Chapter 4

File Systems

Part 1

Reading

Chapter 4: File Systems

Chapter 10: Case Study 1: Linux (& Unix)

Long-Term Storage of Information

Must store large amounts of data

Information must survive the termination of the process using it

"*persistence*"

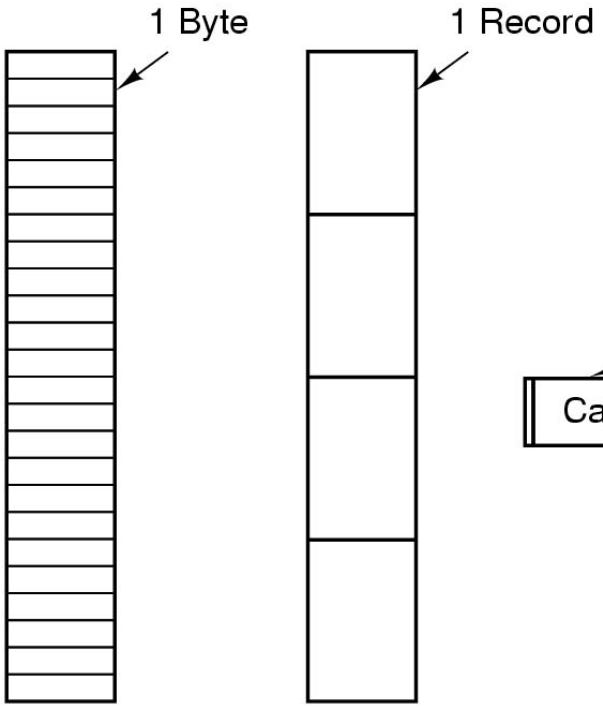
Multiple processes must be able to access the information concurrently

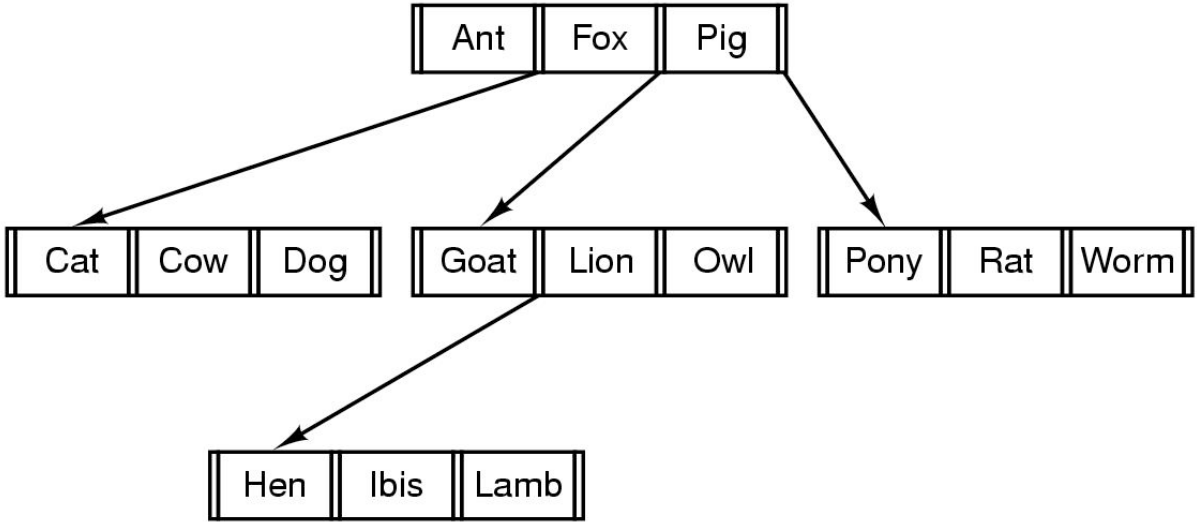
File Naming - Extensions

Typical File Extensions

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

File Structure



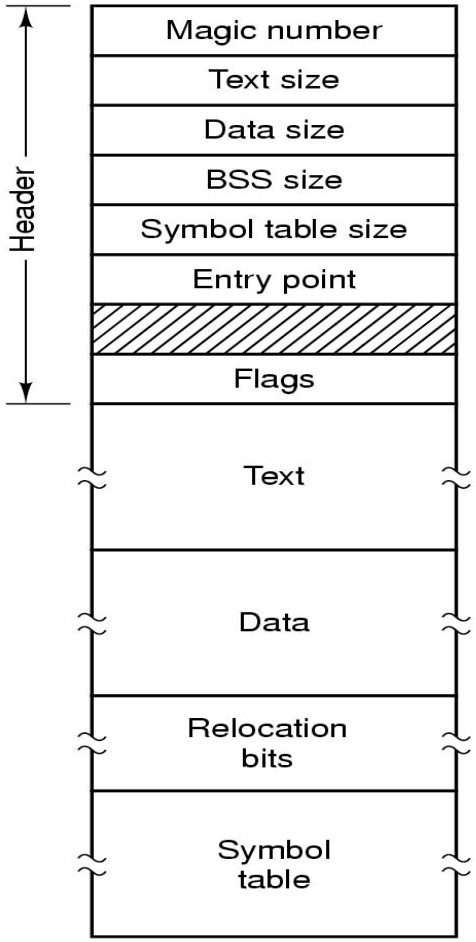


Sequence of bytes

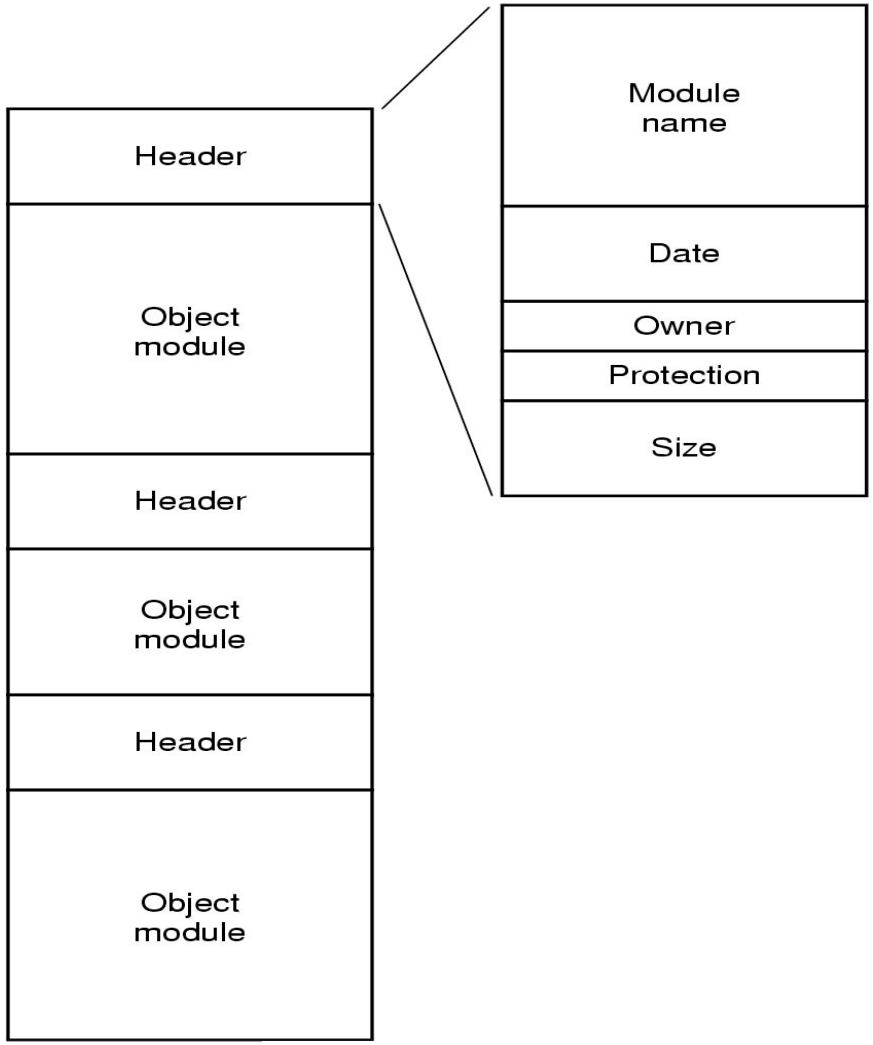
Sequence of records

Tree of records

File Types



An executable file



An archive

File Access

Sequential Access

Read all bytes/records from the beginning

Cannot jump around (but could rewind or back up)

convenient when medium was magnetic tape

Random Access

Can read bytes (or records) in any order

Essential for database systems

Option 1:

move position, then read

Option 2:

perform read, then update current position

File Attributes (some examples)

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 has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Some Important Operations on Files

- Create a file
- Delete a file
- Open
- Close
- Read
- Write
- Append
- Seek (move to new position)
- Get attributes
- Set/modify attributes
- Rename file

A "C" Program to Copy a File

```
/* 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 */
```

(continued)

A "C" Program to Copy a File

```
/* 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 */
        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 */
}
```

Memory-Mapped Files

Before:

Use syscalls (e.g., open, read, write, ...)
to move data from disk to memory

Memory-Mapped Files

Before:

Use syscalls (e.g., open, read, write, ...)
to move data from disk to memory

Notice:

The kernel does this all the time
Pages moved to/from PAGEFILE

Memory-Mapped Files

Before:

Use syscalls (e.g., open, read, write, ...)
to move data from disk to memory

Notice:

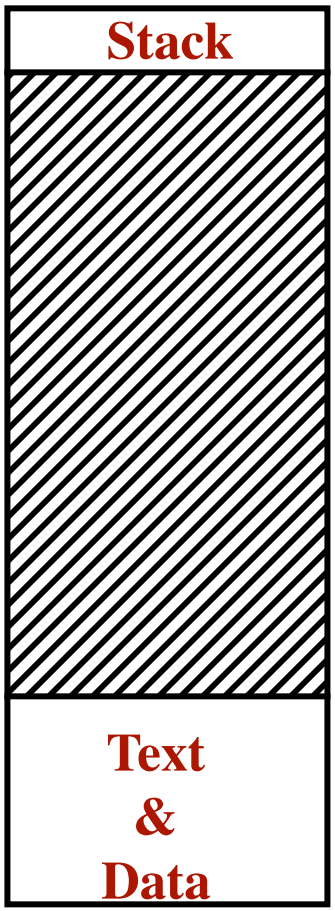
The kernel does this all the time
Pages moved to/from PAGEFILE

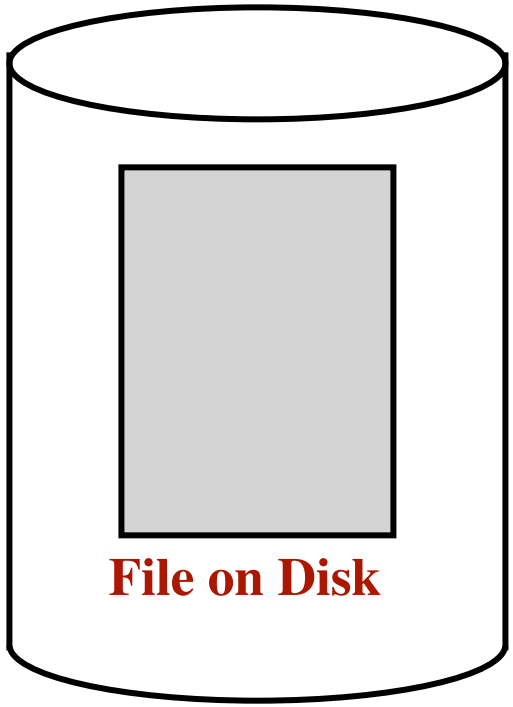
Idea:

"Map" files into the virtual address space
To read from file:
Just access that region of virtual address space
Kernel will fetch pages from disk when needed
To write file:
Modify bytes in memory
Open & Close syscalls → Map & Unmap syscalls

Memory-Mapped Files

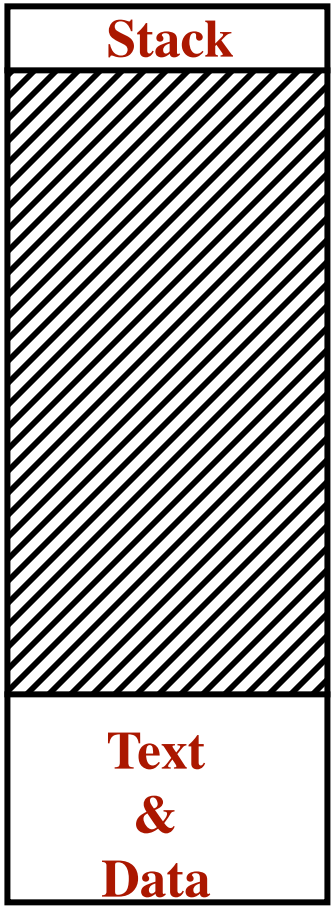
Virtual Address Space



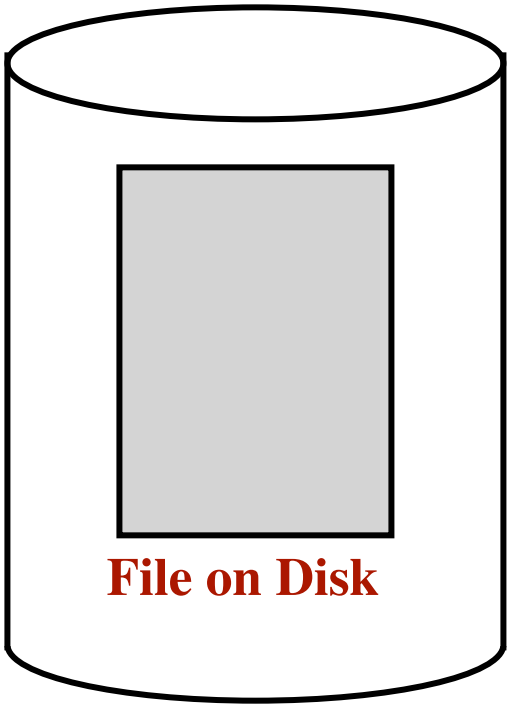


Memory-Mapped Files

Virtual Address Space

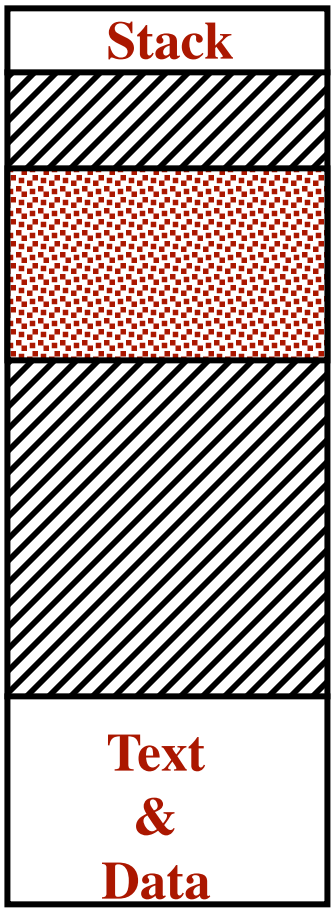


"Map" syscall is made

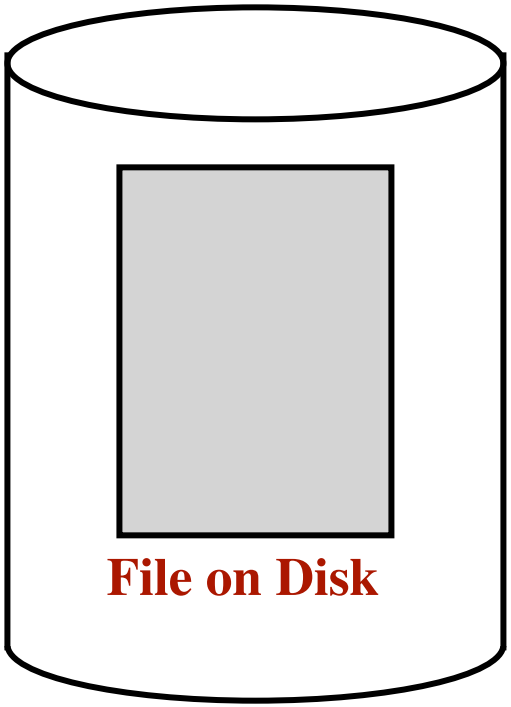


Memory-Mapped Files

Virtual Address Space



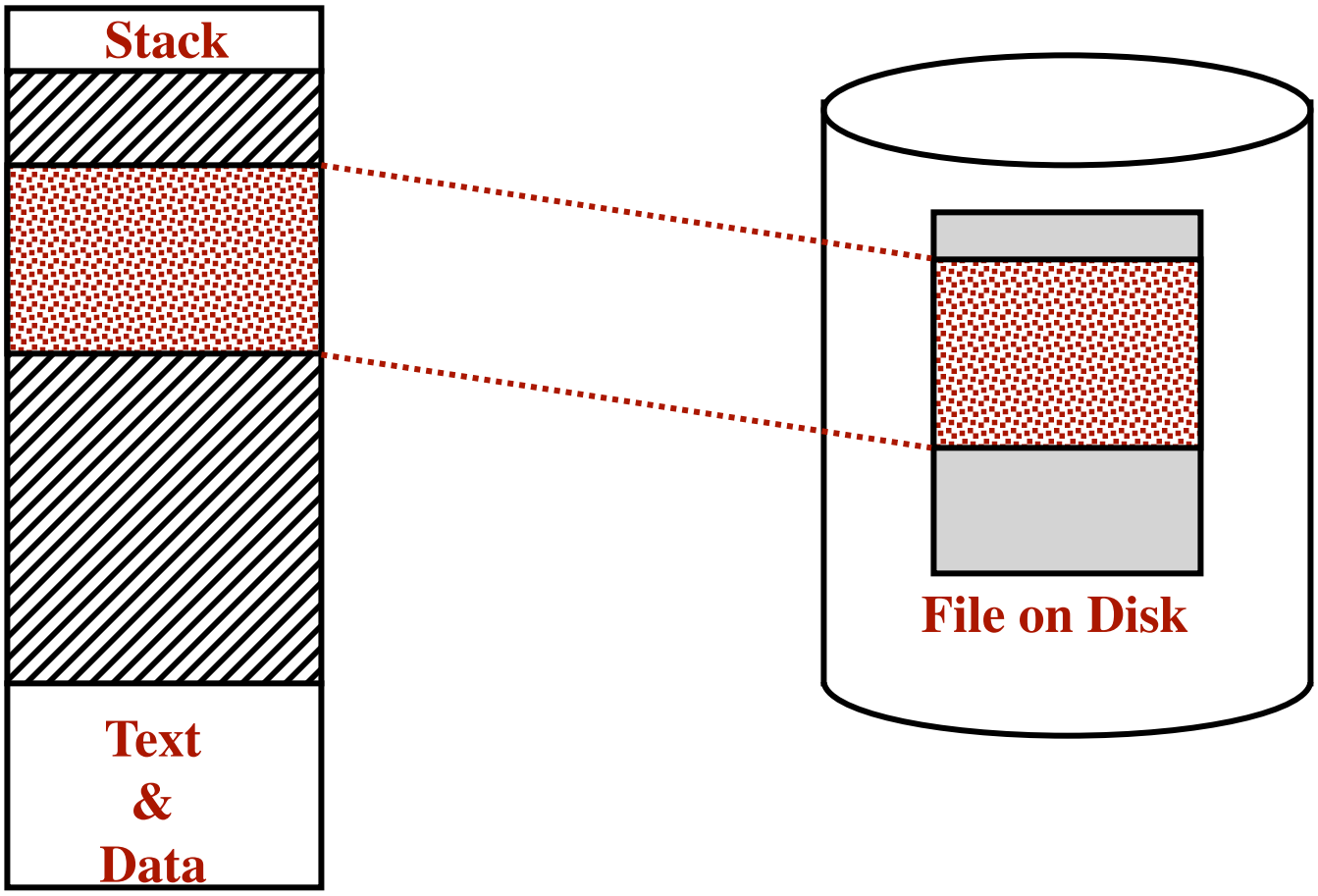
"Map" syscall is made



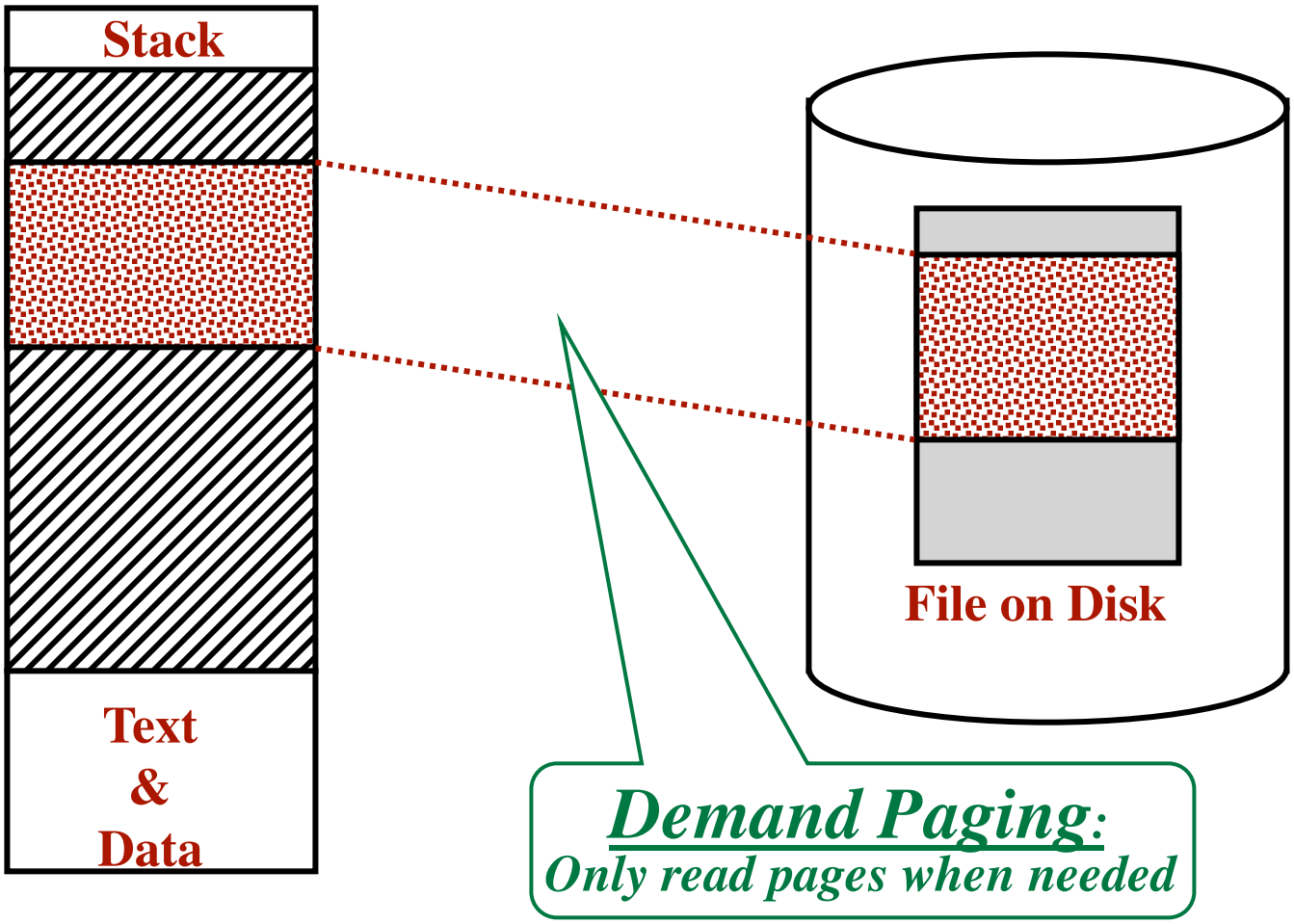
Memory-Mapped Files

Virtual Address Space

"Map" syscall is made



Memory-Mapped Files



Memory-Mapped Files

Unix / Linux:

```
#include <sys/mman.h>

void* mmap (
        void * start,       Address of memory region
        size_t length,      Length of memory region
        int    prot,        Read / write / execute flag
        int    flags,
        int    fd,          File descriptor
        off_t  offset );    Offset in the file

int munmap (
        void * start,       Address of memory region
        size_t length );    Length of memory region
```

Directories

"Folder"

Early OSs

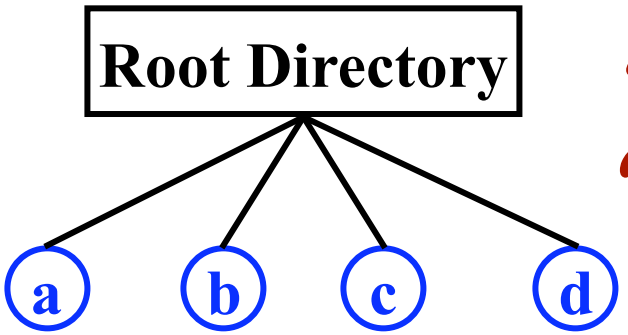
Single-Level Directory Systems

Directories

"Folder"

Early OSs

Single-Level Directory Systems



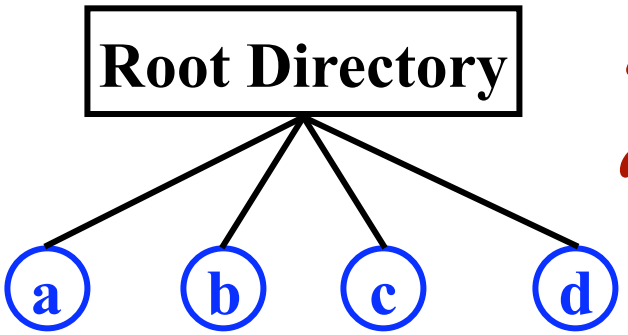
"Files" and "directories" are different, unrelated concepts.

