Files and file systems

Module 5

Lecture

Operating systems 2019

1DT044, 1DT096 and 1DT003

February 2019 karl.marklund@it.uu.se Uppsala university

System and Application Programs

Operating System

Controls the **hardware** and **coordinates** its use among the various application programs for the various user.

Computer Hardware

System and Application Programs

GUI | batch | command line

user interfaces

system calls

program execution | I/O operations | communication

helpful for the user

error detection | file systems

resource allocation | accounting

ensuring the efficient operation of the system itself.

protection and security

Services

Operating System

Computer Hardware

Pyramid of **Menkaure**

Died ca. 2510 BC

Pyramid of **Khafre**

Died ca. 2570 BC

Pyramid of **Khufu**
(aka Cheops)

Died 2566 BC

Persistent data storage

In computer science, persistence refers to the characteristic of state that outlives the process that created it.

Without this capability, state would only exist in RAM, and would be lost when this RAM loses power, such as a computer shutdown.

Adjektivt

Persistent: ihärdig, beständig

Computers can store information on various persistent storage media.

Punched paper cards

Magnetic tapes

Floppy disks

Magnetic disks

Optical disks

USB flash drives

Operating System

The operating system abstracts from the physical properties of the storage devices to define a **logical storage** unit,

the **file**.

Persistent secondary data storage

Secondary storage sometimes called **auxiliary storage**, is all data storage that is not currently in a computer's primary storage (main memory) or memory. An additional synonym is **external storage**.

In a personal computer, secondary storage typically consists of storage on the hard disk and on any removable media, if present, such as a CD, DVD and USB drives.

Computer files

A **file** is a named collection of related information that is recorded on secondary storage.

A file is the **smallest unit of secondary storage** as seen by the user. A user cannot write data to secondary storage unless data are within a file.

Different **types of files** store different types of information. For example, program files store programs, whereas text files stores text, image files stores images.

In Unix, a **directory** is a file with information on how to find other files.

Abstract data type

An abstract data type (ADT) is defined by its behavior (semantics) from the point of view of a user of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations.

This contrasts with data structures, which are concrete representations of data, and are the point of view of an implementer, not a user.

Source: https://en.wikipedia.org/wiki/Abstract_data_type 2018-02-24

File operations

The file is an **abstract data type.**

What operations should be supported by this abstract data type?

File operations

The file is an **abstract data type.** The operating system must provide at least six operations on files.

- **Create**
- **Write**
- **Read**
- **Reposition** within file
- **Delete**
- **Truncate** - erase the content of a file but keep its attributes.

The six basic operations listed above can be combined to perform other operations, for example append.

Directory

A directory is a file system **cataloging structure** which contains references to other computer files, and possibly other directories.

Source: https://en.wikipedia.org/wiki/Directory_(computing) 2018-02-24

Open and close

Most of the file operations involve searching the directory for the entry associated with the named file.

To avoid the constant searching, many systems require that an open system call be made before a file is first used actively.

> `open(f)` – search the directory structure on disk for entry `f`, and move the content of entry to memory.

When a file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table.

> `close(f)` – move the content of entry `f` in memory to directory structure on disk.

File descriptors and open-file table

The operating system keeps a small table, called the open-file table, containing information about all open files.

- When a file operation is requested, the **file** is **specified via** a file descriptor (an **index**) into this table, so **no searching** is required.

- When a file is no longer being actively used, it is **closed** by the process, and the operating system **removes** its **entry** from the open-file table.

A Process

```
int fd;
char buff[BUF_SIZE];
fd = open(”foo.txt”, O_RDONLY);
```

User Space

Kernel Space

Descriptors	
0	stdin
1	stdout
2	stderr
3	?

What information is needed to access a file?

Persistent secondary data storage

File ”foo.txt”

A Process

```
int fd;
char buff[BUF_SIZE];
fd = open(”foo.txt”, O_RDONLY);
read(fd, buf, BUF_SIZE);
close(fd);
```

User Space

Kernel Space

Descriptors	
0	stdin
1	stdout
2	stderr
3	

Persistent secondary data storage

File ”foo.txt”

Open files

Several pieces of data are needed to manage open files.

File pointer

- Pointer to last read/write location, per process that has the file open.

File-open count

- Counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it.

Disk location

- Location of the file on disk: cache of data access information.

Access rights

- Per-process access mode information.

File access methods

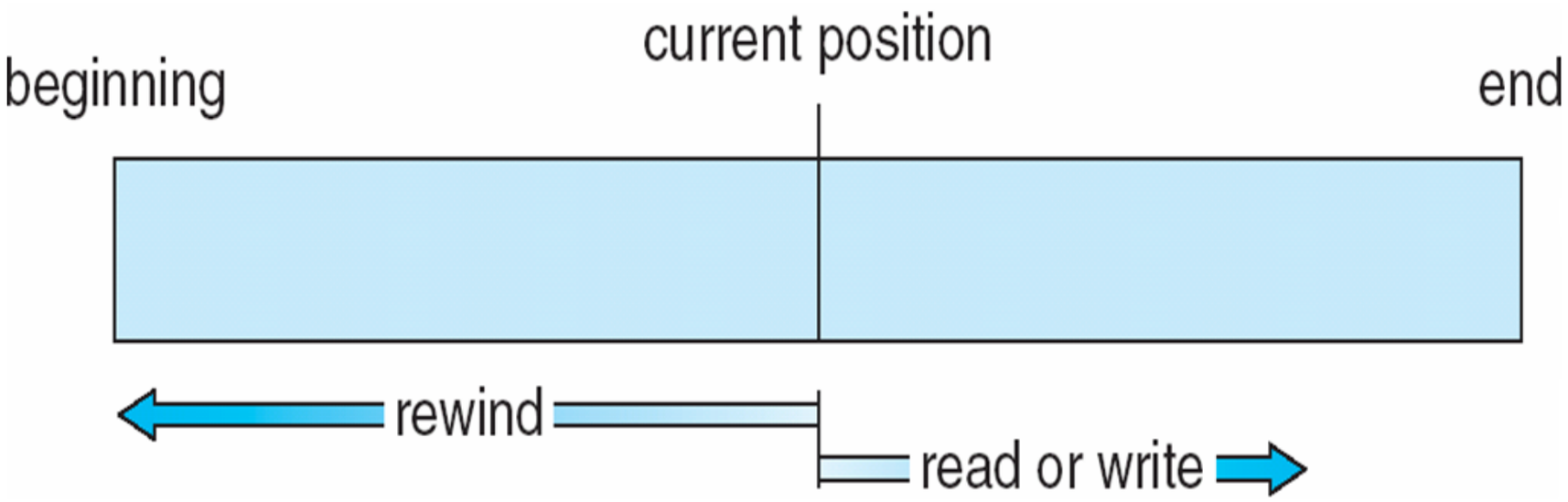
The information in a file can be accessed in several ways.

