File Systems

José R. Ortiz-Ubarri

Modified slides from book

Modern Operating Systems

Long-term Information Storage

1. Must store large amounts of data

2. Information stored must survive the termination of the process using it

3. Multiple processes must be able to access the information concurrently

File naming

Extension	Meaning
*.bak	Backup File
*.c	C source program
*.gif	Compuserve Graphical Interchange Format image
*.hlp	A help file
*.md	A markdown file
*.html	World Wide Web HyperText Markup Language document
*.jpg	Still picture encoded with the JPEG standard

File naming

Extension	Meaning
*.mp3	Music encoded in MPEG layer 3 audio format
*.mpg	Movie encoded with the MPEG standard
*.o	Object file (compiler output, not yet linked)
*.pdf	Portable Document Format file
*.ps	PostScript file
*.tex	Input for the TEX formatting program
*.txt	General text file
*.zip	Compressed archive

File System

Think of a disk as a linear sequence of fixed-size blocks and supporting two operations:

1. Read block k.
2. Write block k

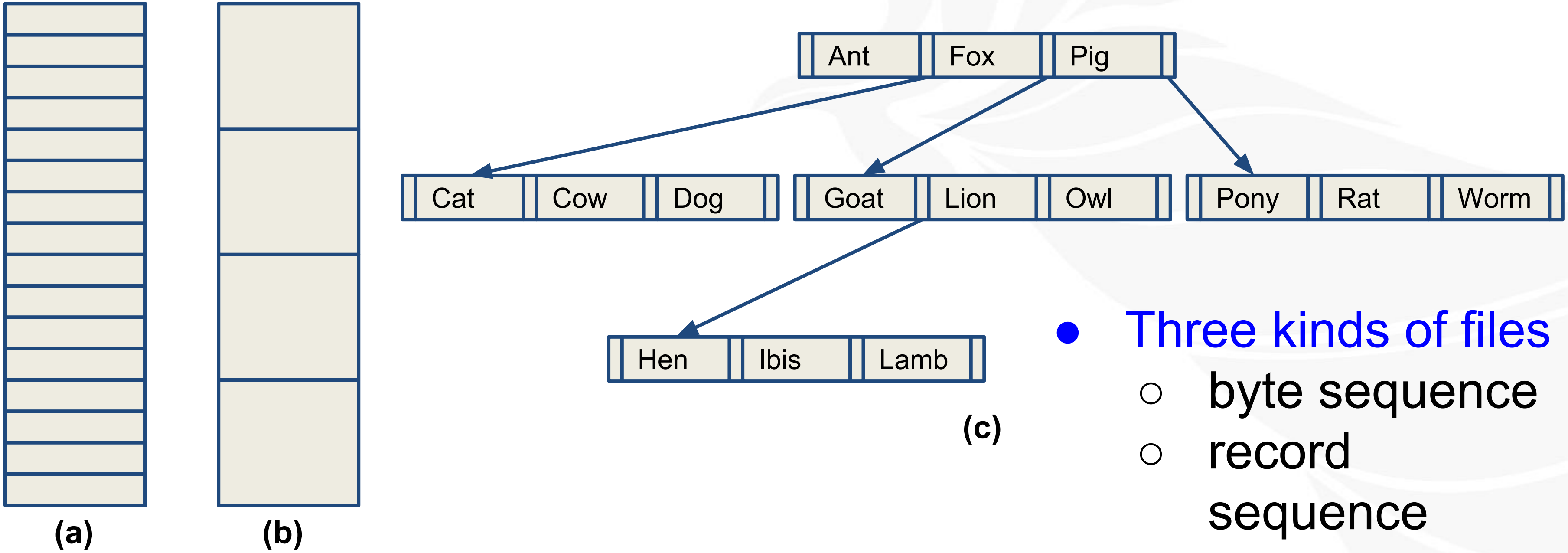
5 MB hard drive being shipped by IBM - 1956

File Sytem

Questions that quickly arise:

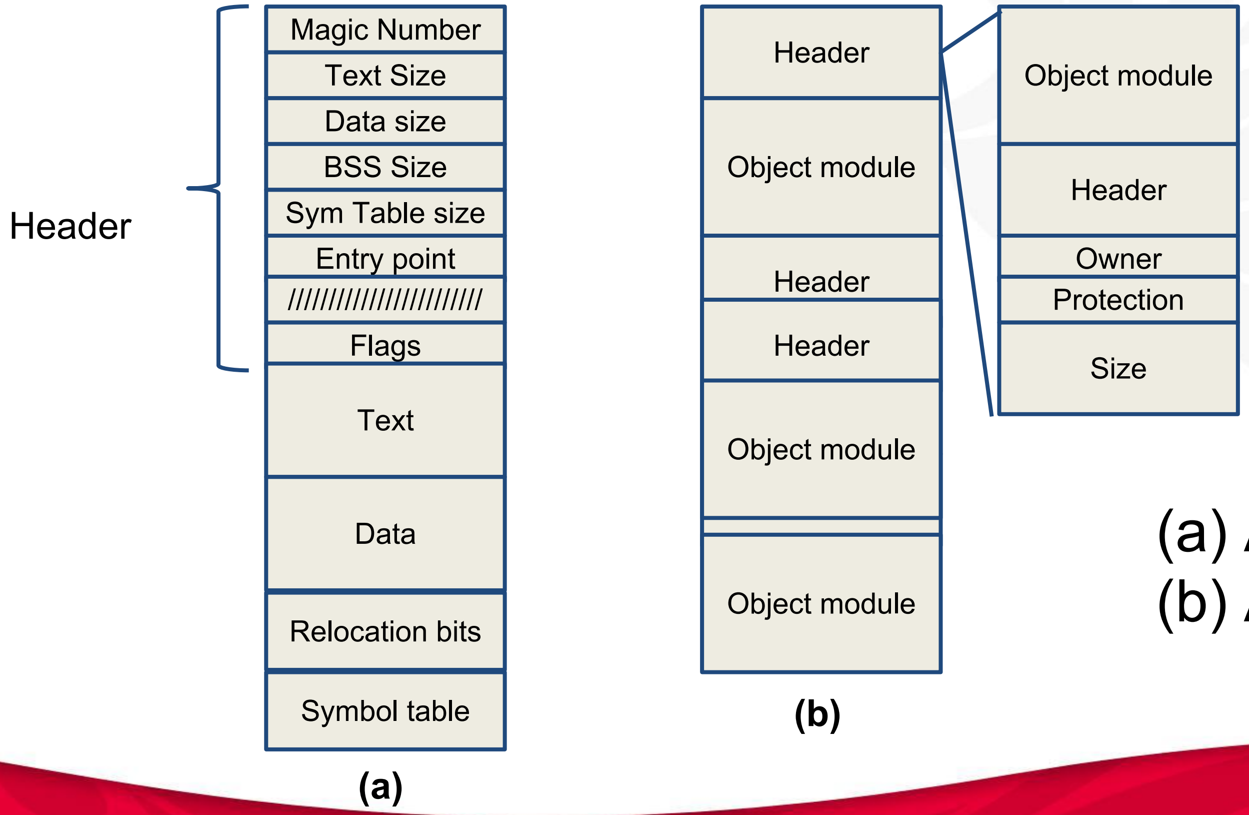
1. How do you find information?
2. How do you keep one user from reading another user's data?
3. How do you know which blocks are free?

File Structure



- Three kinds of files
    - byte sequence
    - record sequence
    - tree

File Types



(a) An executable file
(b) An archive

File Access

- Sequential access
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - convenient when medium was mag tape
- Random access
  - bytes/records read in any order
  - essential for database systems
  - read can be …
  - move file marker (seek), then read or …
  - read and then move file marker

File attributes

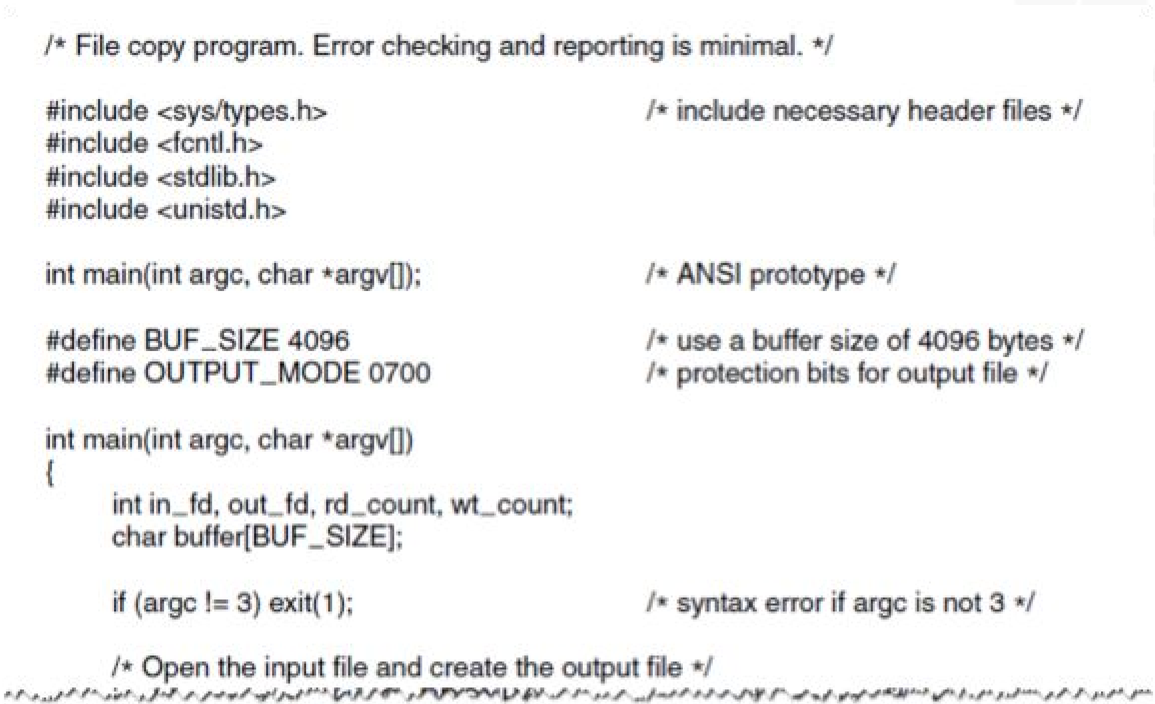
Attribute	Meaning
Protection	Who can access the file and what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up

Attribute	Meaning
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 has last changed

