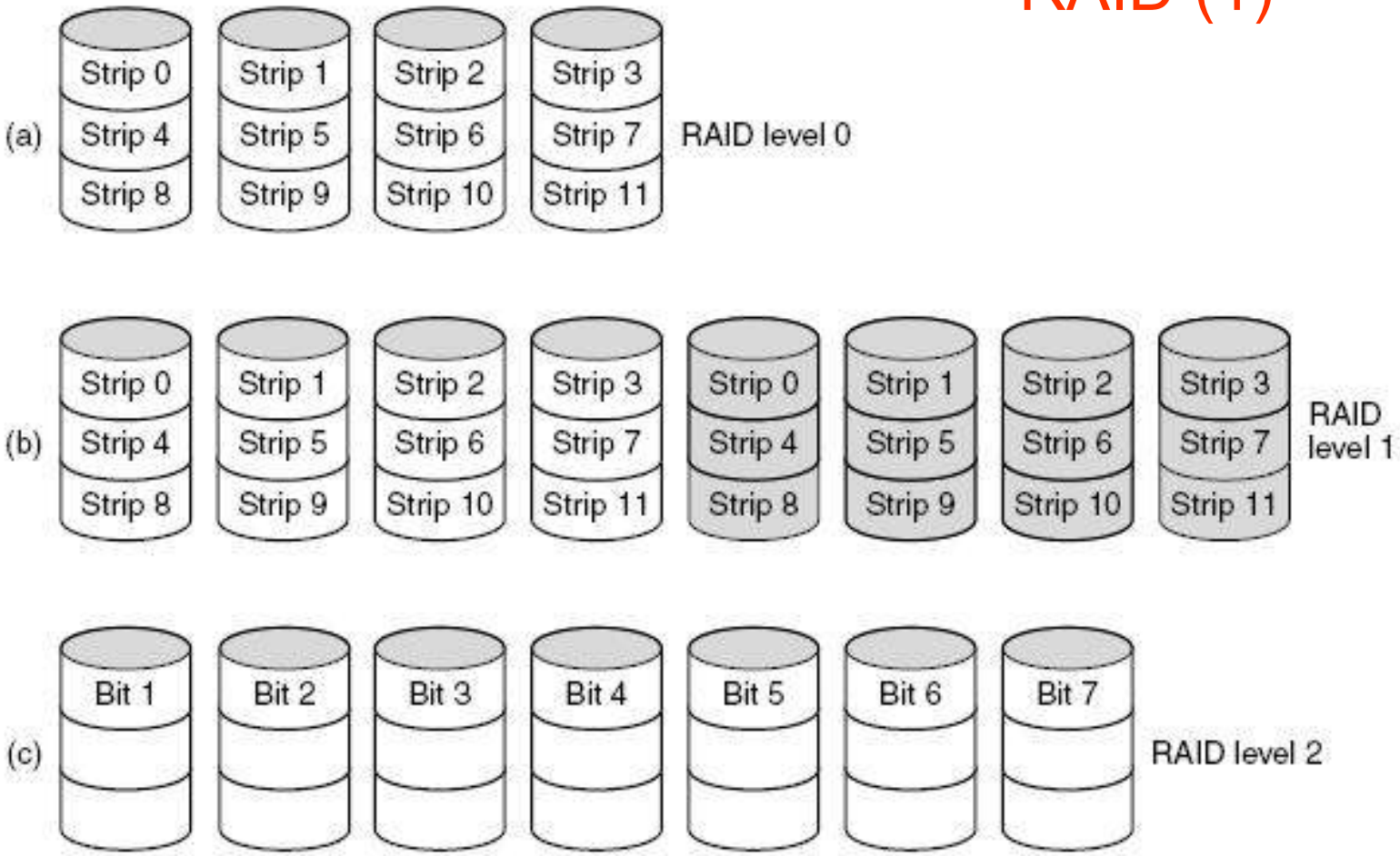


Figure 5-20. RAID levels 0 through 5.
Backup and parity drives are shown shaded.

RAID levels

- RAID levels have different characteristics
  - RAID-5 is not "better" than RAID-1
- Use RAID level according to needs

RAID level	Min # disks	Read performance	Write performance	Redundancy	Data capacity with 3x1GB disk	Other remarks
Linear	2	Equal	Equal	No	3 GB	Can be used if disks are not equal
0	2	Fast	Fast	No	3 GB	
1	2	Fast	Somewhat slower	Yes	1 GB	Can sustain N-1 disk crash(s)
4	3	Somewhat faster	Slow	Yes	2 GB	Can sustain one disk crash Parity disk is bottleneck
5	3	Somewhat faster	Somewhat faster	Yes	2 GB	Can sustain one disk crash CPU intensive

(*) Performance compared to a single disk, for data transfers greater than block size

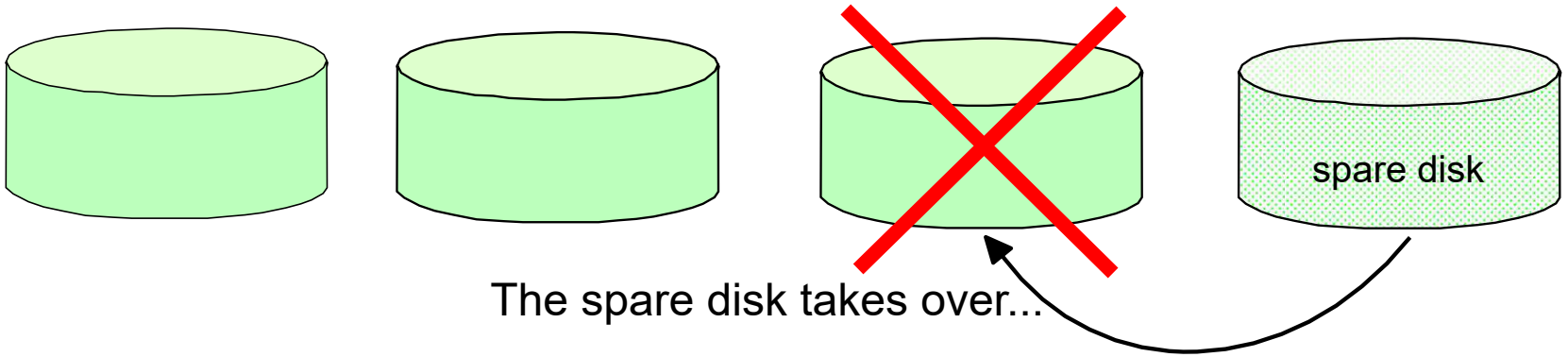
Linux RAID support

- Software RAID
  - Implemented in Linux kernel
  - Needs **mdadm** package
  - Uses disk partitions to create RAID devices
  - Logical device name: /dev/mdn
- Hardware RAID
  - Implemented in special adapter cards
  - Adapter needs to be supported by Linux kernel
  - Generally specific software needed to configure adapter correctly (might not be available under Linux)
  - RAID devices show up as regular SCSI disk

Spare disks

- To make RAID1/RAID5 more failsafe in case of a disk failure, use spare disks!

```
# cat /etc/raidtab
...
nr-spare-disks 1
device  /dev/sdd1
spare-disk 0
...
```



- Remove a failed disk with **`raidhotremove`**
- Add a new disk to the array with **`raidhotadd`**

Additional RAID considerations

- Put RAID partitions on different disks
- Use different SCSI or IDE controllers if possible for different disks that are part of a RAID volume
- Do not use RAID for /boot partition
- If RAID-based filesystems are listed in /etc/fstab, then RAID support needs to be included in the initrd
- Software RAID4 and RAID5 needs a lot of CPU time
- Do not use RAID-linear or RAID0 for swap space
  - The Linux kernel can stripe across swap spaces more efficiently

References

- Chapter 5: Input/Output, *Modern Operating Systems .3rd ed, Andrew S. Tanenbaum*
- Unit 8: Block devices, RAID, and LVM, *Linux System Administration I: Implementation , ERC 6.0, IBM*

Input/Output Devices

I/O Devices

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	3.5 MB/sec
802.11g Wireless	6.75 MB/sec
52x CD-ROM	7.8 MB/sec
Fast Ethernet	12.5 MB/sec
Compact flash card	40 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
SATA disk drive	300 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec

Figure 5-1. Some typical device, network, and bus data rates.

Memory-Mapped I/O (1)

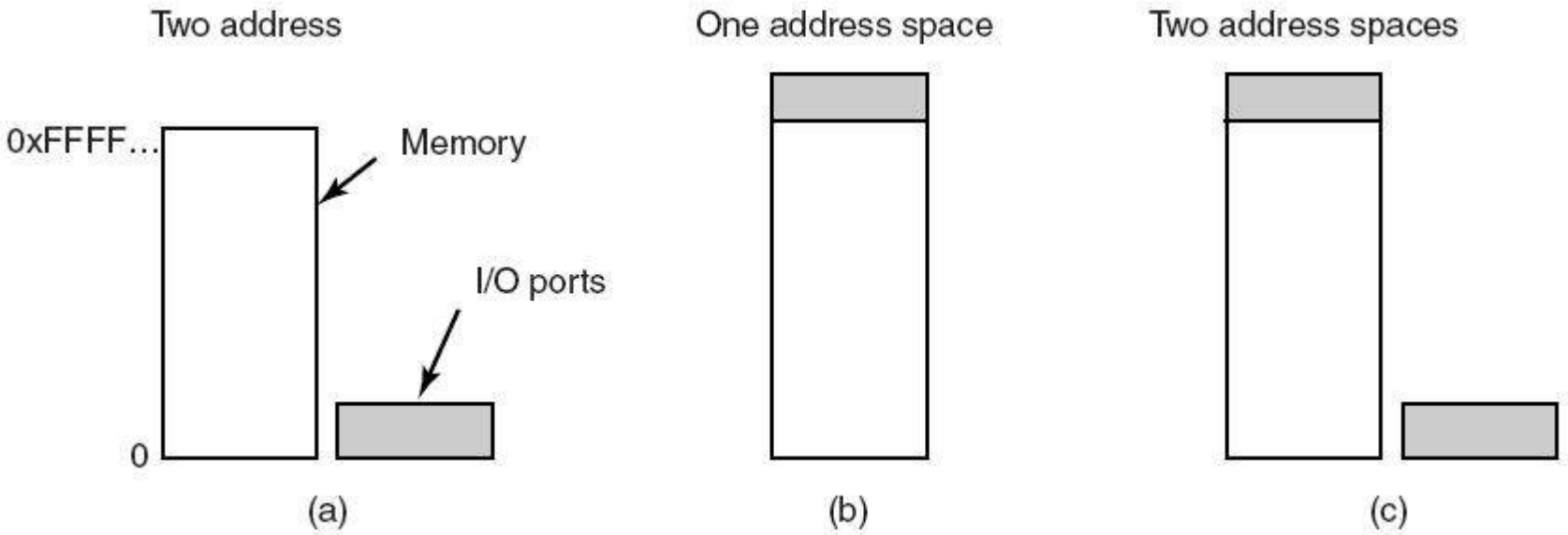


Figure 5-2. (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.