Sequential access

Information in the file is processed in order, one record after other.

Sequential access

Information in the file is processed in order, one record after other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.



- A read operation (**read next**) reads the next portion of the file and automatically advances the file pointer.
- A write operation (**write next**) appends to the end of a file and advances to the end of the newly written material (the new end of the file).
- A rewind operation **reset** the file to the beginning.

Block data storage

A file is made up of fixed length logical records (blocks)

Block data storage

In computing (specifically data transmission and data storage), a block, sometimes called a physical record, is a **sequence of bytes** or bits, having a **fixed length** (a block size).

- Blocking is used to facilitate the handling of the data-stream by the computer program receiving the data.
- Blocked data is normally read a whole block at a time.

Blocking is almost universally employed when storing data to 9-track magnetic tape, to rotating media such as floppy disks, hard disks, optical discs and to NAND flash memory.

Source: http://en.wikipedia.org/wiki/Block_(data_storage) 2014-03-19

Image: http://honeysucklecreek.net/people/bryan_sullivan_files.html 2014-03-19

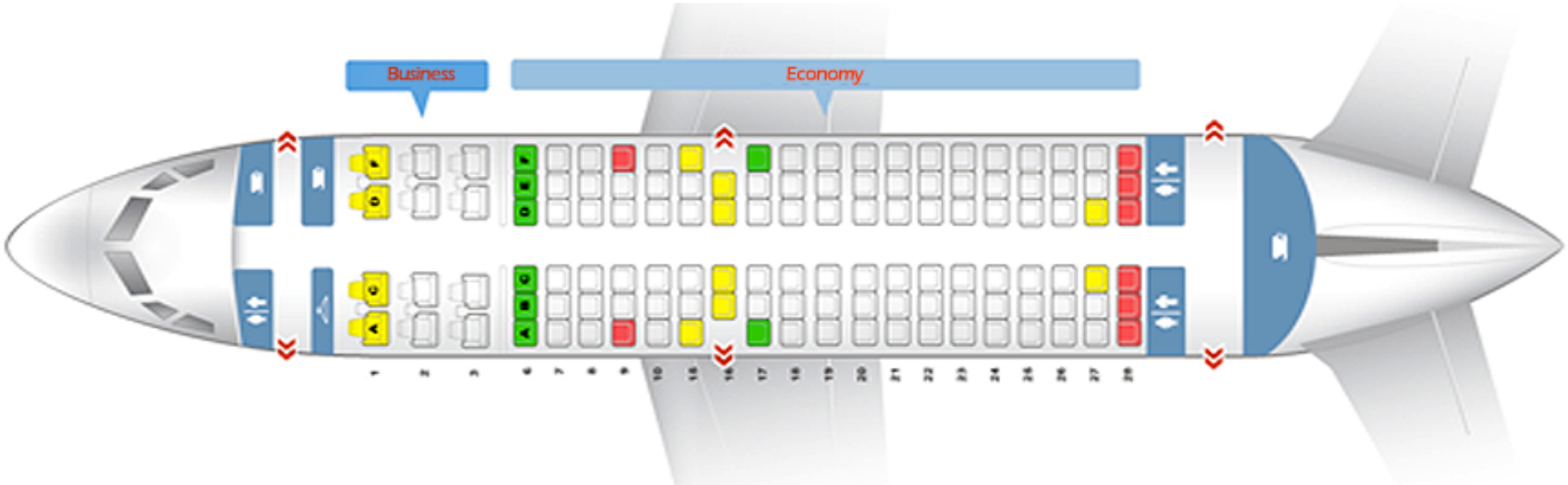
Direct access

A file is made up of fixed length logical records (blocks) with no restriction on the order of access to these blocks.

- The direct access method is based on a disk model of a file, since disks allows **random access** to any file block.
- A file is viewed as a numbered sequence of blocks or records. Thus we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.
- Direct access files are of great use for immediate access to large amounts of information. Databases are often of this type.

Direct access example

A simple airline-reservation system (data base).



The number of available seats for a flight is stored in the disk block identified by the flight number.