File Operations

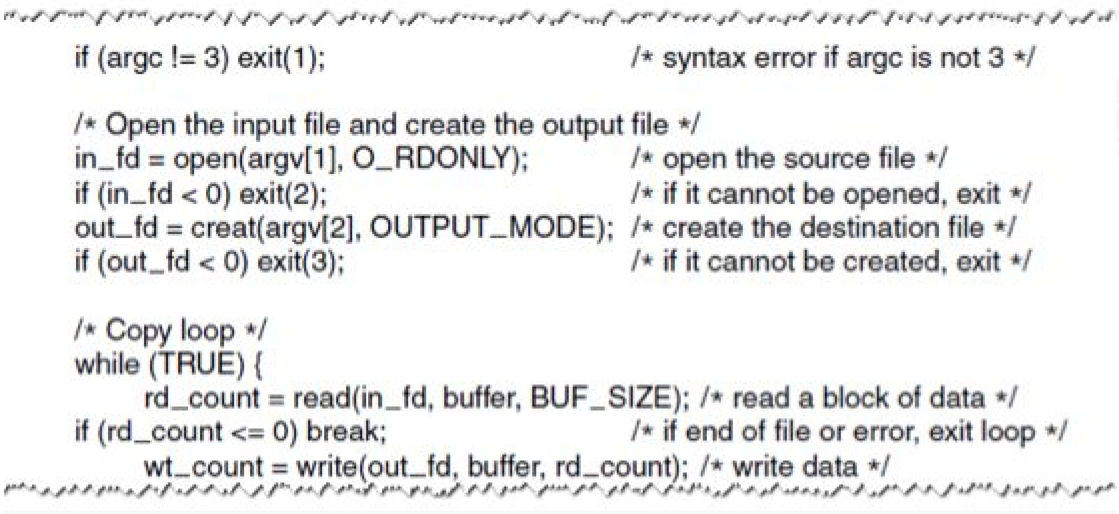
1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

Example Program Using File System Calls



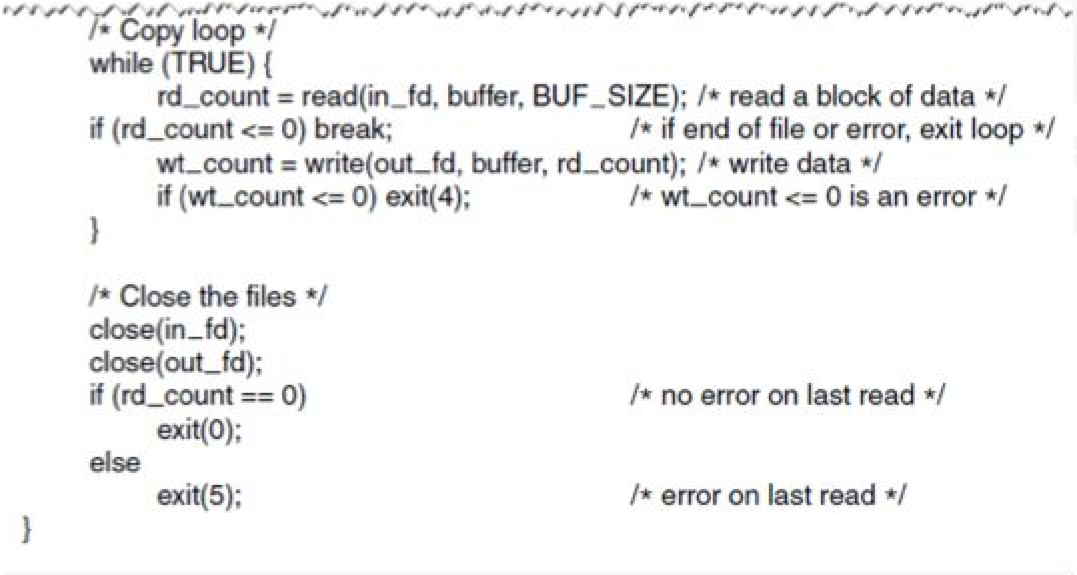
```
/* File copy program. Error checking and reporting is minimal. */

#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 */
```

Example Program Using File System Calls



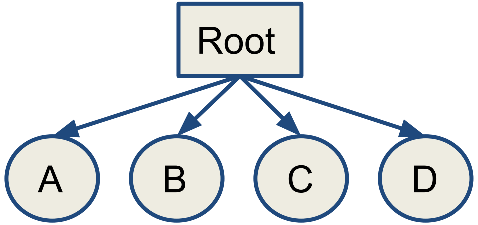
```
if (argc != 3) exit(1);                      /* syntax error if argc is not 3 */

/* 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 */

/* 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 */
```