Directories

"Folder"

Early OSs

Single-Level Directory Systems



"Files" and "directories" are different, unrelated concepts.

Problem:

Sharing amongst users

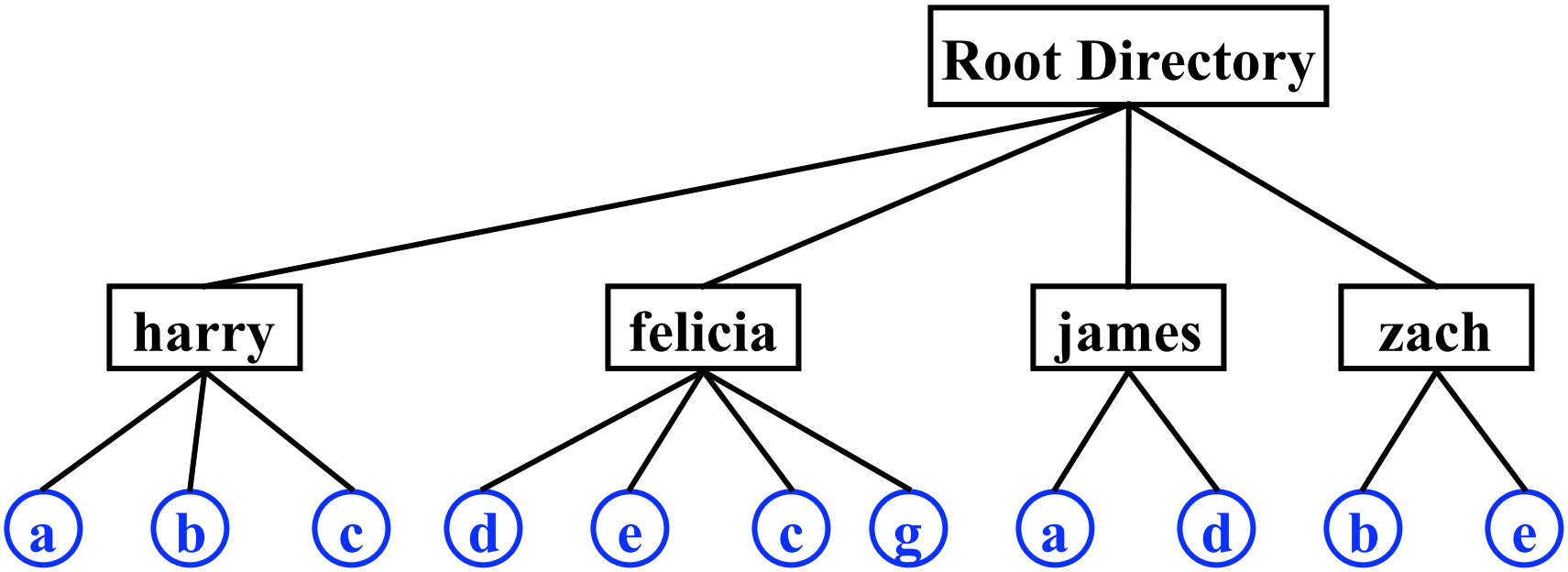
Appropriate for small, embedded systems

Two-Level Directory Systems

Each user has a directory.

Files accessed with user/filename.

/james/d



Two-Level Directory Systems

Each user has a directory.

Files accessed with user/filename.

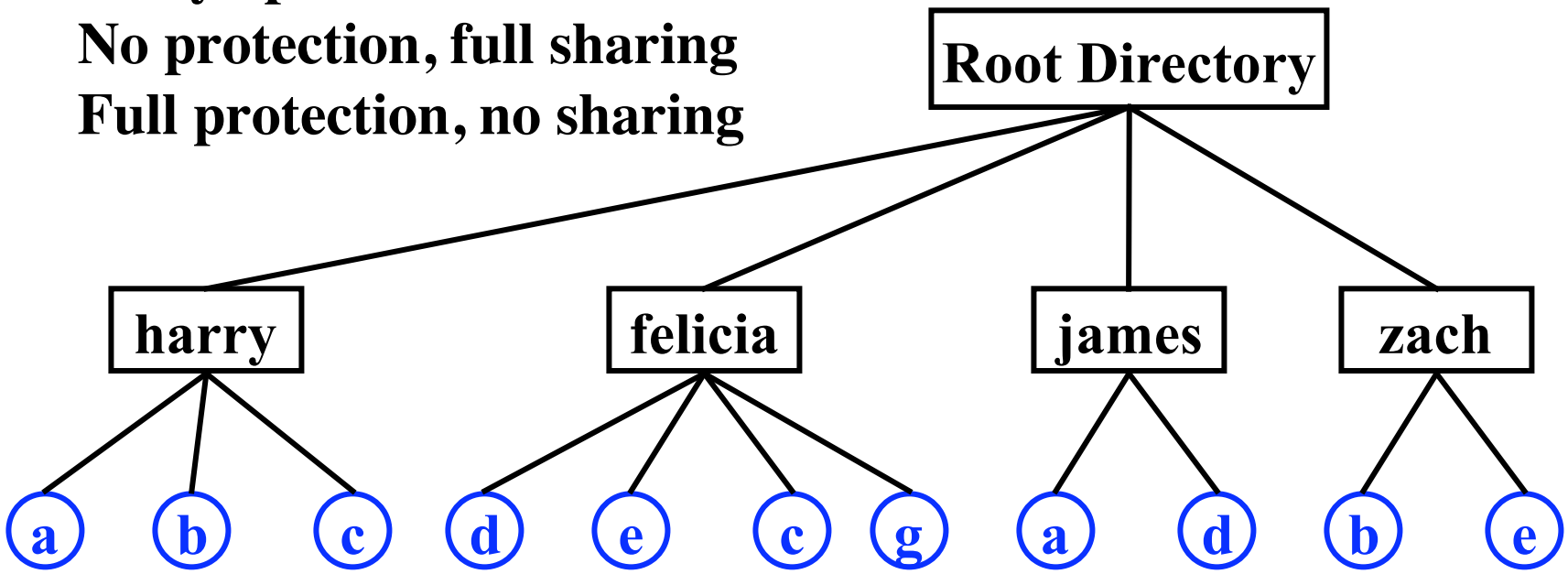
/james/d

Directories and files are seen as “different” creatures.

Security options:

No protection, full sharing

Full protection, no sharing

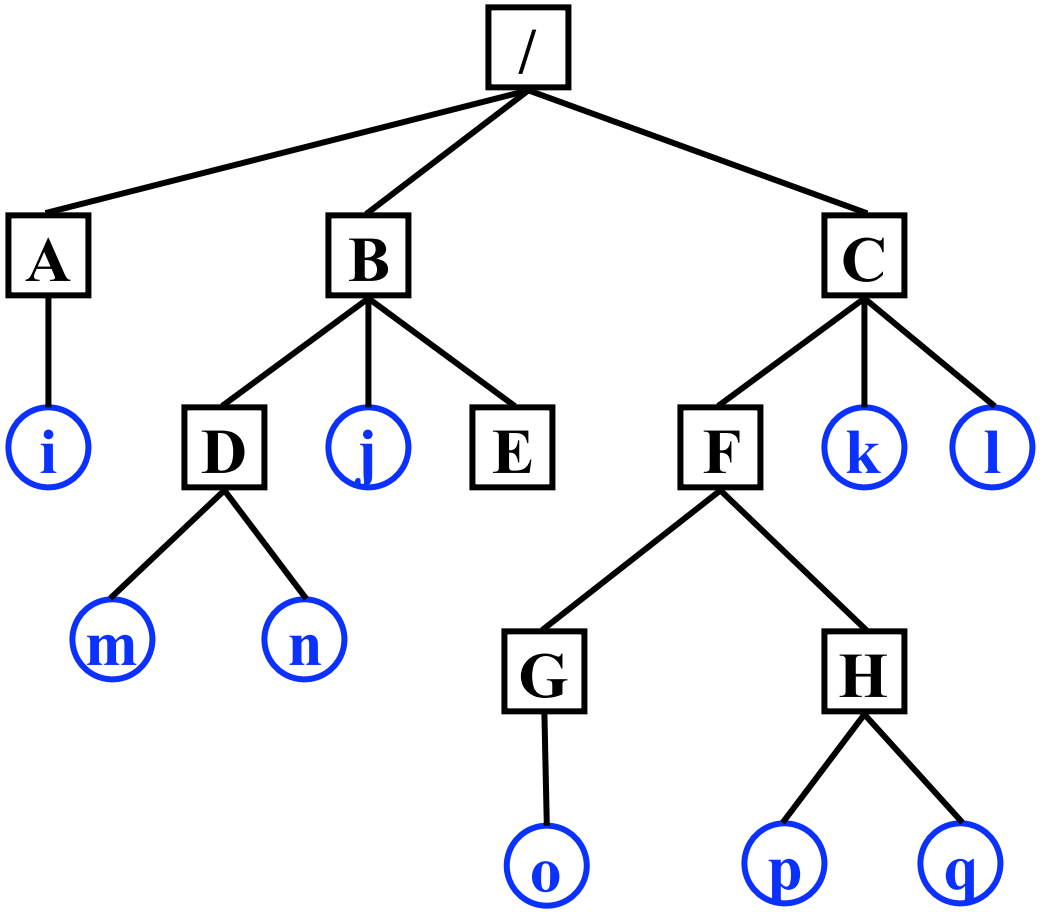


Hierarchical Directory Systems

A tree of directories

Interior nodes: Directories

Leaves: Files

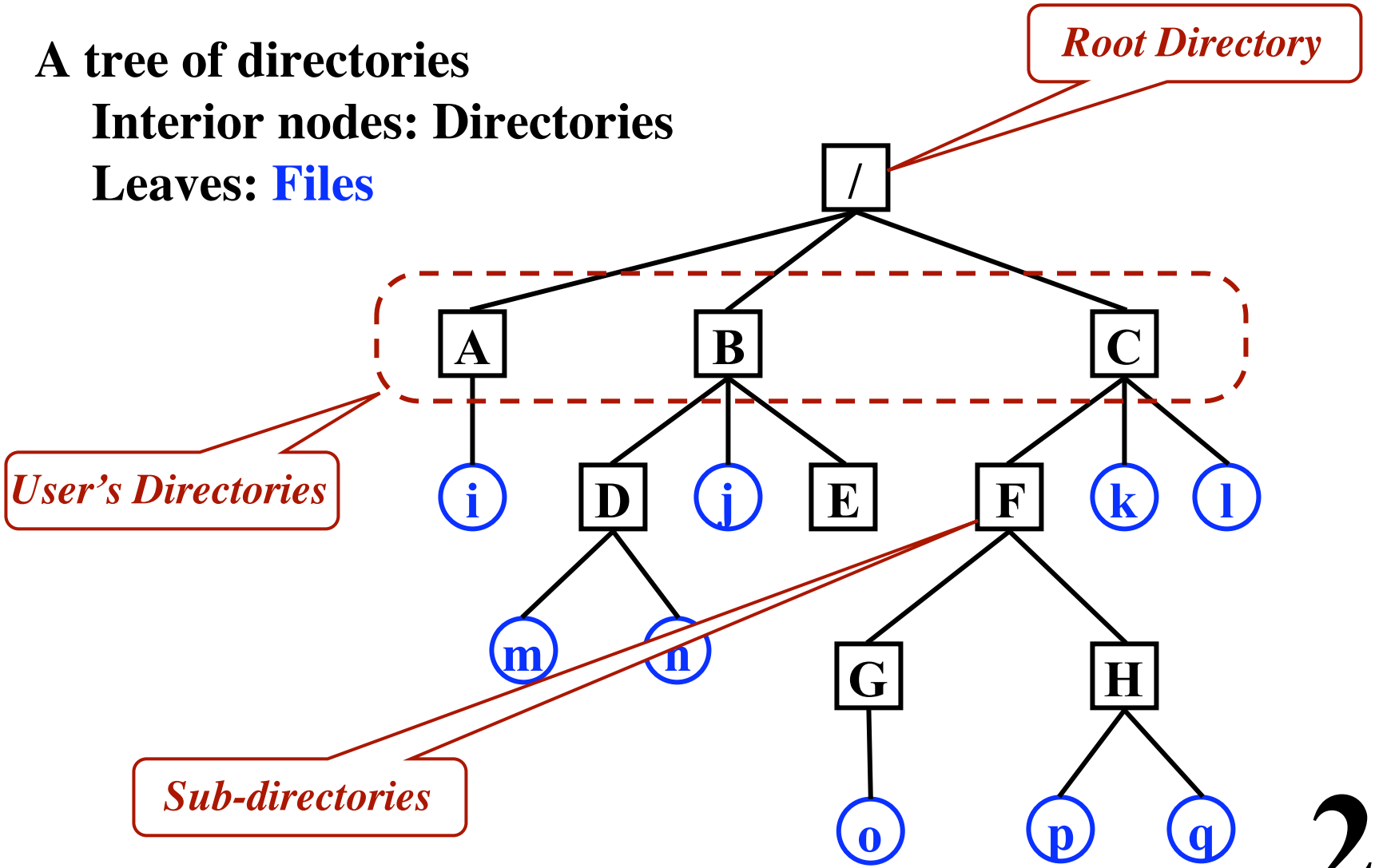


Hierarchical Directory Systems

A tree of directories

Interior nodes: Directories

Leaves: Files



Path Names

MULTICS

>usr>harry>mailbox

Unix

/usr/harry/mailbox

Windows

\usr\harry\mailbox

Path Names

MULTICS

>usr>harry>mailbox

Unix

/usr/harry/mailbox

Windows

\usr\harry\mailbox

Absolute Path Name

/usr/harry/mailbox

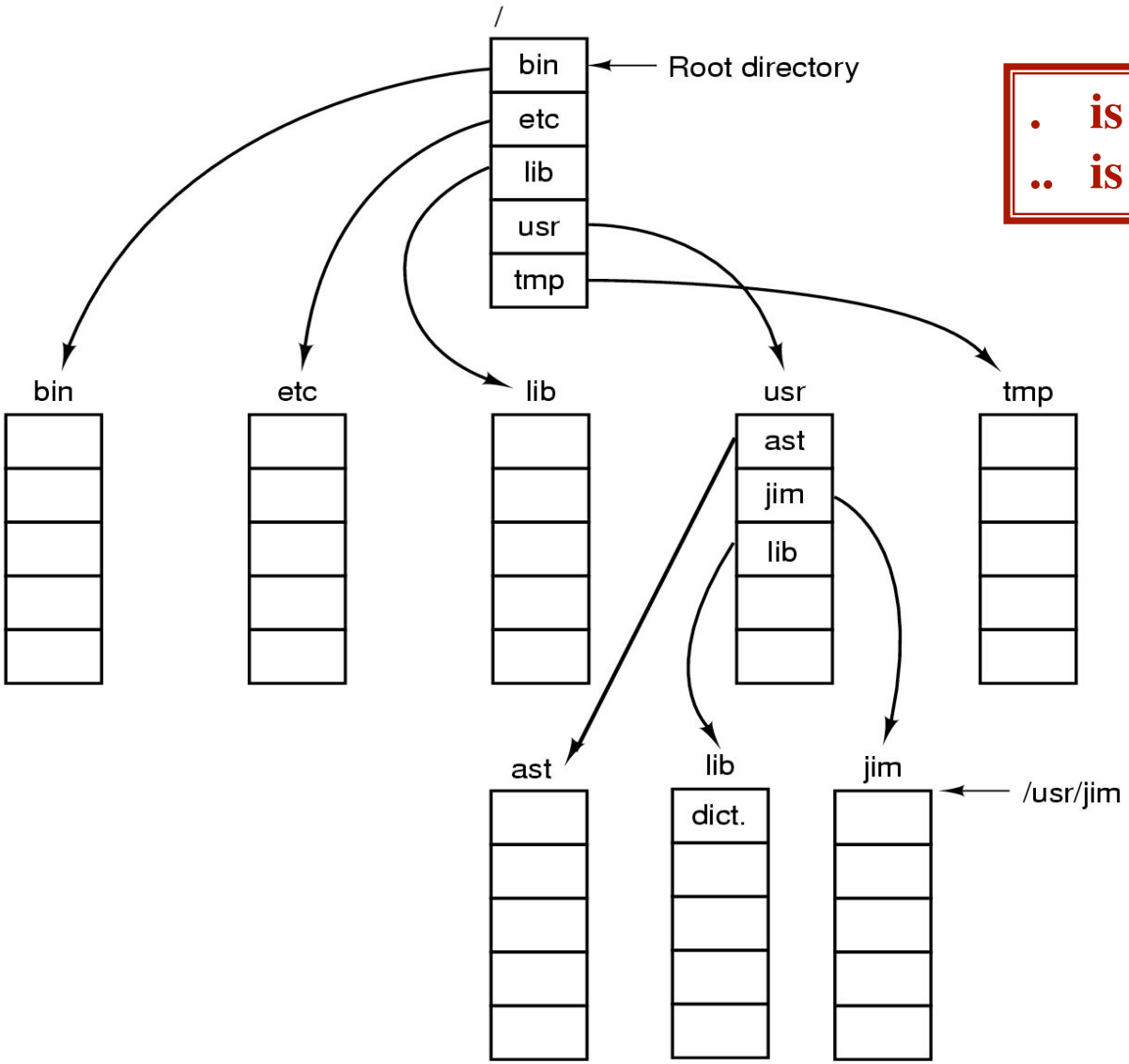
Relative Path Name

"working directory" (or "current directory")

mailbox

Each process has its own working directory

A Unix Directory Tree



. is the "current directory"
.. is the parent

Typical Directory Operations

Create a new directory
Delete a directory
Open a directory for reading
Close
Readdir - Return next entry in the directory
(Returns the entry in a standard format, regardless of the internal representation)
Rename a directory
Link - Add this directory as a sub directory in another directory. (Make a "hard link".)
Unlink - Remove a "hard link"

Unix Directory-Related Syscalls

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

s = error code
dir = directory stream
dirent = directory entry

File System Implementation

Sector 0: "Master Boot Record" (MBR)

Contains the partition map

Rest of disk divided into "partitions"

Partition: sequence of consecutive sectors.

Each partition can hold its own file system.