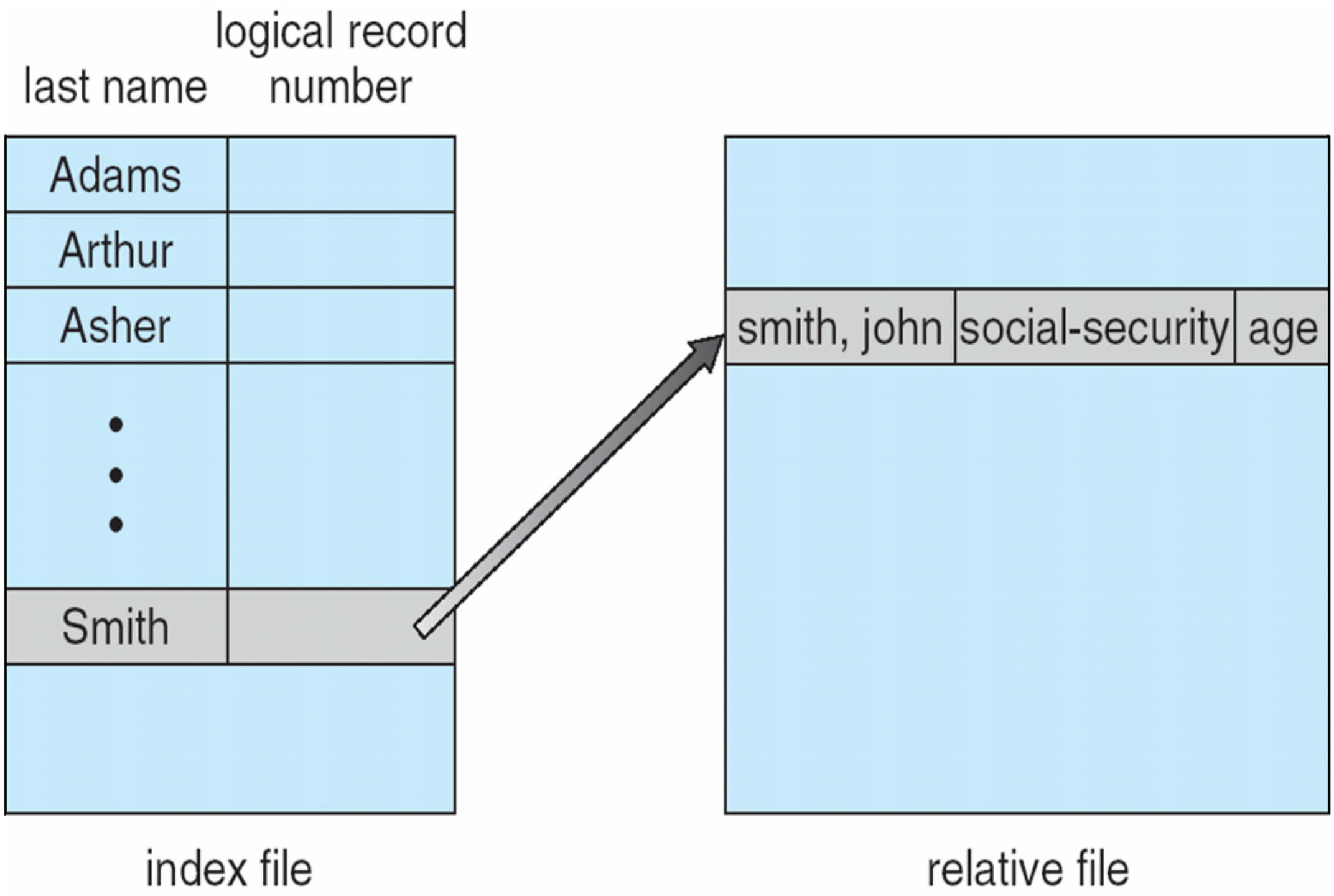
Direct access - hash based lookup

To store information about a larger set, such as people, we might compute a hash (of the name) and use this to find a block for further search.

Index files

Other access methods can be built on top of a direct-access method. One example is to construct an index for the file.

- The index contain pointers to the various blocks.
- To find a record in the file, we first search the index and then use the pointer to access the file directly.


logical record
last name
number
Adams
Arthur
Asher
Smith
smith, john
social-security
age
index file
relative file

File control block

A File Control Block (FCB) is a file system structure in which the state of an open file is maintained.

A typical file control block (FCB)

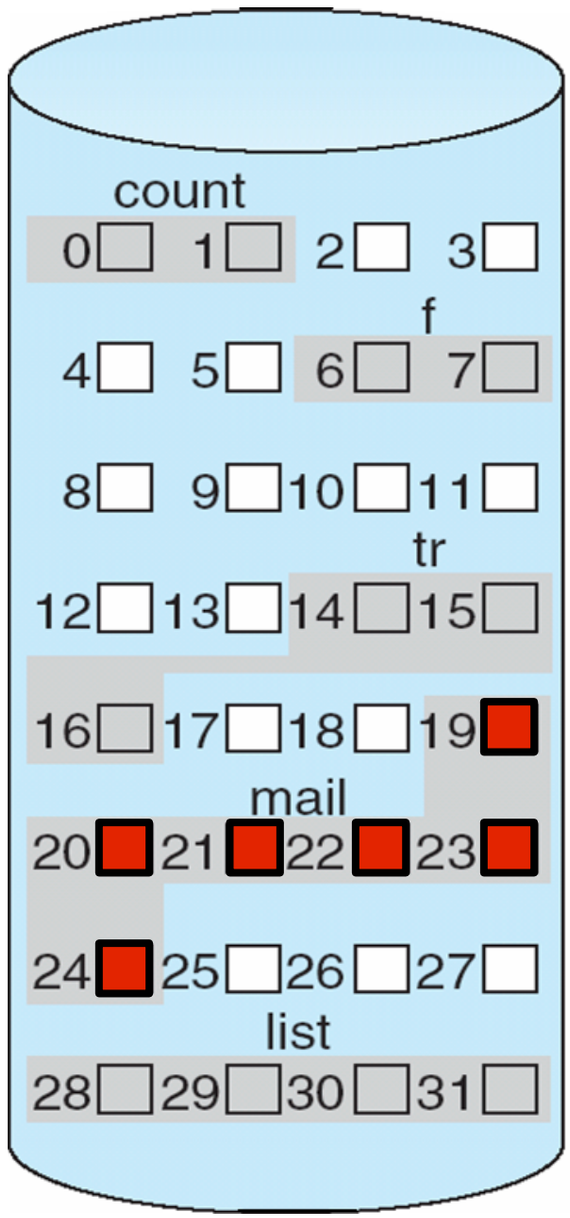
file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Disk allocation

How can blocks on disk be allocated to files?

Contiguous allocation

Each file occupies a set of contiguous blocks on the disk.