Memory-Mapped I/O (2)

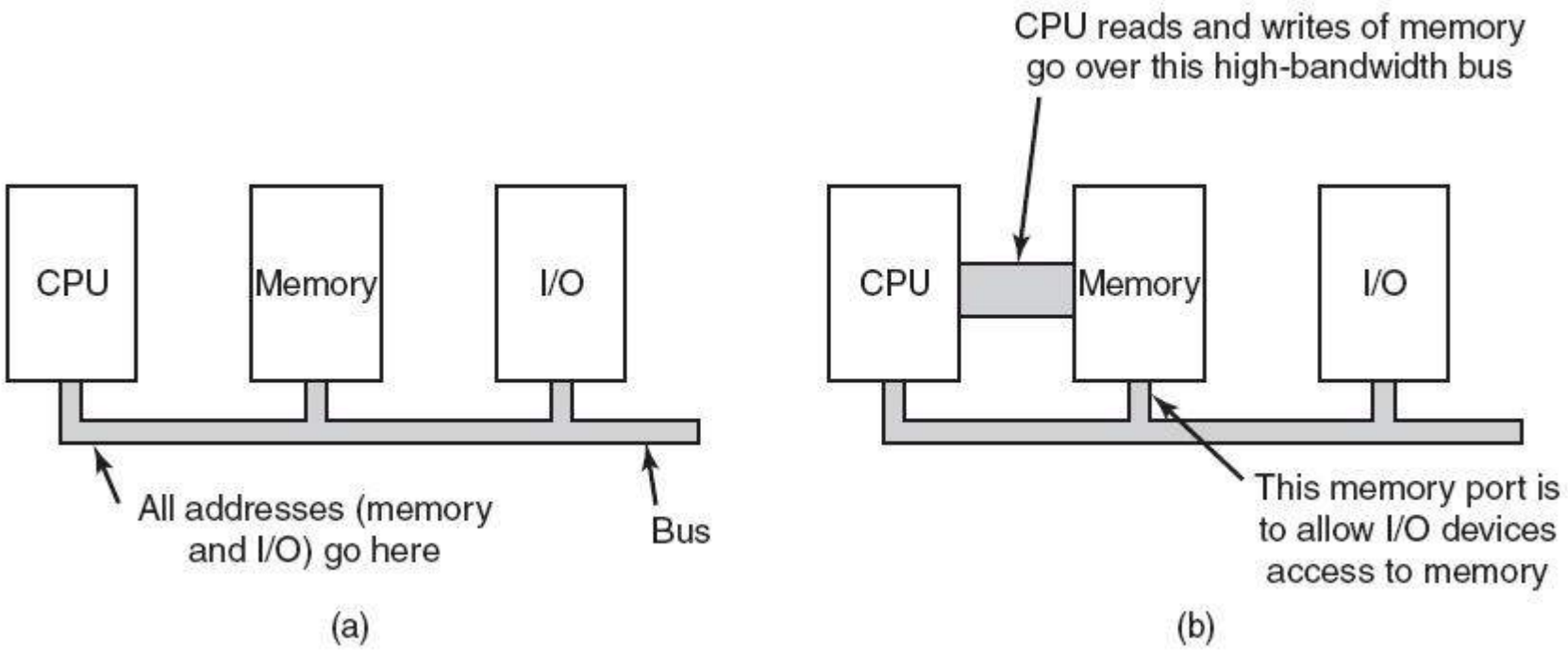


Figure 5-3. (a) A single-bus architecture. (b) A dual-bus memory architecture.

Direct Memory Access (DMA)

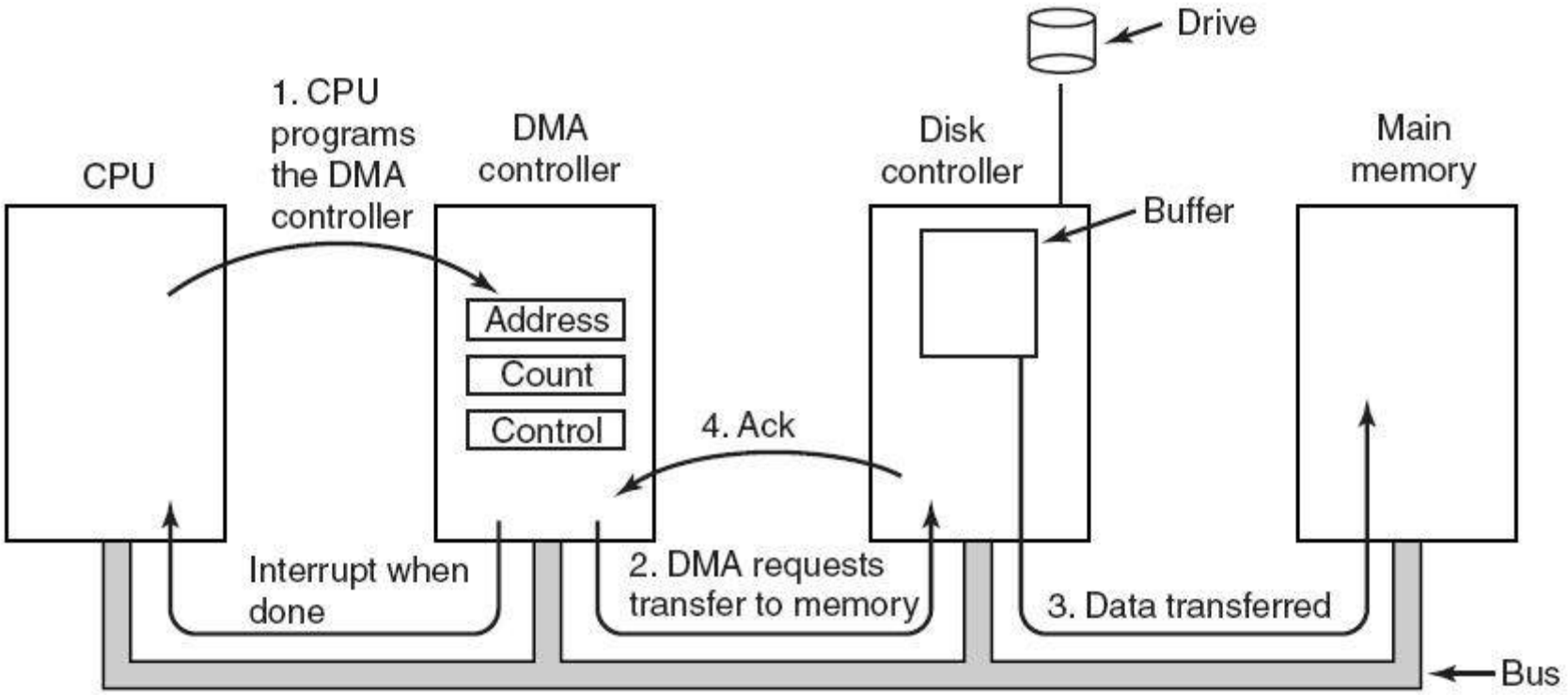


Figure 5-4. Operation of a DMA transfer.

Interrupts Revisited

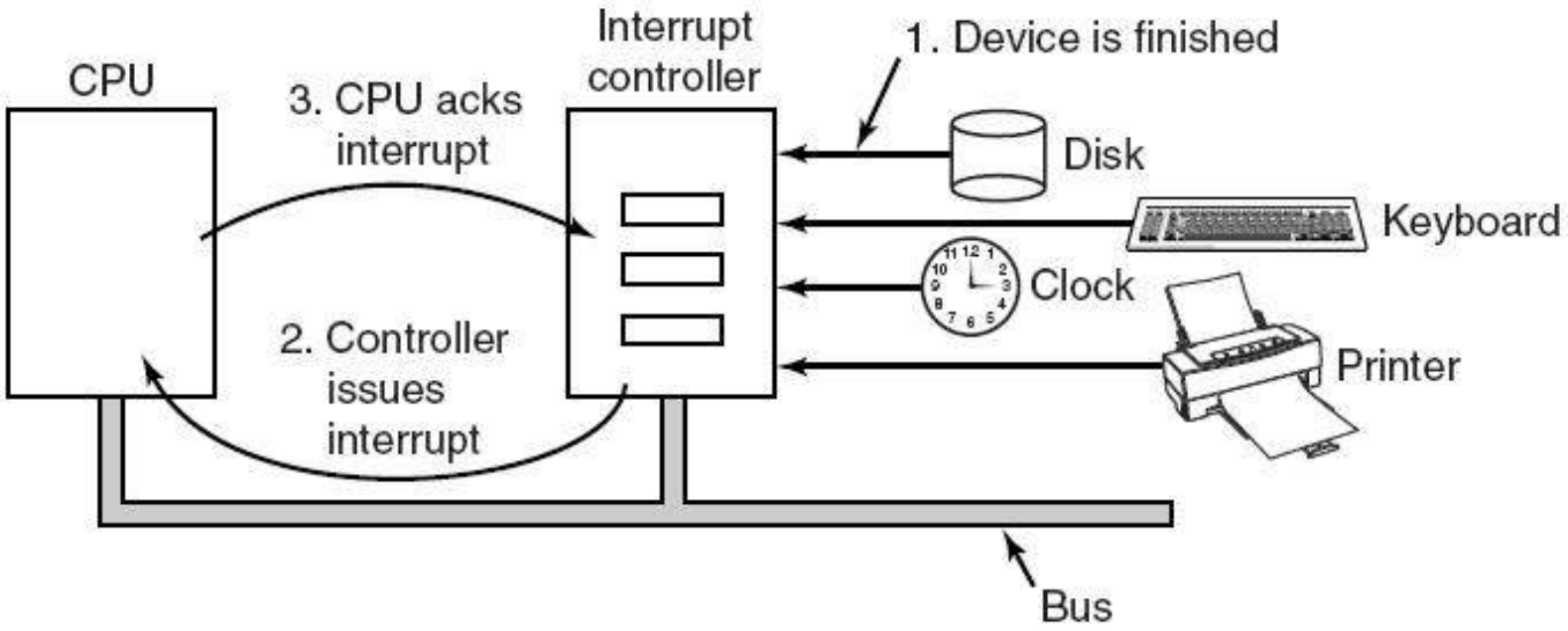


Figure 5-5. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.

Precise and Imprecise Interrupts (1)

Properties of a *precise interrupt*

1. PC (Program Counter) is saved in a known place.
2. All instructions before the one pointed to by the PC have fully executed.
3. No instruction beyond the one pointed to by the PC has been executed.
4. Execution state of the instruction pointed to by the PC is known.

Precise and Imprecise Interrupts (2)

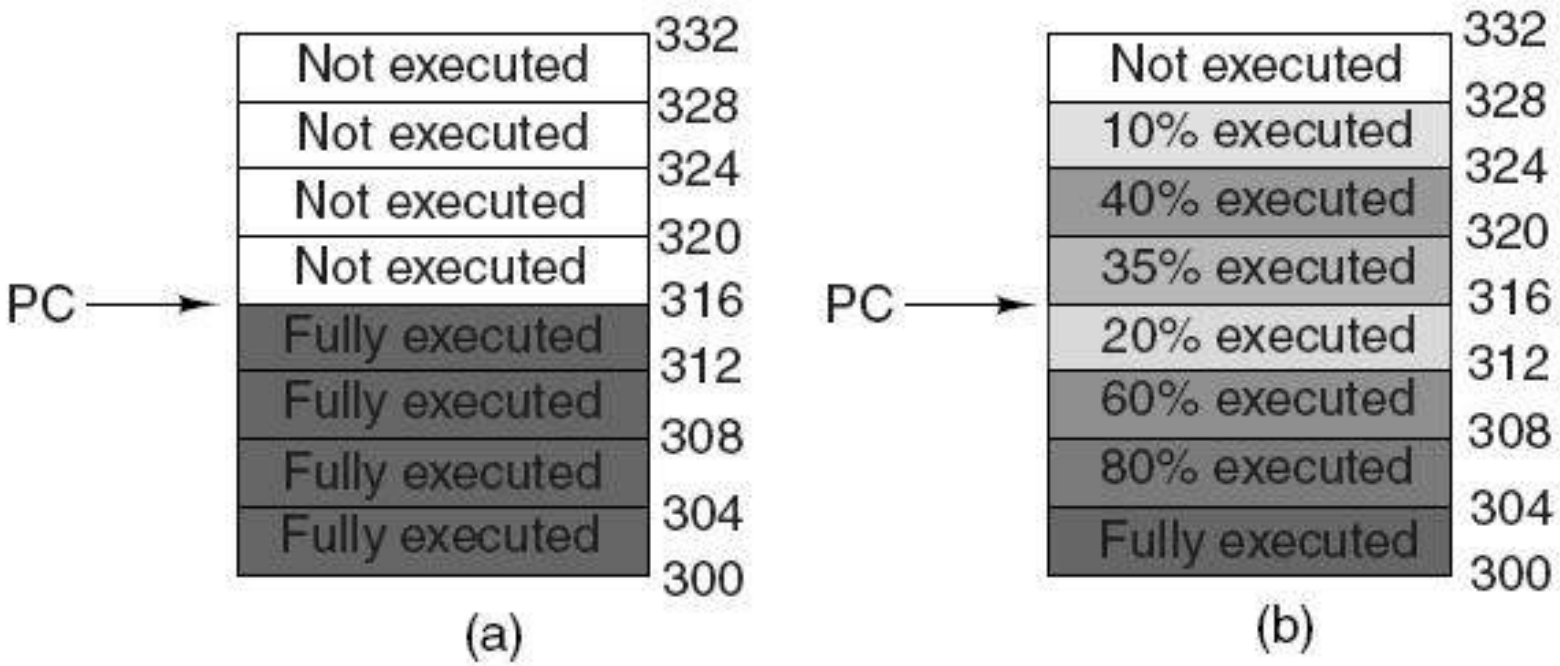


Figure 5-6. (a) A precise interrupt. (b) An imprecise interrupt.