- **Unix file system**
- **Window file system**
- **Apple file system**

Every partition starts with a "boot block"

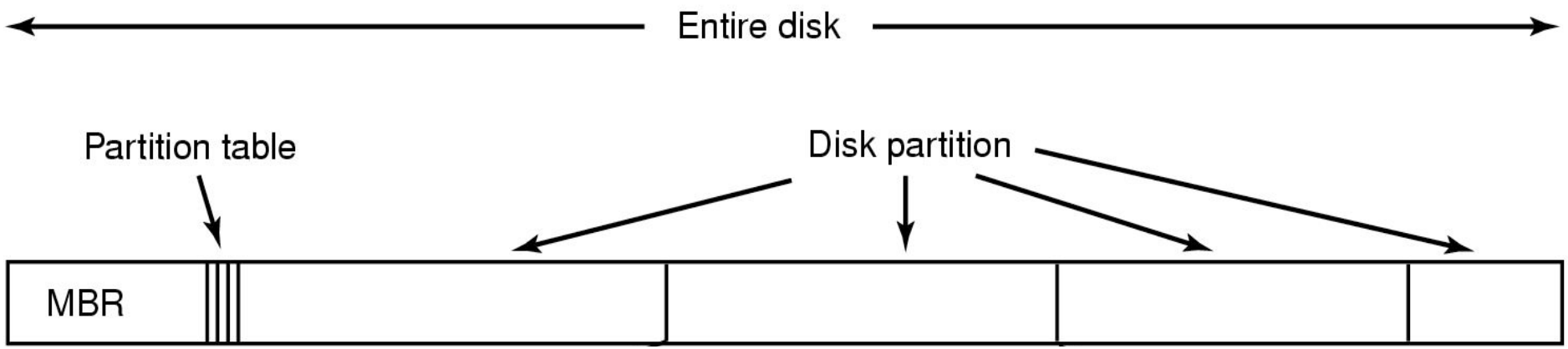
Contains a small program

This "boot program" reads in an OS from the file system in that partition

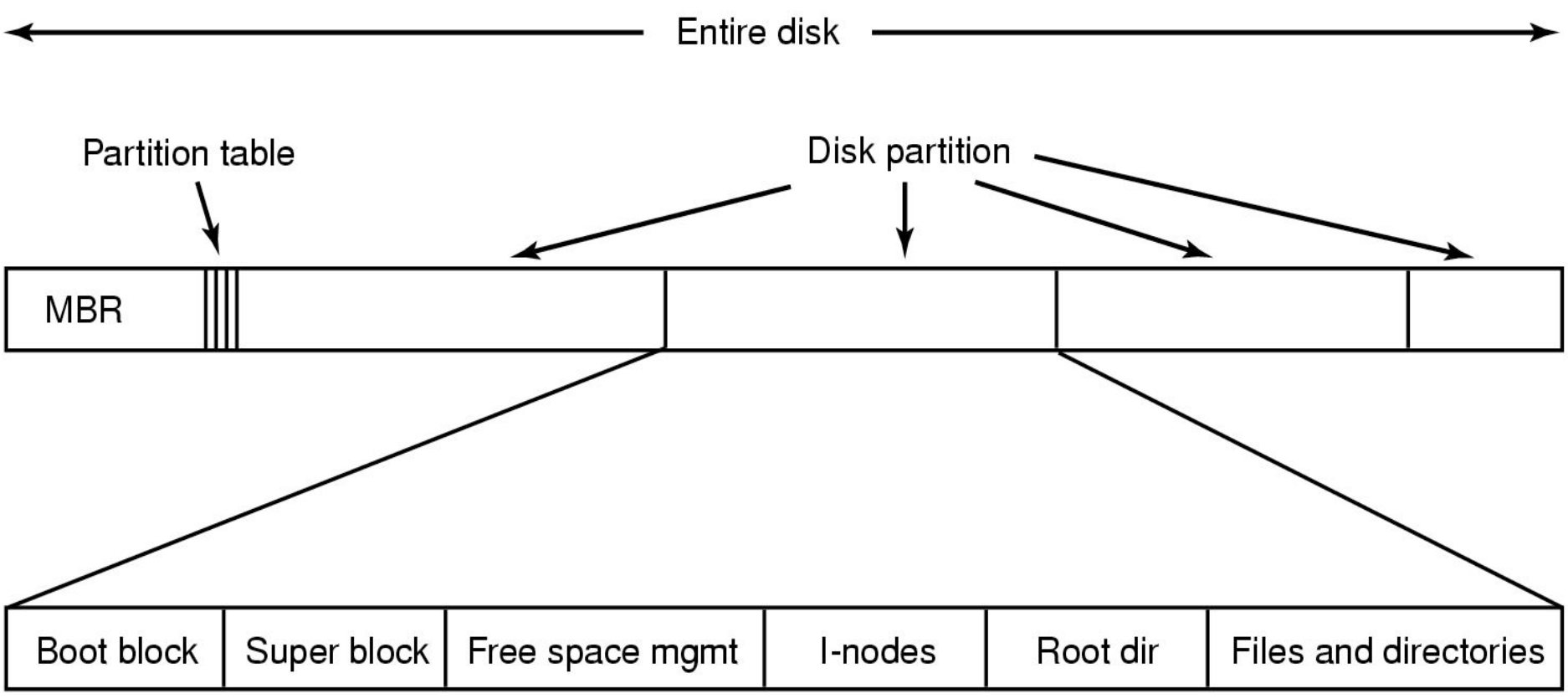
OS Startup

BIOS reads MBR , then reads & execs a boot block

An Example Disk



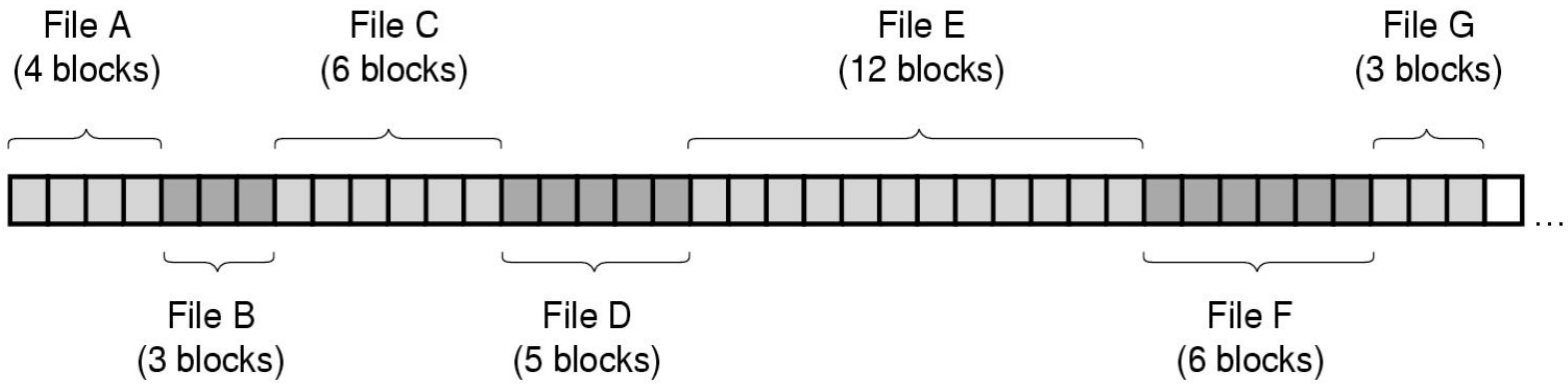
An Example Disk



Contiguous Allocation

Idea:

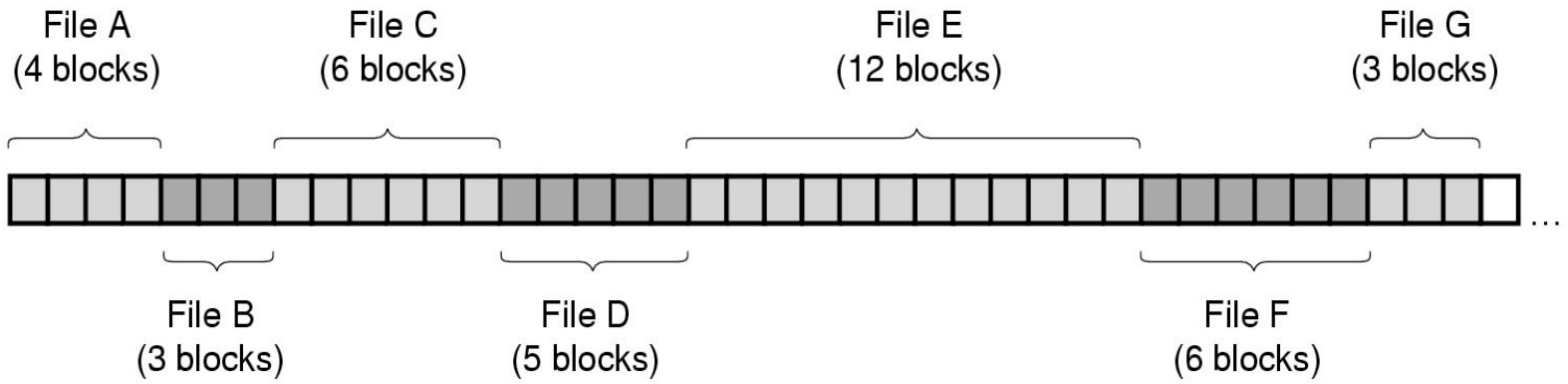
All blocks in a file are contiguous on the disk.



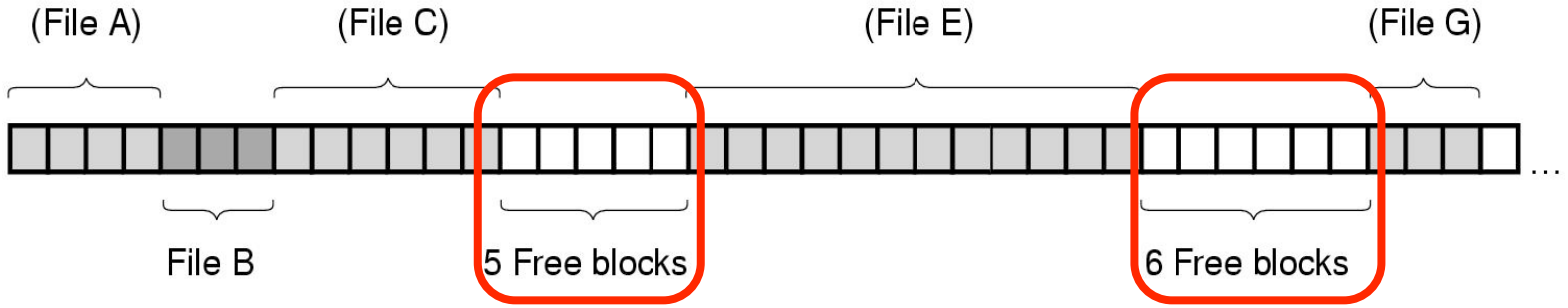
Contiguous Allocation

Idea:

All blocks in a file are contiguous on the disk.



After deleting D and F...



Contiguous Allocation

Advantages:

- **Simple to implement**
  (Need only starting sector & length of file)
- **Performance is good (for reading)**

Contiguous Allocation

Advantages:

- **Simple to implement**
  (Need only starting sector & length of file)
- **Performance is good (for reading)**

Disadvantages:

- **After deletions, disk becomes fragmented**
- **Will need periodic compaction (time-consuming)**
- **Will need to manage free lists**
  If new file put at end of disk...
  No problem
  If new file is put into a "hole"...
  Must know a file's maximum possible size
  ... at the time it is created

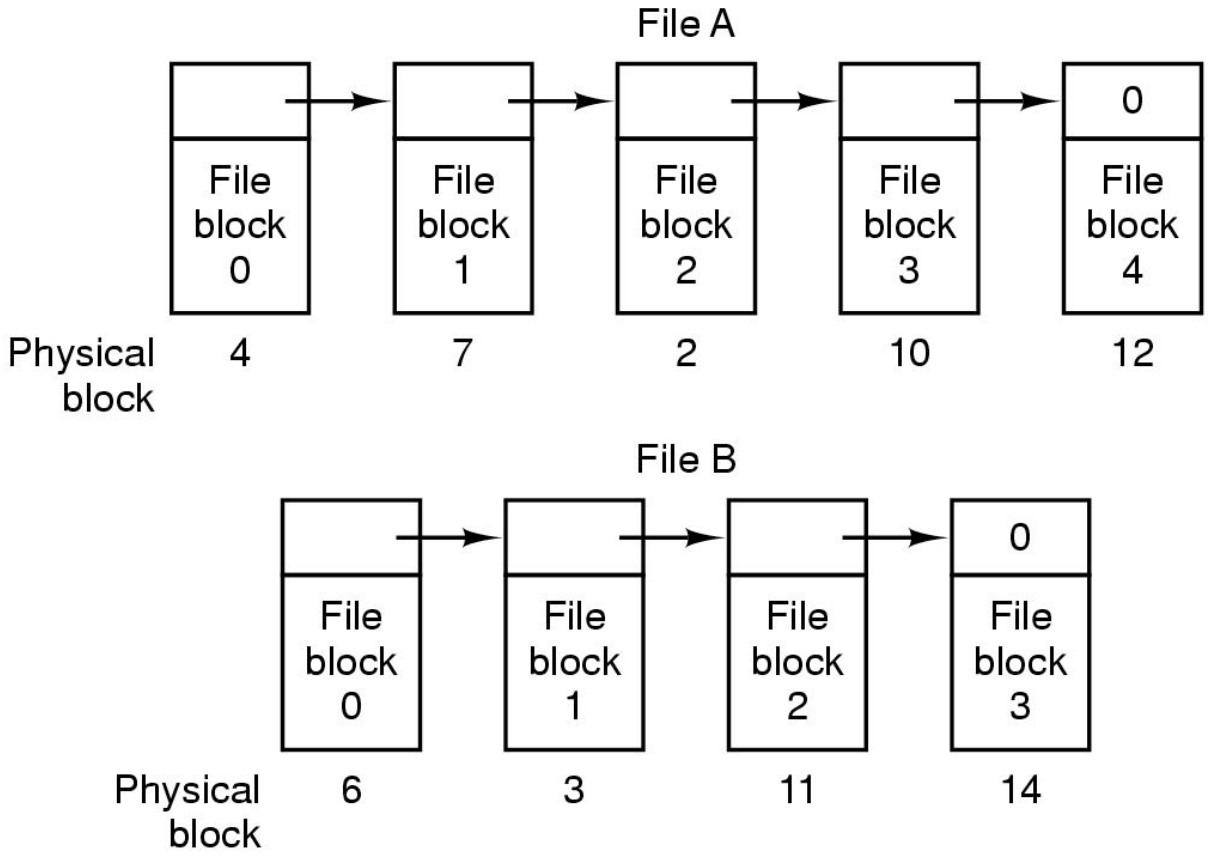
Contiguous Allocation

Good for CD-ROMs

- **All file sizes are known in advance**
- **Files are never deleted**

Linked List Allocation

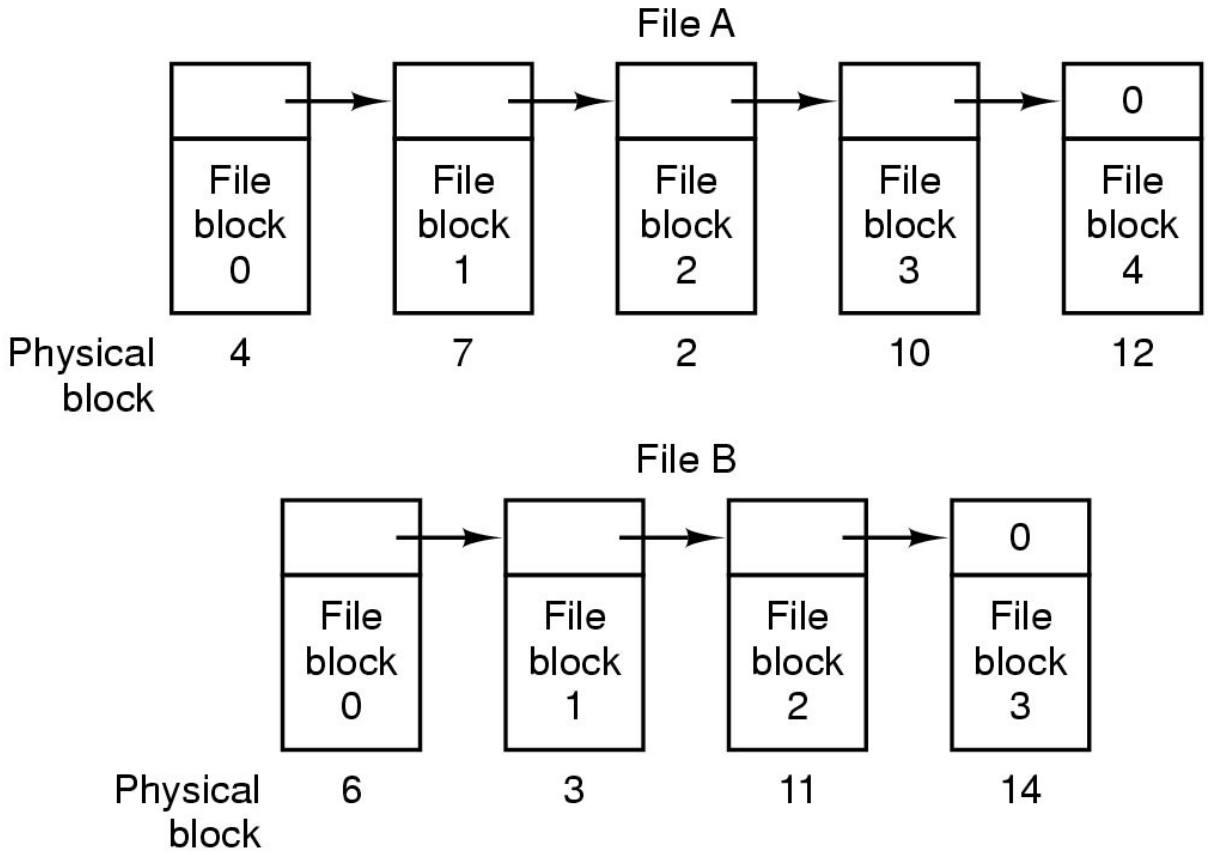
Each file is a sequence of blocks
First word in each block contains number of next block



Linked List Allocation

Each file is a sequence of blocks

First word in each block contains number of next block

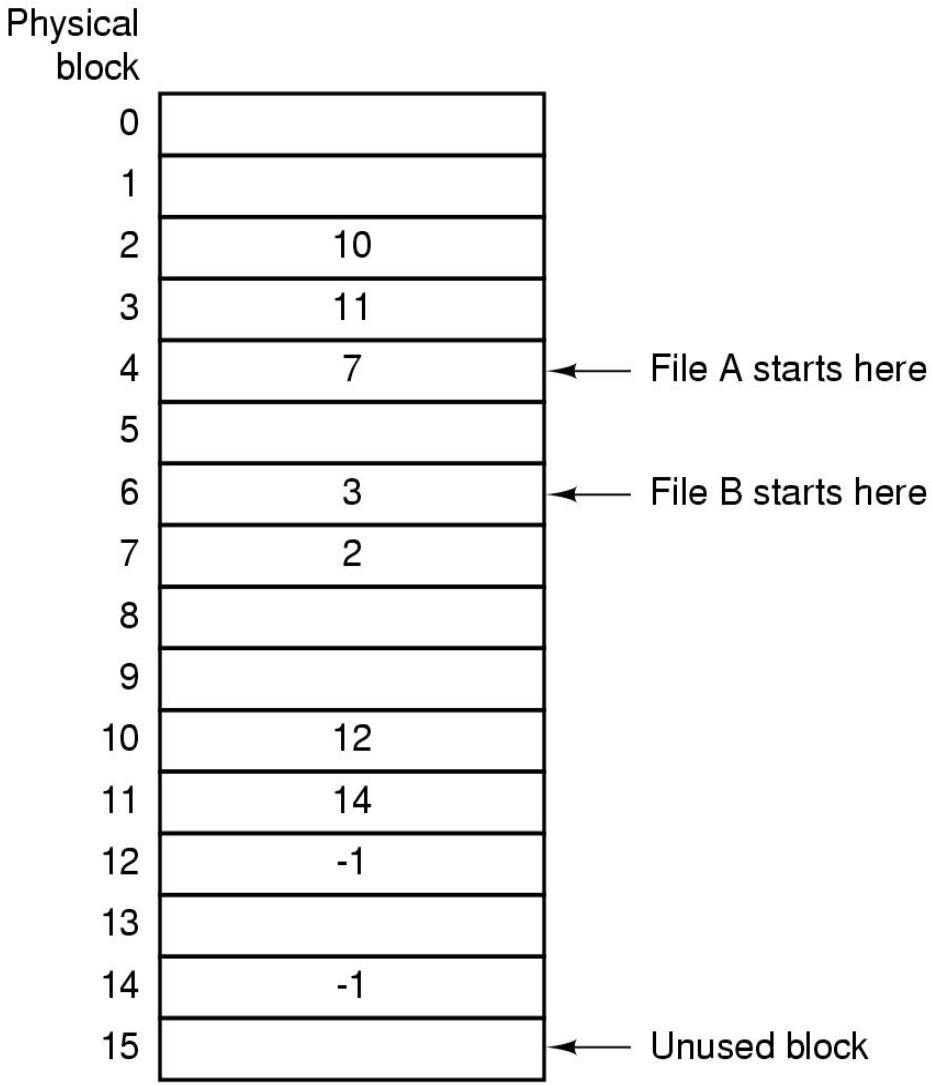


Random access into the file is slow!

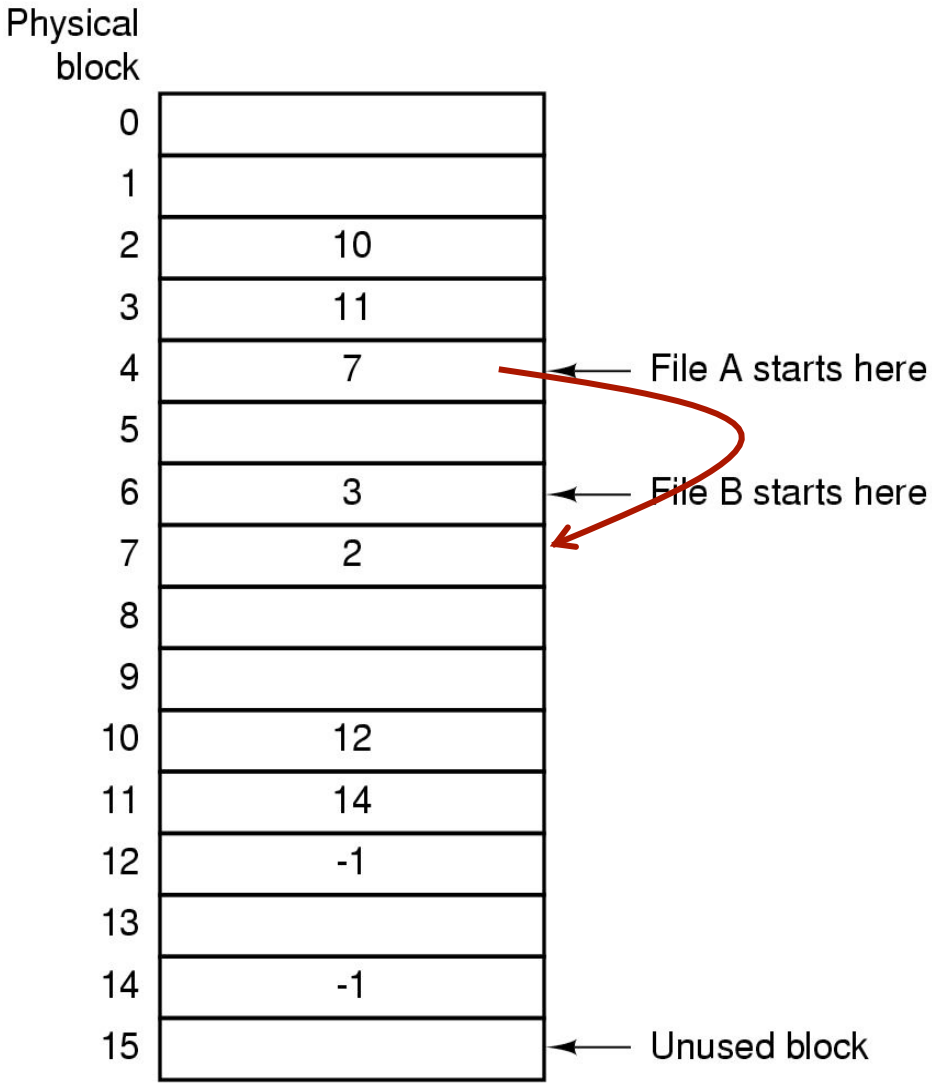
File Allocation Table (FAT)

Keep a table in memory
On entry per block on the disk
Each entry contains the address of the “next” block
A special value (-2) indicates the block is free