Example Program Using File System Calls



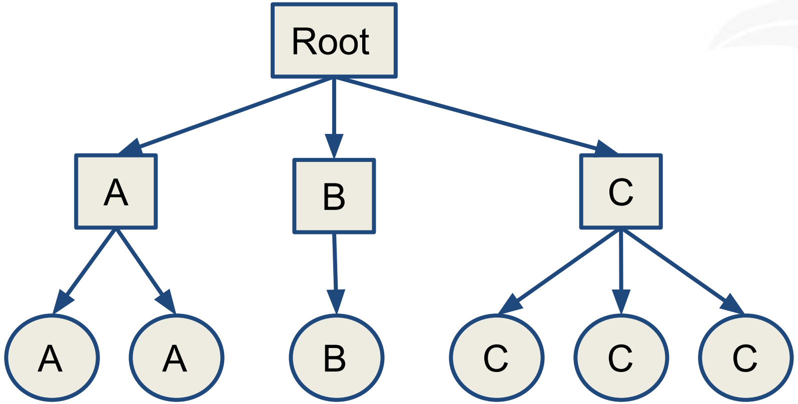
```
    /* Copy loop */
    while (TRUE) {
         rd_count = read(in_fd, buffer, BUF_SIZE); /* read a block of data */
    if (rd_count <= 0) break;                 /* if end of file or error, exit loop */
         wt_count = write(out_fd, buffer, rd_count); /* write data */
         if (wt_count 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

Single Level Directory System



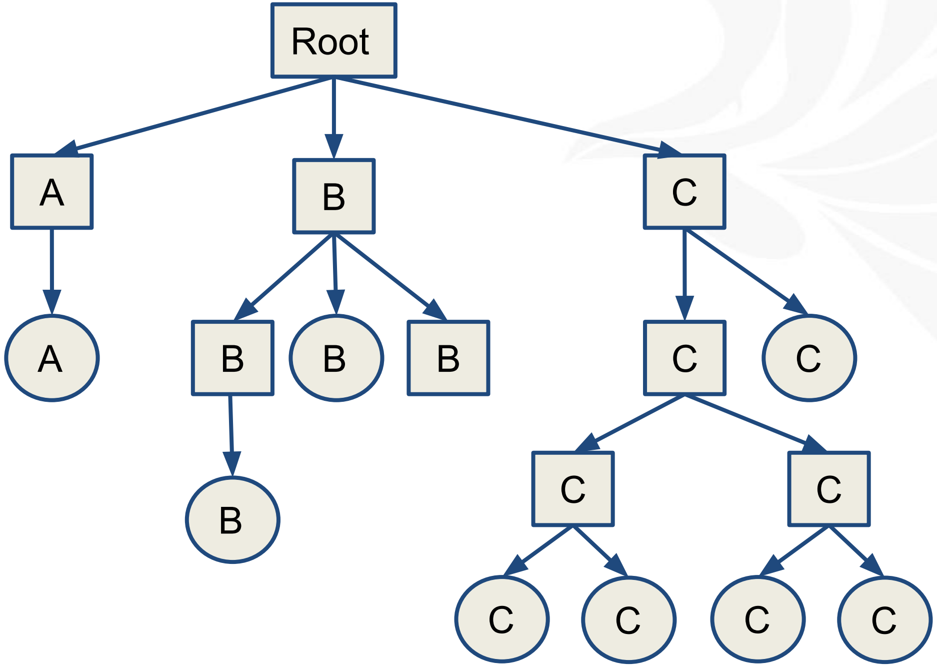
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

Two Level Directory System



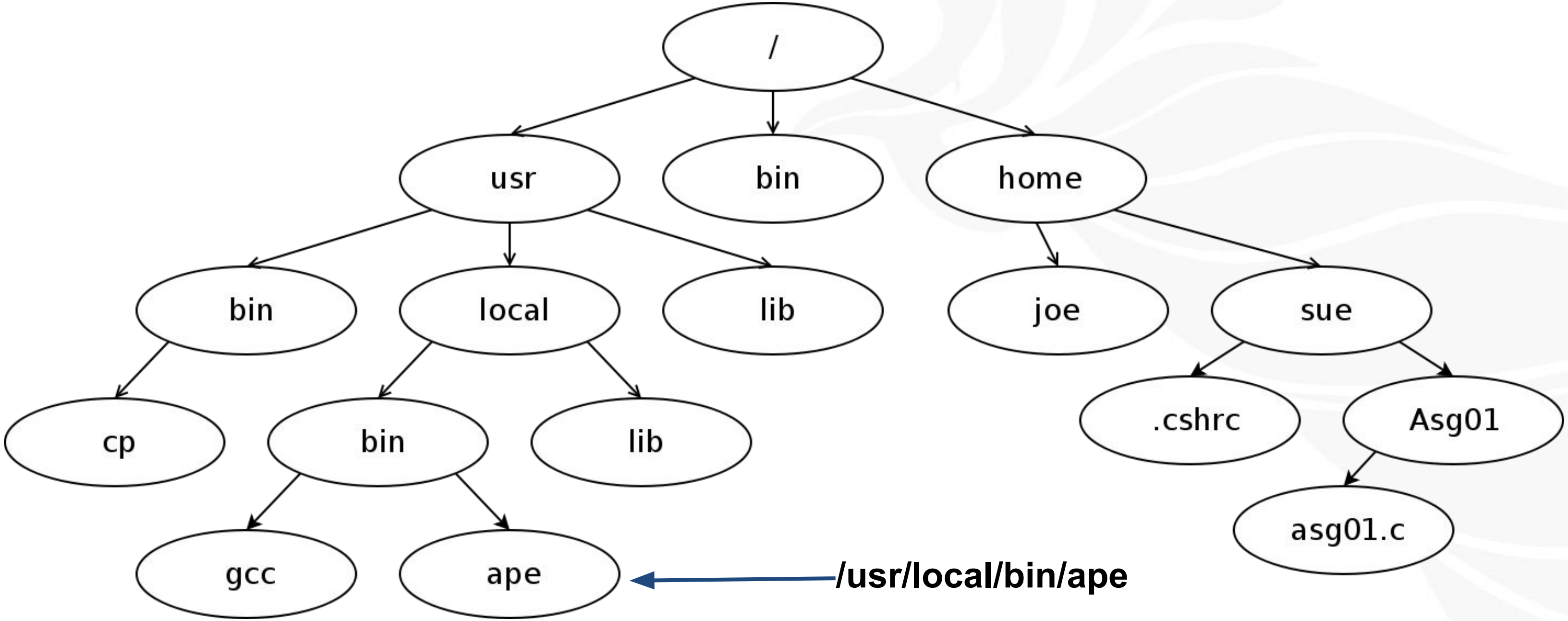
Letters indicate owners of the directories and files

Hierarchical Directory Systems



A hierarchical directory system

A UNIX directory tree (Path Names)

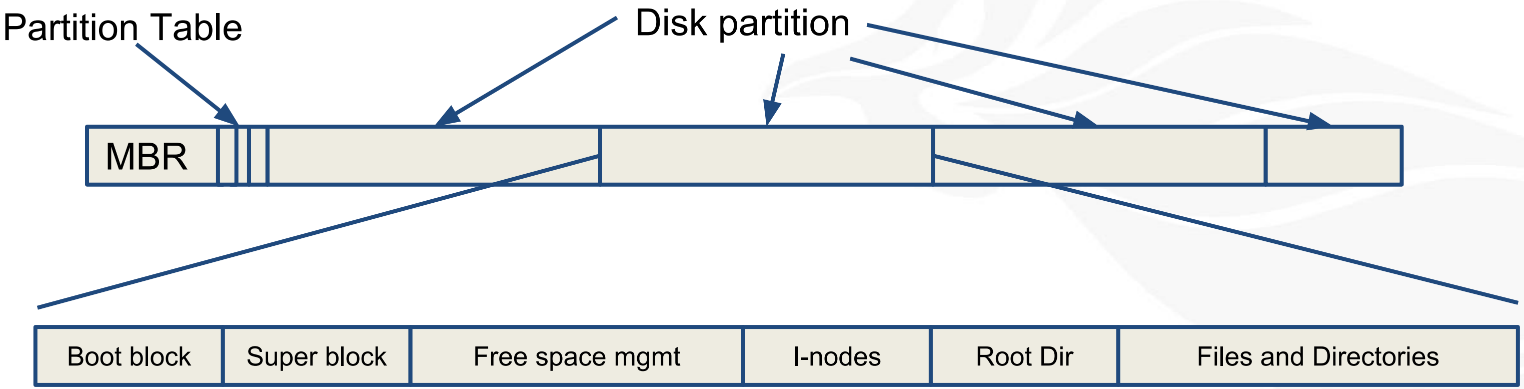


/usr/local/bin/ape

Directory Operations

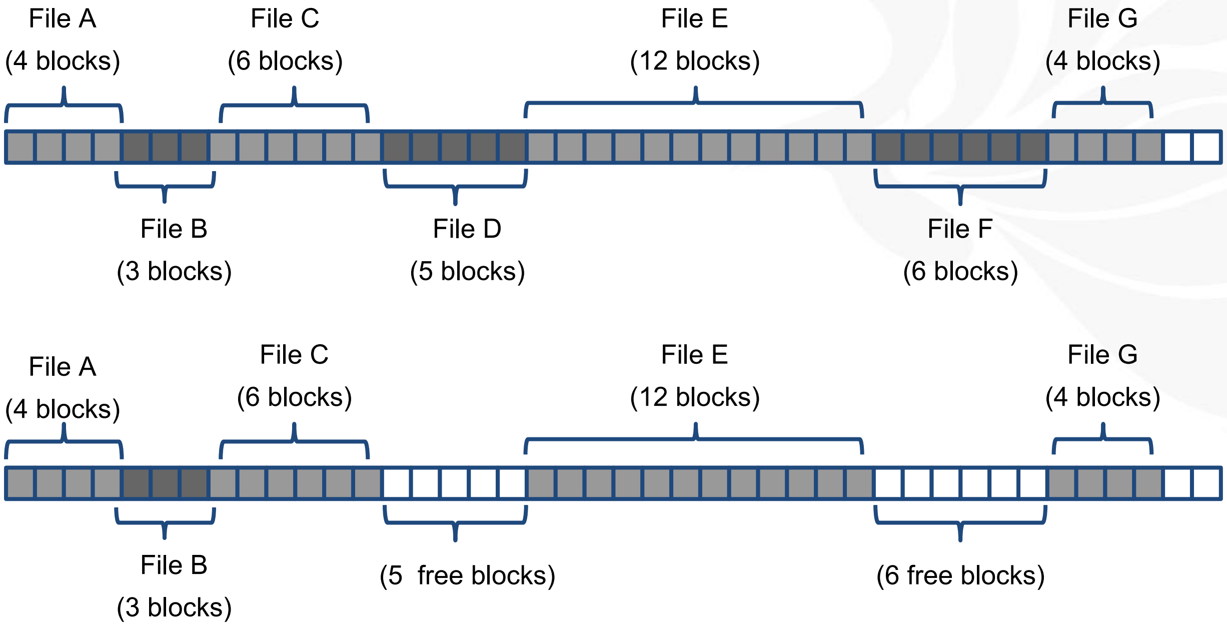
1. Create
2. Delete
3. Open Dir
4. Close Dir
5. Read Dir
6. Rename
7. Link
8. Unlink

File System Implementation



A possible file system layout

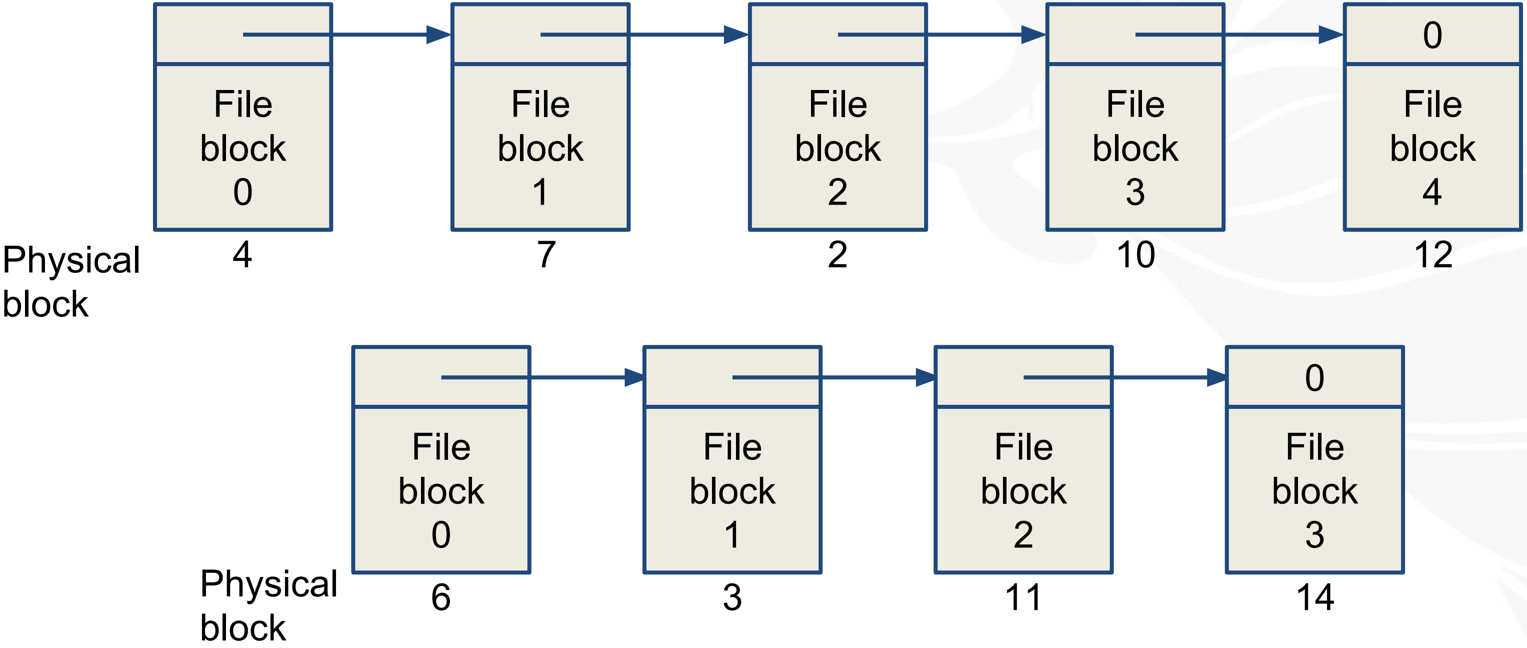
Implementing Files



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files D and E have been removed

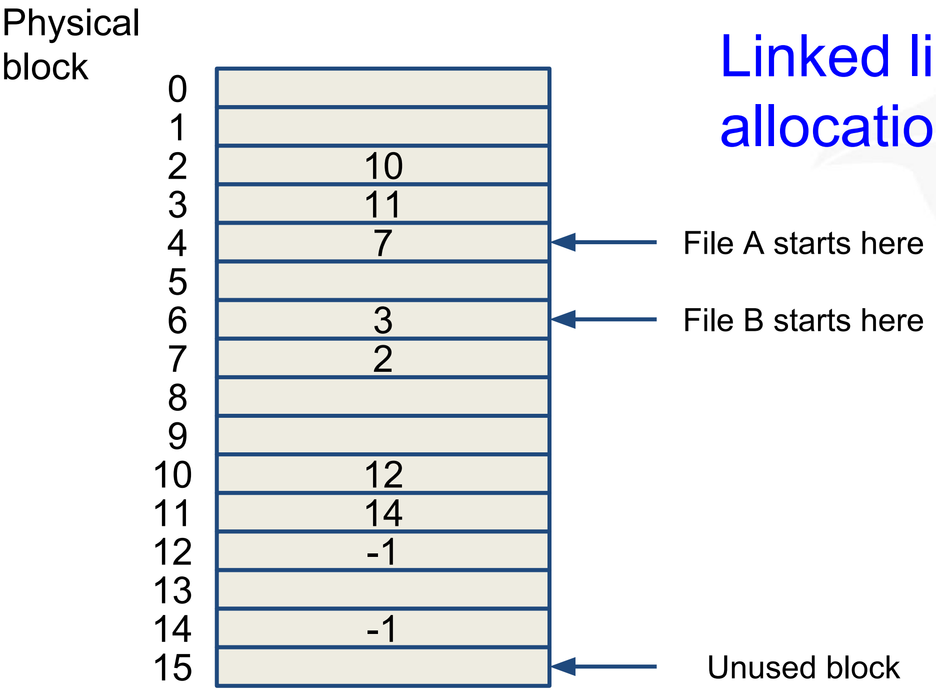
Implementing files