Programmed I/O (1)

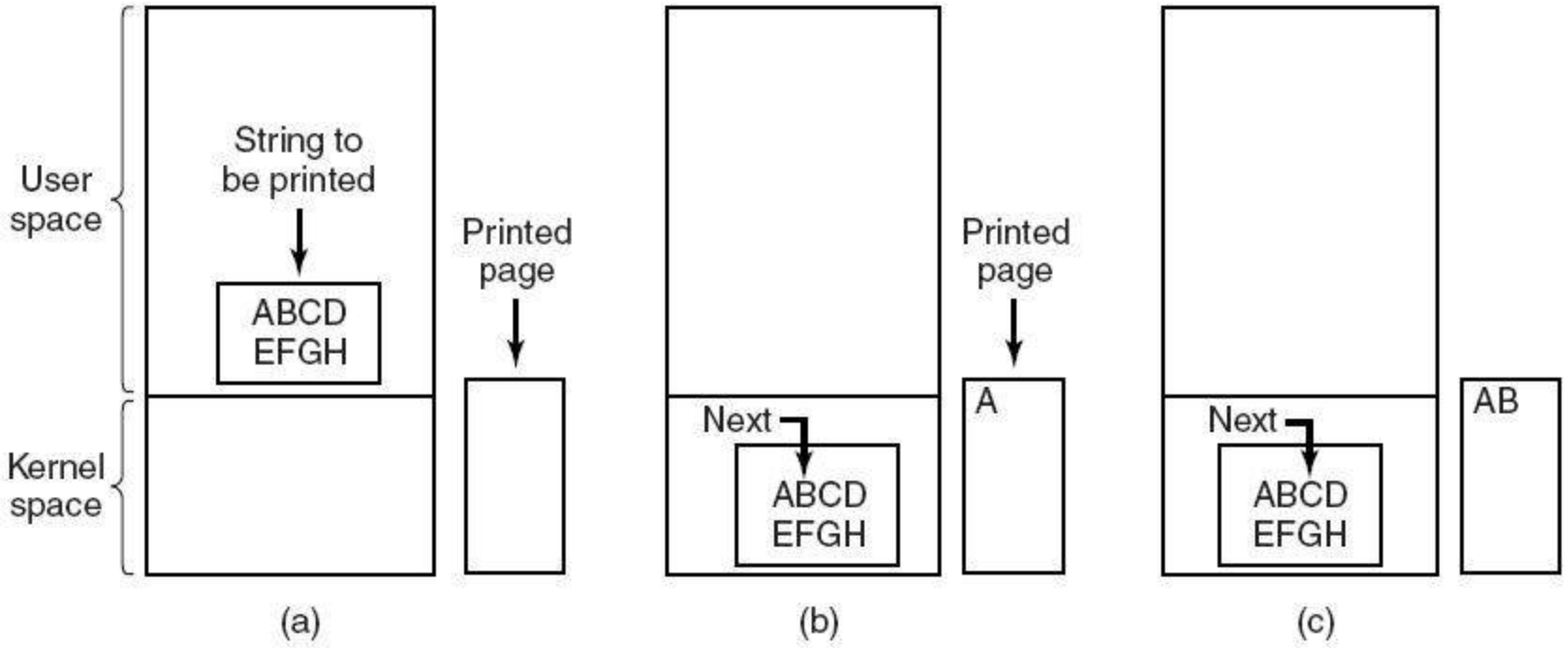


Figure 5-7. Steps in printing a string.

Interrupt-Driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Figure 5-9. Writing a string to the printer using interrupt-driven I/O. (a) Code executed at the time the print system call is made. (b) Interrupt service procedure for the printer.

I/O Using DMA

```
copy_from_user(buffer, p, count);
set_up_DMA_controller( );
scheduler( );
```

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Figure 5-10. Printing a string using DMA. (a) Code executed when the print system call is made. (b) Interrupt service procedure.

I/O Software Layers

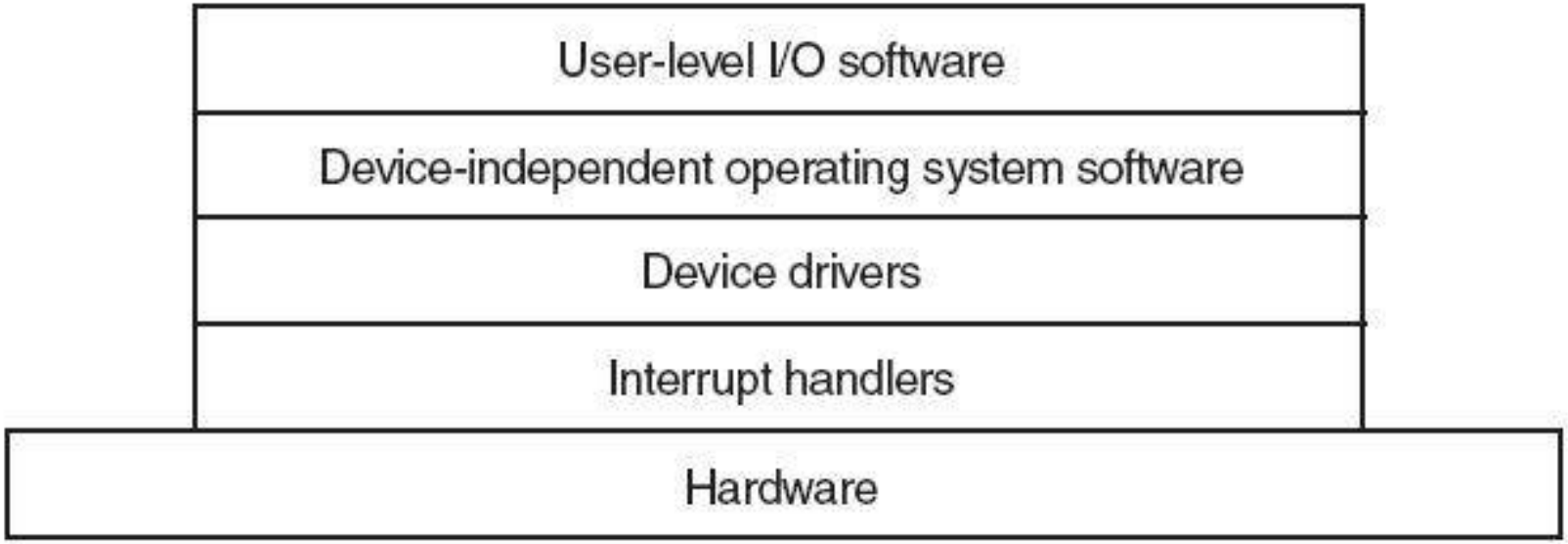


Figure 5-11. Layers of the I/O software system.

Interrupt Handlers (1)

1. Save registers not already been saved by interrupt hardware.
2. Set up a context for the interrupt service procedure.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved to the process table.

Interrupt Handlers (2)

6. Run the interrupt service procedure.
7. Choose which process to run next.
8. Set up the MMU context for the process to run next.
9. Load the new process' registers, including its PSW.
10. Start running the new process.

Device Drivers

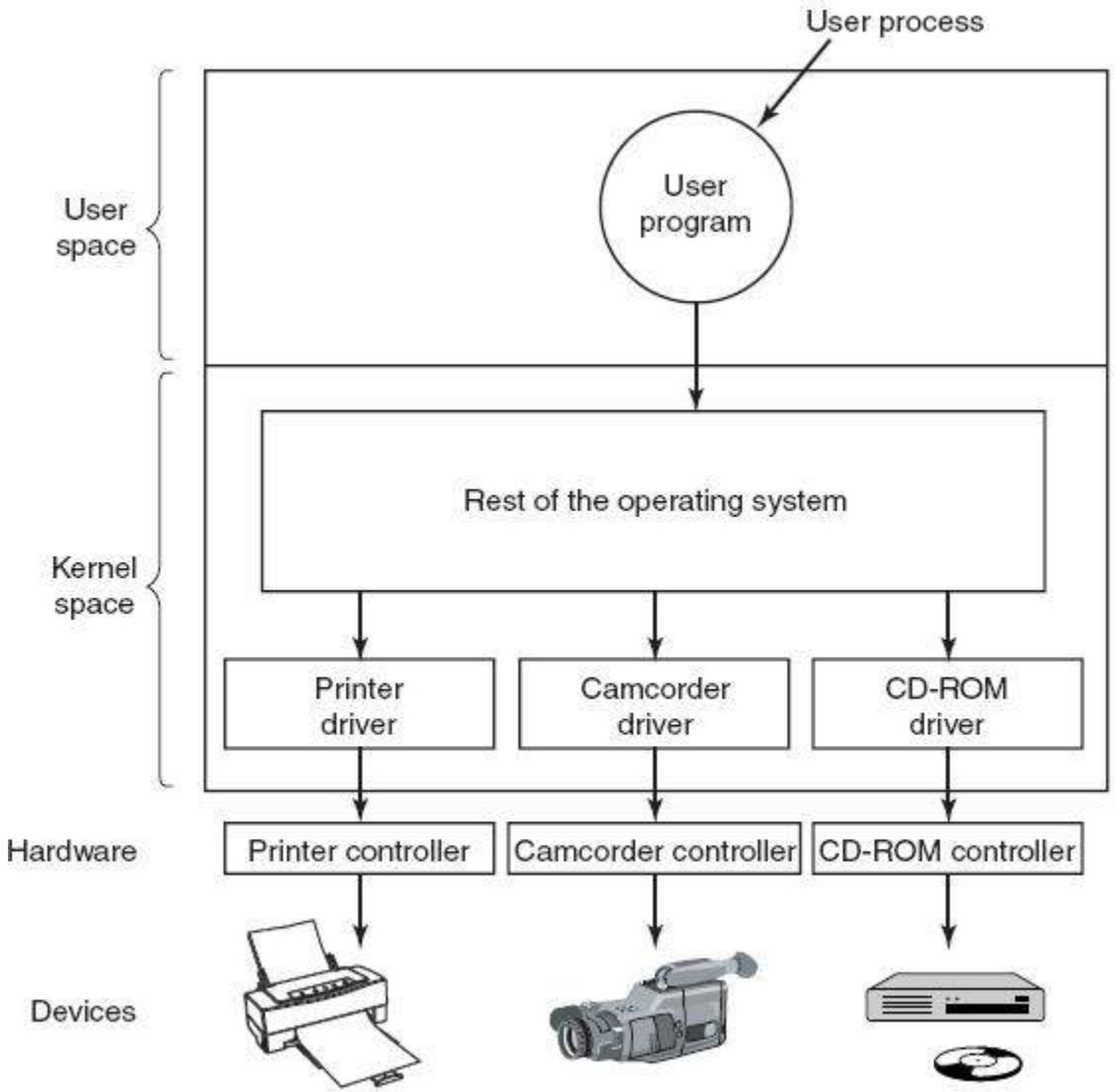


Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.

Device-Independent I/O Software

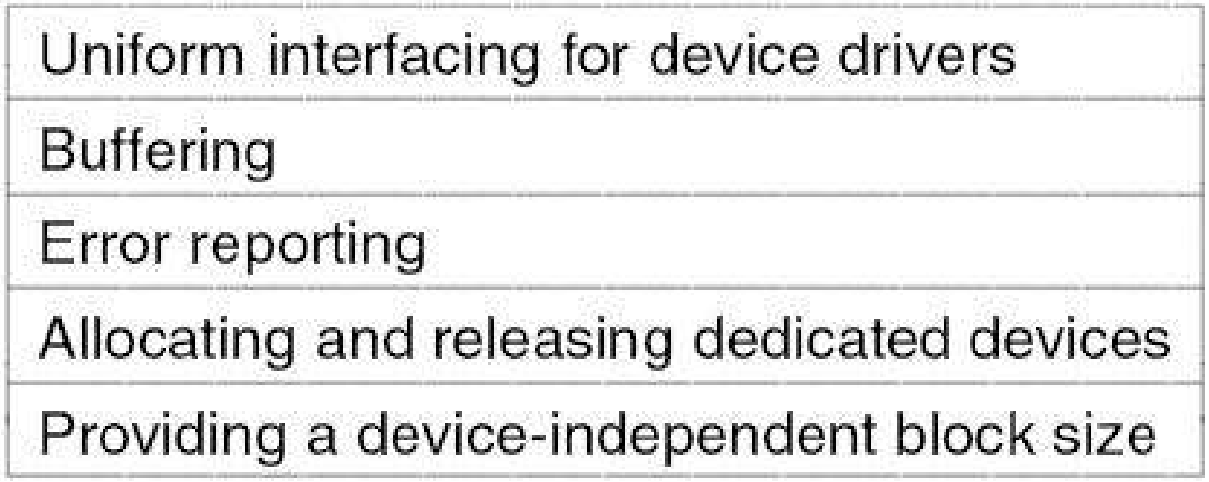


Figure 5-13. Functions of the device-independent I/O software.