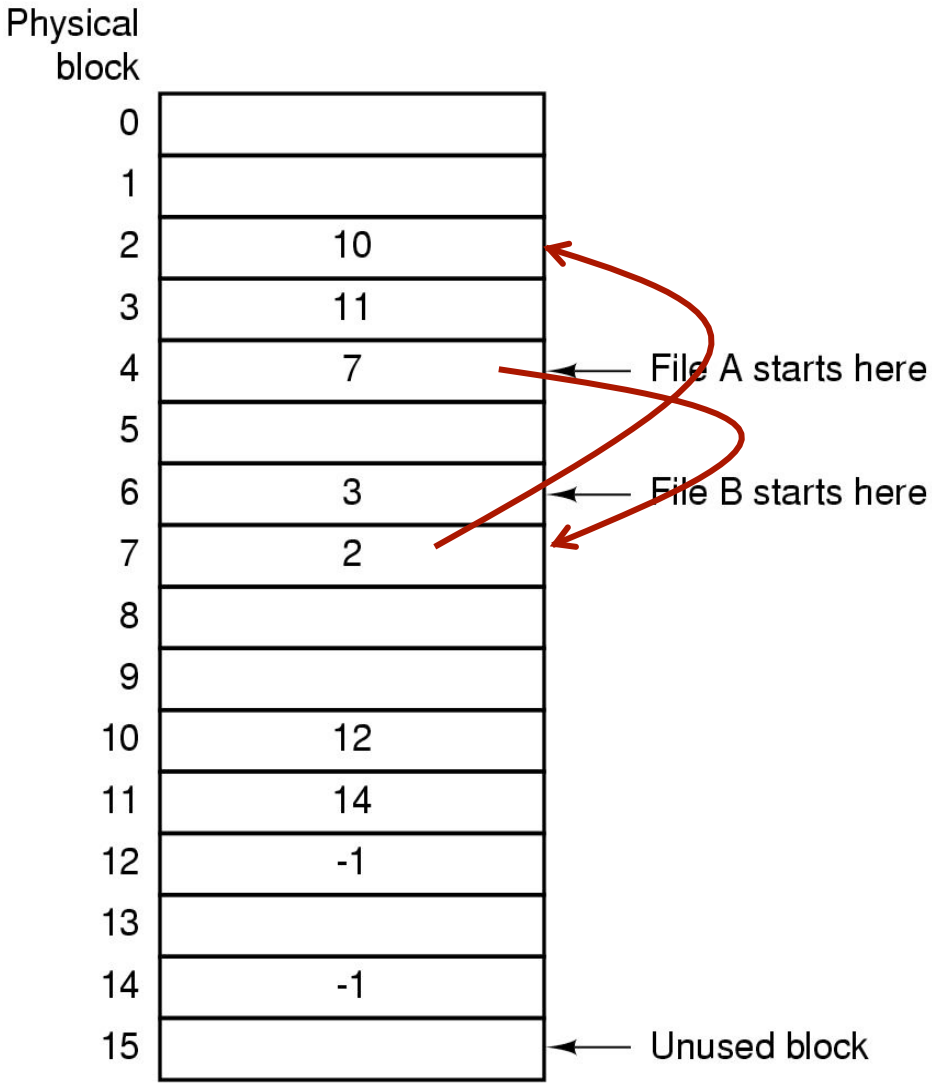
File Allocation Table (FAT)



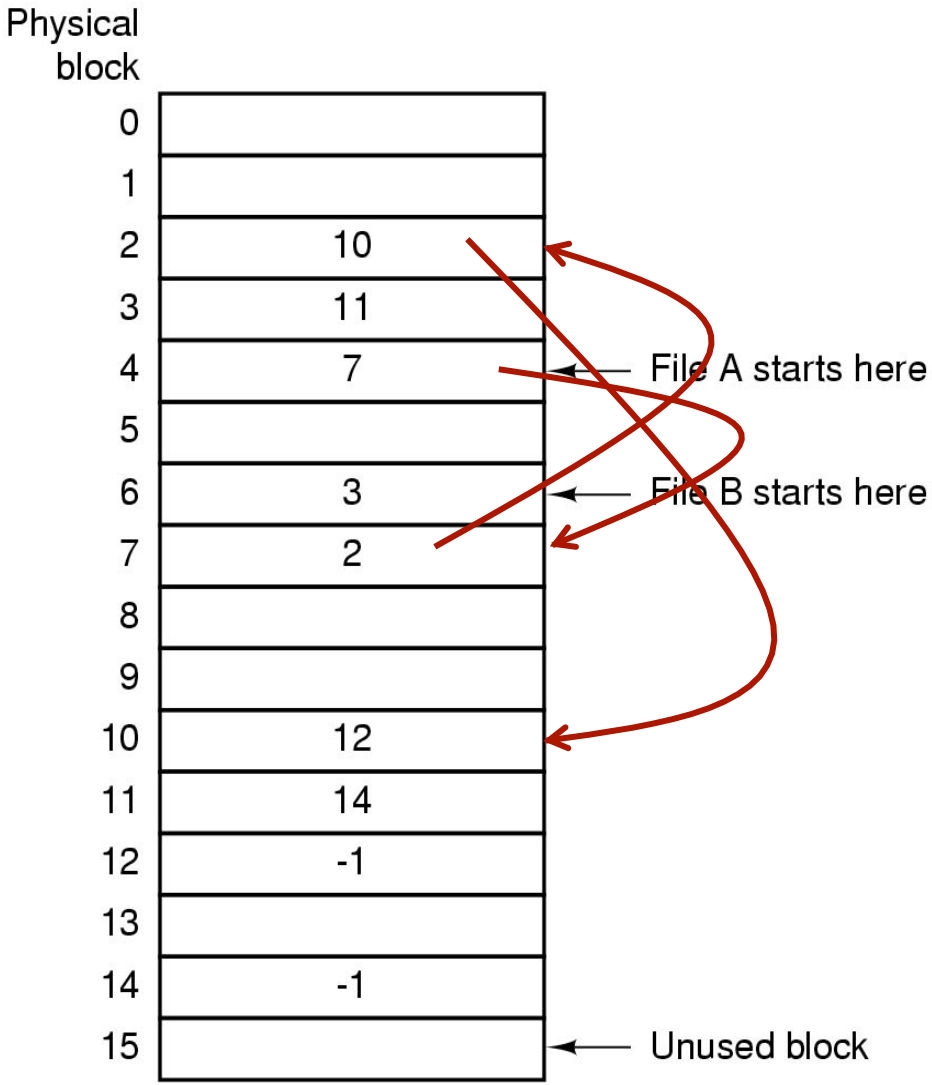
File Allocation Table (FAT)



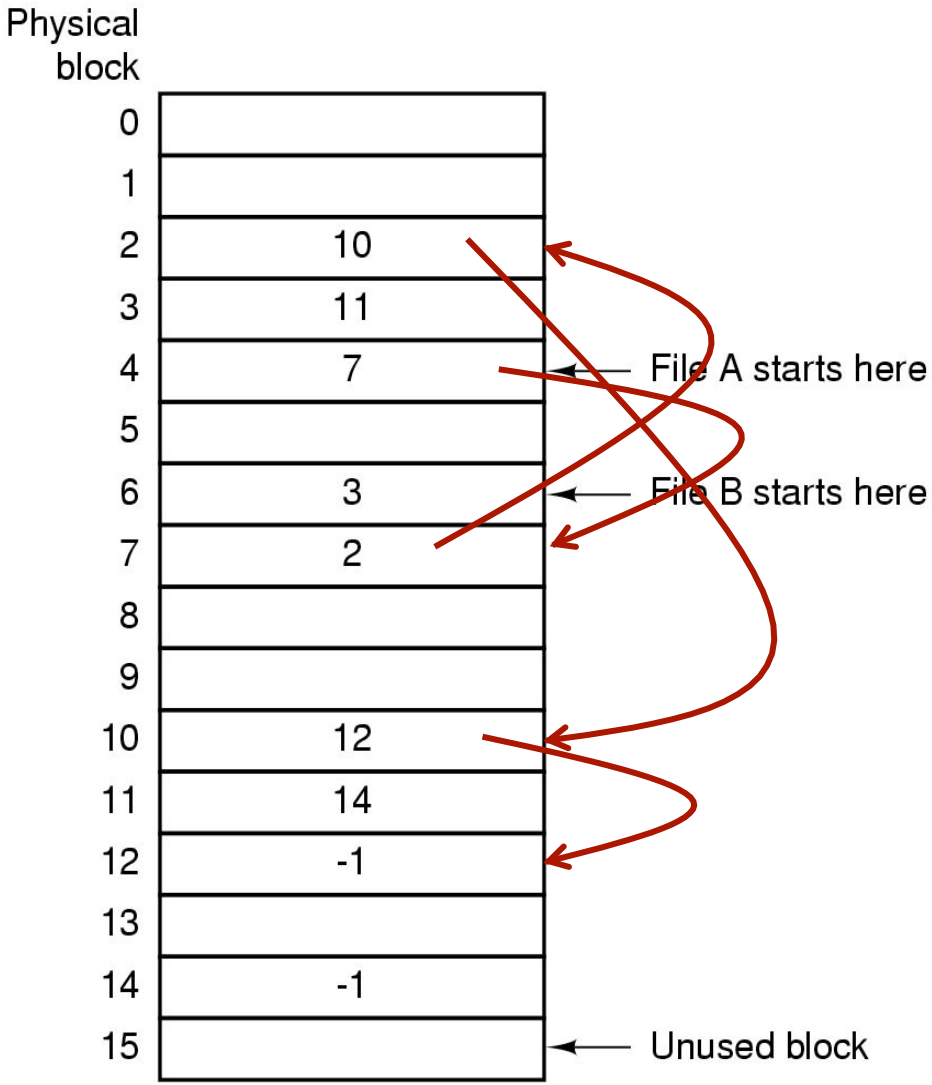
File Allocation Table (FAT)



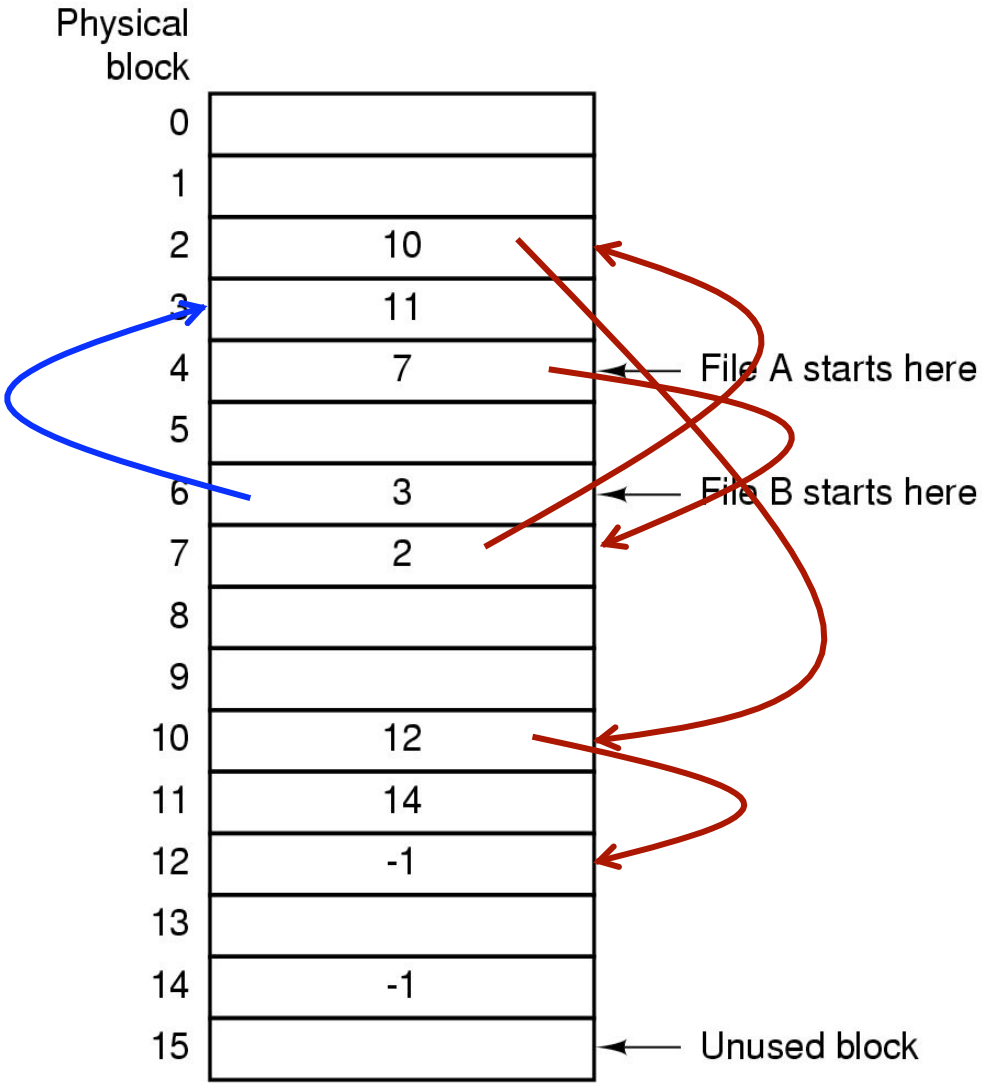
File Allocation Table (FAT)



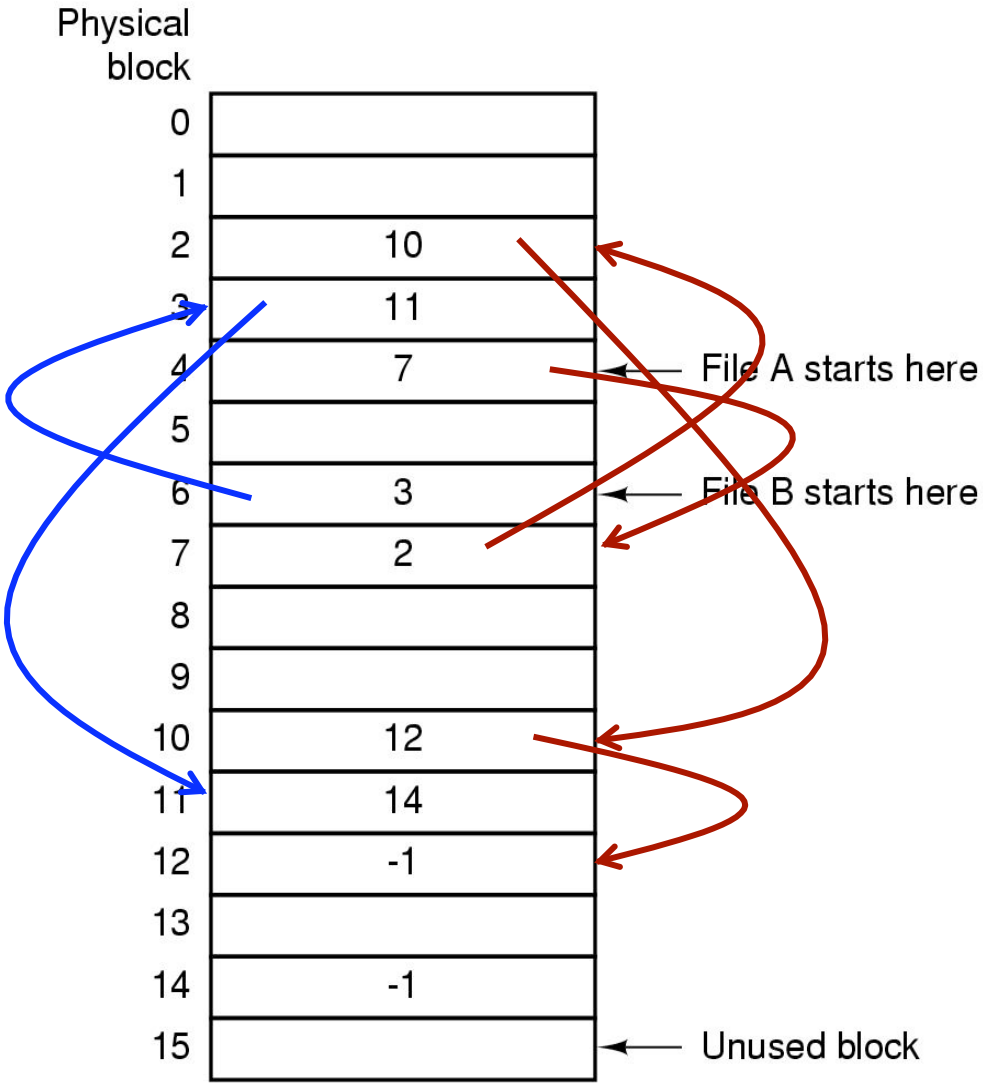
File Allocation Table (FAT)



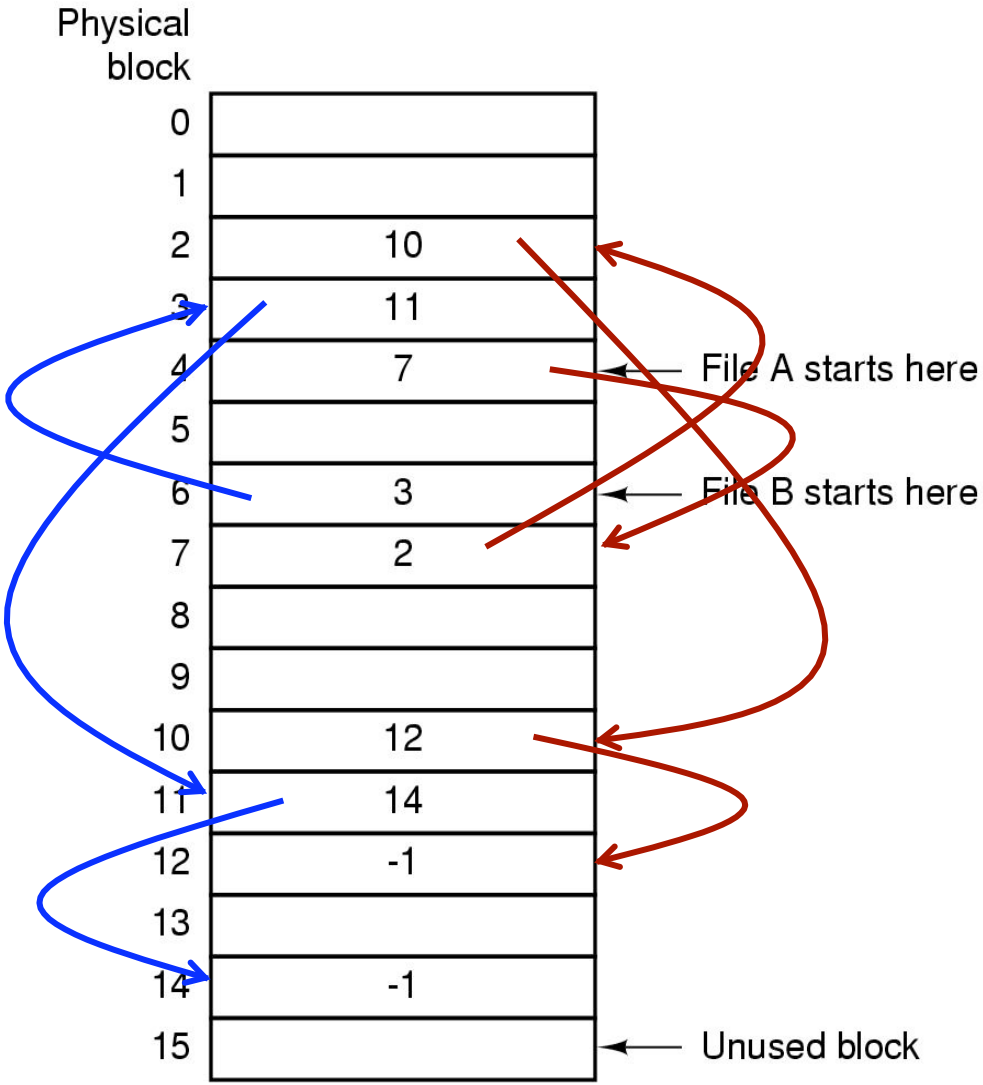
File Allocation Table (FAT)



File Allocation Table (FAT)



File Allocation Table (FAT)



File Allocation Table (FAT)

Random access...

- **Search the linked list (but all in memory)**

Directory Entry needs only one number

- **Starting block number**

File Allocation Table (FAT)

Random access...

- **Search the linked list (but all in memory)**

Directory Entry needs only one number

- **Starting block number**

Disadvantage:

- **Entire table must be in memory all at once!**

File Allocation Table (FAT)

Random access...

Search the linked list (but all in memory)

Directory Entry needs only one number

Starting block number

Disadvantage:

Entire table must be in memory all at once!