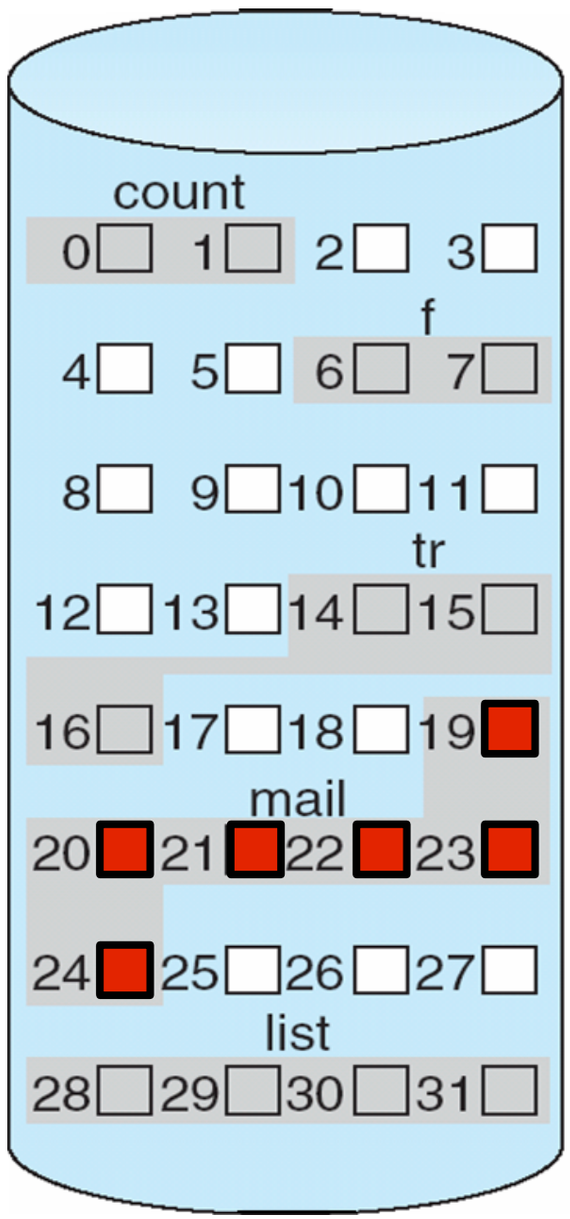


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

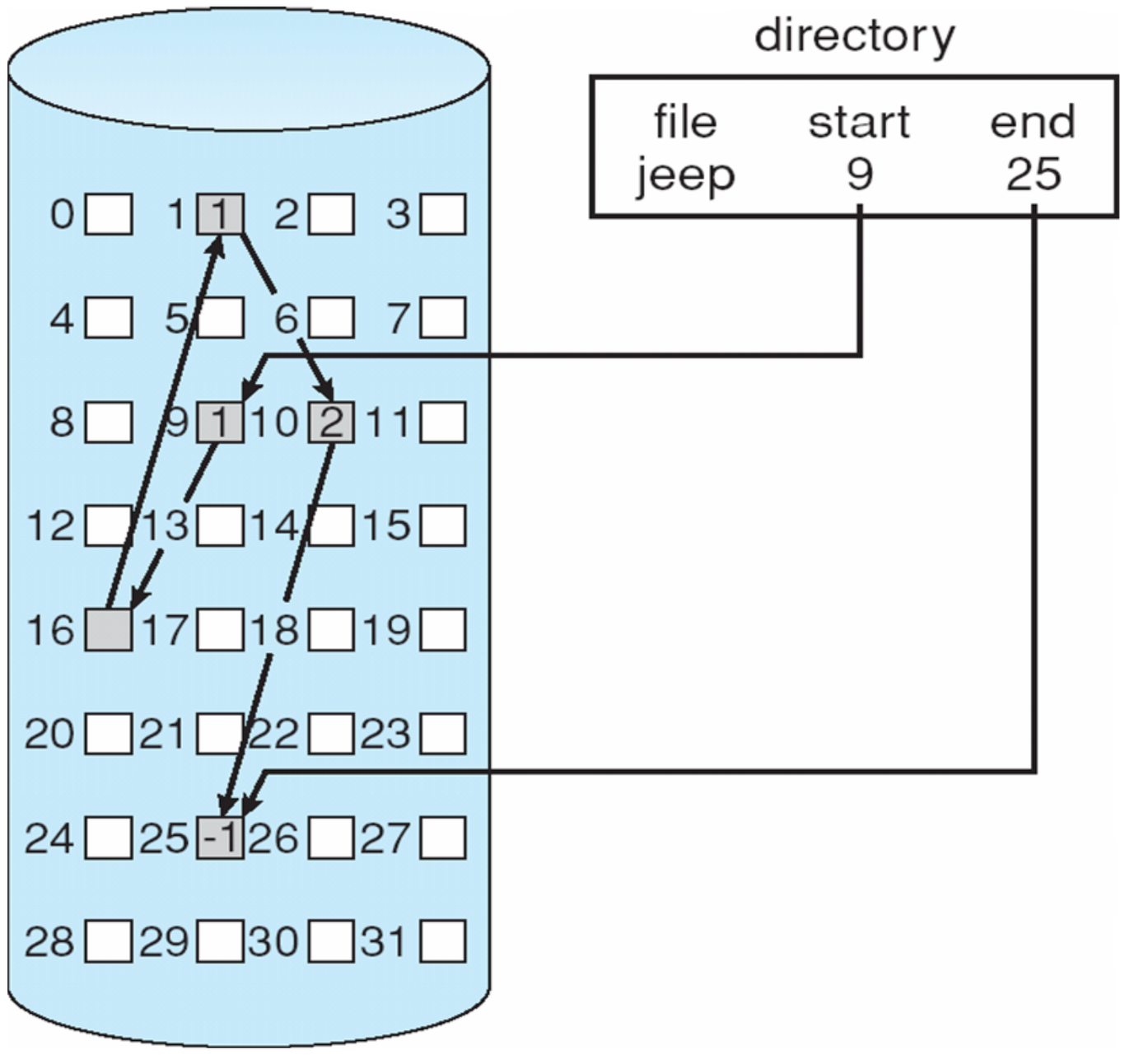


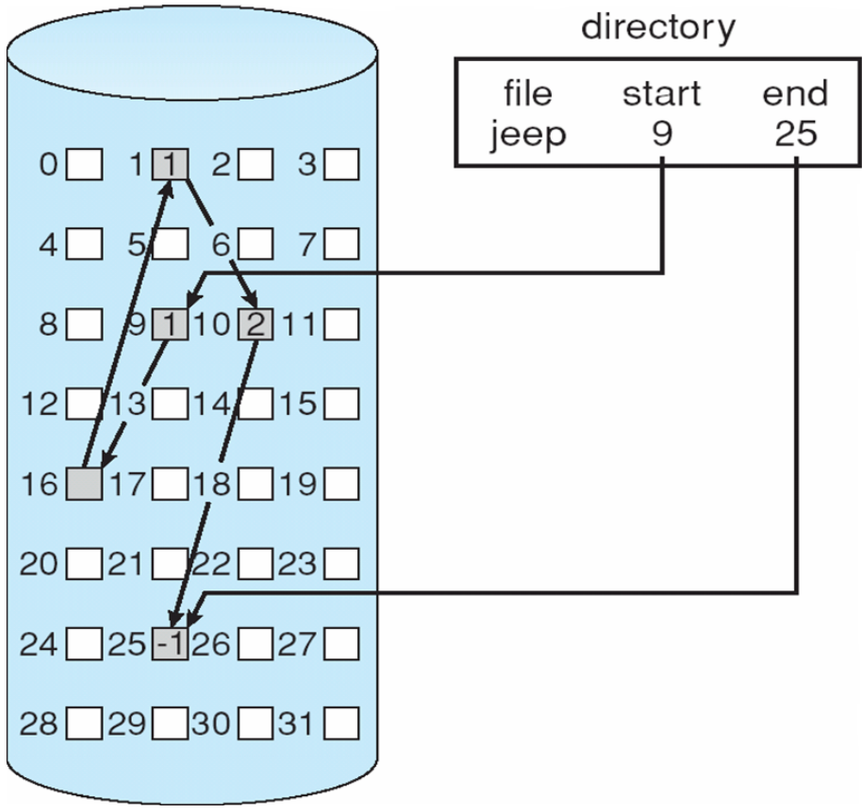
- **Simple** – only starting location (block #) and length (number of blocks) are required
- **Sequential access** - easy!
- **Random access** - easy! (start + offset)
- **Wasteful** of space (dynamic storage-allocation problem) - **external fragmentation**
- Files cannot grow