Uniform Interfacing for Device Drivers

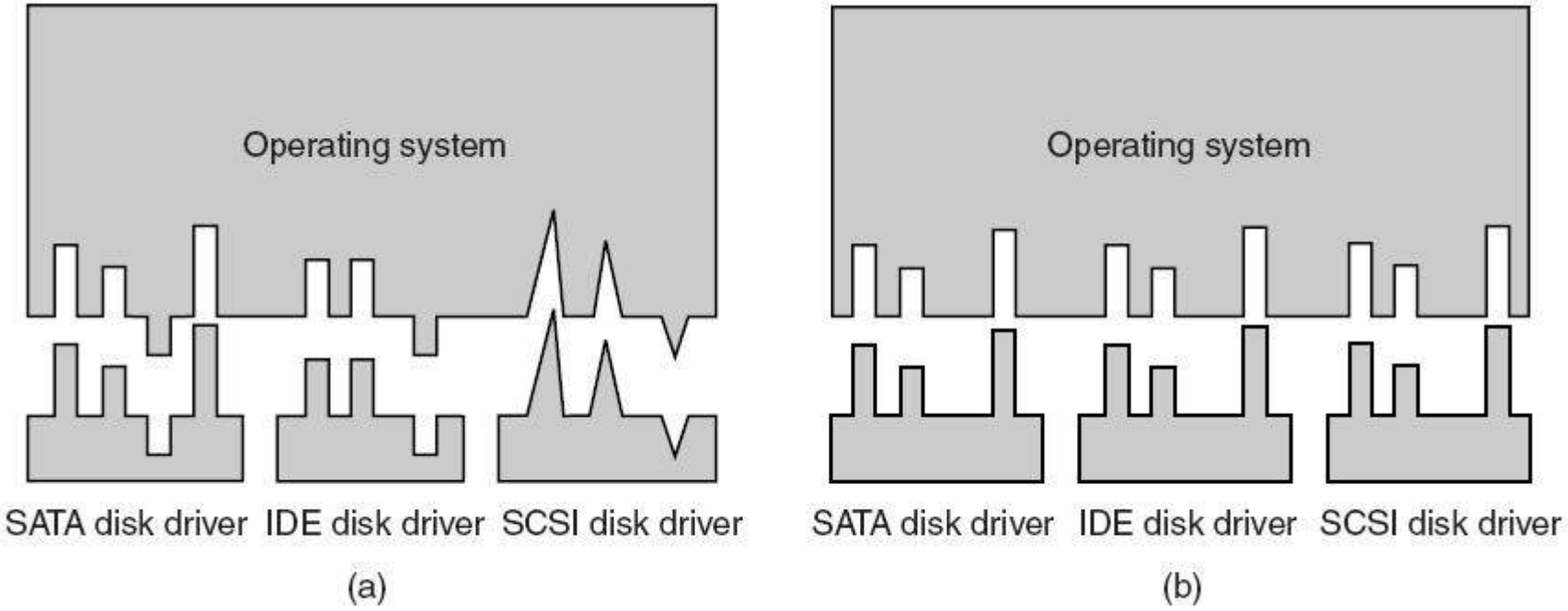


Figure 5-14. (a) Without a standard driver interface. (b) With a standard driver interface.

Buffering (1)

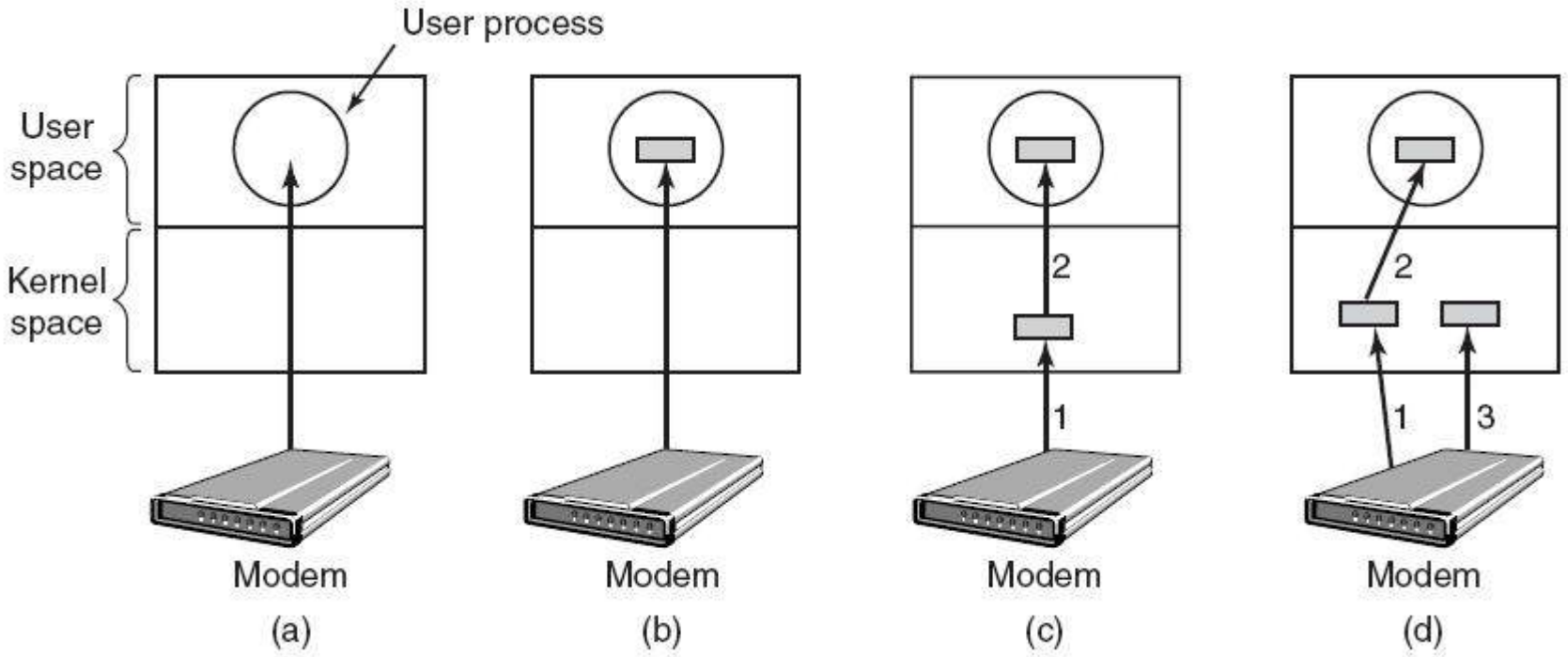


Figure 5-15. (a) Unbuffered input. (b) Buffering in user space. (c) Buffering in the kernel followed by copying to user space. (d) Double buffering in the kernel.

Buffering (2)

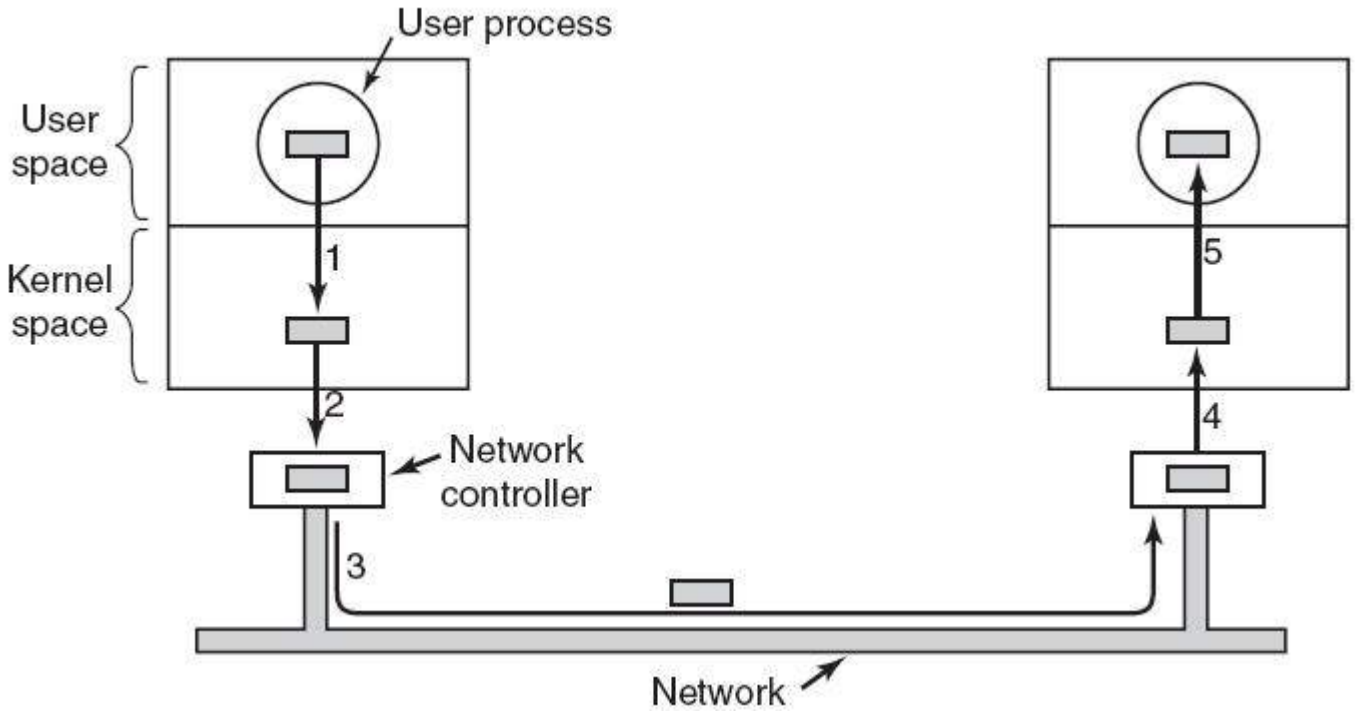


Figure 5-16. Networking may involve many copies of a packet.