Storing a file as a linked list of disk blocks

Implementing Files

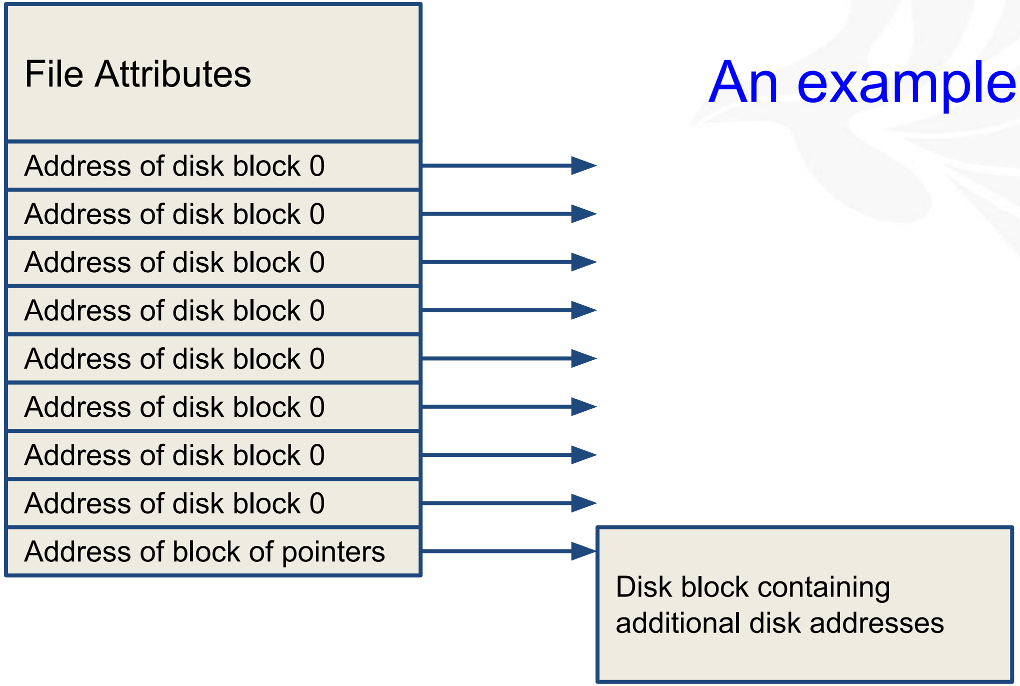
Linked list allocation using a file allocation table in RAM



Physical block		
0		
1		
2	10	
3	11	
4	7	File A starts here
5		
6	3	File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	
13		
14	-1	
15		Unused block

Implementing Files

An example of an iNode



File Attributes

Address of disk block 0

Address of disk block 0

Address of disk block 0

Address of disk block 0

Address of disk block 0

Address of disk block 0

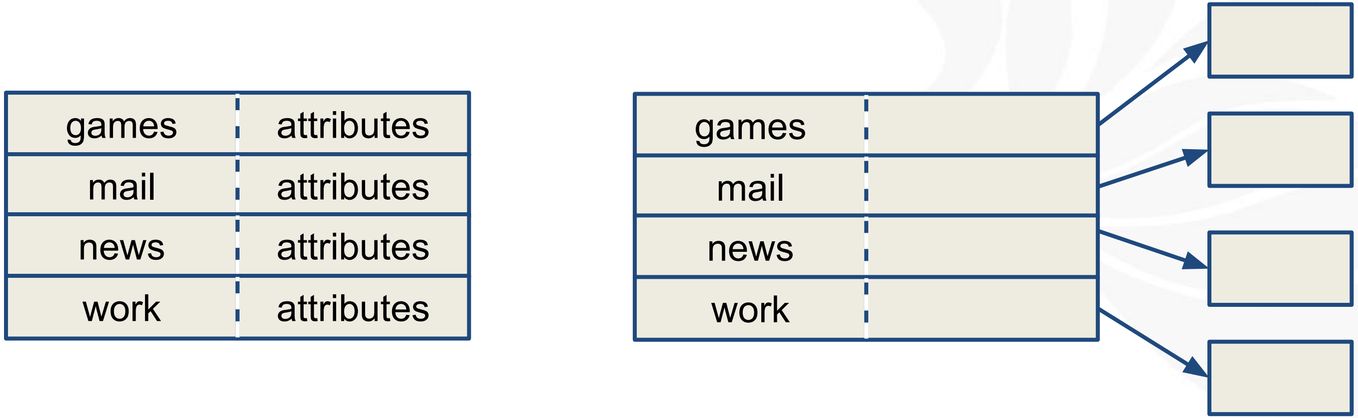
Address of disk block 0

Address of disk block 0

Address of block of pointers

Disk block containing additional disk addresses

Implementing Directories



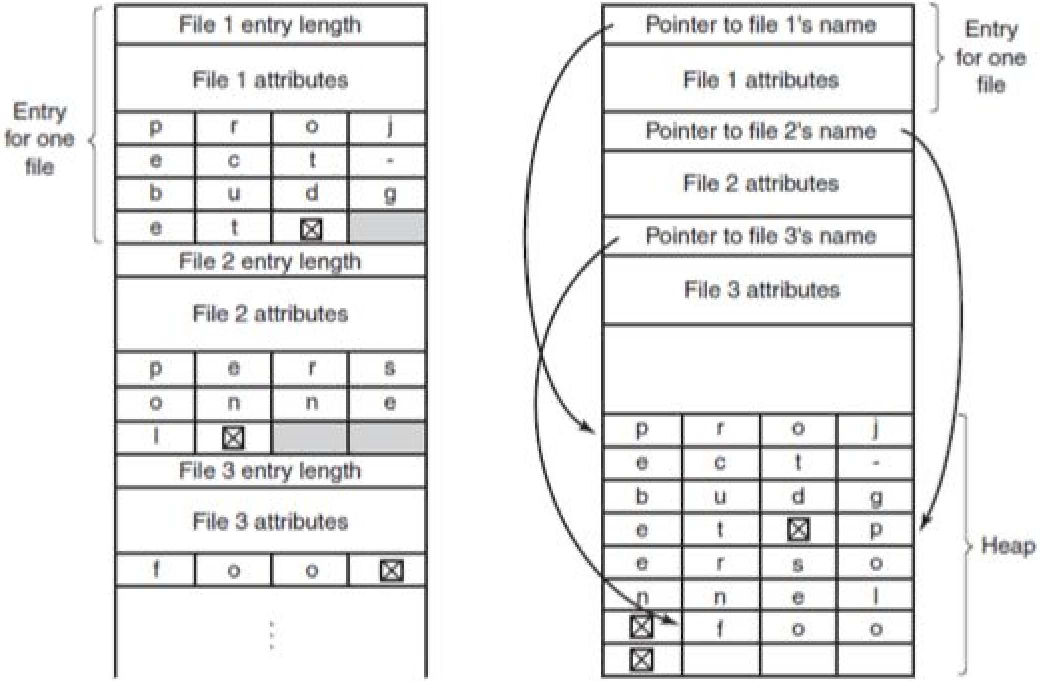
games	attributes
mail	attributes
news	attributes
work	attributes

games	
mail	
news	
work	

(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

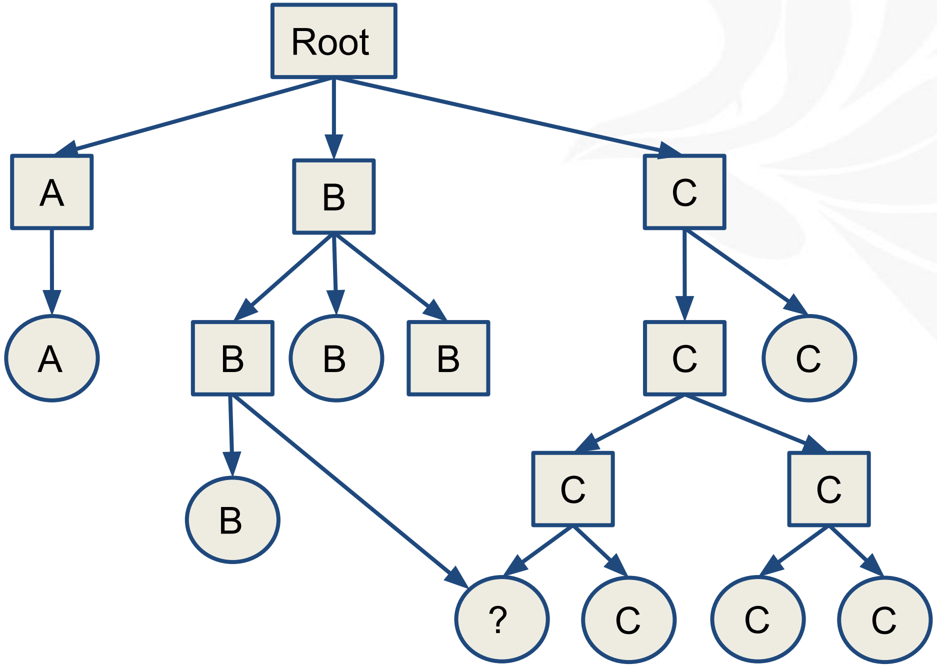


(a) (b)

Two ways of handling long file names in a directory.

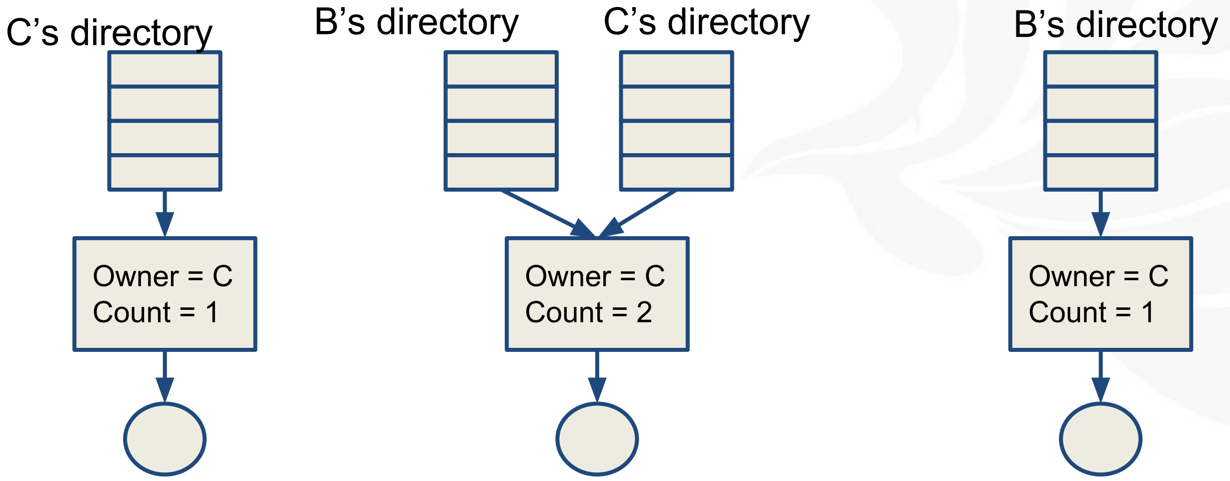
(a) In-line.
(b) In a heap.

Shared Files



File system containing a shared file

Shared Files

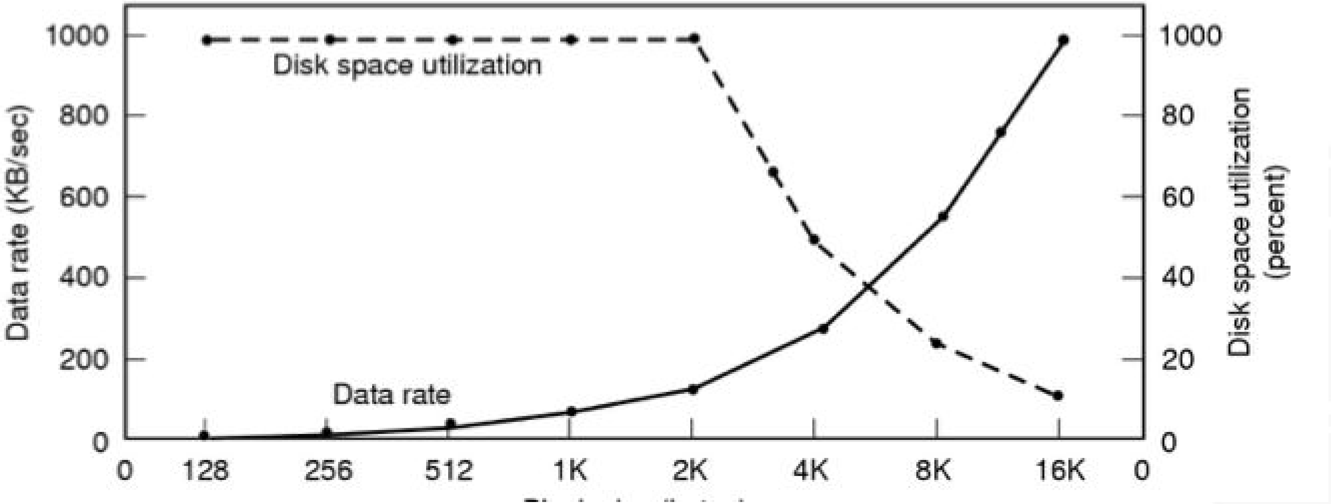


(a) Situation prior to linking.

(b) After the link is created.