Alternative?

Linked allocation

Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.





- **Simple** – need only starting address
- Free-space management system – **no waste** of space - **no external fragmentation**.
- **No random access**

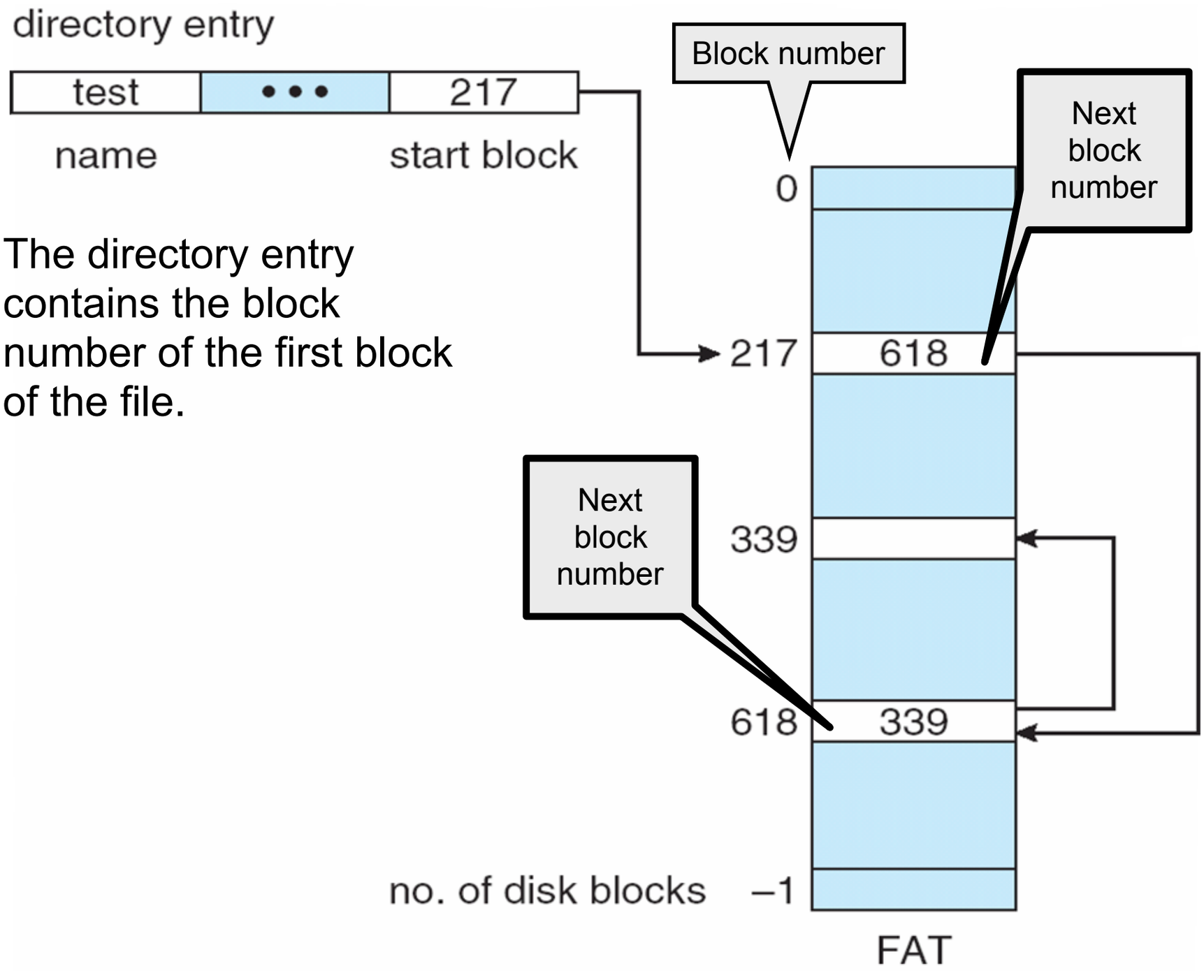
Alternative?

FAT

File Allocation Table (FAT) is a computer file system architecture and a family of industry-standard file systems utilizing it. The FAT file system is a legacy file system which is simple and robust.