User-Space I/O Software

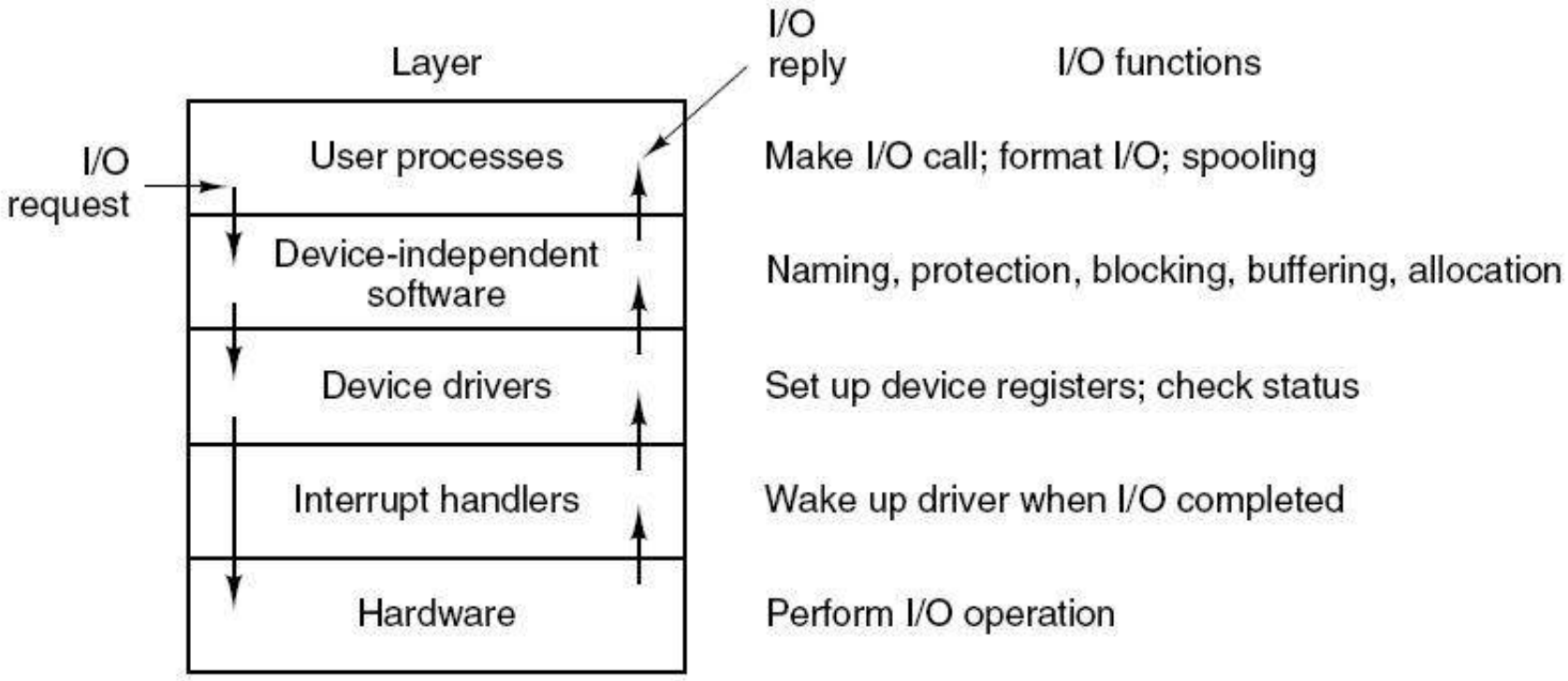


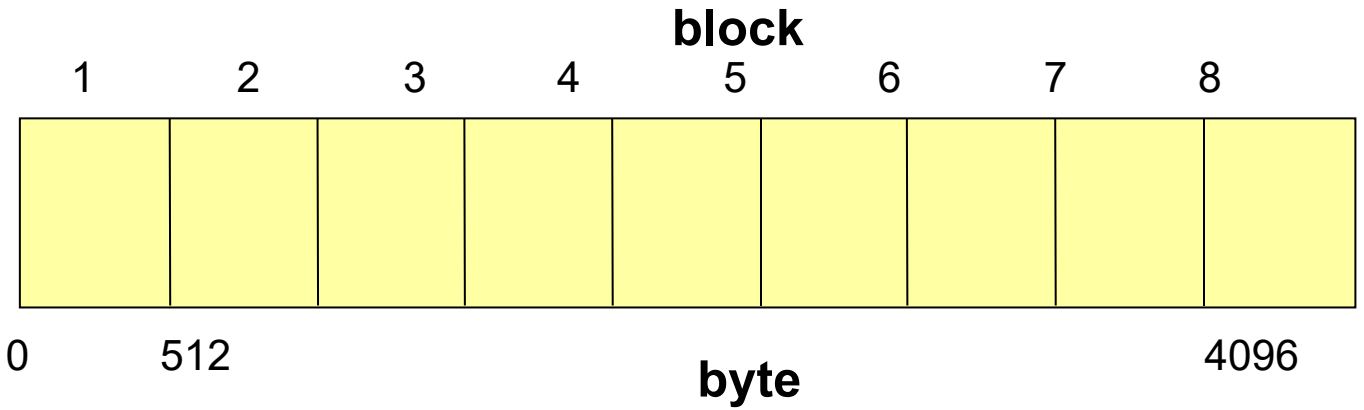
Figure 5-17. Layers of the I/O system and the main functions of each layer.

Character devices

- A *character device* is any device which does not allow random access (seeks)
- Examples:
  - Console (keyboard, mouse)
  - Serial terminals
  - Printers
  - Sound card
  - Random number generator

Block devices

- A *block device* is any device which allows random access ("seeks") and which is divided into "blocks" of a given size.



- Typical block devices:
  - Hard disks (and partitions)
  - Floppy disks
  - Virtual block devices (RAID and LVM)

Magnetic Disks (1)

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

Figure 5-18. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 18300 hard disk.

Magnetic Disks (2)

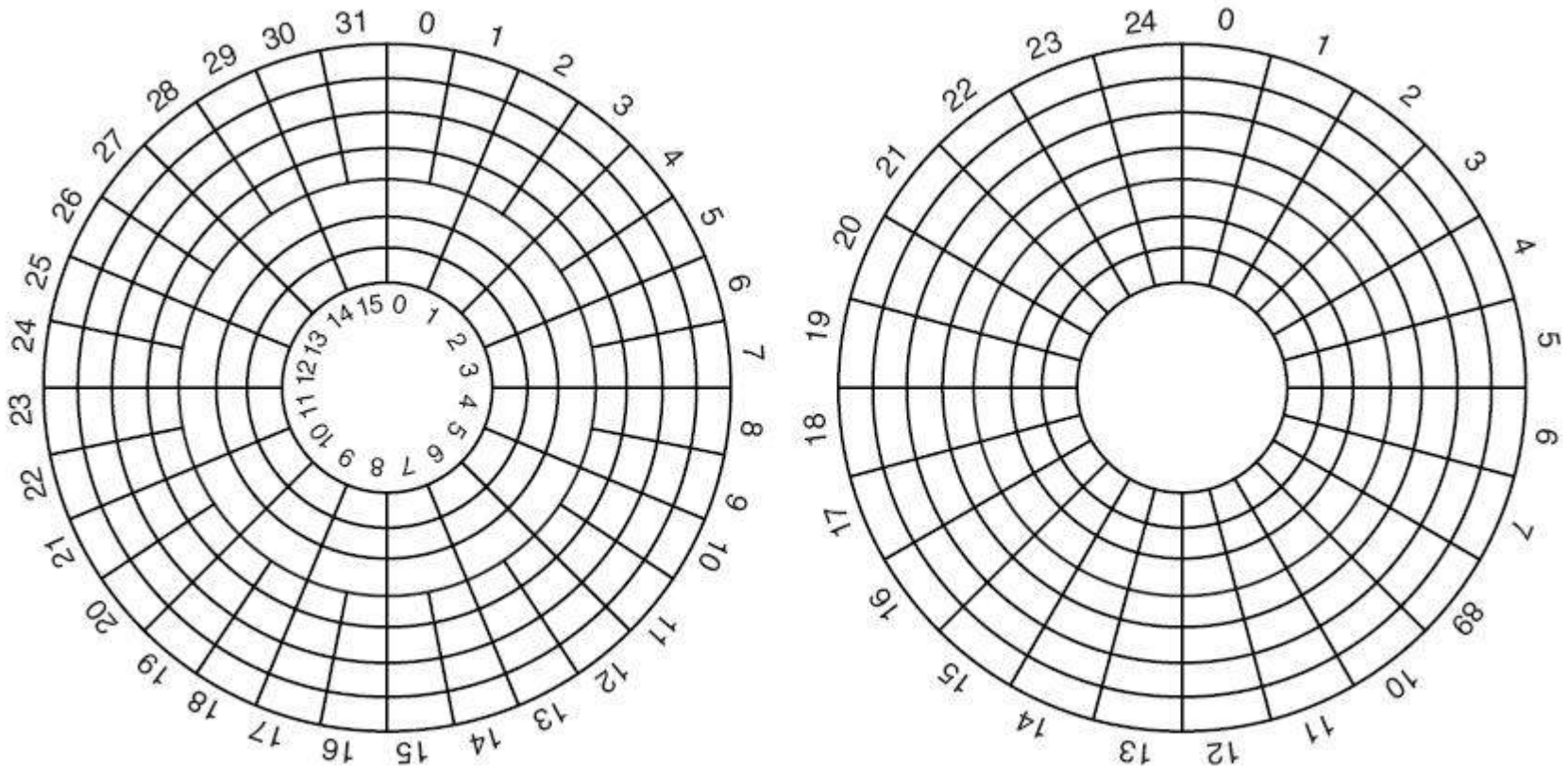


Figure 5-19. (a) Physical geometry of a disk with two zones. (b) A possible virtual geometry for this disk.

CD-ROMs (1)

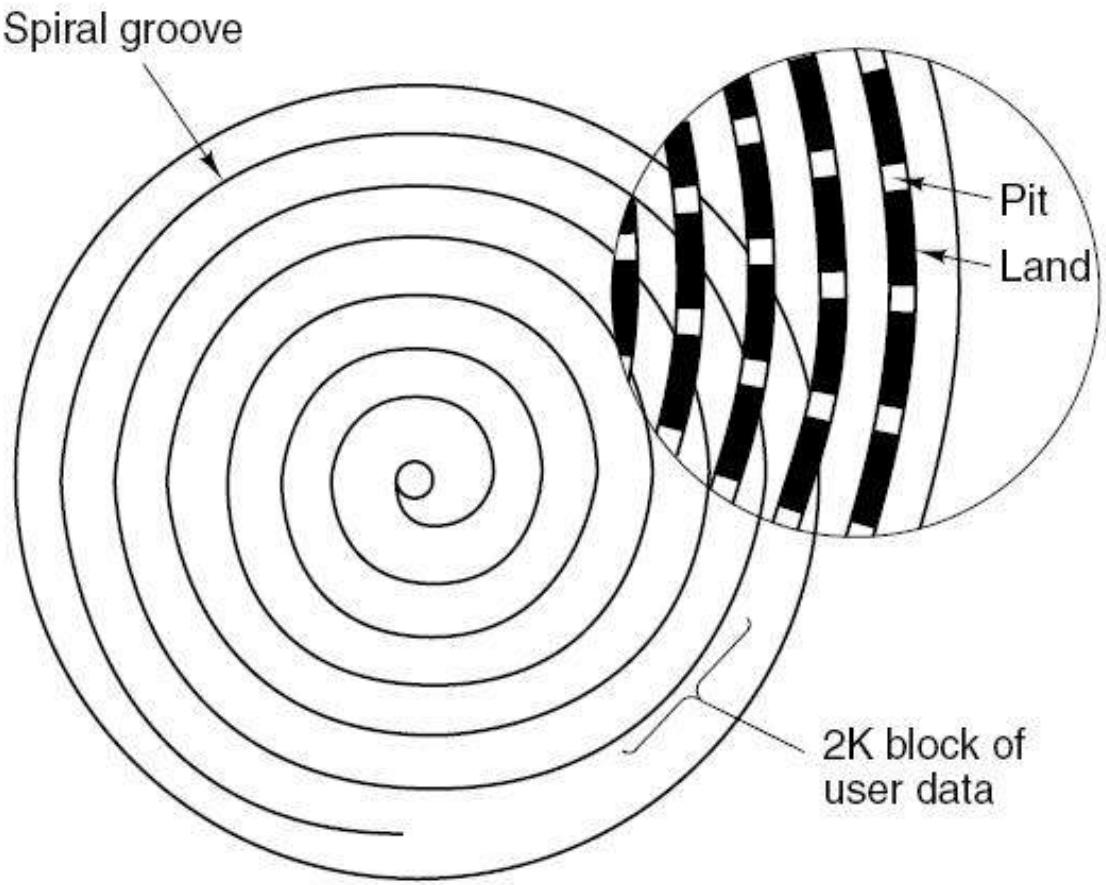


Figure 5-21. Recording structure of a compact disc or CD-ROM.

DVD (1)

DVD Improvements on CDs

1. Smaller pits
   (0.4 microns versus 0.8 microns for CDs).
2. A tighter spiral
   (0.74 microns between tracks versus 1.6 microns for CDs).
3. A red laser
   (at 0.65 microns versus 0.78 microns for CDs).

DVD (2)

DVD Formats

1. Single-sided, single-layer (4.7 GB).
2. Single-sided, dual-layer (8.5 GB).
3. Double-sided, single-layer (9.4 GB).
4. Double-sided, dual-layer (17 GB).

Disk Formatting (1)

Figure 5-25. A disk sector.