(c) After the original owner removes the file.

Disk Space Management

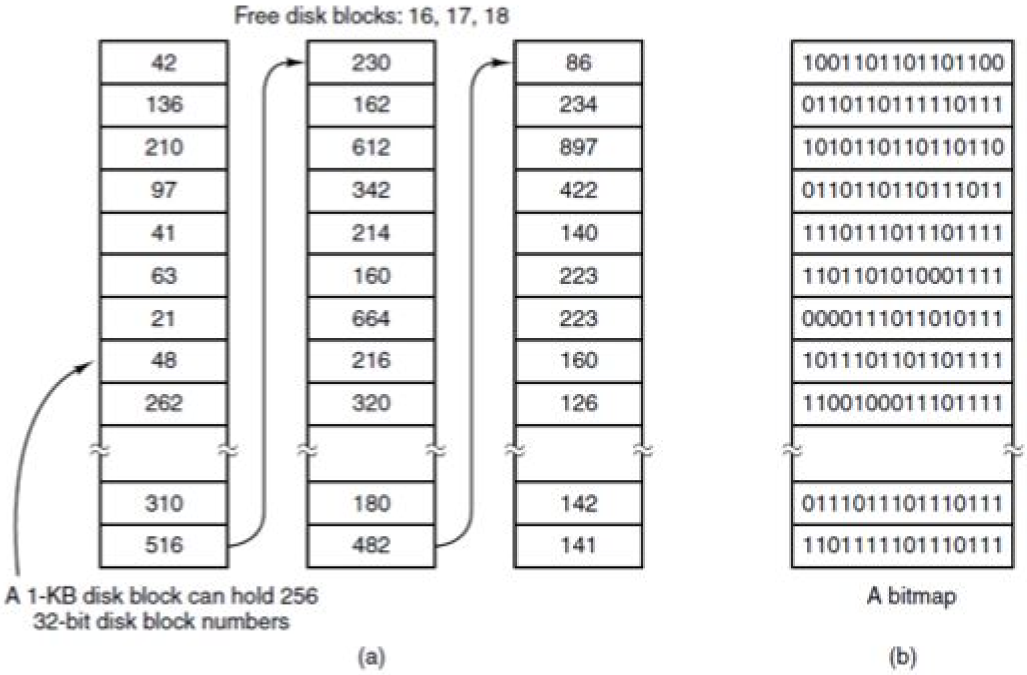


Dark line (left hand scale) gives data rate of a disk

Dotted line (right hand scale) gives disk space efficiency

All files 2KB

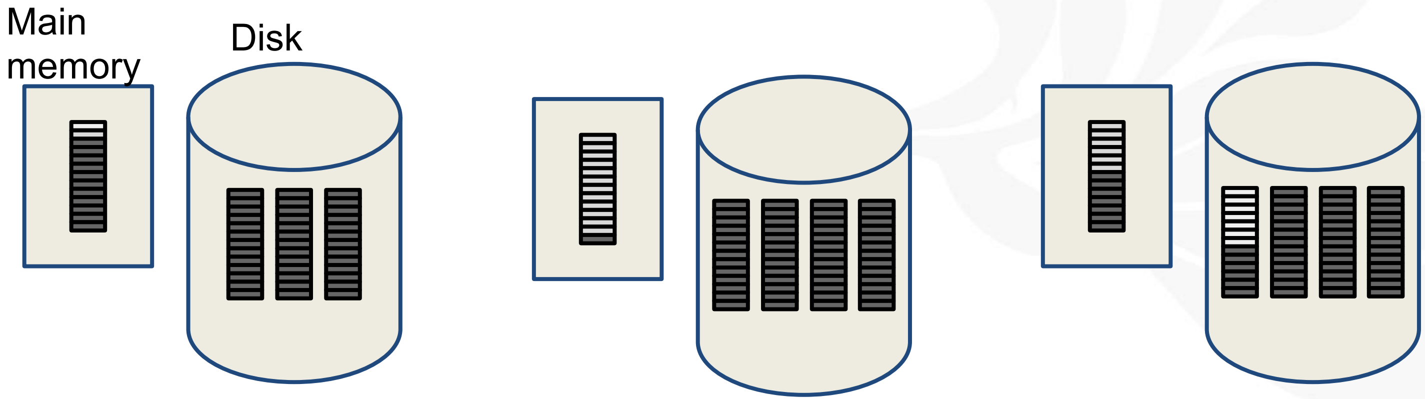
Keeping Track of Free Blocks



(a) Storing the free list on a linked list.

(b) A bitmap.

Disk Space Management



(a) Almost-full block of pointers to free disk blocks in RAM

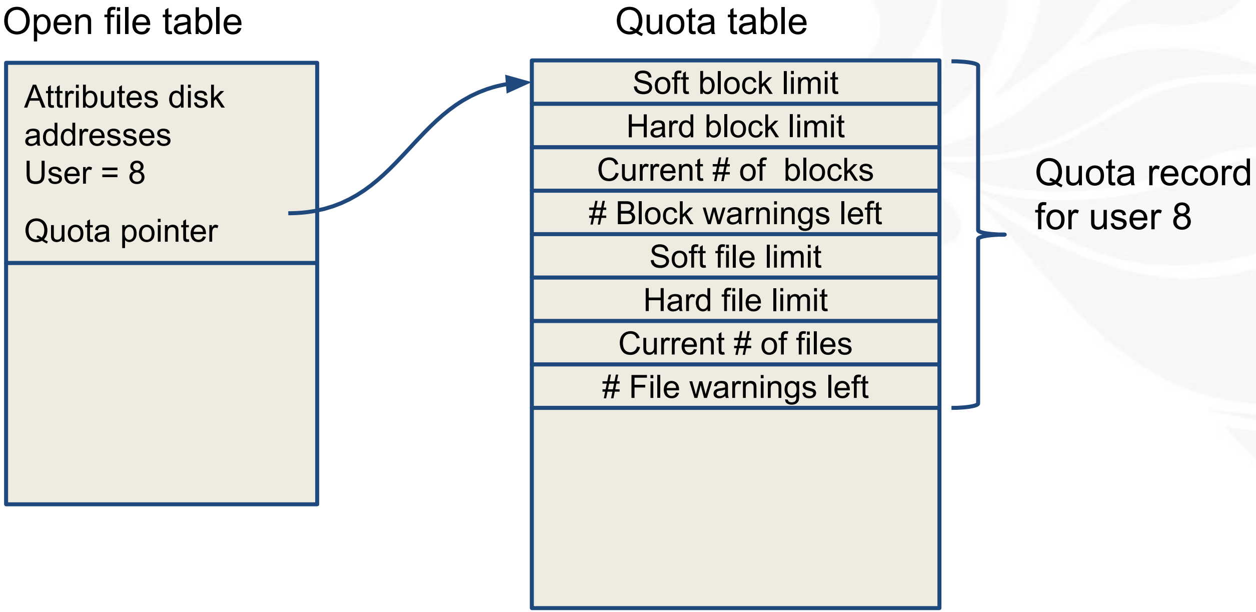
- three blocks of pointers on disk

(b) Result of freeing a 3-block file

(c) Alternative strategy for handling 3 free blocks

- shaded entries are pointers to free disk blocks

Disk space management



Open file table

Attributes disk addresses
User = 8

Quota pointer

Quota table

Soft block limit
Hard block limit
Current # of blocks
# Block warnings left
Soft file limit
Hard file limit
Current # of files
# File warnings left

Quota record for user 8

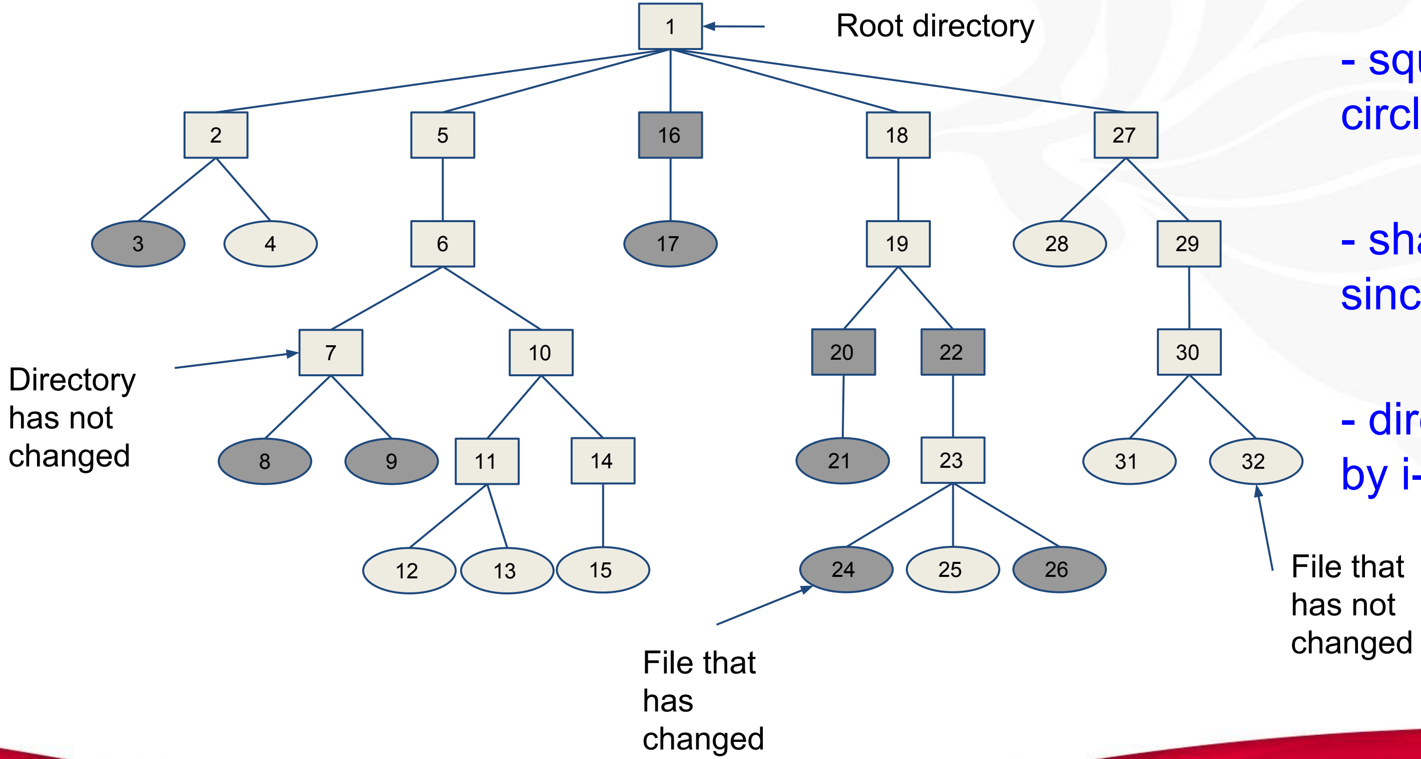
Quotas for keeping track of each user’s disk use

FS Reliability - Backing up

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

1. Recover from disaster.
2. Recover from stupidity.

FS Reliability - Backing up

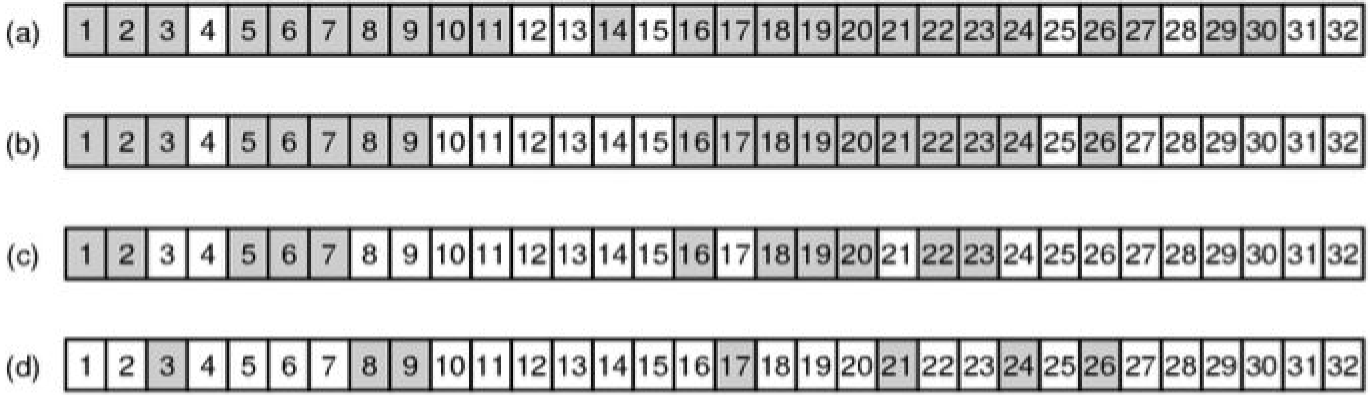


- squares are directories, circles are files