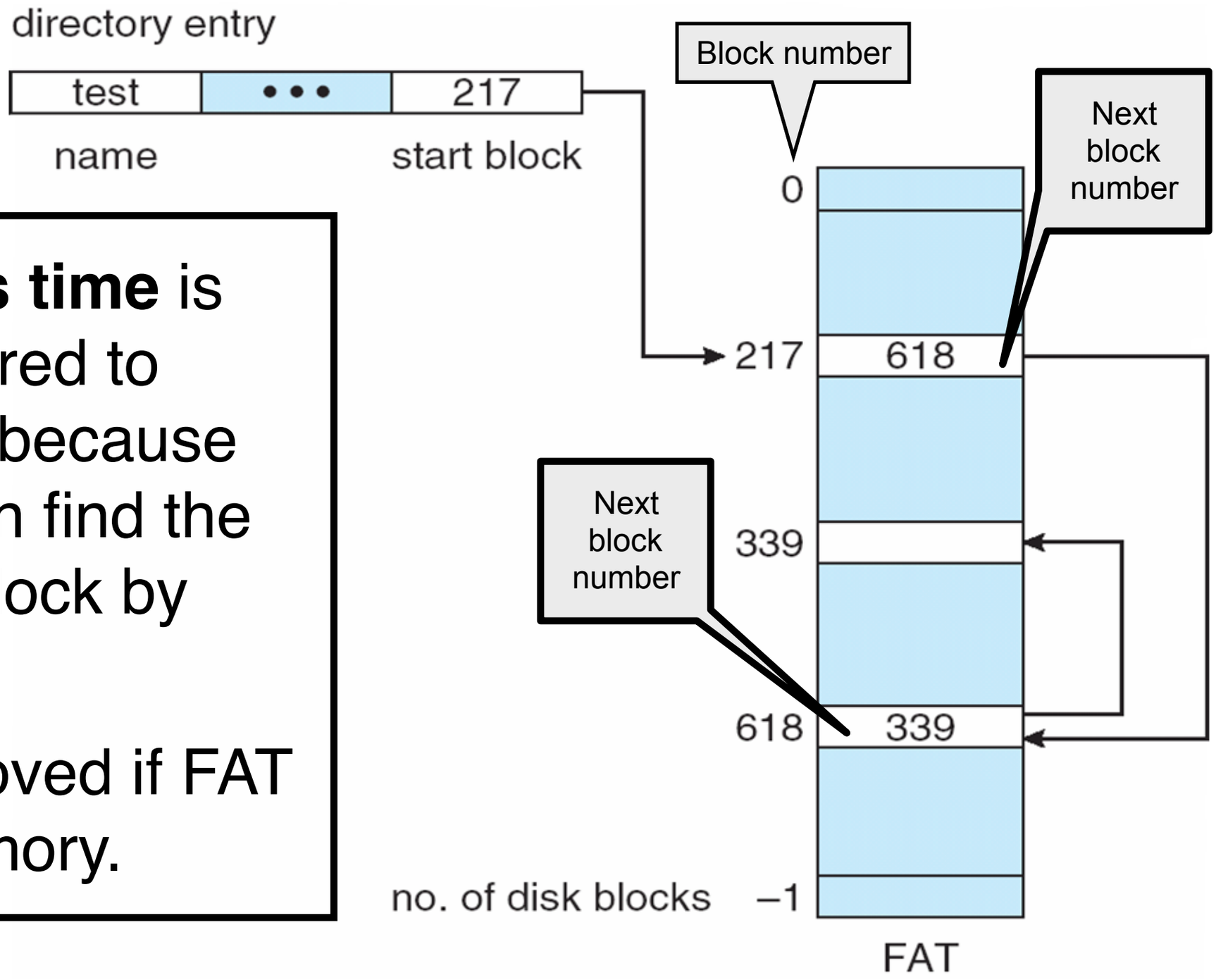
Source: https://en.wikipedia.org/wiki/File_Allocation_Table 2018-02-23

FAT is a **variation on linked allocation**. The file allocation table has one entry for each disk block and is indexed by block number.



The directory entry contains the block number of the first block of the file.

The FAT is located in contiguous space on disk.



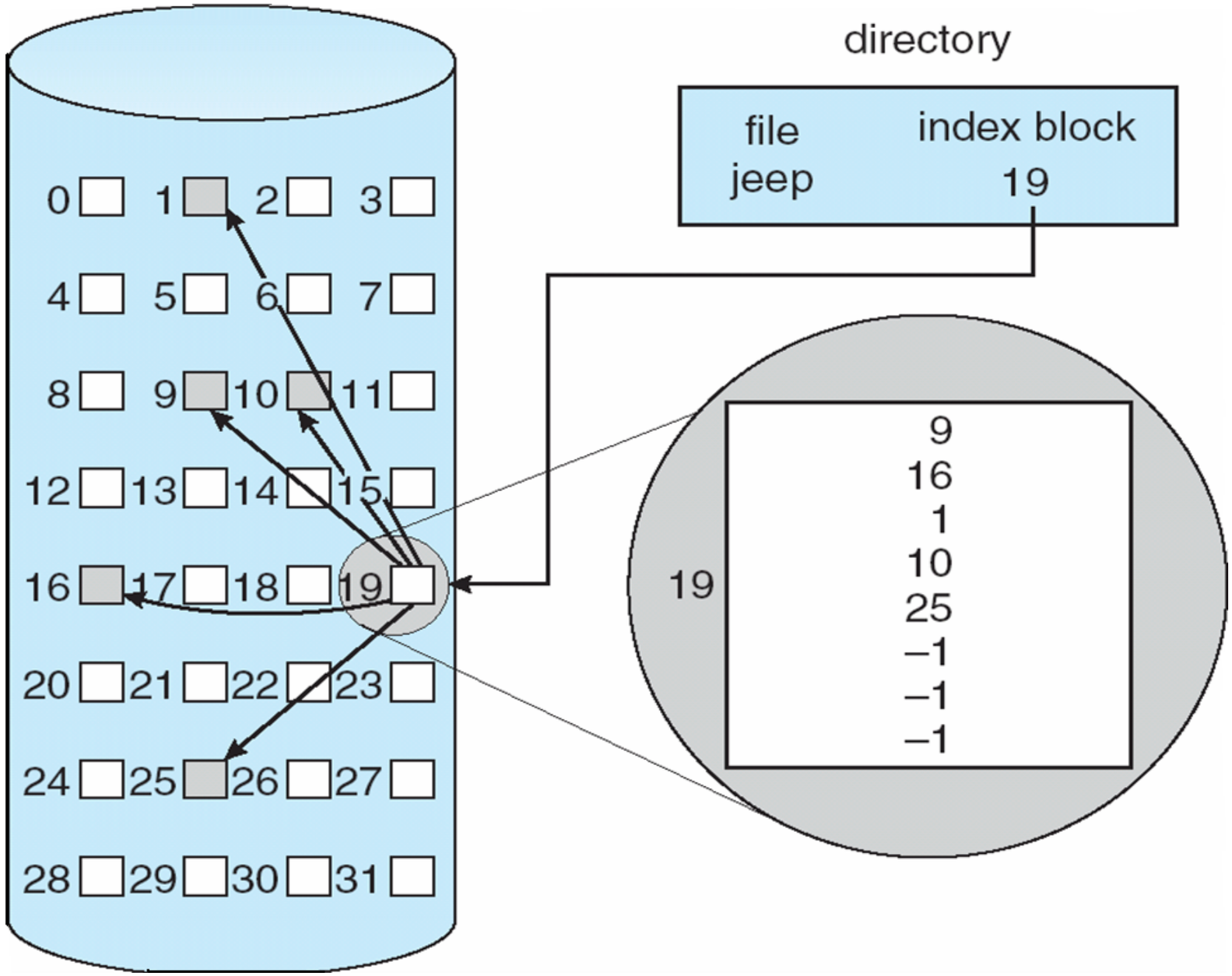
Random access time is improved compared to linked allocation because the disk head can find the location of any block by reading the FAT.