Example:

20 GB = disk size

1 KB = block size

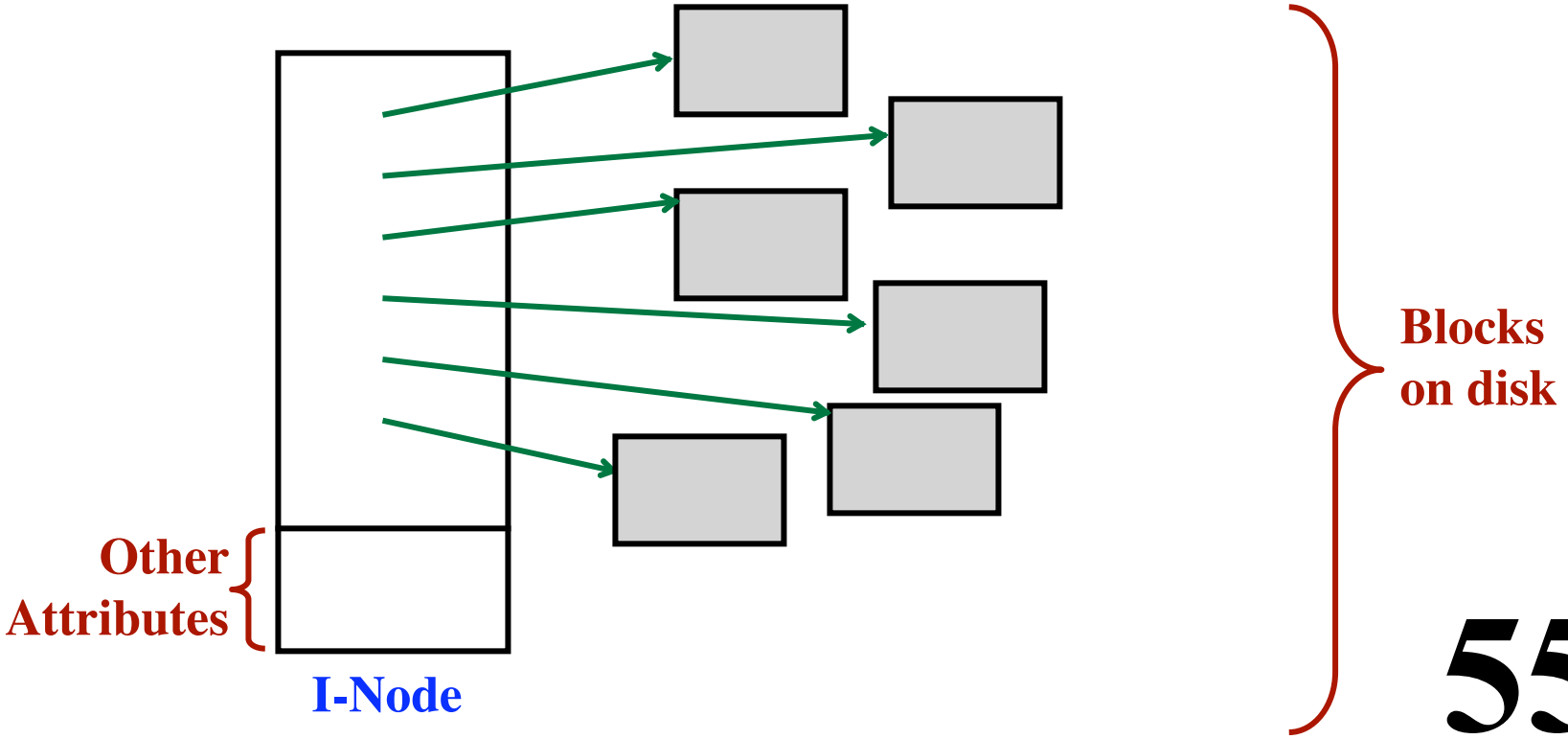
4 bytes = FAT entry size

80 MB of memory used to store the FAT

I-Nodes

Each I-Node ("index-node") is a structure / record
Contains info about the file

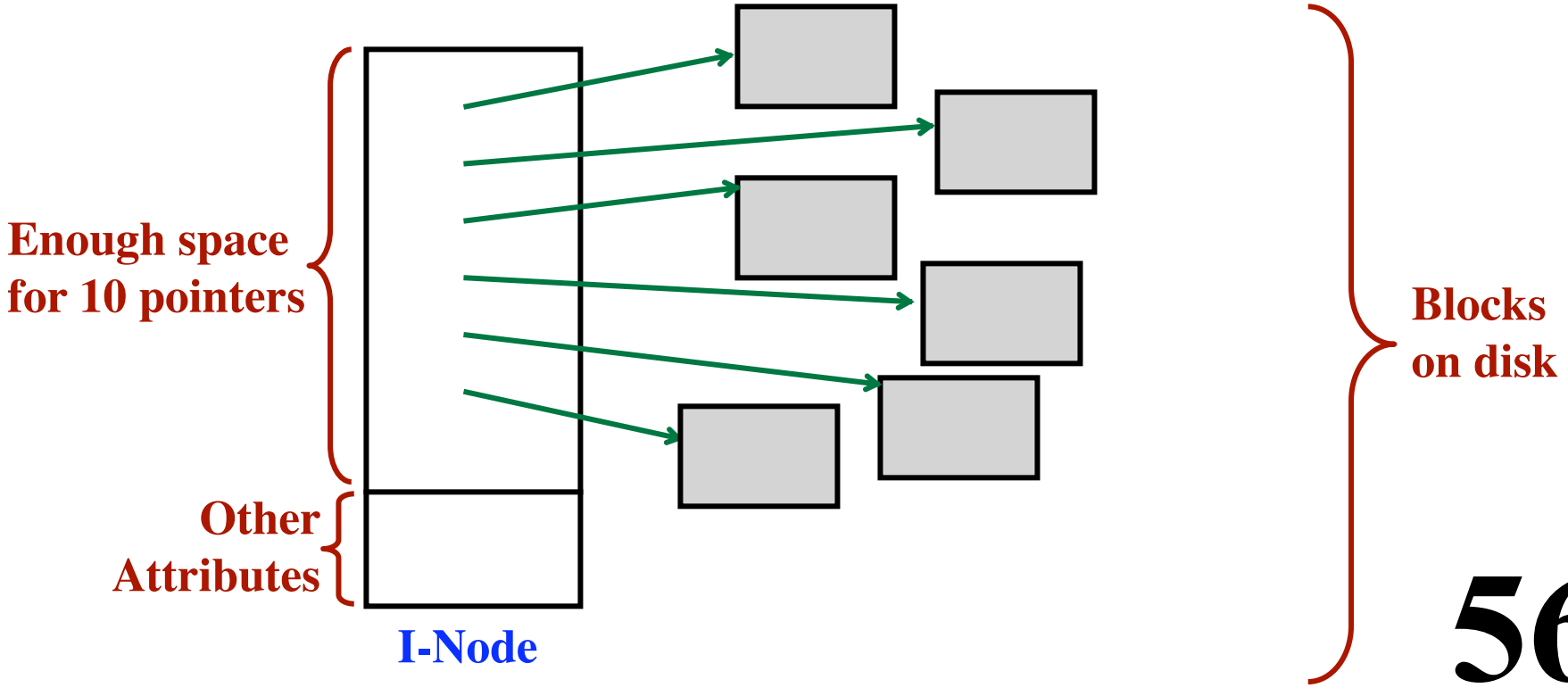
- **Attributes**
- **Location of the blocks containing the file**



I-Nodes

Each I-Node ("index-node") is a structure / record
Contains info about the file

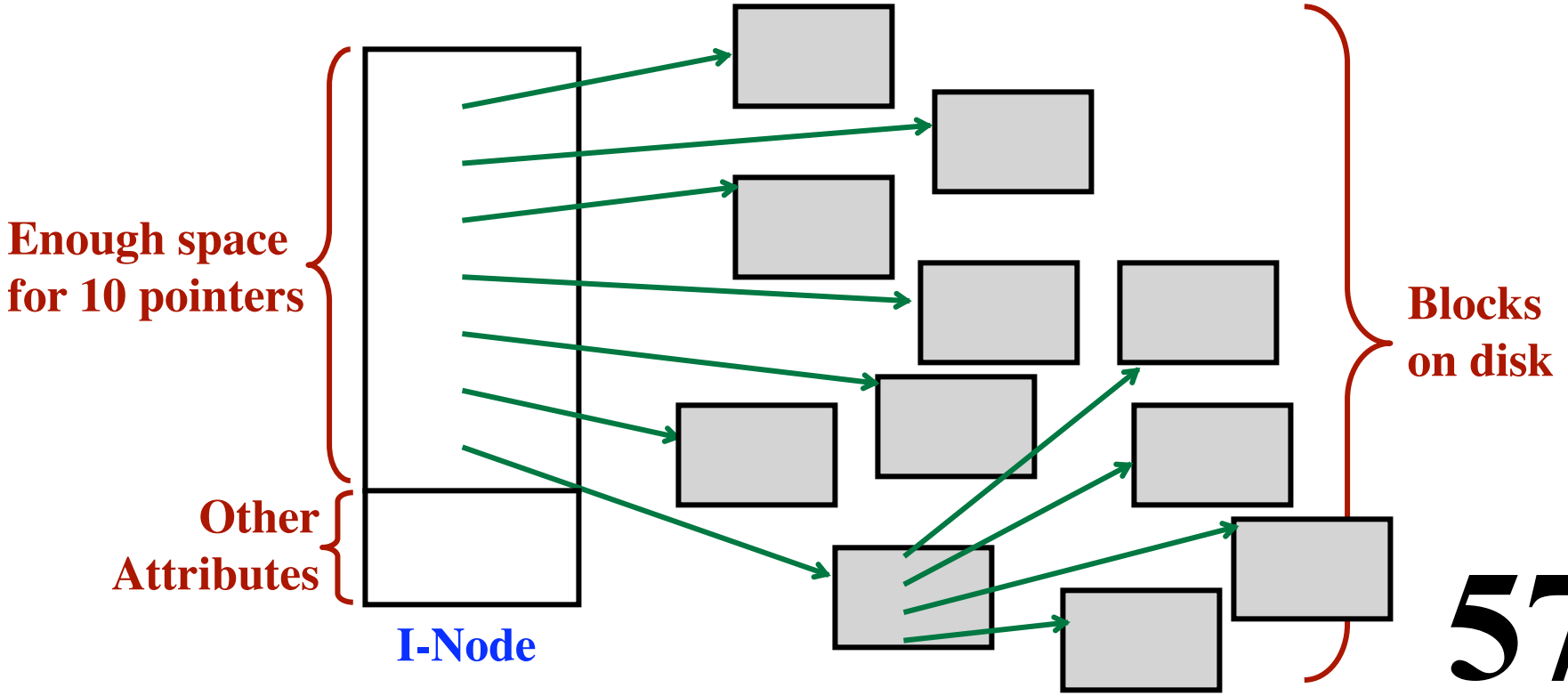
- **Attributes**
- **Location of the blocks containing the file**



I-Nodes

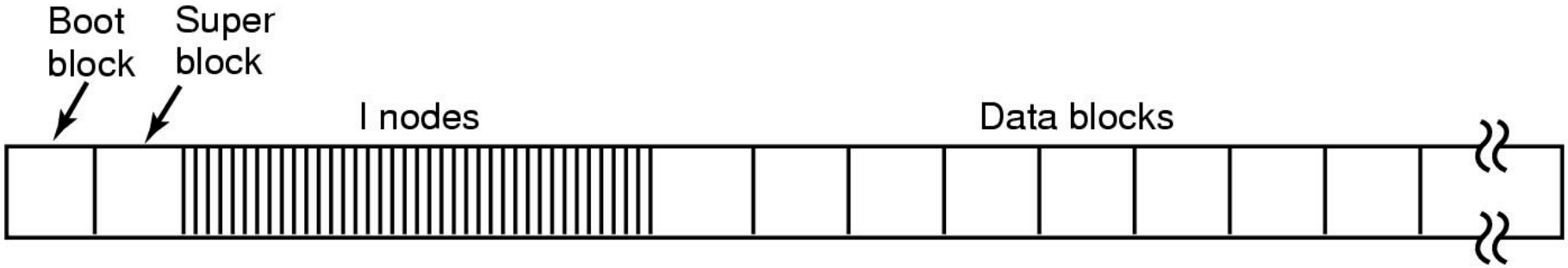
Each I-Node ("index-node") is a structure / record
Contains info about the file

- **Attributes**
- **Location of the blocks containing the file**

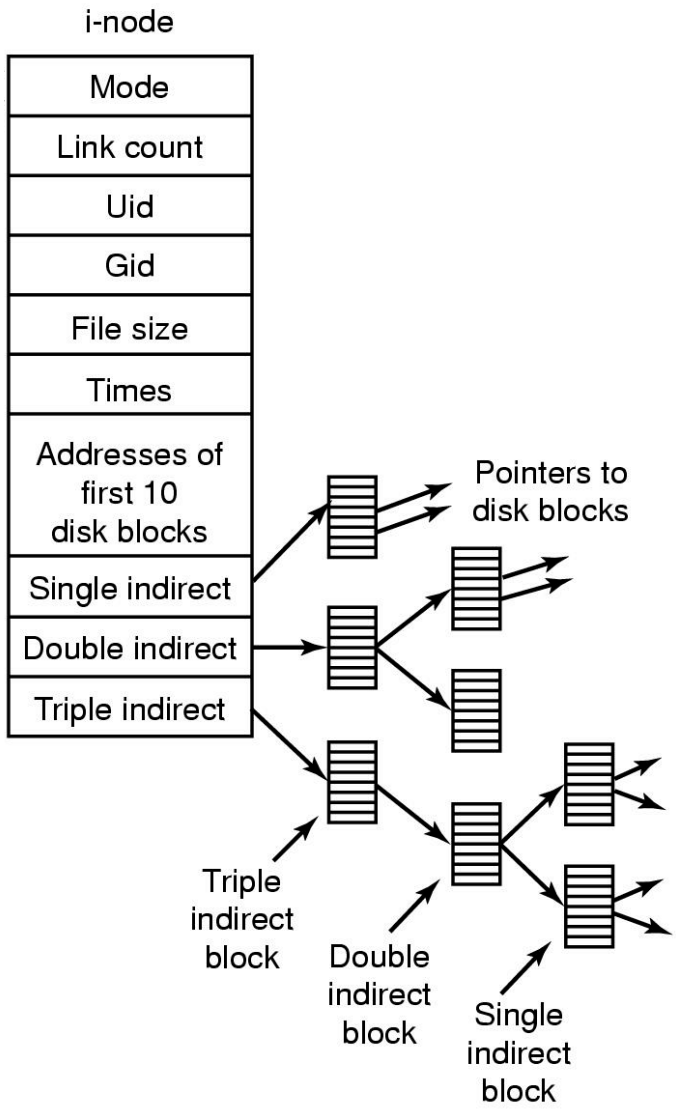


The UNIX File System

The layout of the disk:



The UNIX File System

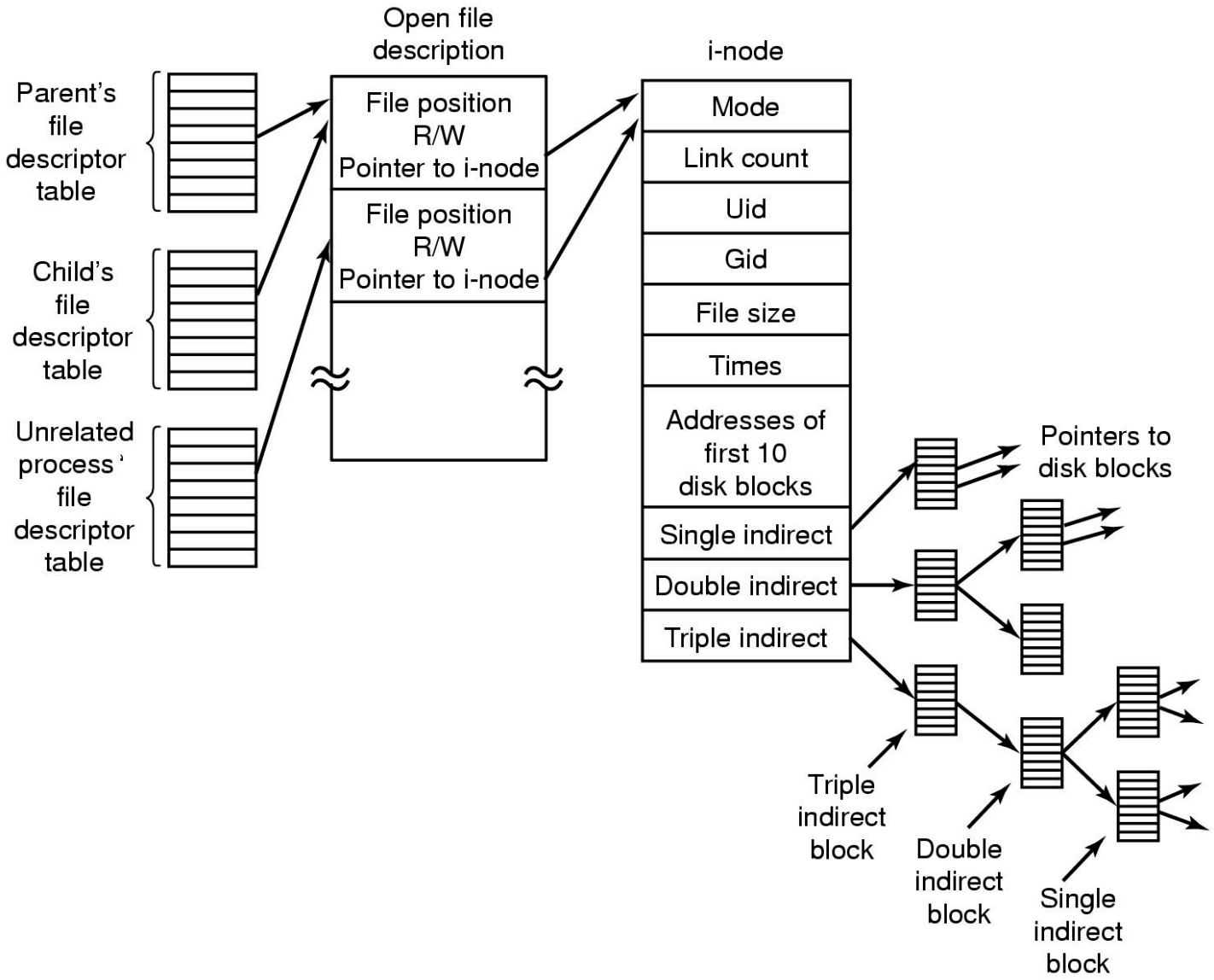


The UNIX File System

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

Structure of an I-Node

The UNIX File System



Directories

List of files

- **File name**
- **File Attributes**

Directories

List of files

- **File name**
- **File Attributes**

Simple Approach:

Put all attributes in the directory

Directories

List of files

- **File name**
- **File Attributes**

Simple Approach:

Put all attributes in the directory

Unix Approach:

Directory contains

File name

I-Node number

I-Node contains

File Attributes

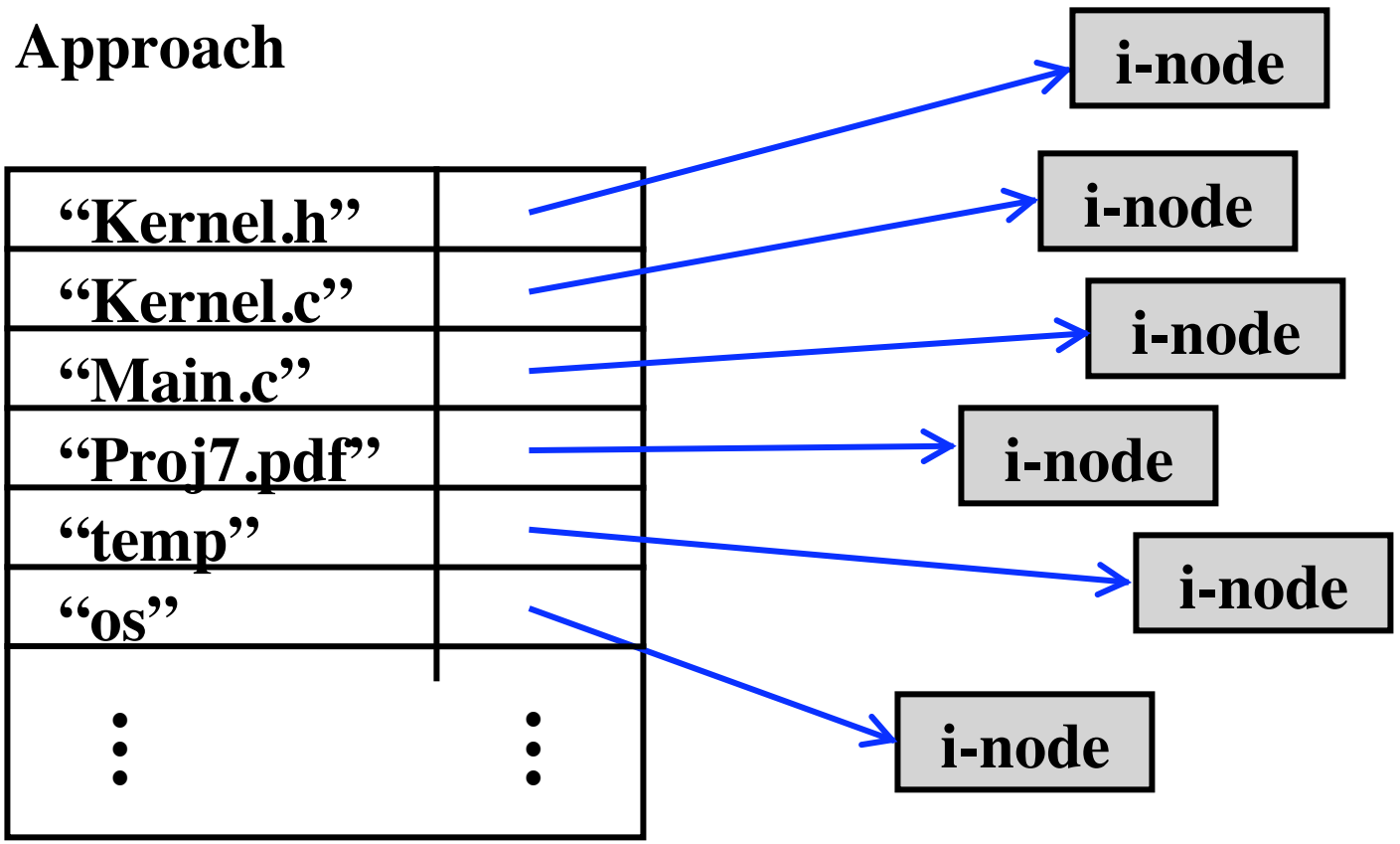
Directories

Simple Approach

“Kernel.h”	attributes
“Kernel.c”	attributes
“Main.c”	attributes
“Proj7.pdf”	attributes
“temp”	attributes
“os”	attributes
⋮	⋮

Directories

Unix Approach



Filenames

Short, Fixed Length Names

- **MS-DOS/Windows**
  - **8 + 3 “FILE3.BAK”**
  - **Each directory entry has 11 bytes for the name**
- **Unix (original)**
  - **Max 14 chars**

Filenames

Short, Fixed Length Names

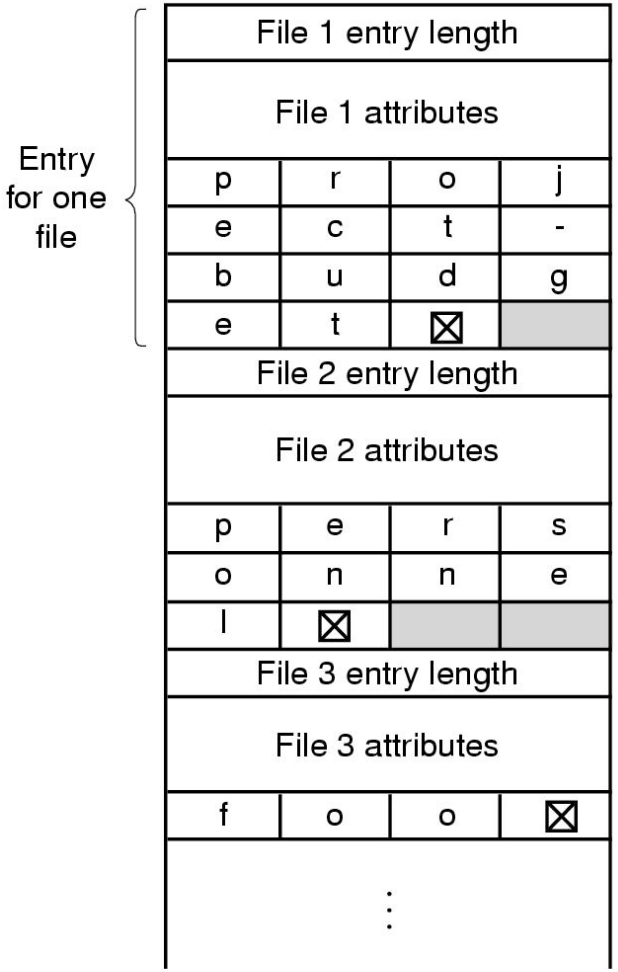
- MS-DOS/Windows
  - 8 + 3 “FILE3.BAK”
  - Each directory entry has 11 bytes for the name
- Unix (original)
  - Max 14 chars