- shaded items, modified since last dump

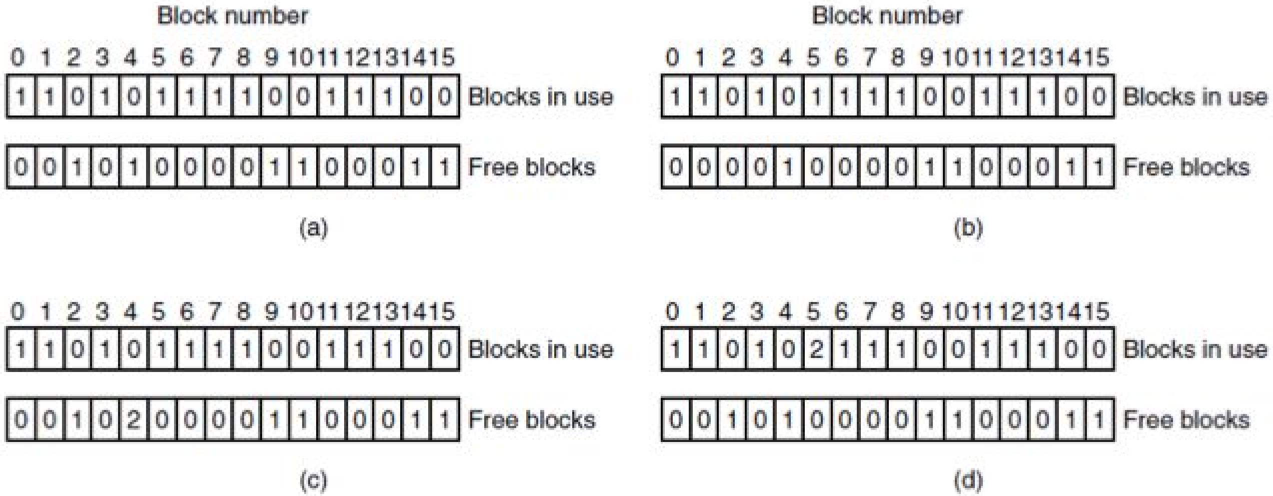
- directories & files labeled by i-node number

FS Reliability - Backing up



Bit maps used by the logical dumping algorithm

File System Consistency



Block number

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(a)

Block number

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(b)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	2	0	0	0	0	1	1	0	0	0	1	1	Free blocks

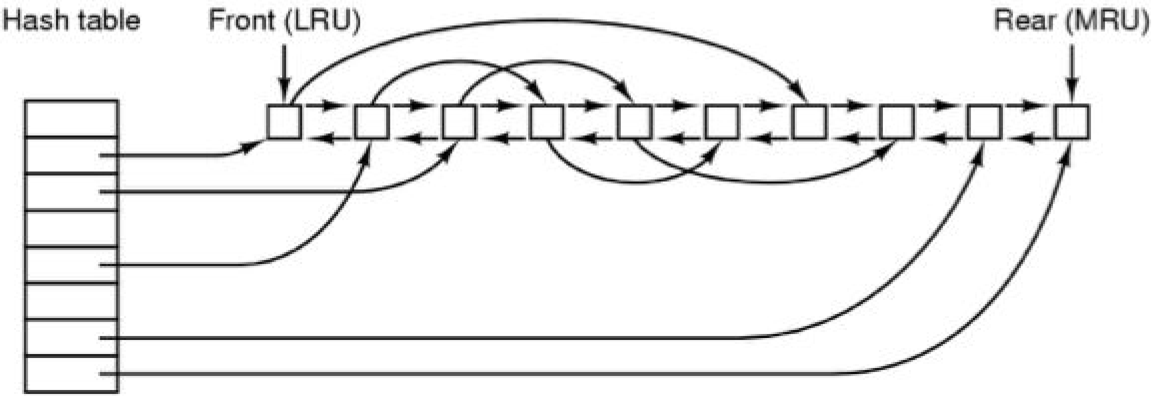
(c)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	1	0	1	0	2	1	1	1	0	0	1	1	1	0	0	Blocks in use
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1	Free blocks

(d)

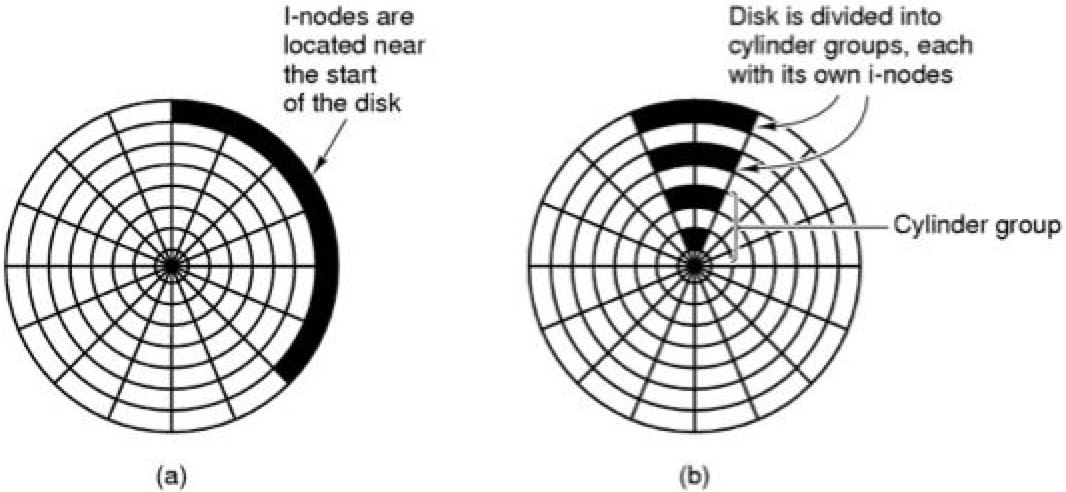
File system states. (a) Consistent. (b) Missing block. (c) Duplicate block in free list. (d) Duplicate data block

File System Performance



The block cache data structures

File System Performance

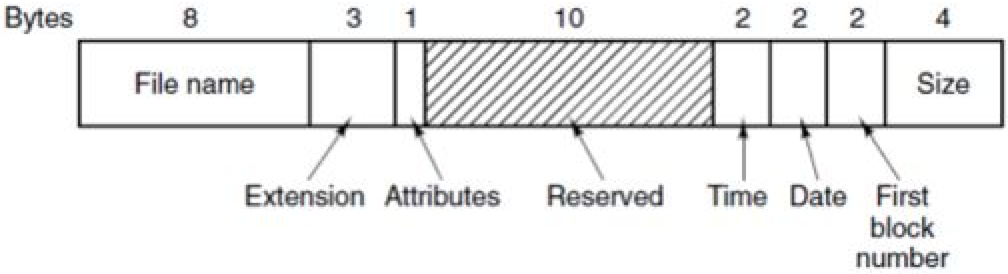

I-nodes are located near the start of the disk
Disk is divided into cylinder groups, each with its own i-nodes
Cylinder group
(a)
(b)

- I-nodes placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes

Journal File Systems

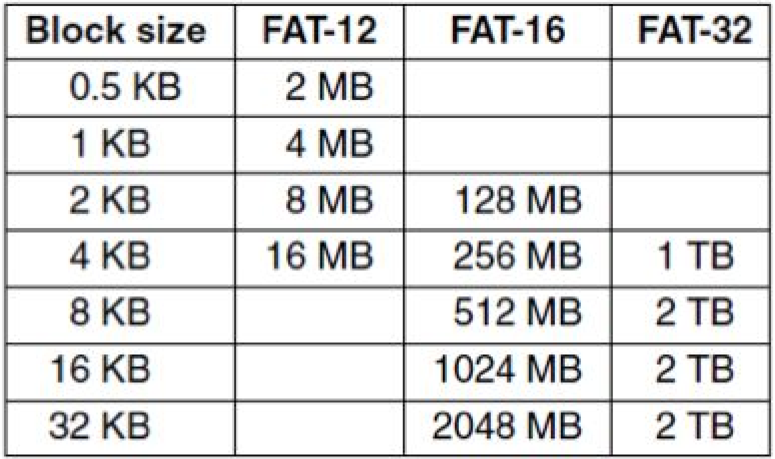
- Ensure robustness in the face of failures