Disk Formatting (2)

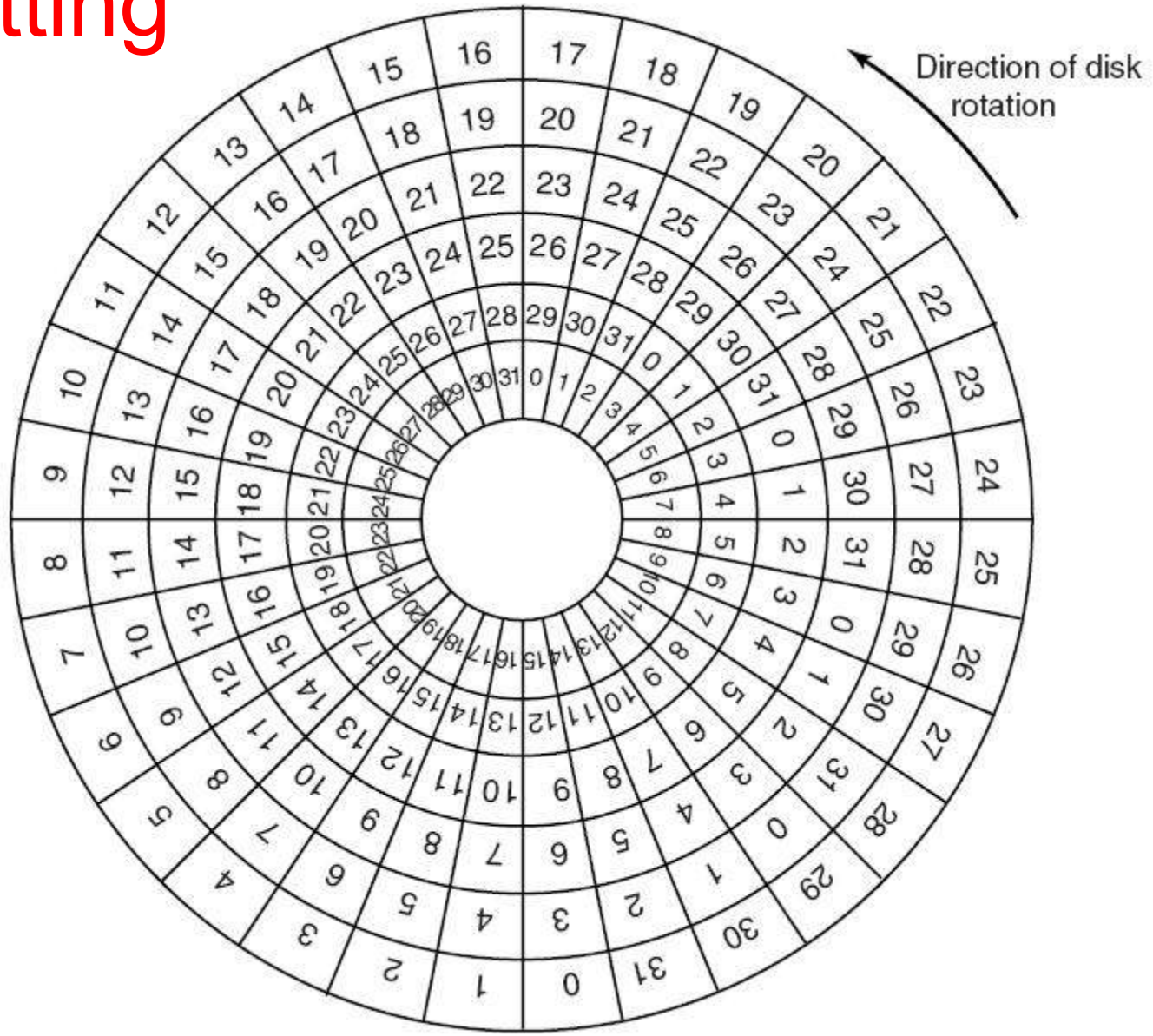


Figure 5-26. An illustration of cylinder skew.

Disk Formatting (3)

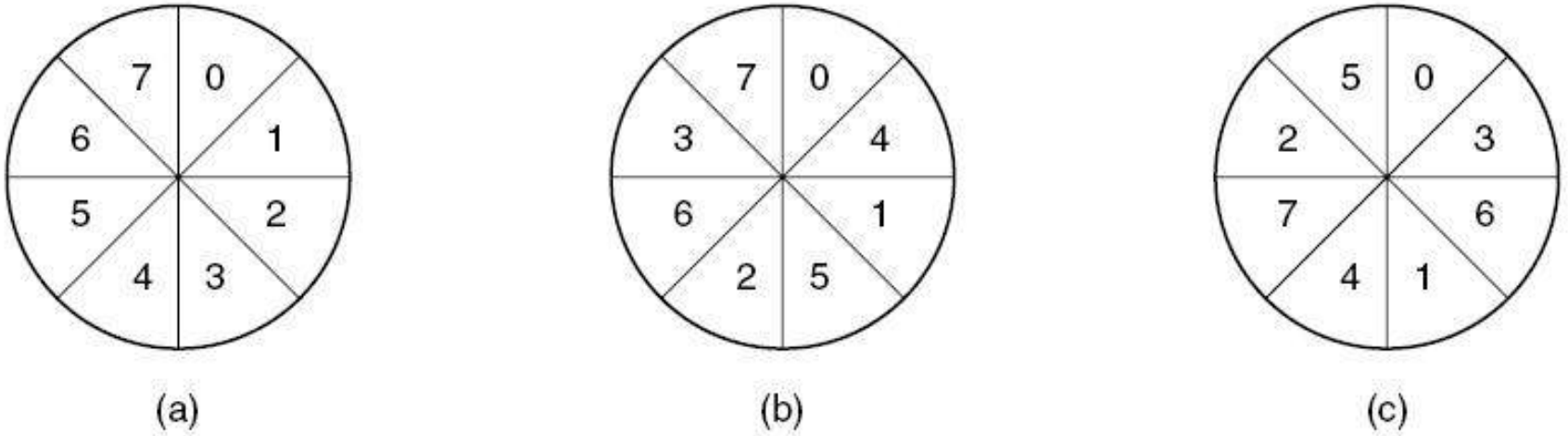


Figure 5-27. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.

Disk Arm Scheduling Algorithms (1)

Read/write time factors

1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (the time for the proper sector to rotate under the head).
3. Actual data transfer time.

Disk Arm Scheduling Algorithms (2)

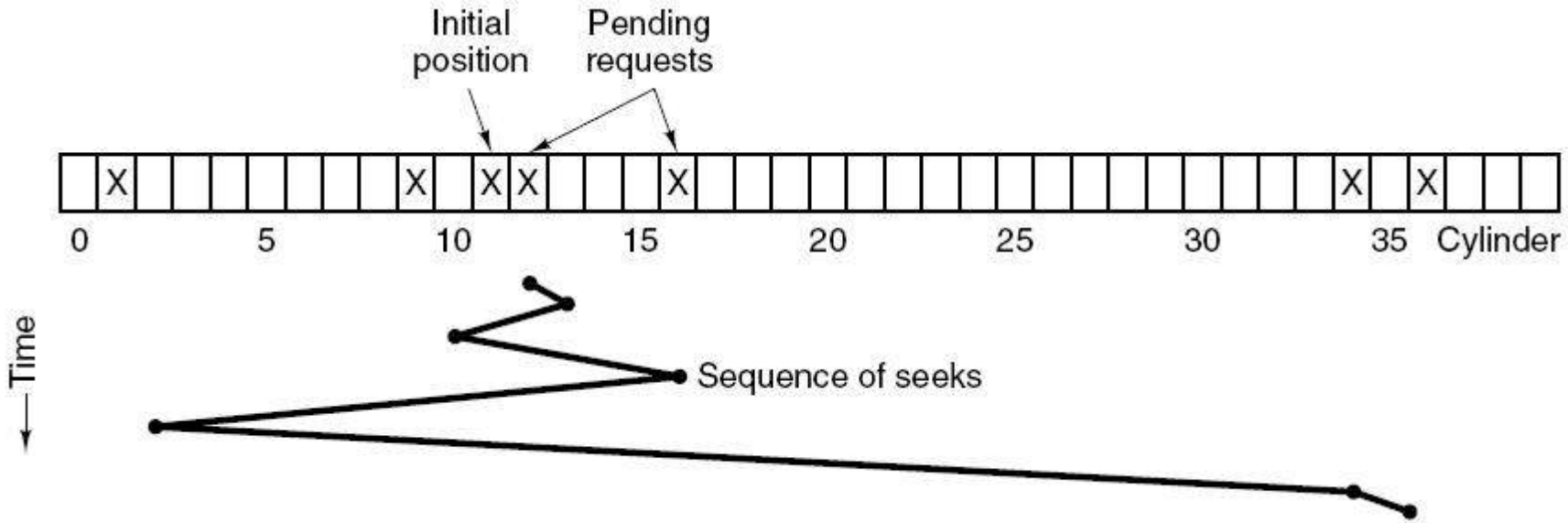


Figure 5-28. Shortest Seek First (SSF) disk scheduling algorithm.

Disk Arm Scheduling Algorithms (3)

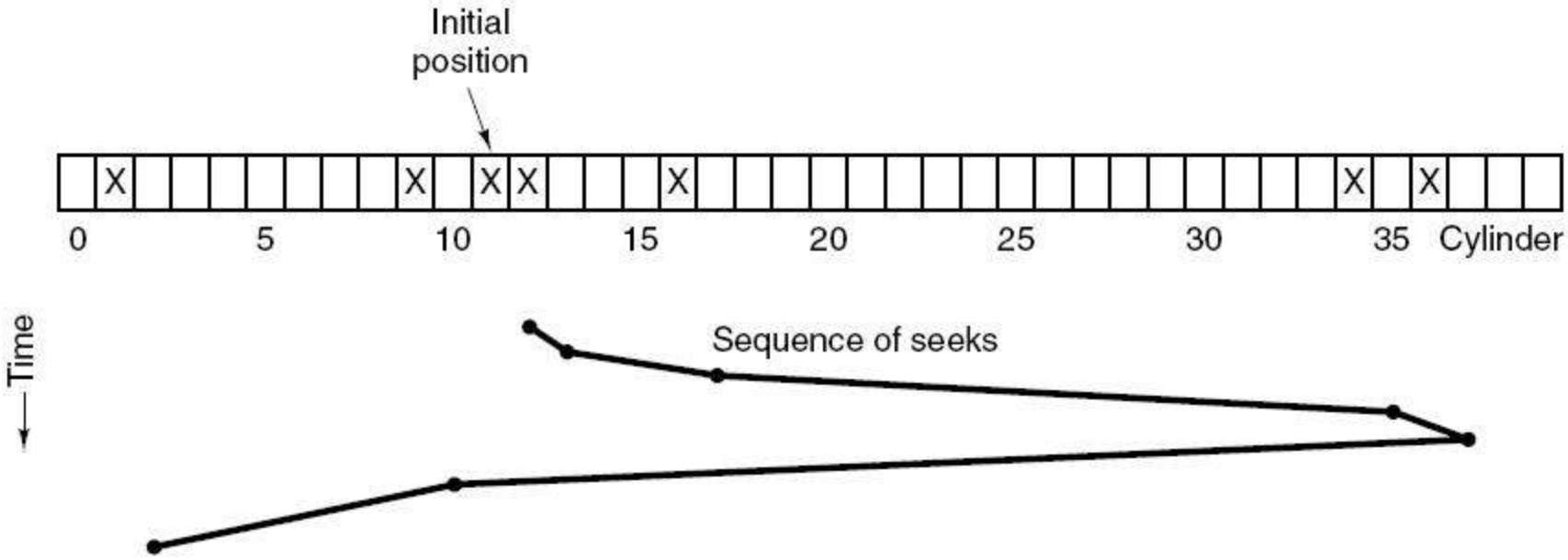


Figure 5-29. The elevator algorithm for scheduling disk requests.