Variable Length Names

- Unix (today)
  - Max 255 chars
  - Directory structure gets more complex

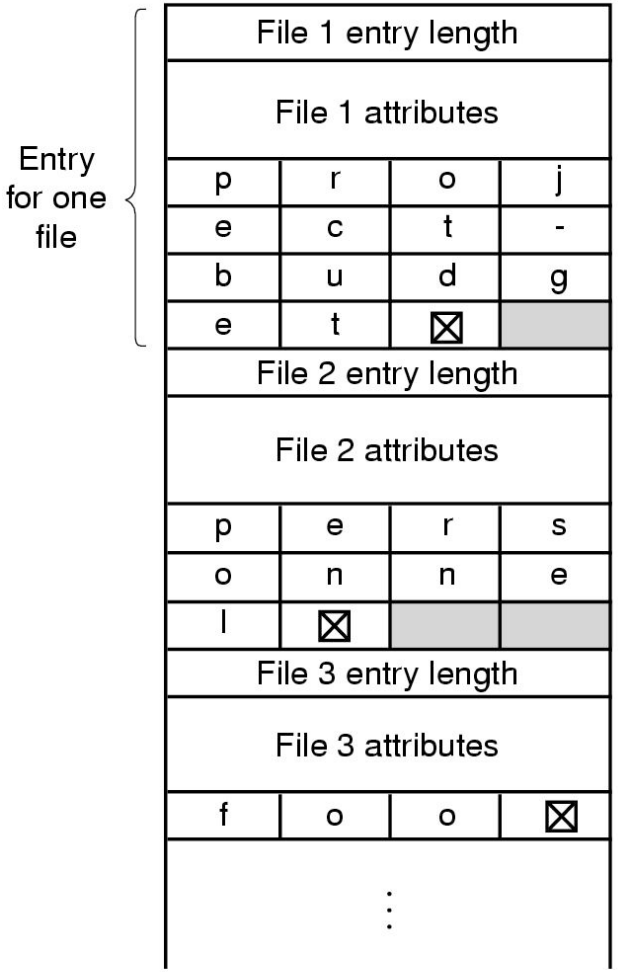
Variable-Length Filenames

Approach #1

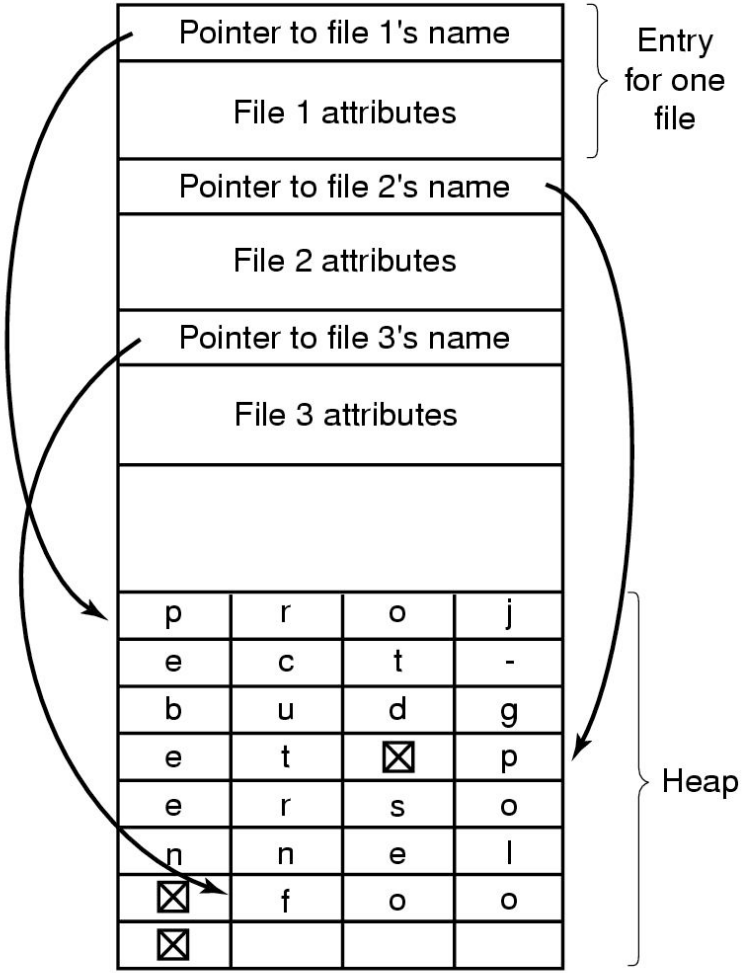


Variable-Length Filenames

Approach #1

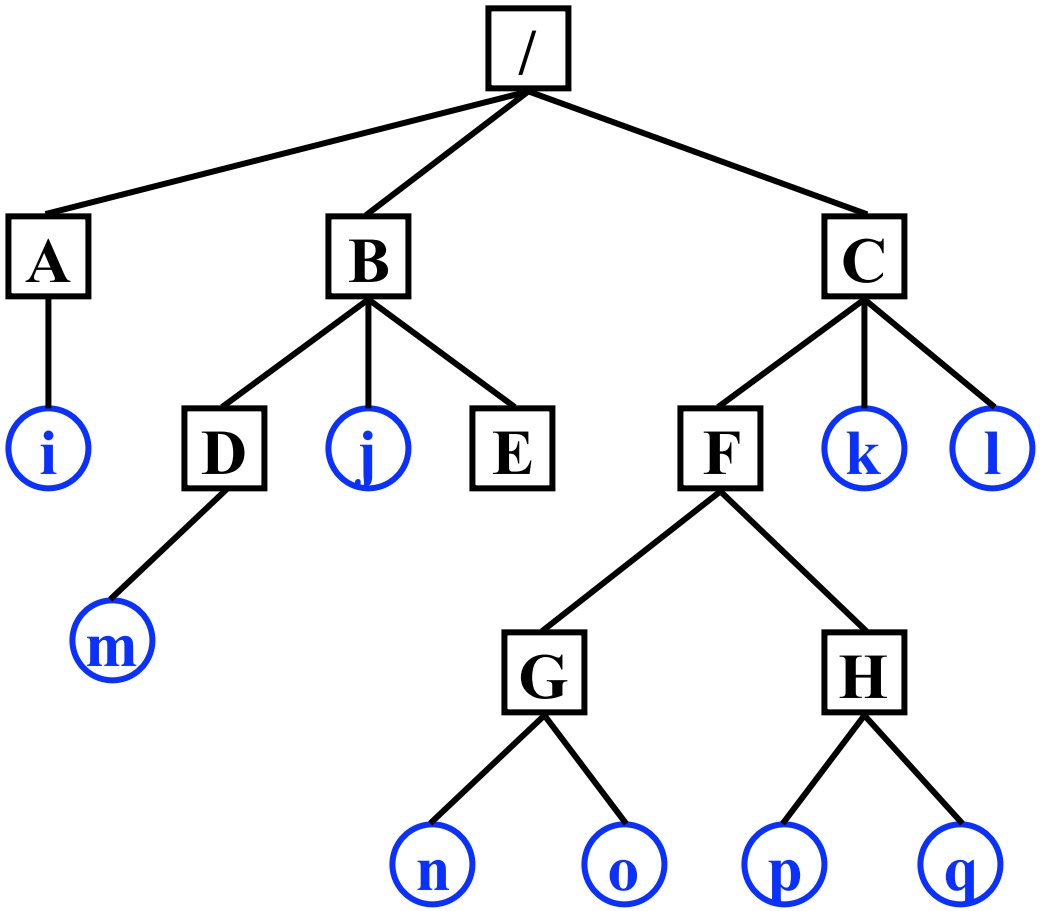


Approach #2



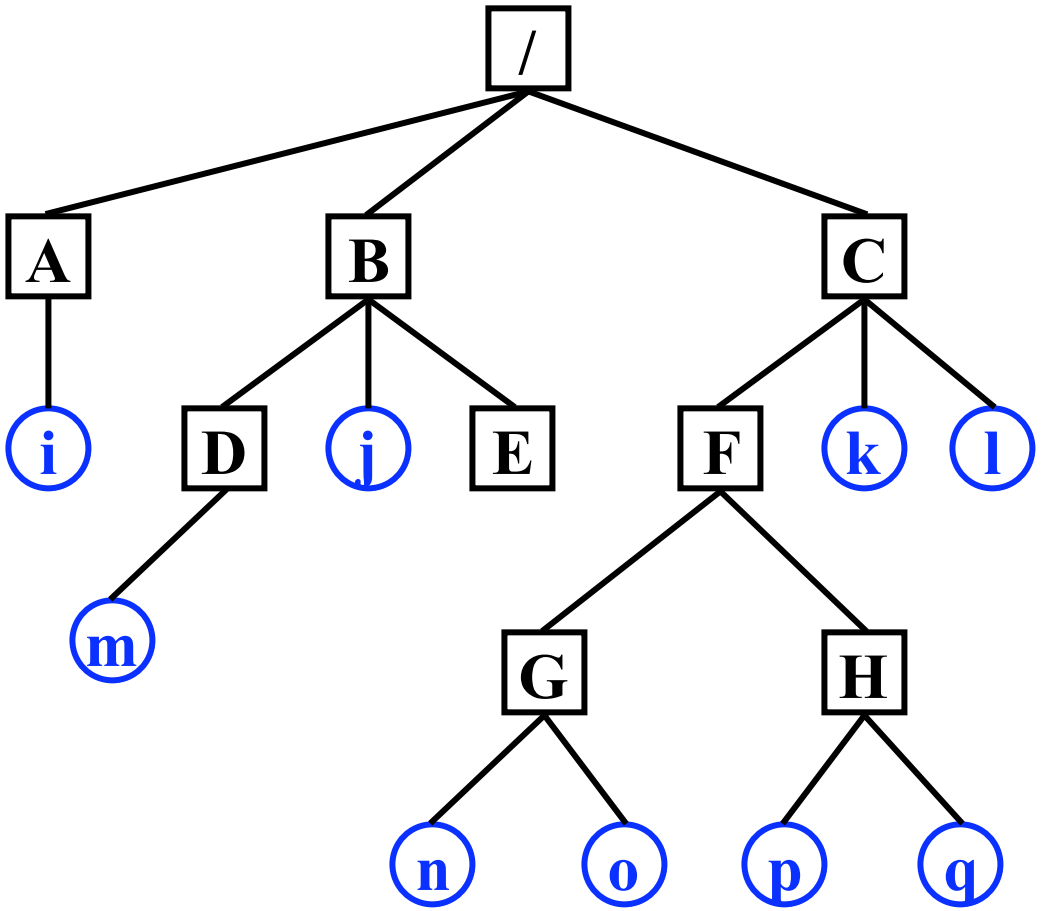
Sharing Files

One file appears in several directories.
Tree → DAG



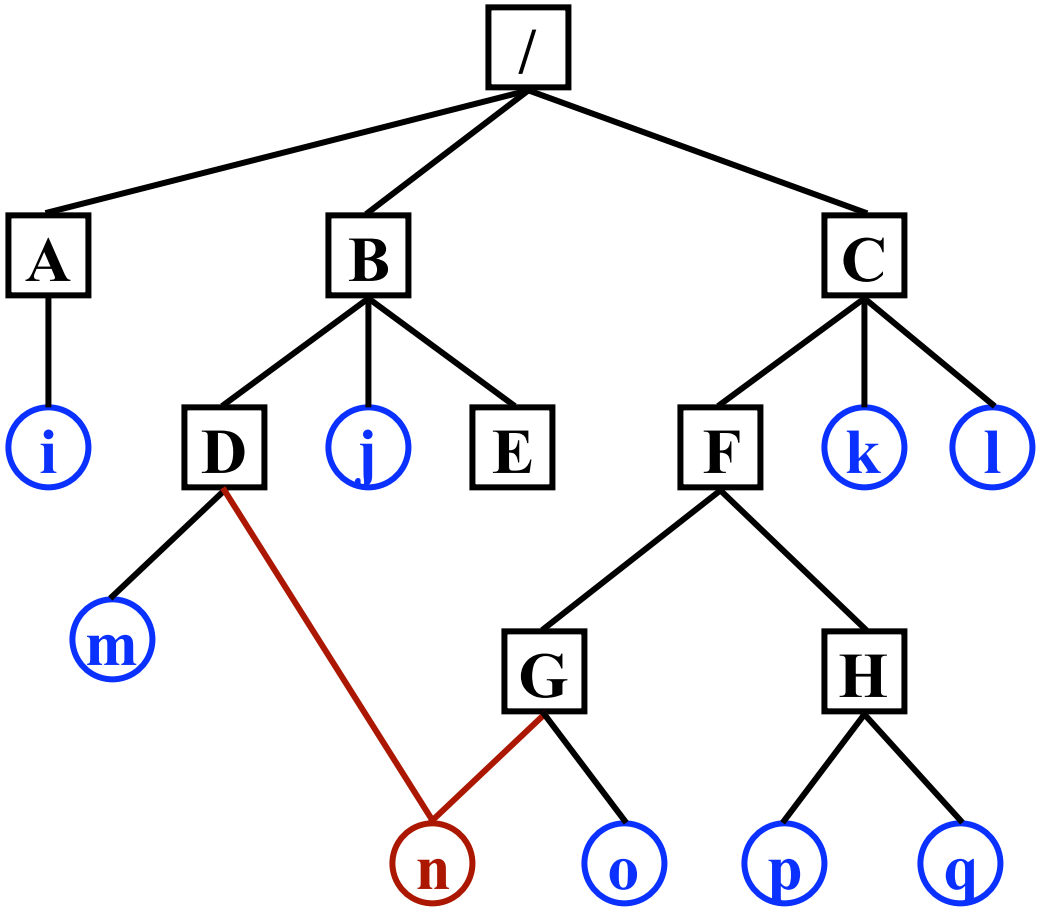
Sharing Files

One file appears in several directories.
Tree → DAG (Directed Acyclic Graph)



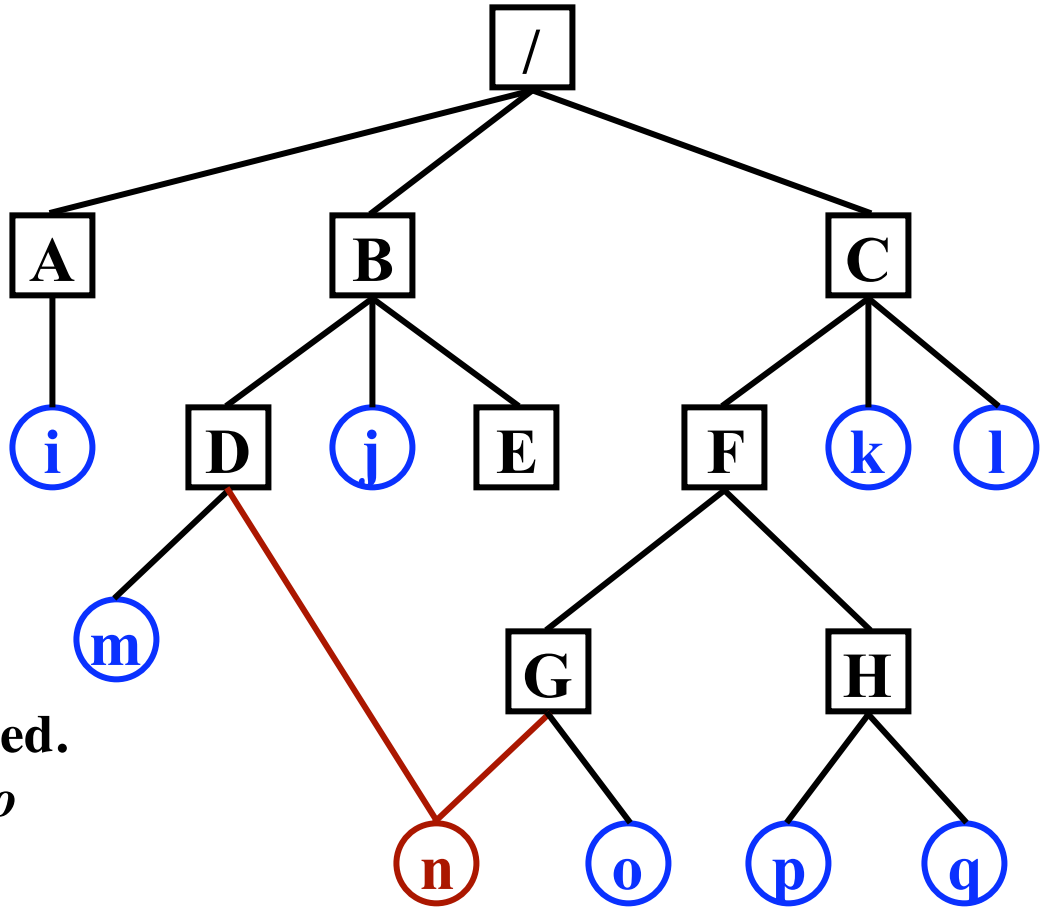
Sharing Files

One file appears in several directories.
Tree → DAG (Directed Acyclic Graph)



Sharing Files

One file appears in several directories.
Tree → DAG (Directed Acyclic Graph)



What if the file changes?

New disk blocks are used.

Better not store this info in the directories!!!

Hard Links and Symbolic Links

In Unix:

Hard links

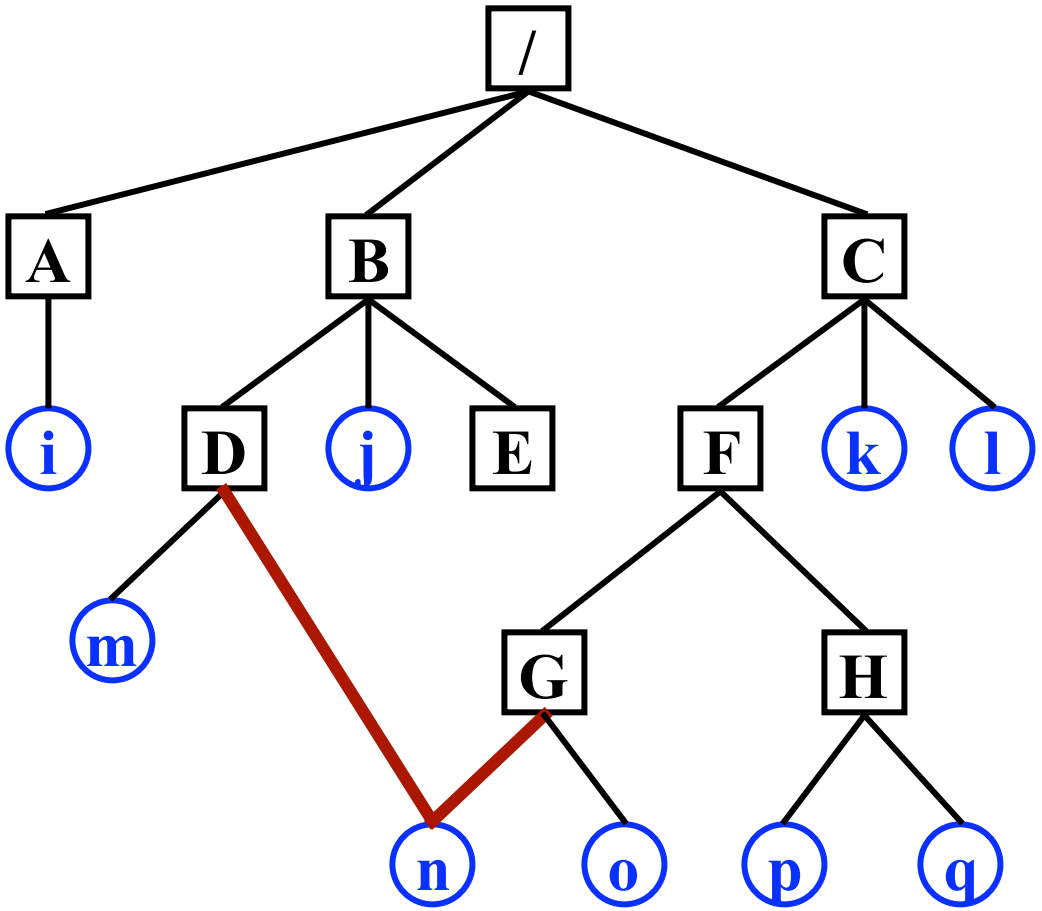
Both directories point to the same i-node

Symbolic links

One directory points to the file's i-node

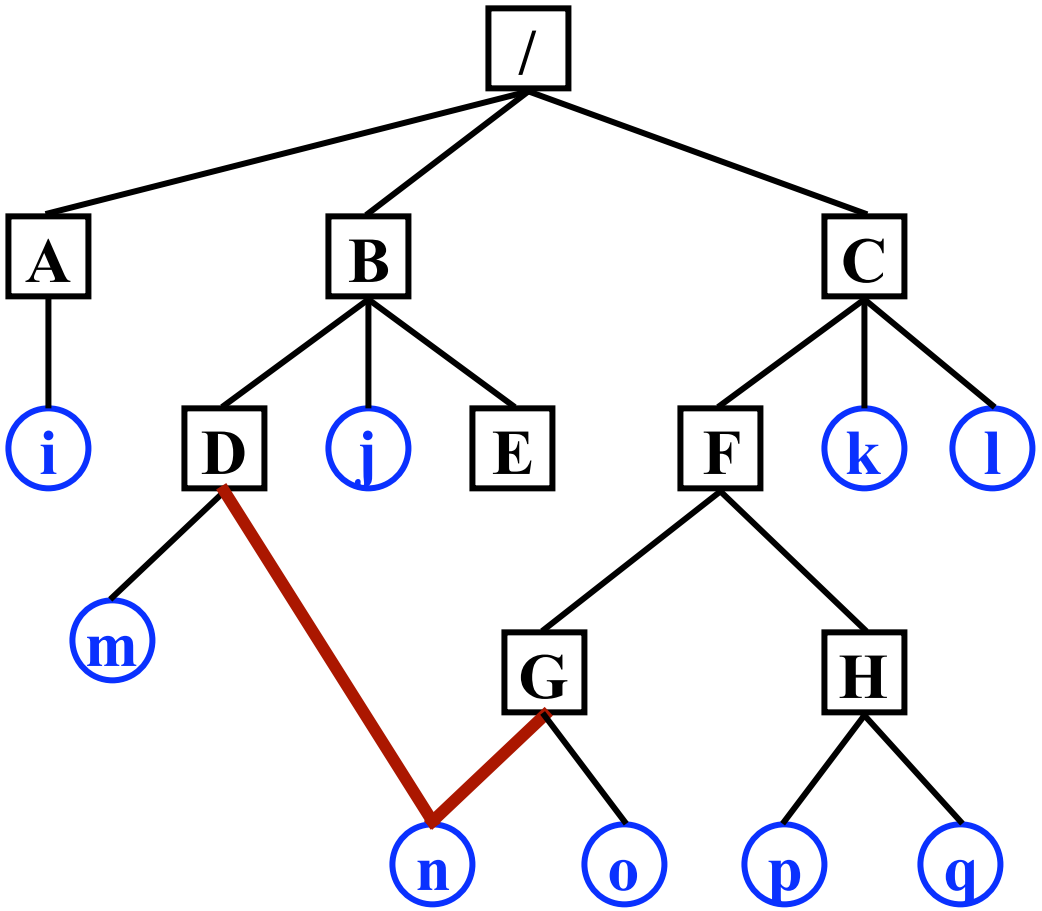
Other directory contains the "path"

Hard Links



Hard Links

Assume i-node number of "n" is 45.