- Consider steps for file removal (ex. Unix)
  - Remove the file from its directory
  - Release the i-node to the pool of the free i-nodes
  - Reclaim deleted file’s disk blocks
- Order of steps is irrelevant when no failures
- What happens when failures occur?
- Solution
  - Journal steps before their execution
  - On failure recover see which operation is still pending

MS-DOS File System



The MS-DOS directory entry.

MS-DOS File System



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

Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

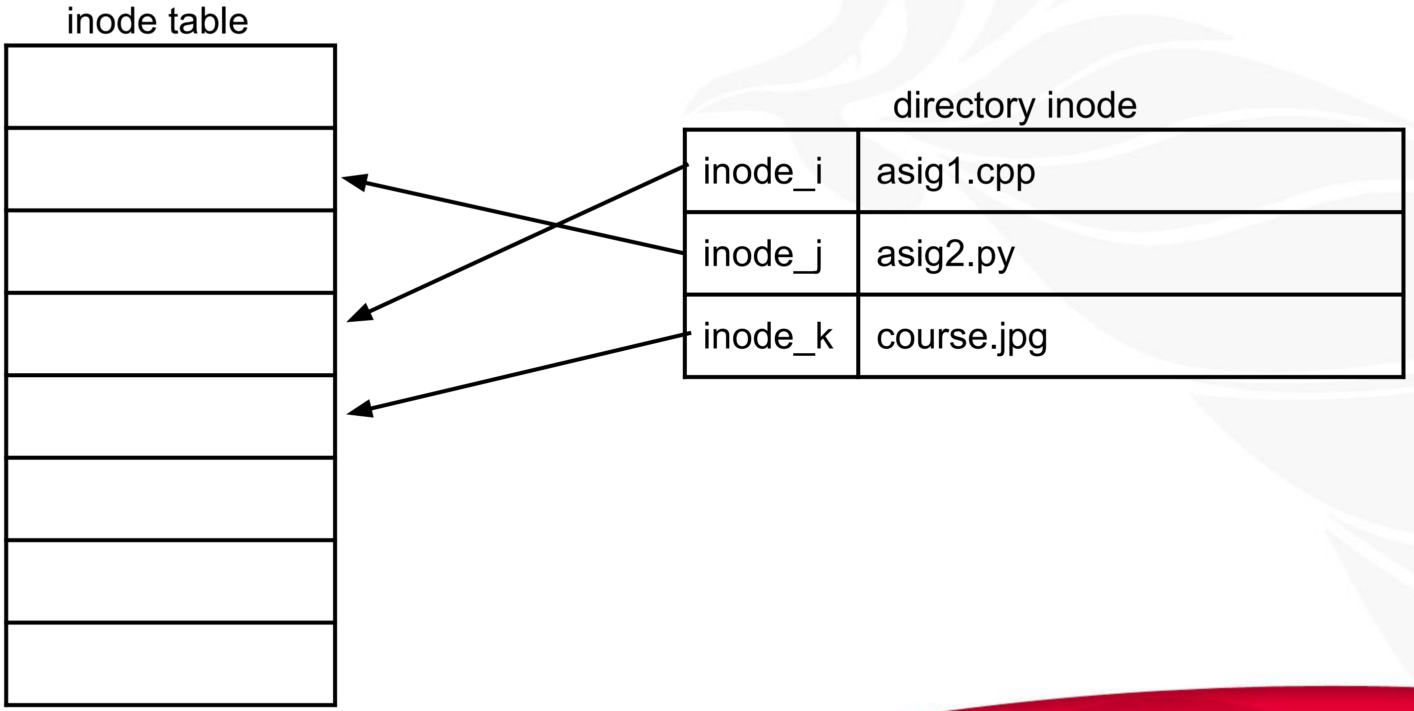
The UNIX V7 File System

directory entry

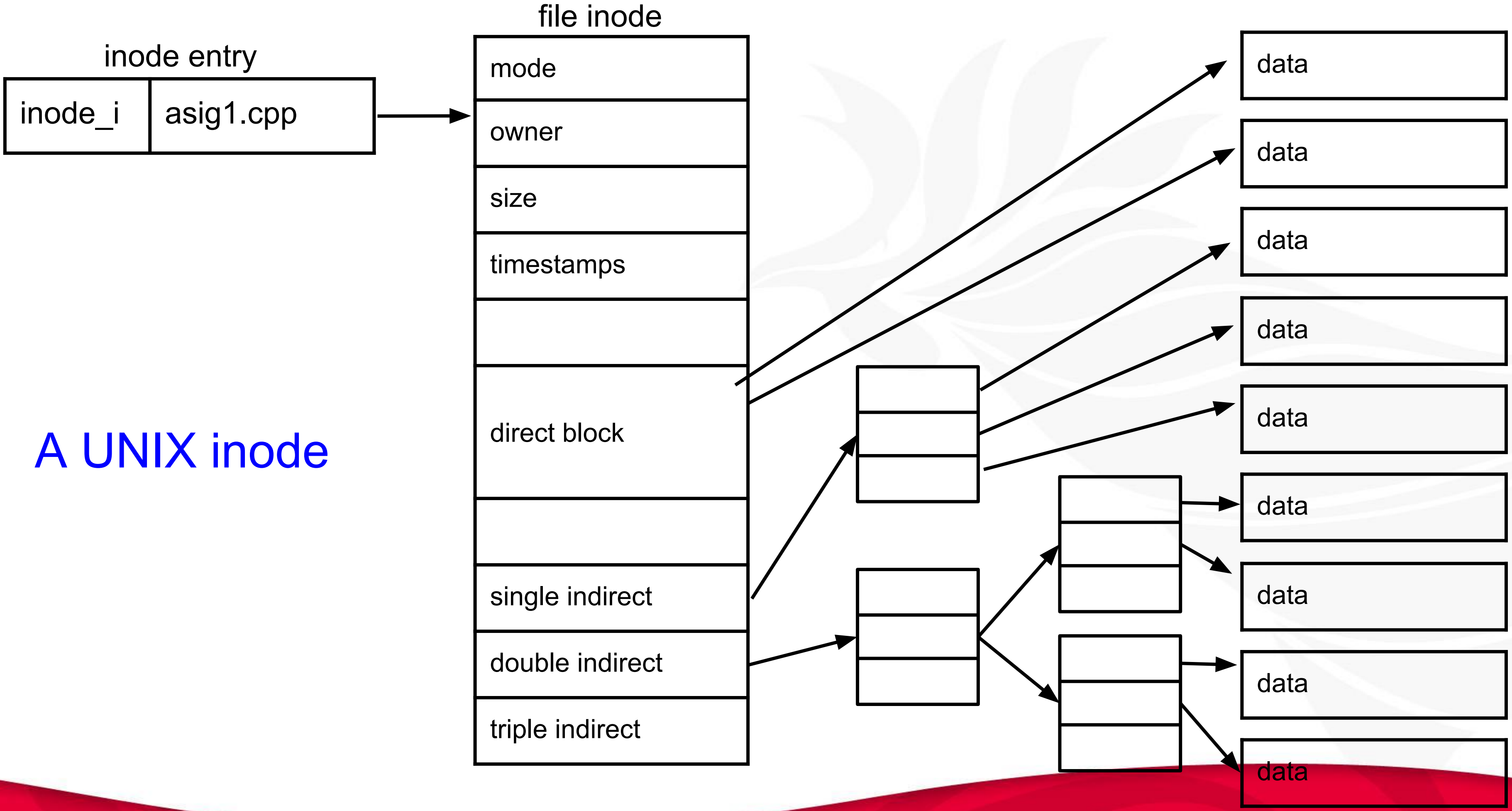
inode_i	asig1.cpp

Unix directory entry

The UNIX V7 File System



A UNIX inode


inode entry
inode_i
asig1.cpp
file inode
mode
owner
size
timestamps
direct block
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data

RAIDs

Redundant Array of Independent Disk a.k.a

Redundant Array of Inexpensive Disk

RAIDs

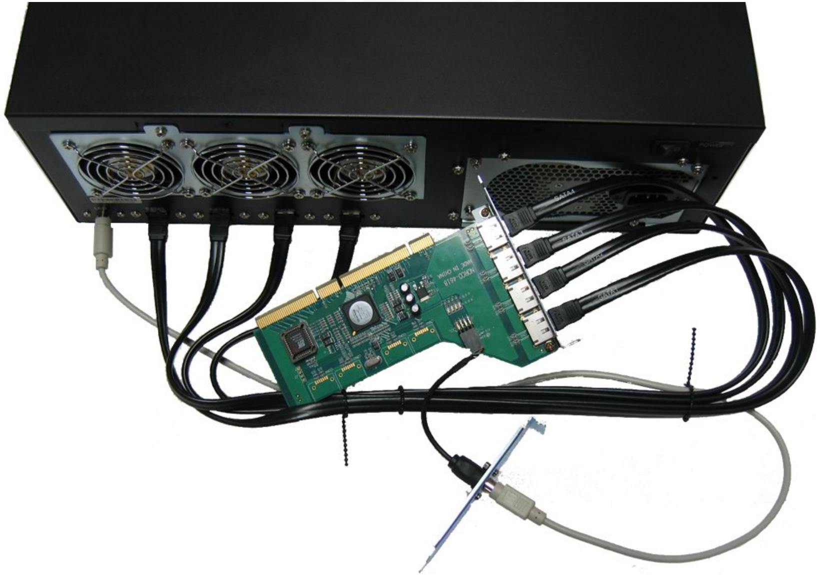
Not to be confused with

RAID Controller

The PCI Card

SATA Cables

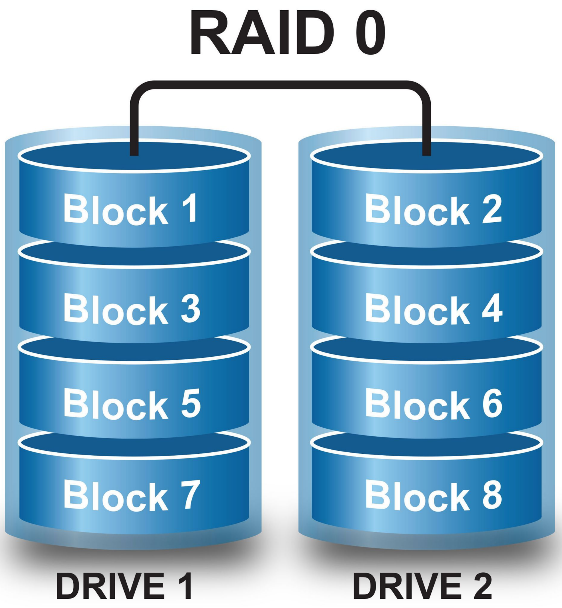
Box with disks



RAID 0

Striping

- large disk
- no redundancy

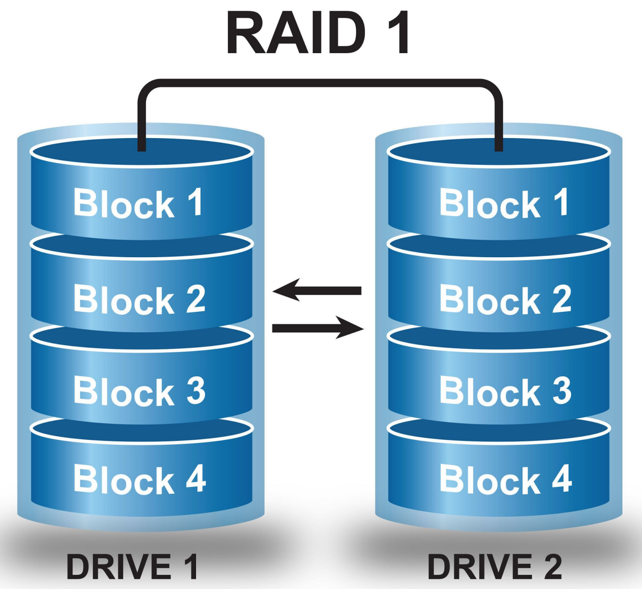