Error Handling

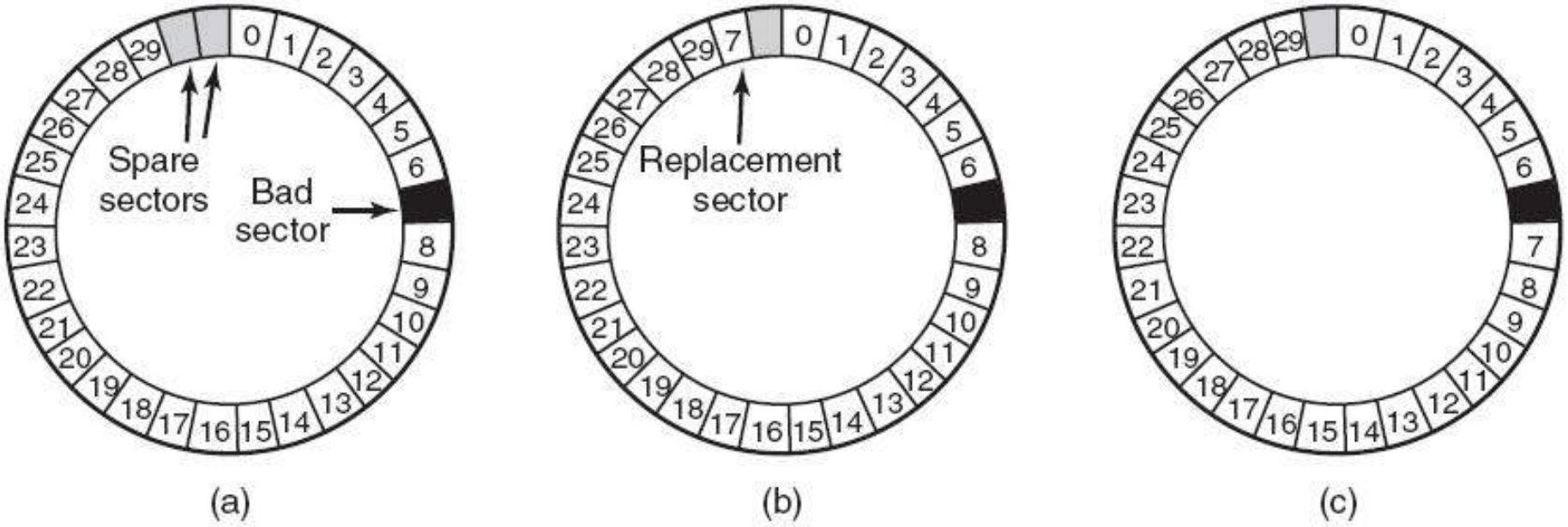


Figure 5-30. (a) A disk track with a bad sector.
(b) Substituting a spare for the bad sector.
(c) Shifting all the sectors to bypass the bad one.

Clock Hardware

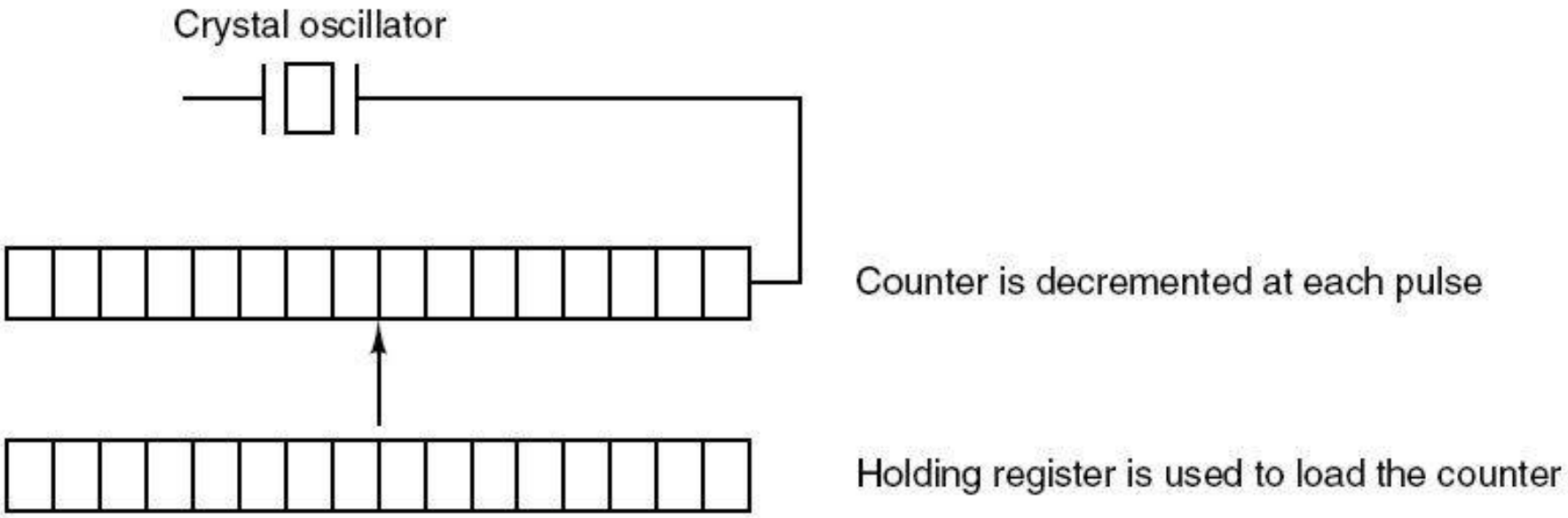


Figure 5-32. A programmable clock.

Clock Software (1)

Typical duties of a clock driver

1. Maintaining the time of day.
2. Preventing processes from running longer than they are allowed to.
3. Accounting for CPU usage.
4. Handling alarm system call made by user processes.
5. Providing watchdog timers for parts of the system itself.
6. Doing profiling, monitoring, statistics gathering.

Clock Software (2)

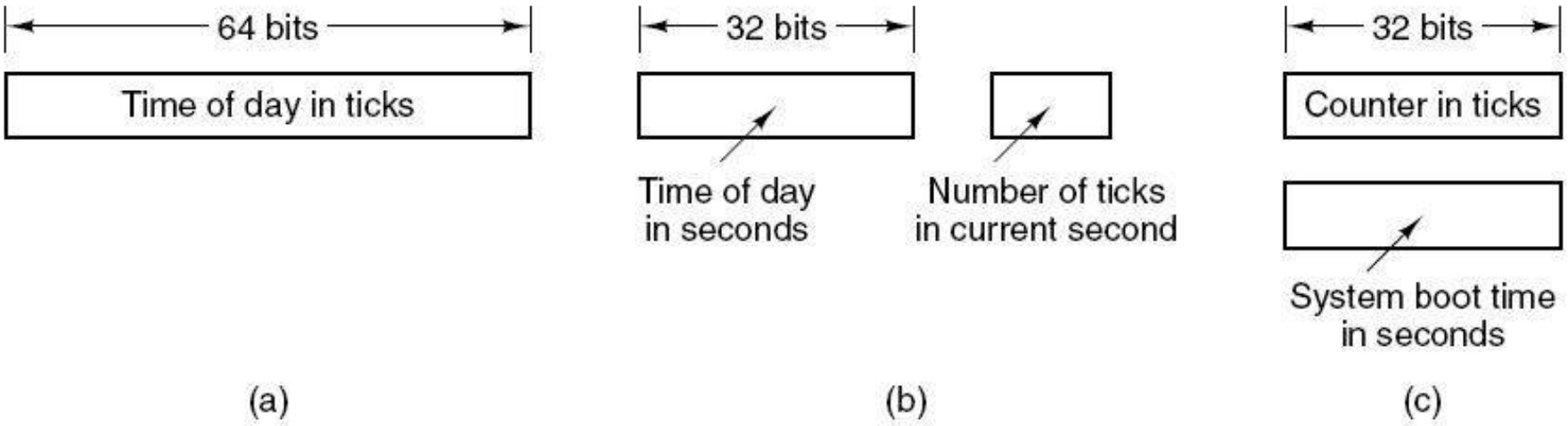


Figure 5-33. Three ways to maintain the time of day.

Keyboard Software

Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return (unchangeable)
CTRL-J	NL	Linefeed (unchangeable)

Figure 5-35. Characters that are handled specially in canonical mode.

The X Window System (1)

Escape sequence	Meaning
ESC [*n* A	Move up *n* lines
ESC [*n* B	Move down *n* lines
ESC [*n* C	Move right *n* spaces
ESC [*n* D	Move left *n* spaces
ESC [*m* ; *n* H	Move cursor to (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 1from start, 2 all)
ESC [*s* K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n* L	Insert *n* lines at cursor
ESC [*n* M	Delete *n* lines at cursor
ESC [*n* P	Delete *n* chars at cursor
ESC [*n* @	Insert *n* chars at cursor
ESC [*n* m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

Figure 5-36. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and *n*, *m*, and *s* are optional numeric parameters.

The X Window System (2)

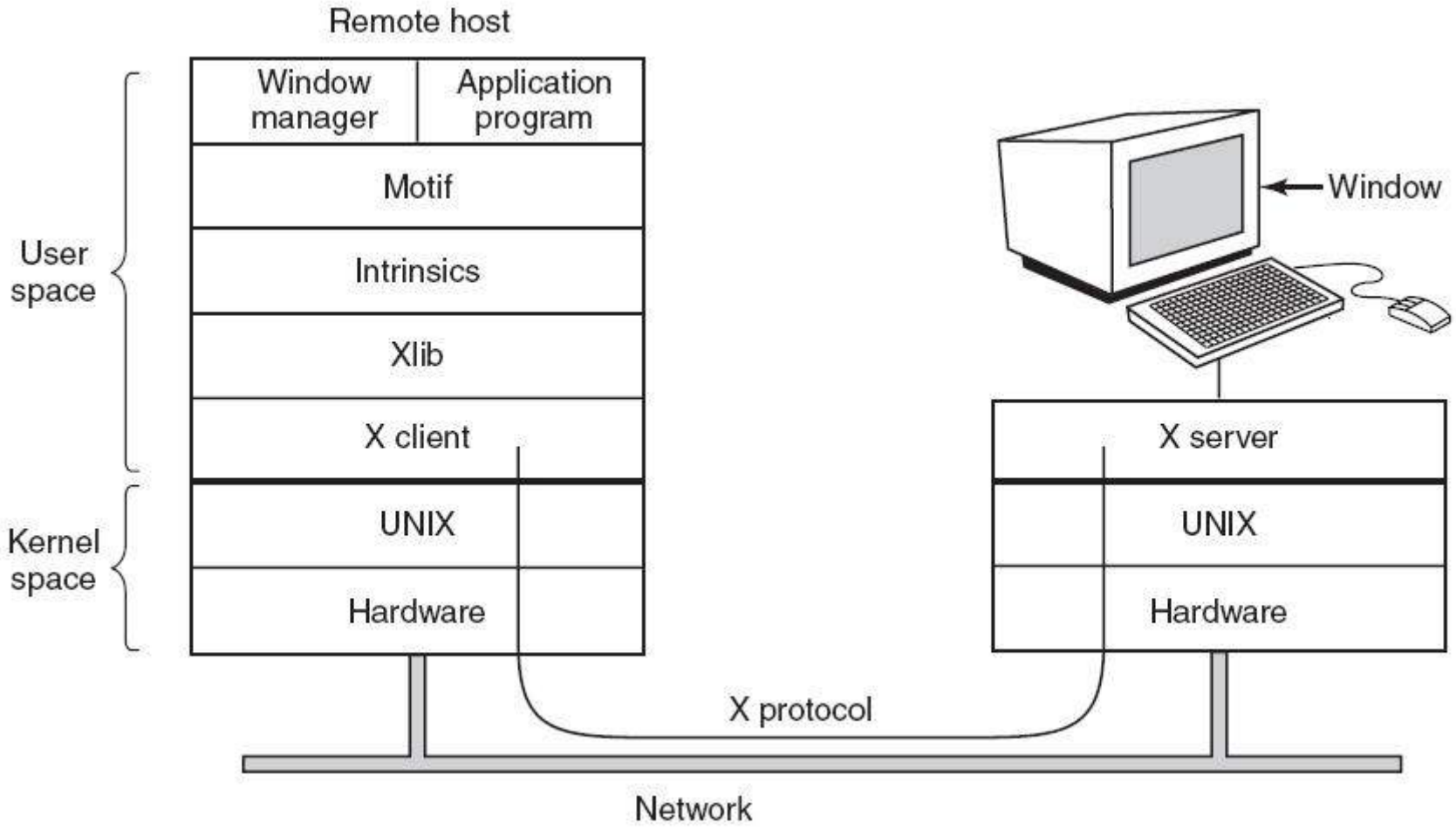


Figure 5-37. Clients and servers in the M.I.T. X Window System.

The X Window System (3)

Types of messages between client and server:

1. Drawing commands from the program to the workstation.
2. Replies by the workstation to program queries.
3. Keyboard, mouse, and other event announcements.
4. Error messages.