Hard Links

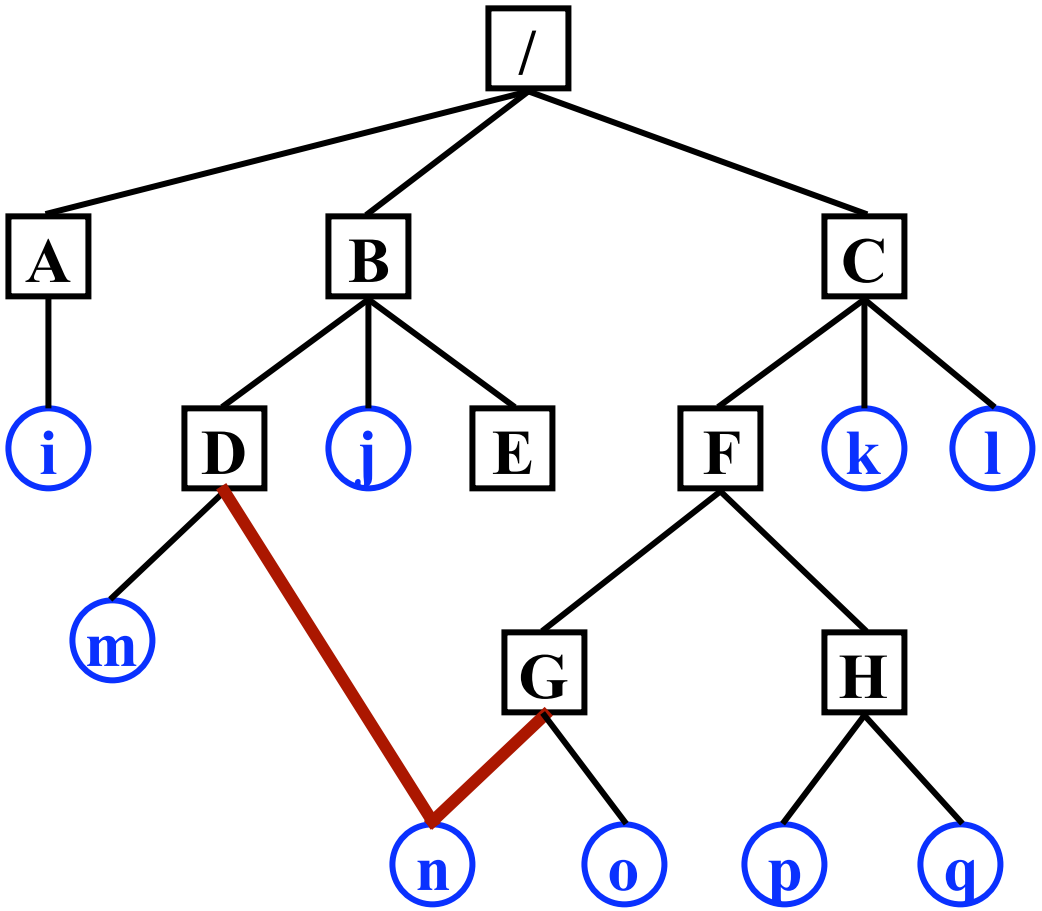
Assume i-node number of "n" is 45.

Directory "D"

"m"	123
"n"	45
⋮	⋮

Directory "G"

"n"	45
"o"	87
⋮	⋮



Hard Links

Assume i-node numb

The file may have a different name in each directory

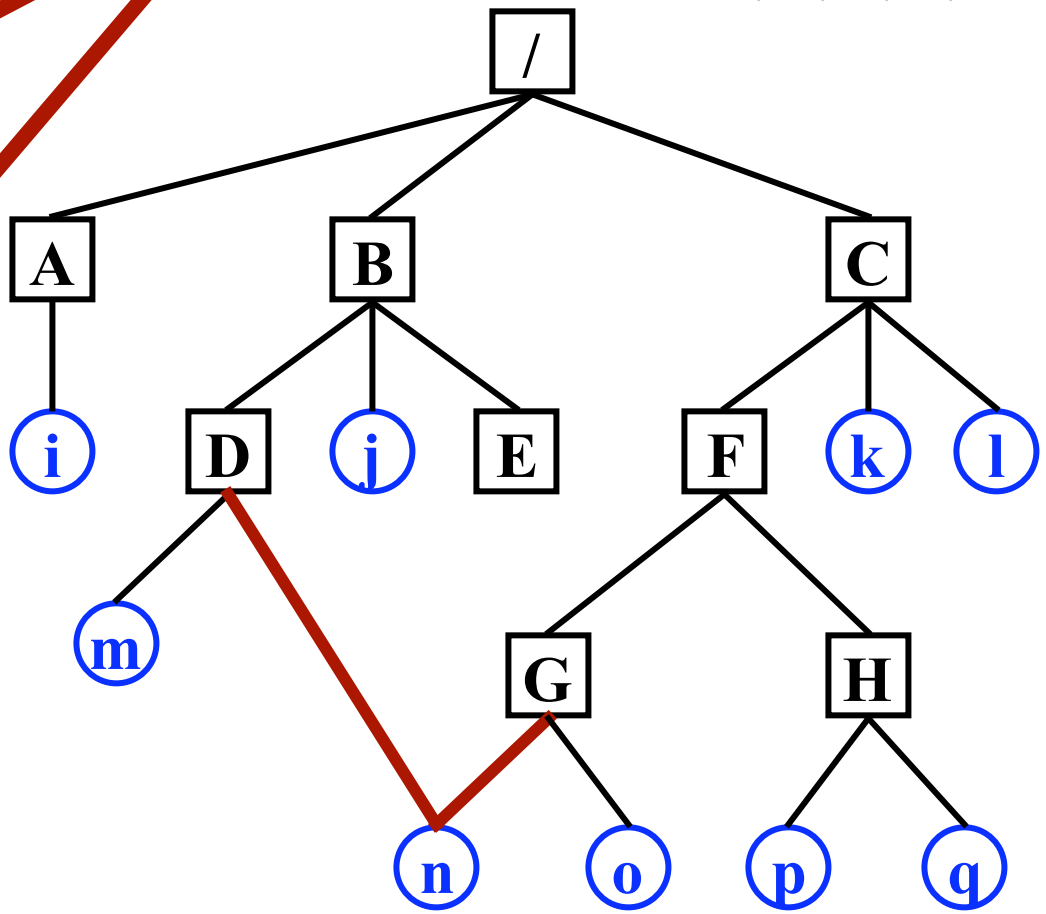
/B/D/n1
/C/F/G/n2

Directory "D"

"m"	123
"n1"	45
⋮	⋮

Directory "G"

"n2"	45
"o"	87
⋮	⋮

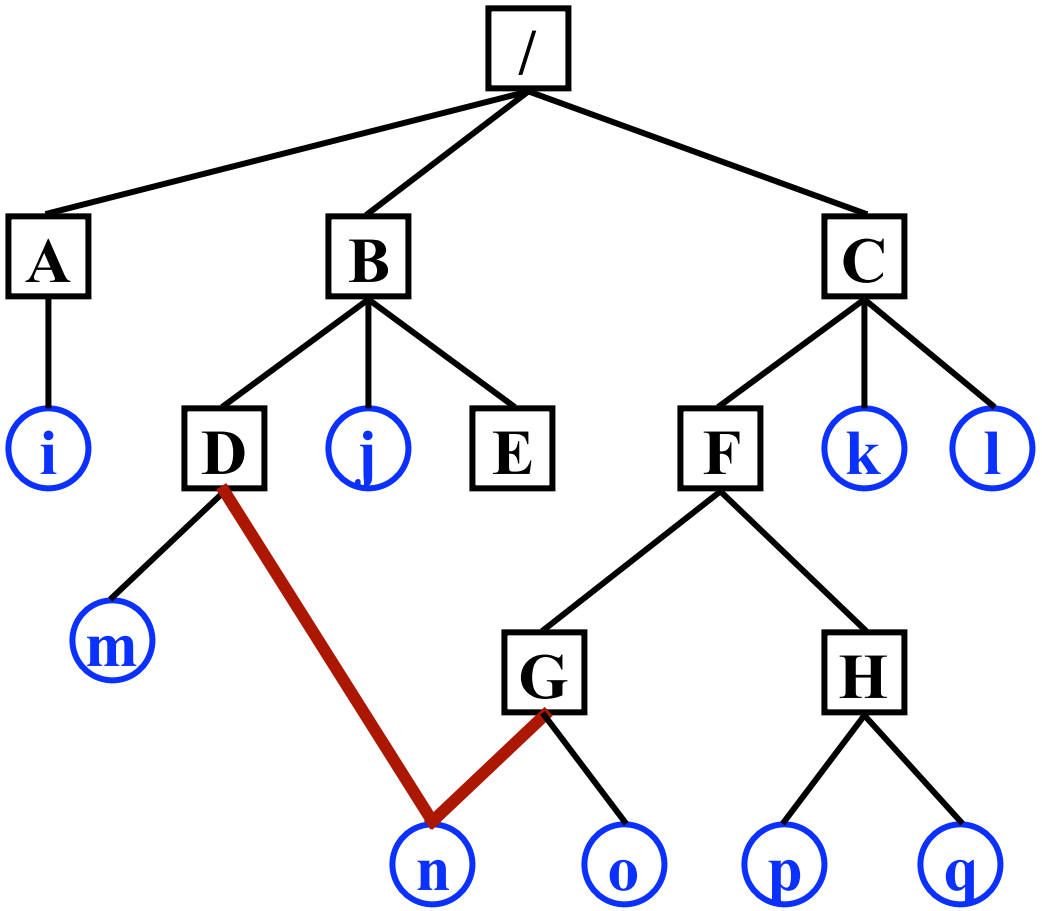


Hard Links

The name of a file is stored in the directory that points to the file.

Edges are labeled.

```
/B/D/n1
/C/F/G/n2
```

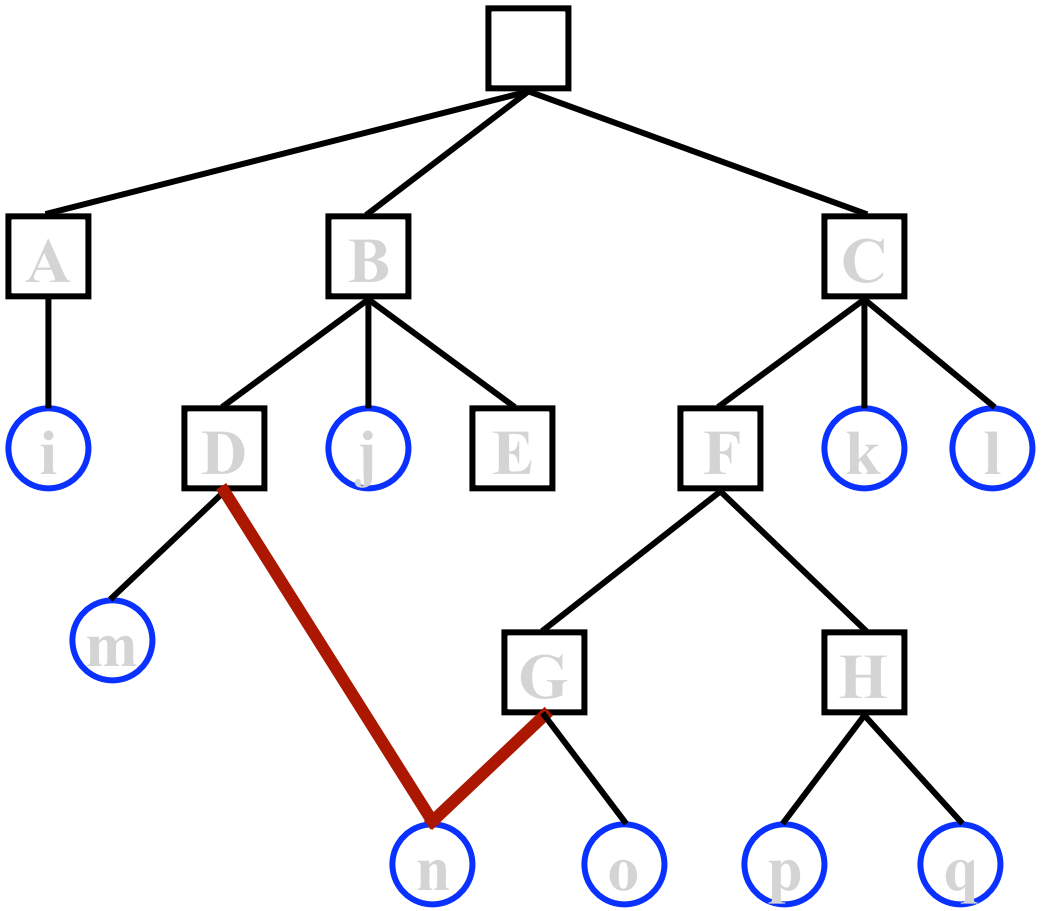


Hard Links

The name of a file is stored in the directory that points to the file.

Edges are labeled.

```
/B/D/n1
/C/F/G/n2
```

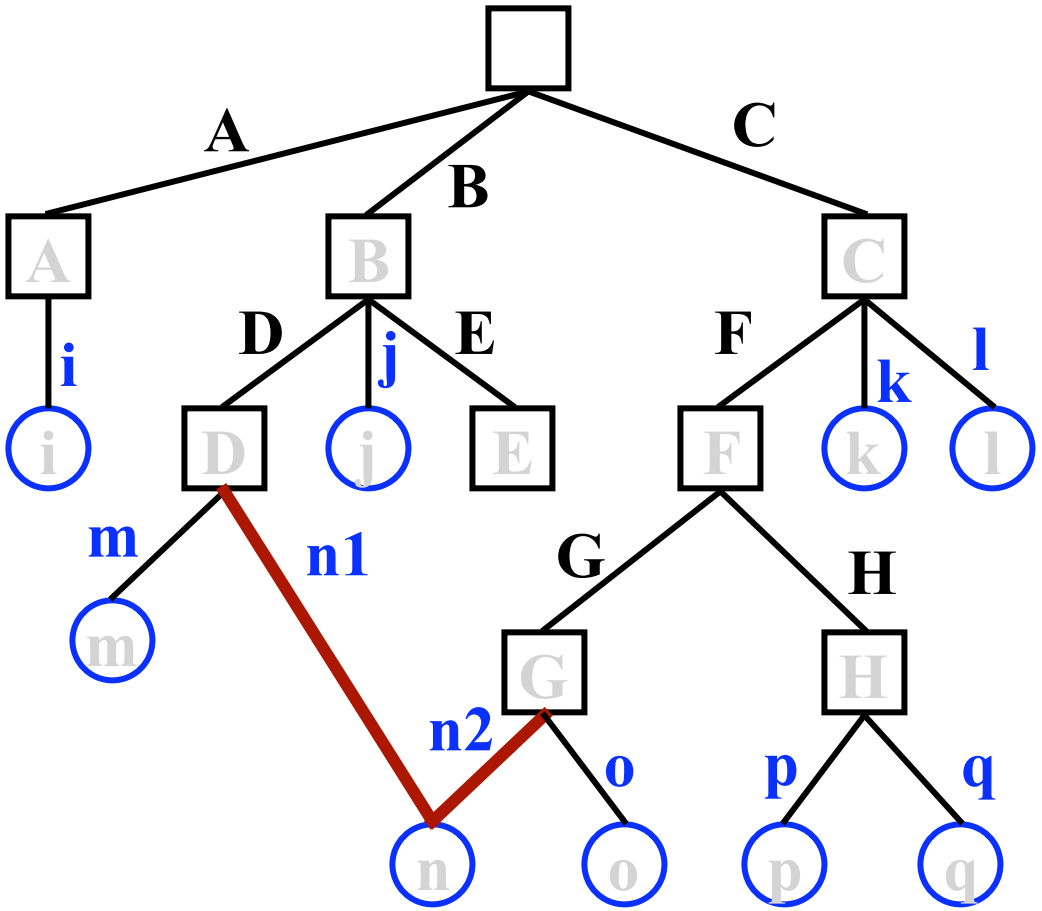


Hard Links

The name of a file is stored in the directory that points to the file.

Edges are labeled.

```
/B/D/n1
/C/F/G/n2
```

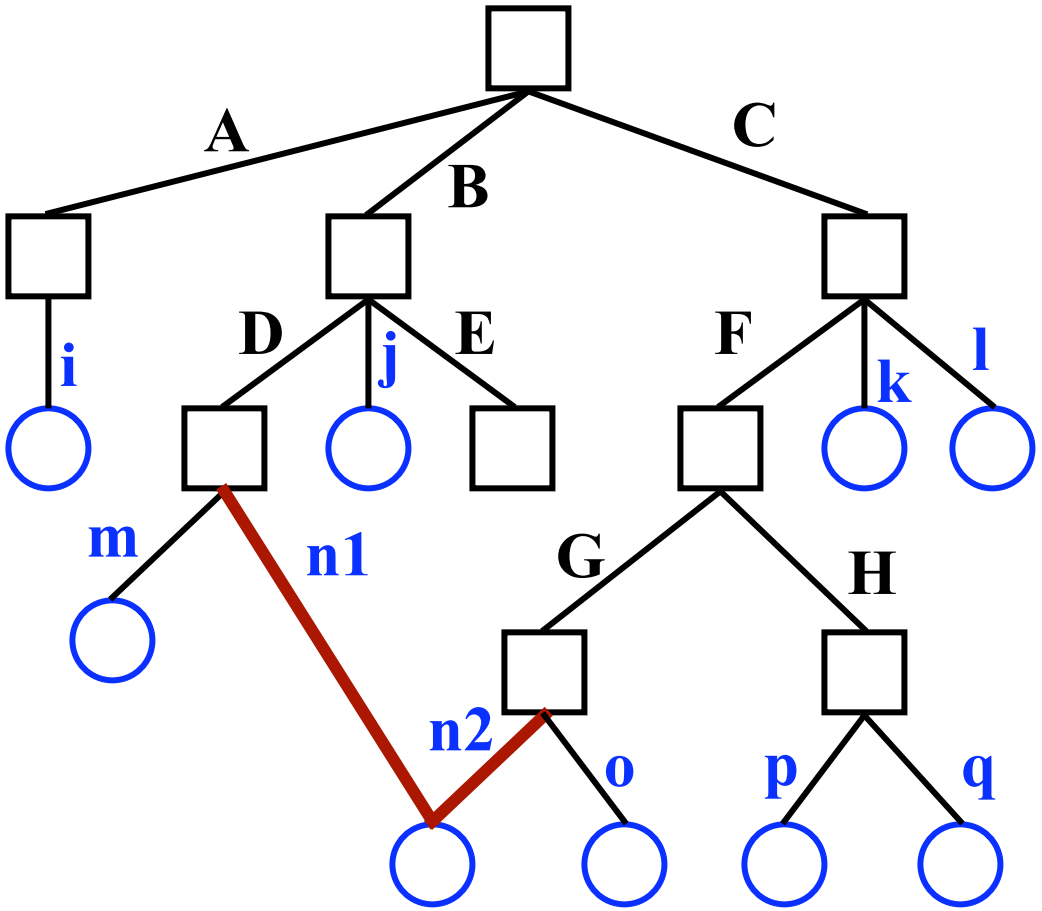


Hard Links

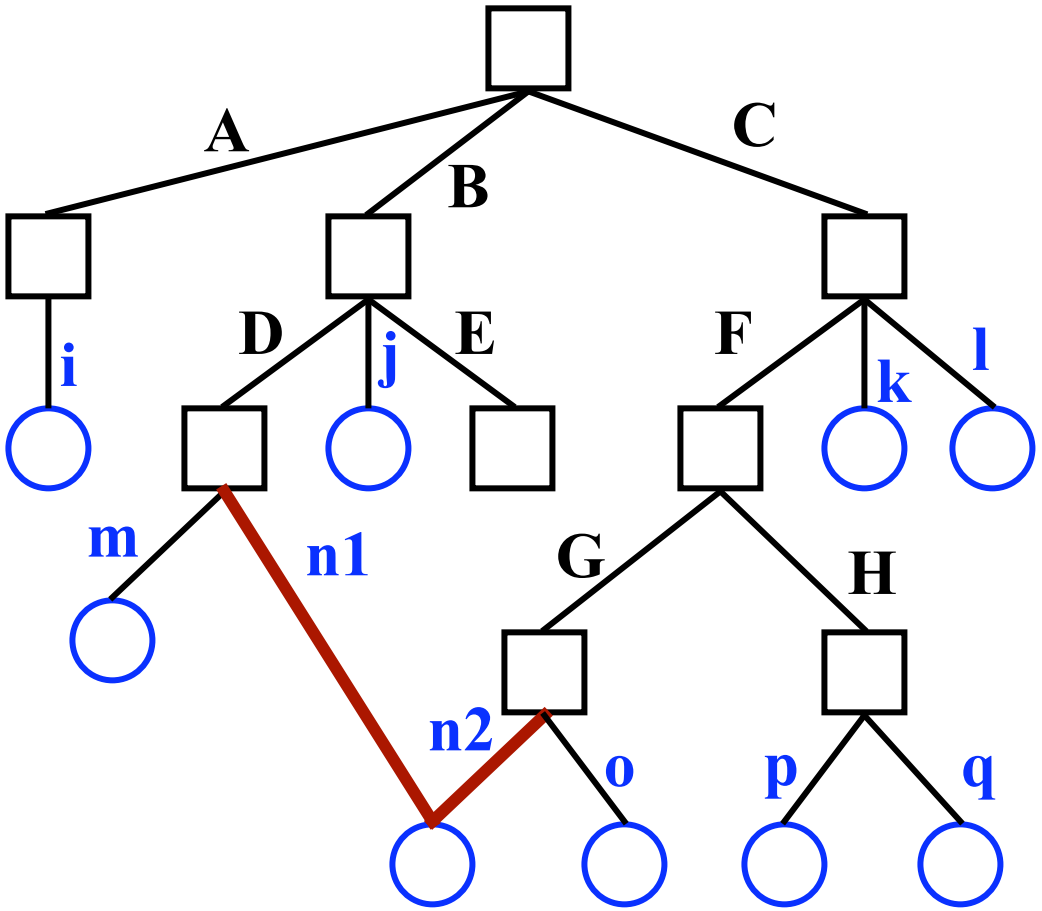
The name of a file is stored in the directory that points to the file.

Edges are labeled.