Note in the book the use the term strip instead of block.

RAID 1

Mirror data in both disks.

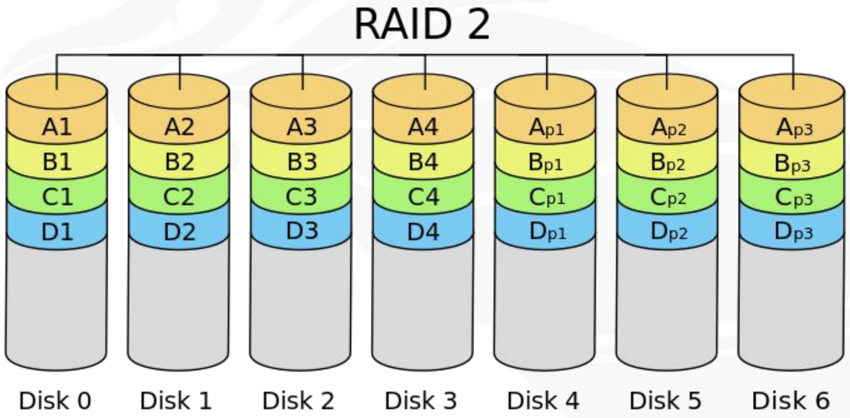
- large disk
- redundancy
- read performance or reliability i performance



Mirrored Data to both Drives

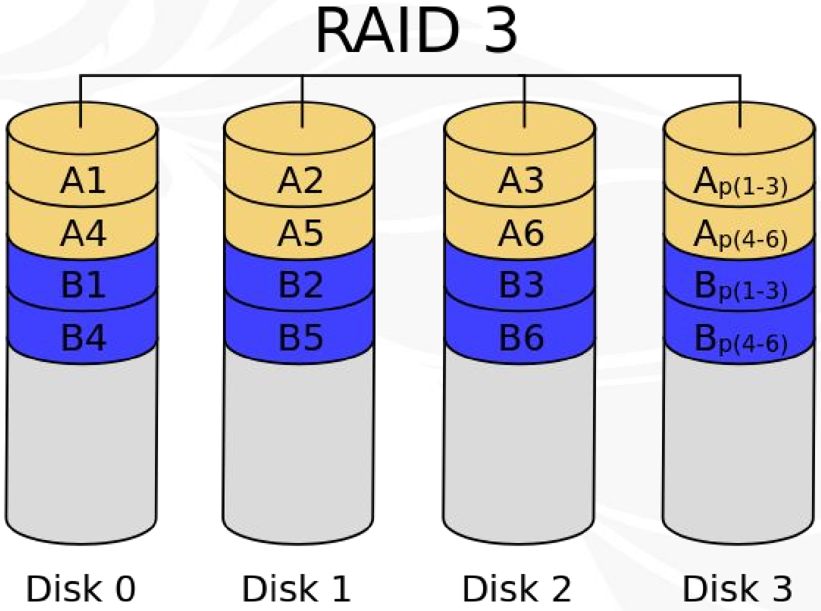
RAID 2

- Stripes data at the bit level
- redundancy - hamming code for error correction
- The disks are synchronized by the controller to spin at the same angular orientation



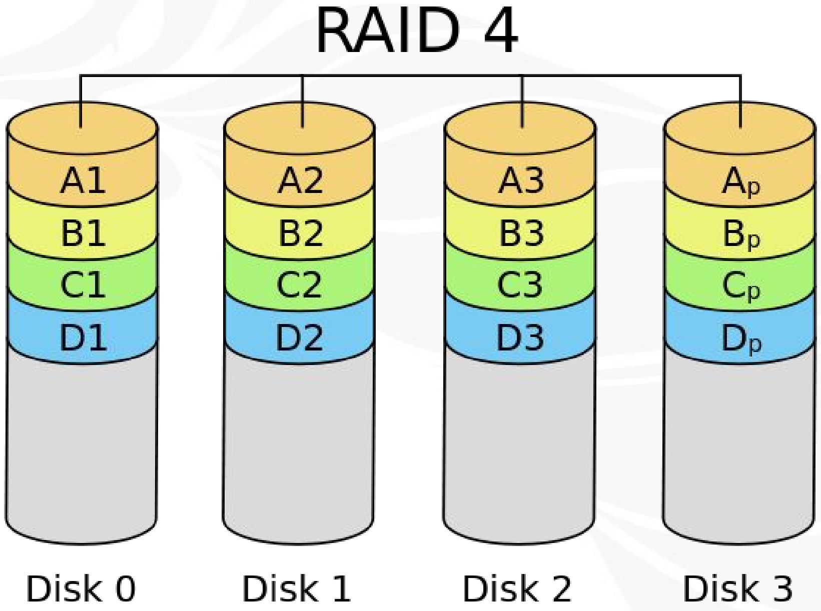
RAID 3

- byte level striping
- redundancy - dedicated parity disk
- cannot service multiple requests simultaneously
- requires synchronized spindles



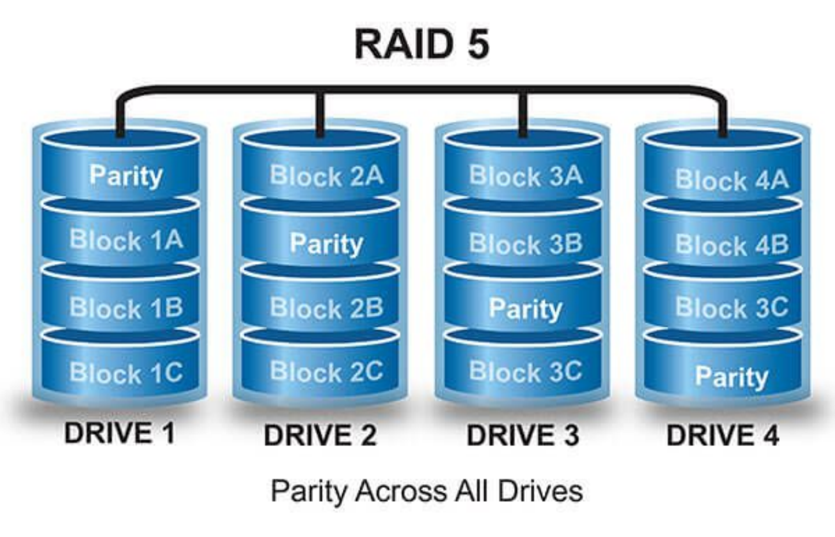
RAID 4

- block level striping
- dedicated parity disk
- provides good random reads performance
- performance of random writes is low due to the need to write all parity data to a single disk.



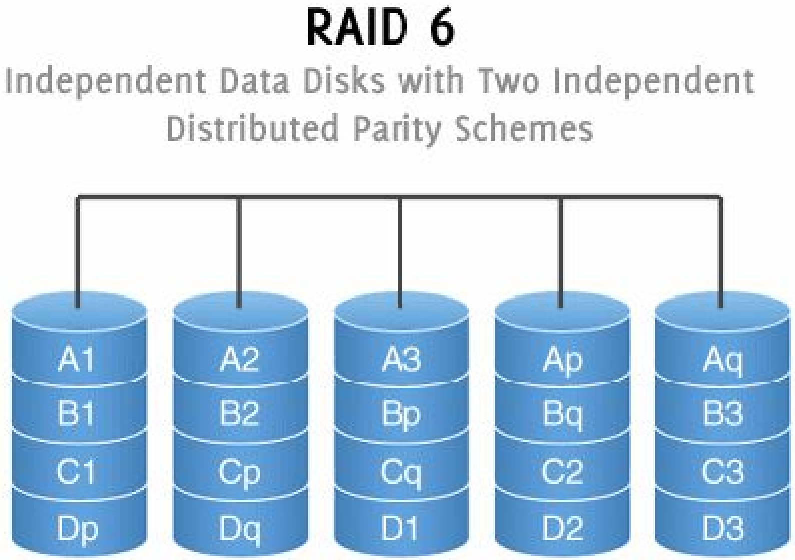
RAID 5

- block level striping
- distributed parity disk
- provides good random reads performance
- better performance of random writes



RAID 6

- block level striping
- two distributed parity disk
- tolerates two concurrent disk failures


RAID 6
Independent Data Disks with Two Independent Distributed Parity Schemes
A1
B1
C1
Dp
A2
B2
Cp
Dq
A3
Bp
Cq
D1
Ap
Bq
C2
D2
Aq
B3
C3
D3