Graphical User Interfaces (1)

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(int argc, char *argv[])
{
        Display disp;                                   /* server identifier */
        Window win;                                     /* window identifier */
        GC gc;                                          /* graphic context identifier */
        XEvent event;                                   /* storage for one event */
        int running = 1;

        disp = XOpenDisplay("display_name");            /* connect to the X server */
        win = XCreateSimpleWindow(disp,  ... );         /* allocate memory for new window */
        XSetStandardProperties(disp, ...);      /* announces window to window mgr */
        gc = XCreateGC(disp, win, 0, 0);        /* create graphic context */
        XSelectInput(disp, win, ButtonPressMask | KeyPressMask | ExposureMask);
        XMapRaised(disp, win);                  /* display window; send Expose event */

        . . .
```

Figure 5-38. A skeleton of an X Window application program.

Graphical User Interfaces (2)

```
. . .
    while (running) {
        XNextEvent(disp, &event);          /* get next event */
        switch (event.type) {
            case Expose:        ...;  break;    /* repaint window */
            case ButtonPress:   ...;  break;    /* process mouse click */
            case Keypress:      ...;  break;    /* process keyboard input */
        }
    }

    XFreeGC(disp, gc);                     /* release graphic context */
    XDestroyWindow(disp, win);             /* deallocate window's memory space */
    XCloseDisplay(disp);                   /* tear down network connection */
}
```

Figure 5-38. A skeleton of an X Window application program.

Graphical User Interfaces (3)

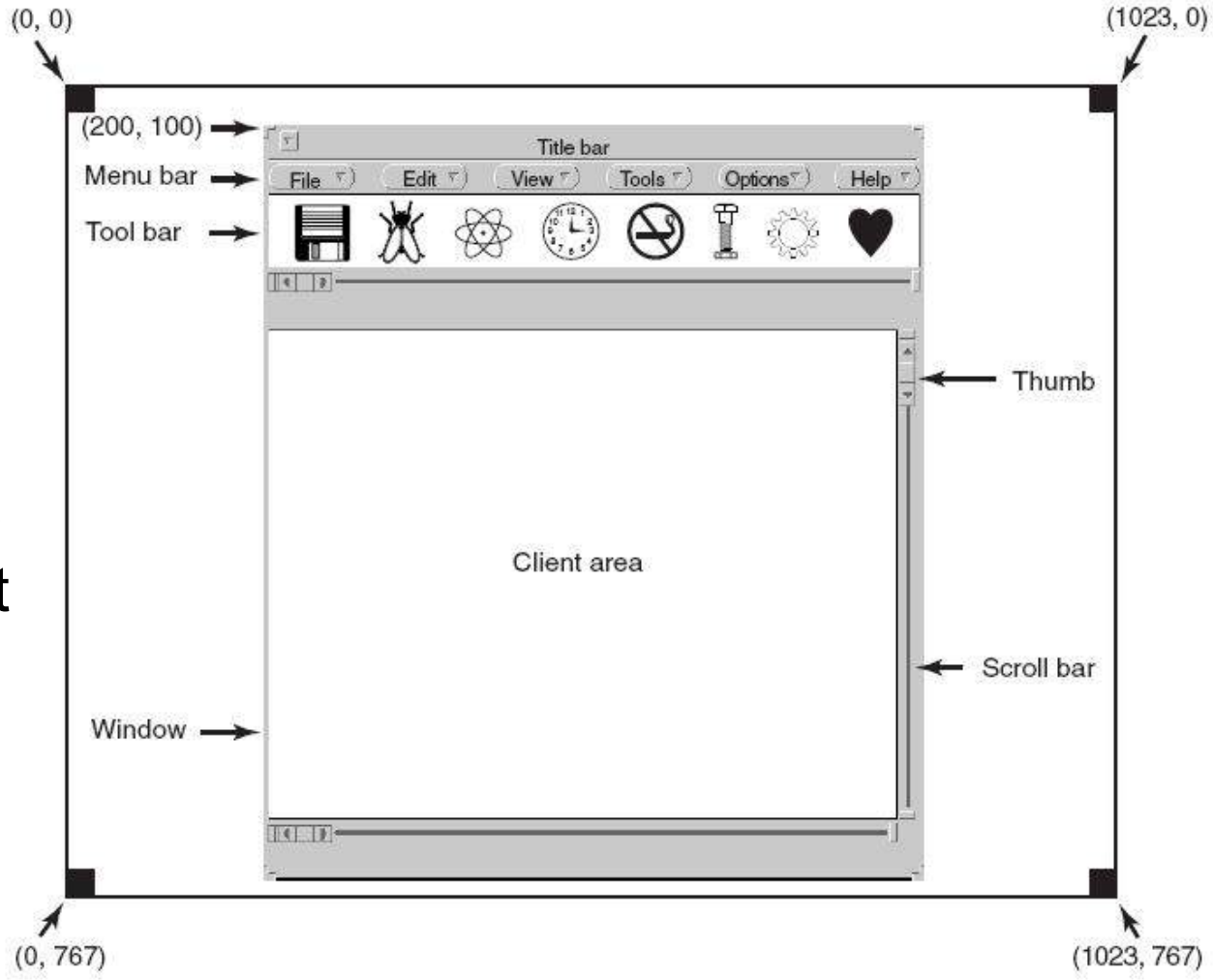


Figure 5-39. A sample window located at (200, 100) on an XGA display.

Graphical User Interfaces (4)

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE h, HINSTANCE, hprev, char *szCmd, int iCmdShow)
{
    WNDCLASS wndclass;                  /* class object for this window */
    MSG msg;                            /* incoming messages are stored here */
    HWND hwnd;                          /* handle (pointer) to the window object */

    /* Initialize wndclass */
    wndclass.lpfnWndProc = WndProc;     /* tells which procedure to call */
    wndclass.lpszClassName = "Program name";    /* Text for title bar */
    wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);    /* load program icon */
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);      /* load mouse cursor */

    RegisterClass(&wndclass);           /* tell Windows about wndclass */
    hwnd = CreateWindow ( ... )         /* allocate storage for the window */
    ShowWindow(hwnd, iCmdShow);         /* display the window on the screen */
    UpdateWindow(hwnd);                 /* tell the window to paint itself */
  . . .
```

Figure 5-40. A skeleton of a Windows main program.

Graphical User Interfaces (5)

```
. . .
        while (GetMessage(&msg, NULL, 0, 0)) {          /* get message from queue */
                TranslateMessage(&msg);         /* translate the message */
                DispatchMessage(&msg);          /* send msg to the appropriate procedure */
        }
        return(msg.wParam);
}

long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
        /* Declarations go here. */

        switch (message) {
                case WM_CREATE:     ... ;   return ... ;    /* create window */
                case WM_PAINT:      ... ;   return ... ;    /* repaint contents of window */
                case WM_DESTROY:    ... ;   return ... ;    /* destroy window */
        }
        return(DefWindowProc(hwnd, message, wParam, lParam));       /* default */
}
```

Figure 5-40. A skeleton of a Windows main program.

Power Management Hardware Issues

Device	Li et al. (1994)	Lorch and Smith (1998)
Display	68%	39%
CPU	12%	18%
Hard disk	20%	12%
Modem		6%
Sound		2%
Memory	0.5%	1%
Other		22%

Figure 5-45. Power consumption of various parts of a notebook computer.

Power Management
The Display

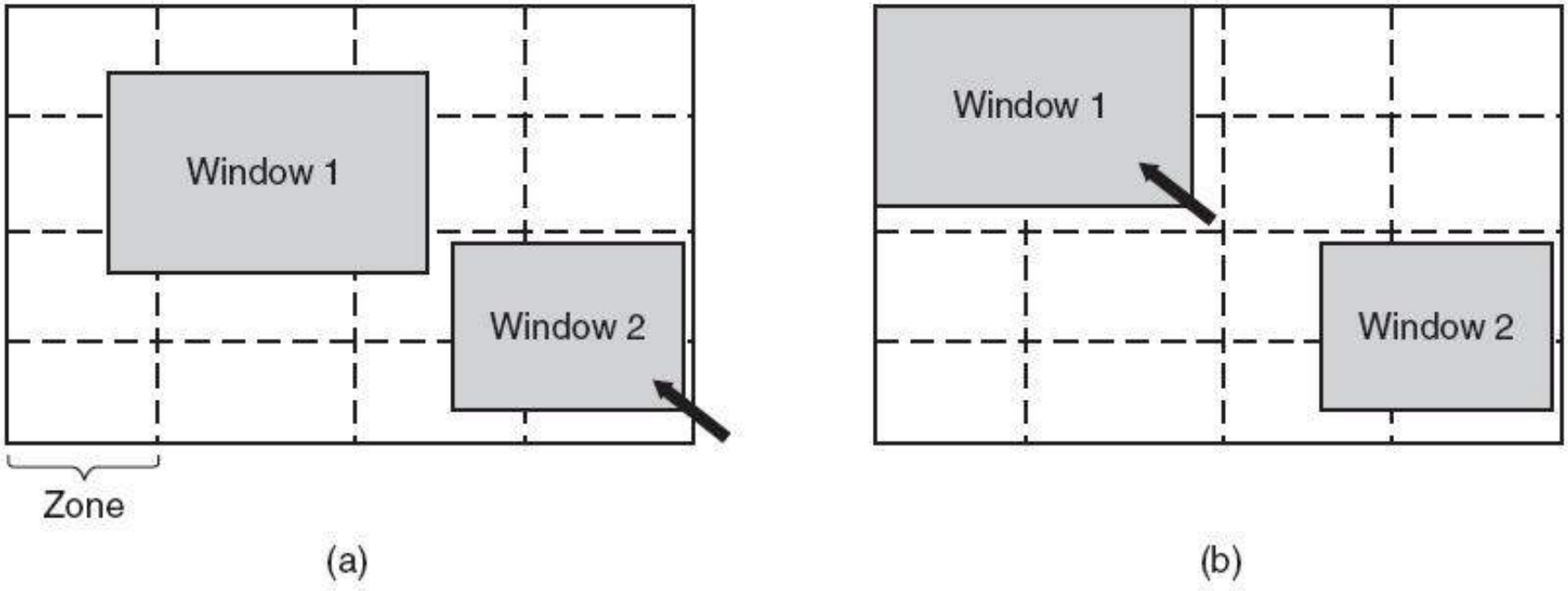


Figure 5-46. The use of zones for backlighting the display. (a) When window 2 is selected it is not moved. (b) When window 1 is selected, it moves to reduce the number of zones illuminated.

Power Management
The CPU

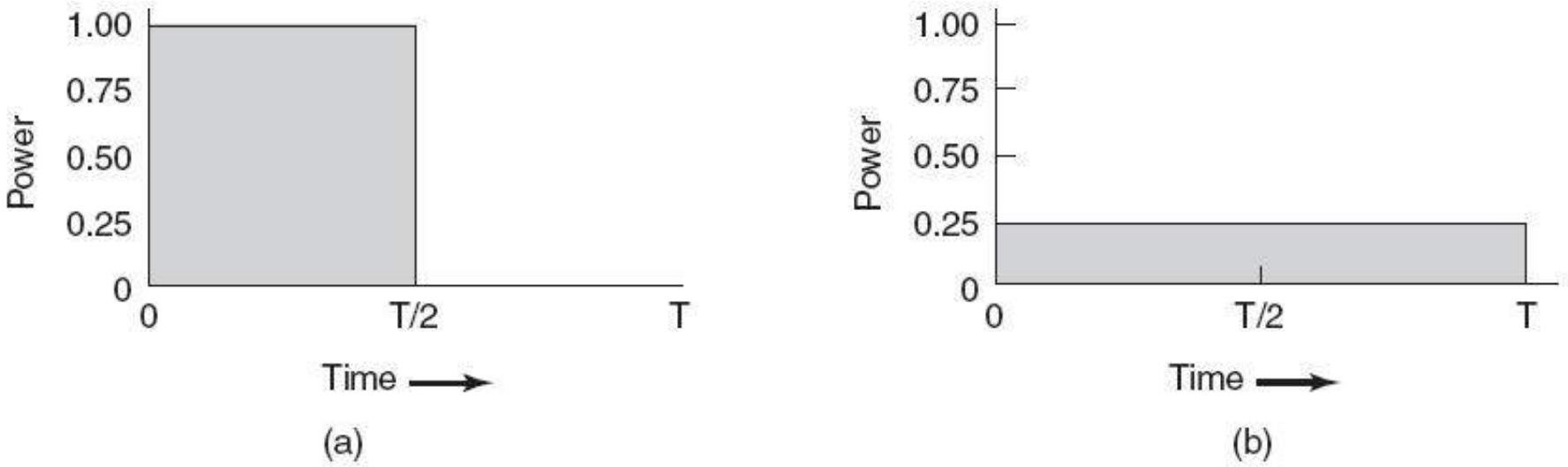


Figure 5-47. (a) Running at full clock speed. (b) Cutting voltage by two cuts clock speed by two and power consumption by four.

作业

- P241 11， 24