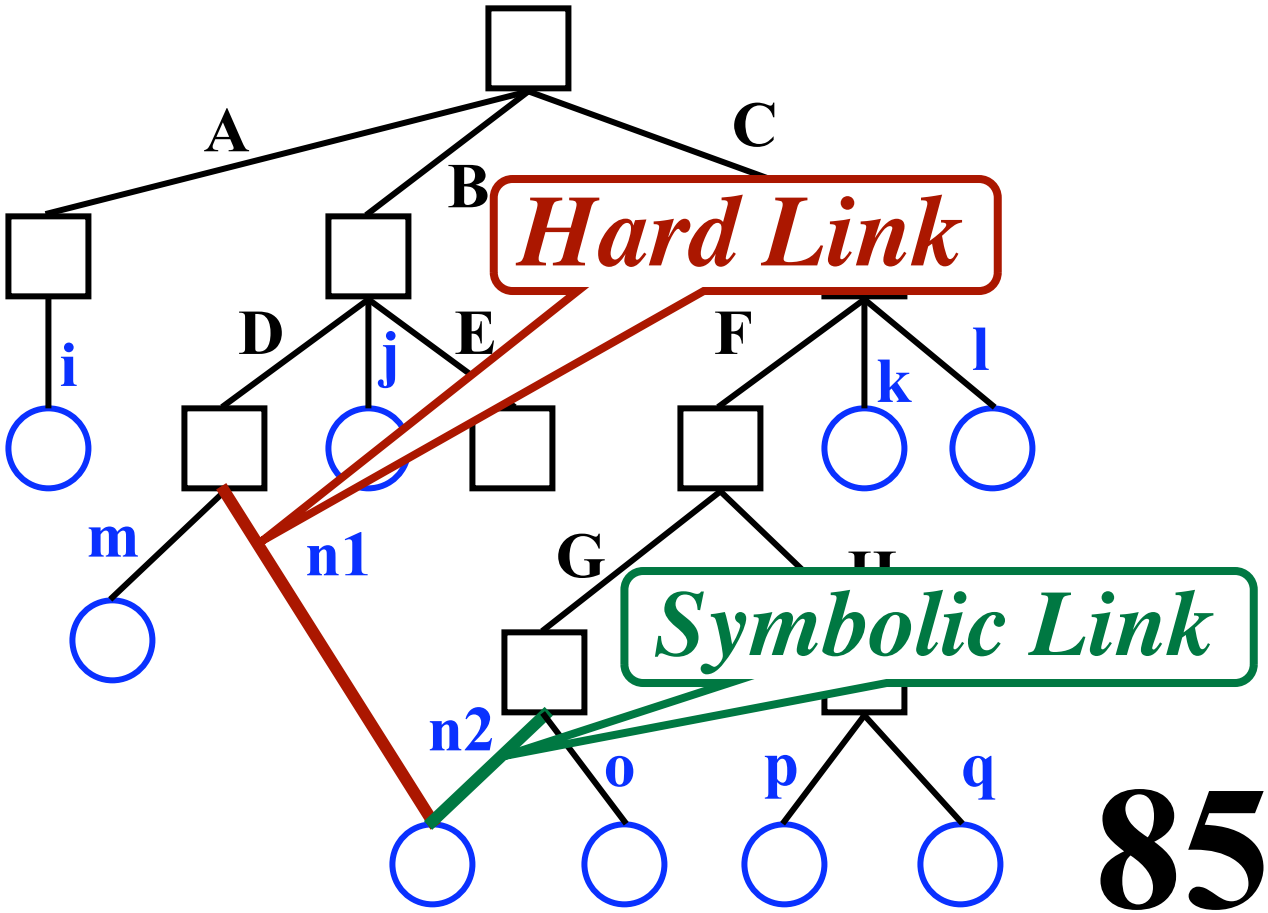
```
/B/D/n1
/C/F/G/n2
```



Symbolic Links



Symbolic Links

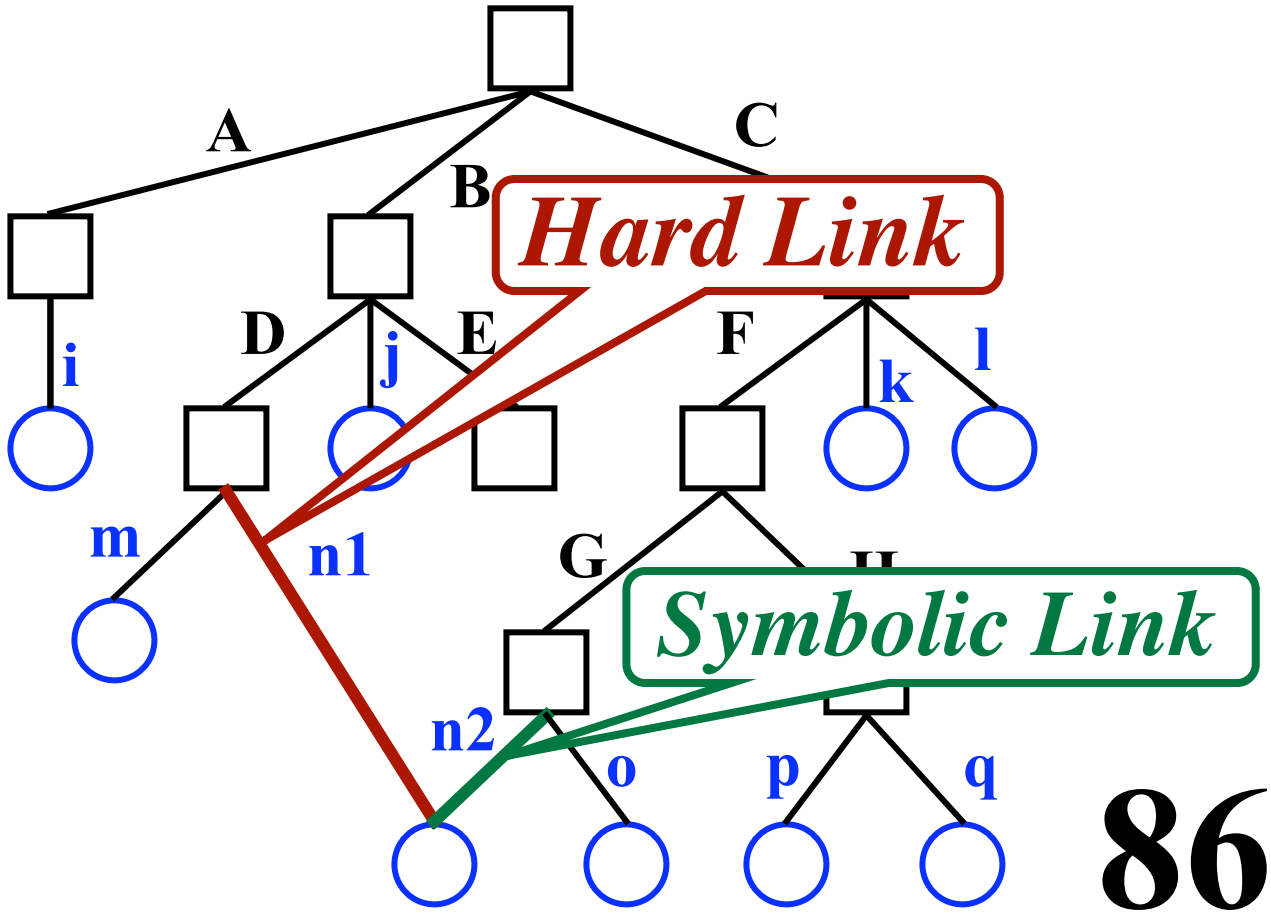


Symbolic Links

Assume i-node number of "n" is 45

Directory "D"

"m"	123
"n1"	45
⋮	⋮



Symbolic Links

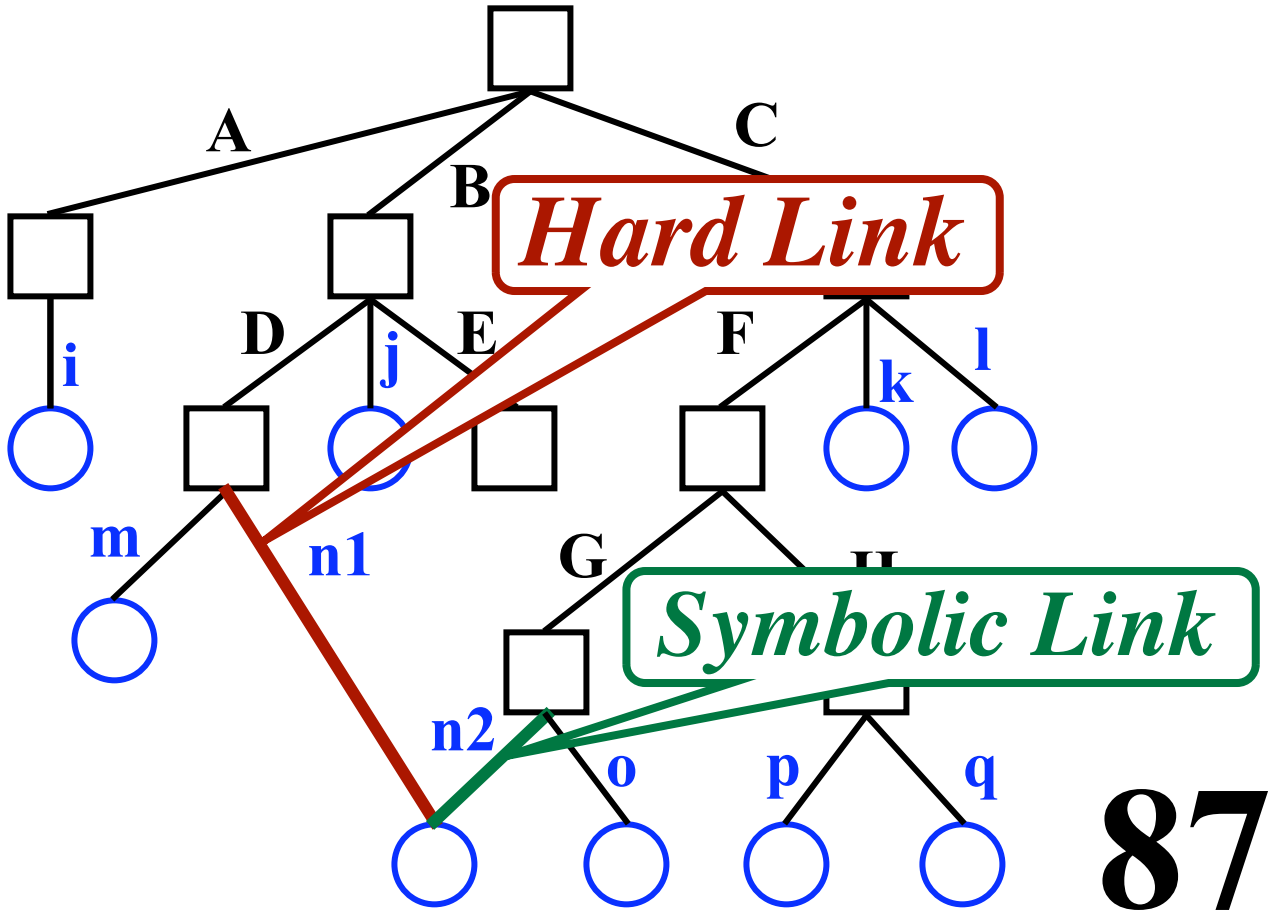
Assume i-node number of "n" is 45

Directory "D"

"m"	123
"n1"	45
⋮	⋮

Directory "G"

"n2"	/B/D/n1
"o"	87
⋮	⋮



Symbolic Links

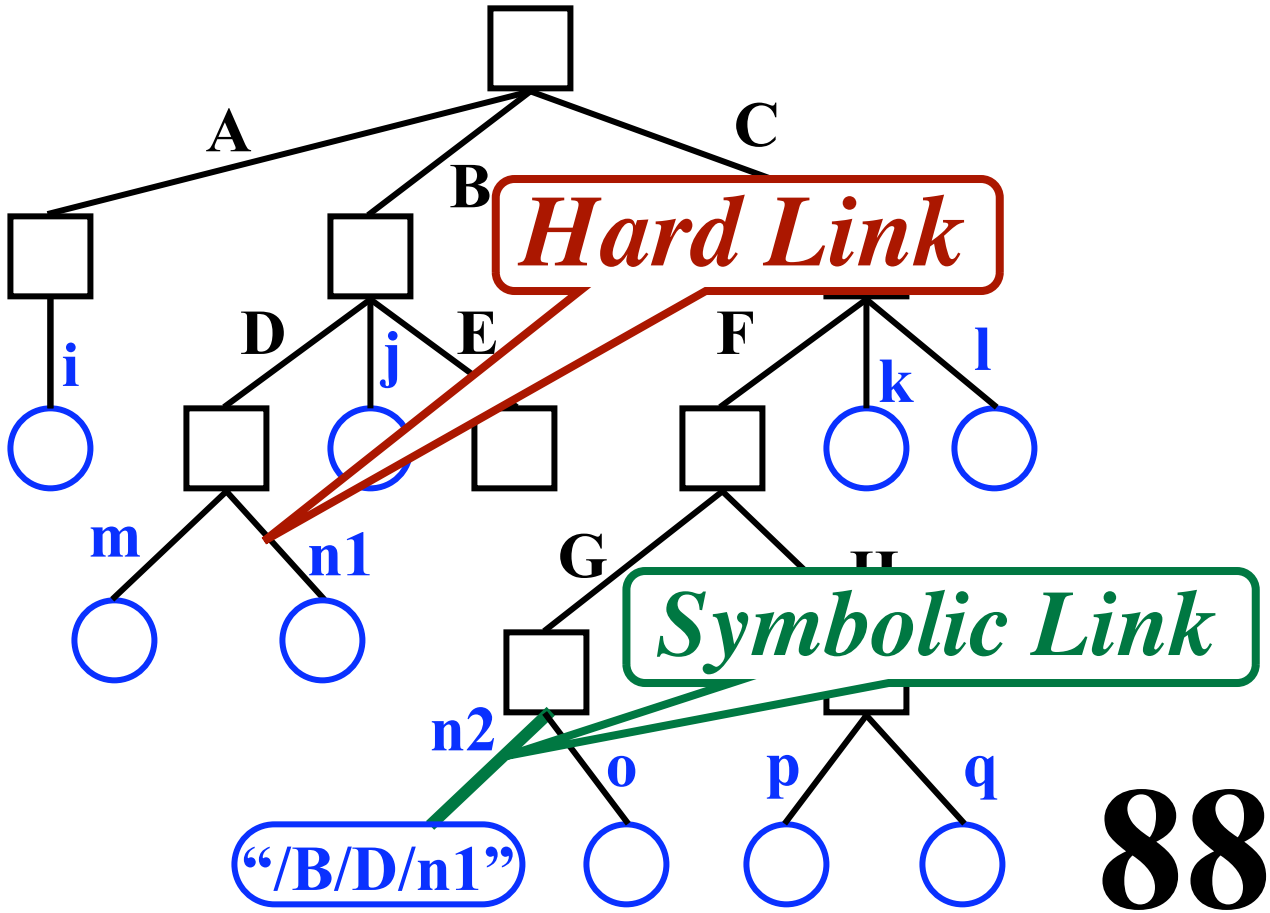
Assume i-node number of "n" is 45

Directory "D"

"m"	123
"n1"	45
⋮	⋮

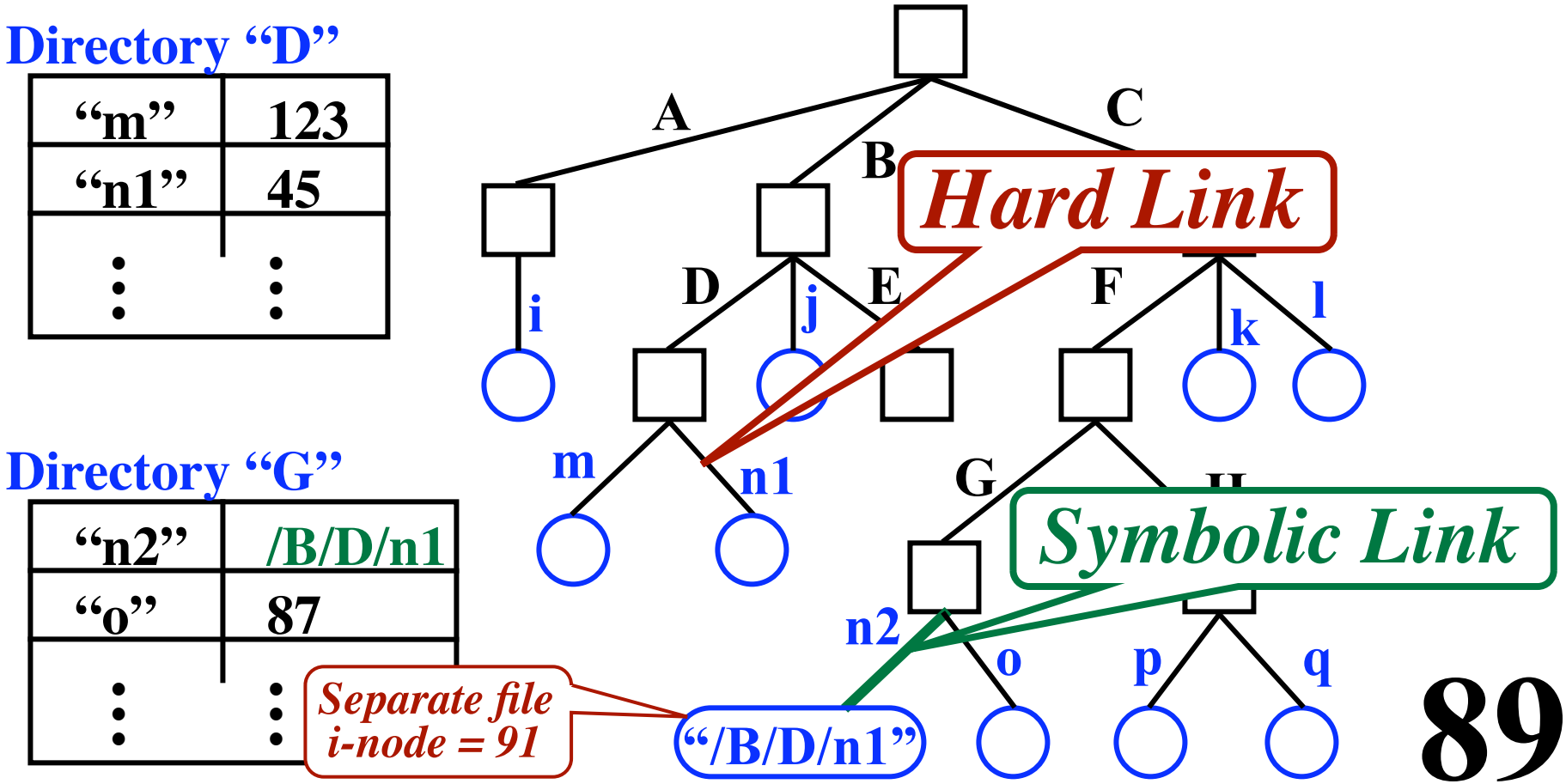
Directory "G"

"n2"	/B/D/n1
"o"	87
⋮	⋮



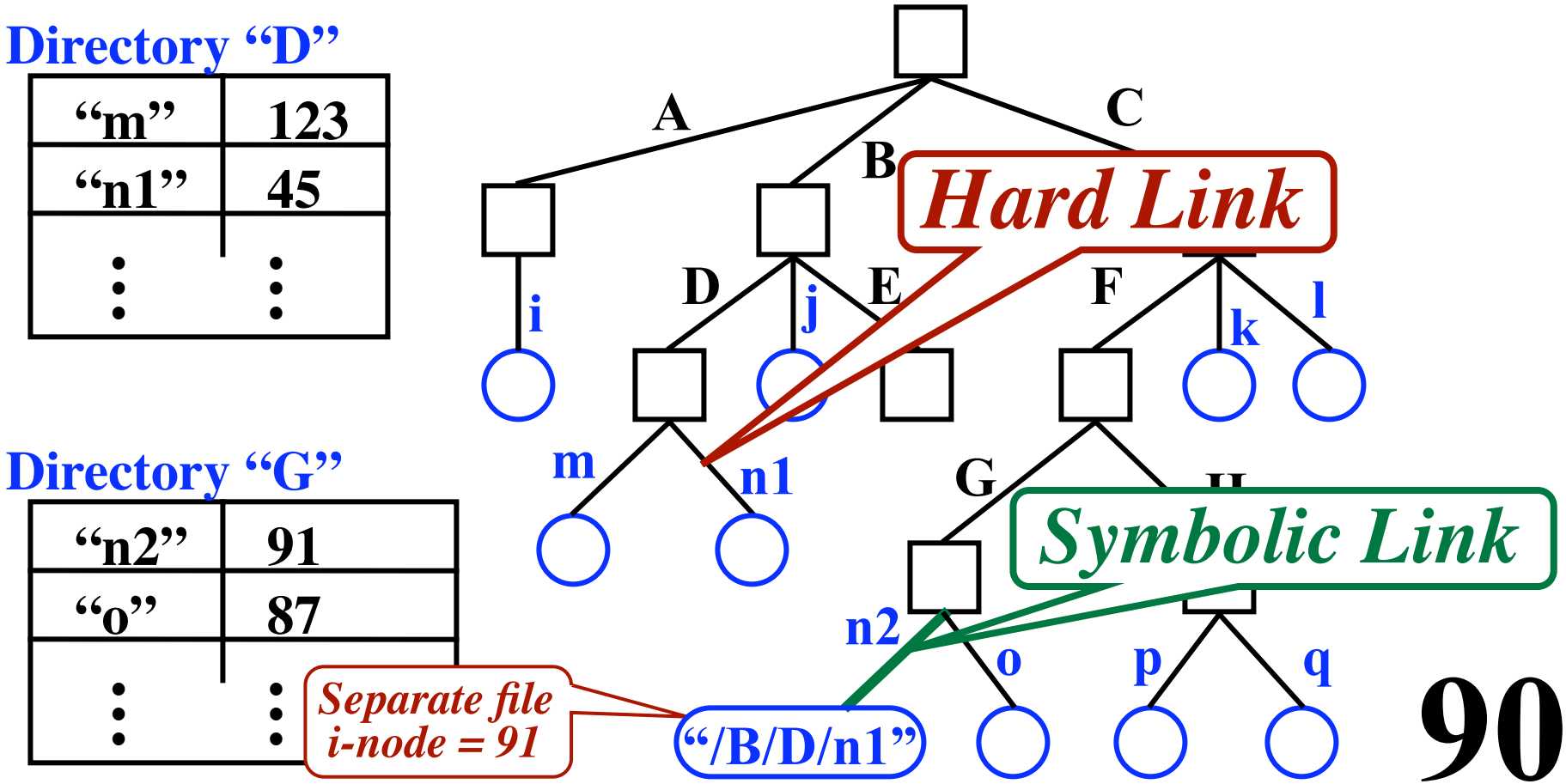
Symbolic Links

Assume i-node number of "n" is 45



Symbolic Links

Assume i-node number of "n" is 45



Deleting a File

Directory entry is removed from directory
All blocks in file are returned to free list

Deleting a File

Directory entry is removed from directory
All blocks in file are returned to free list

What about sharing???

- **Multiple links to one file (in Unix)**

Deleting a File

Directory entry is removed from directory
All blocks in file are returned to free list

What about sharing???

- **Multiple links to one file (in Unix)**

Hard Links

- **Put a "reference count" field in each i-node**
- **Counts number of directories that point to the file**
- **When removing file from directory, decrement count**
- **When count goes to zero, reclaim all blocks in the file**

Deleting a File

Directory entry is removed from directory
All blocks in file are returned to free list

What about sharing???

- **Multiple links to one file (in Unix)**

Hard Links

- **Put a "reference count" field in each i-node**
- **Counts number of directories that point to the file**
- **When removing file from directory, decrement count**
- **When count goes to zero, reclaim all blocks in the file**

Symbolic Link

- **Remove the real file... (normal file deletion)**
- **Symbolic link becomes "broken"**