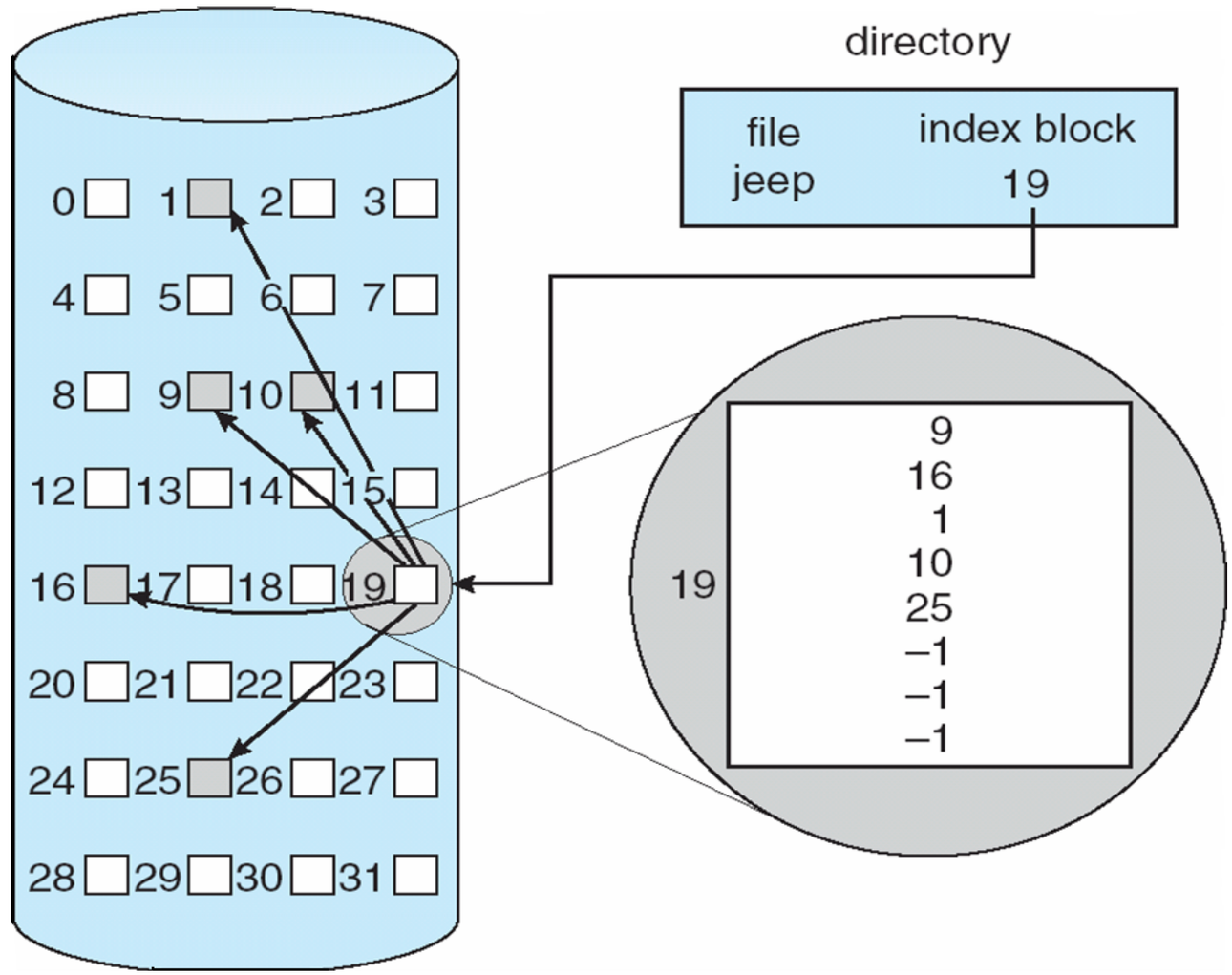
Even more improved if FAT is cached in memory.

Allocating a new block to a file becomes simple, search the FAT for an unused block.

Indexed allocation

Indexed allocation brings all pointers together into an index block.

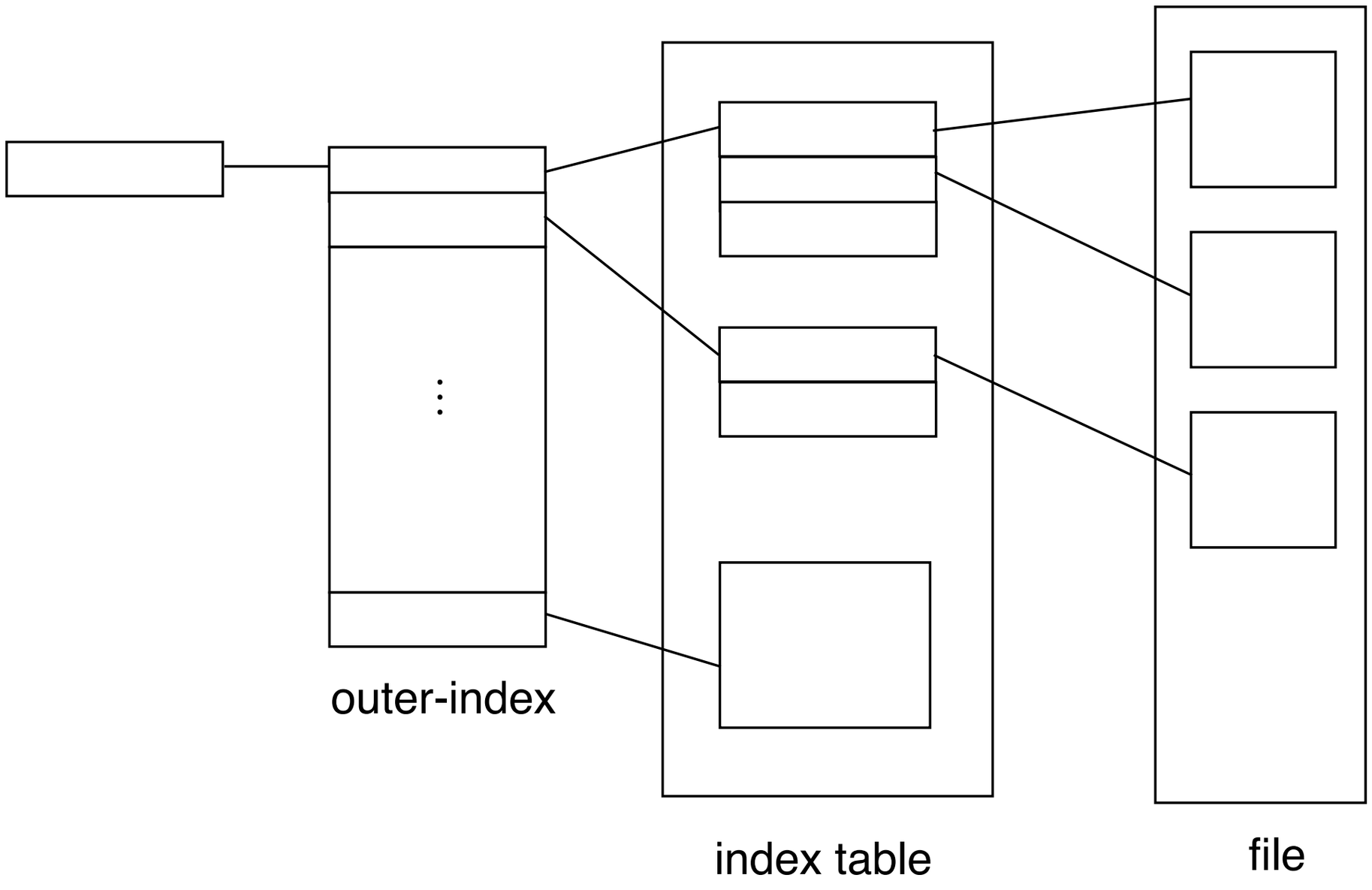




- Free-space management system – **no waste** of space - **no external fragmentation**
- **Random access**
- **Limited file size**

Two-level index

The size of the index block puts a limit on the file size. An alternative is to use a multi level index.



iNode

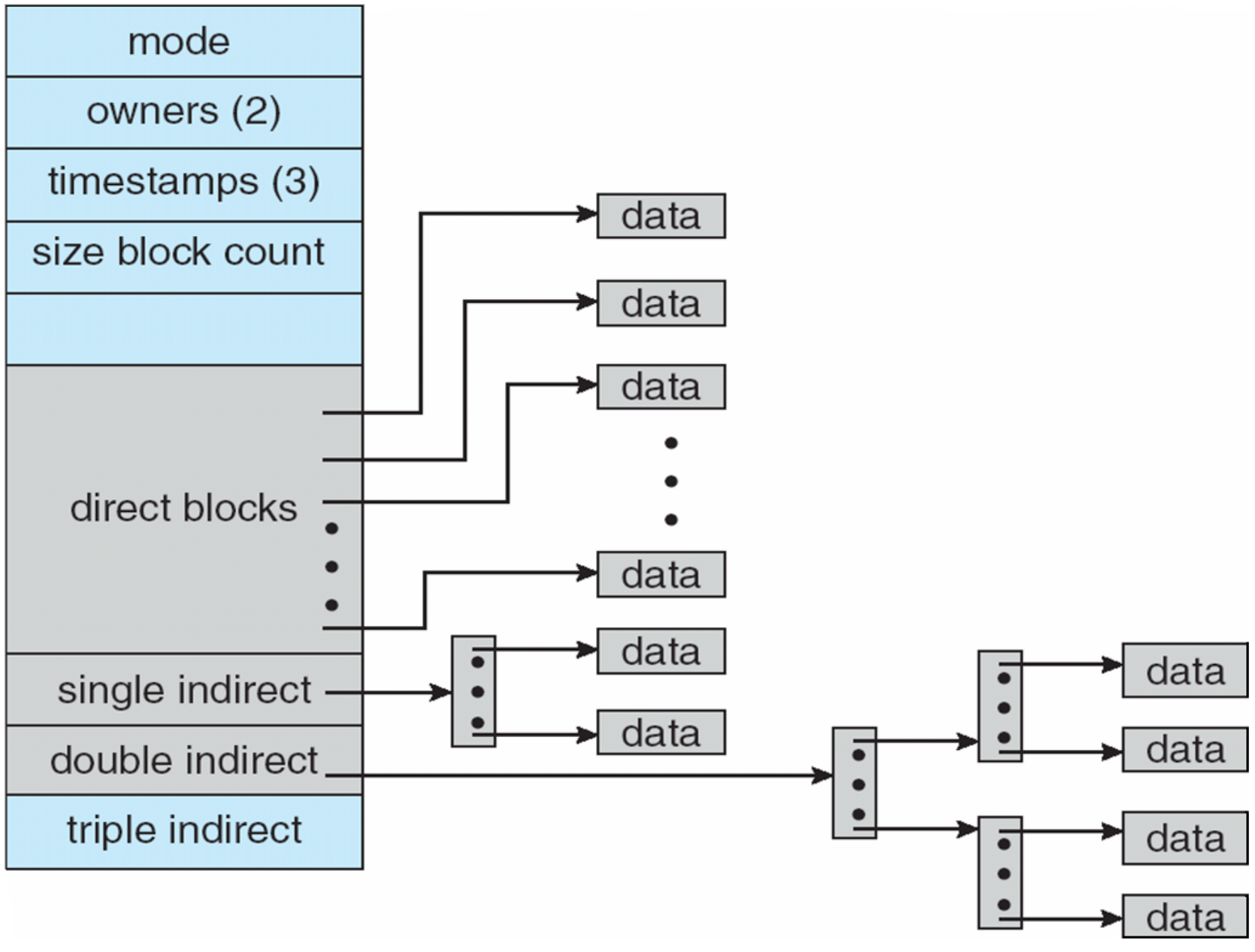
In Unix-style file systems the file control block is called iNode.

The inode is a data structure that describes a **filesystem object** such as a **file** or a **directory**.

Source: https://en.wikipedia.org/wiki/Inode 2018-02-23

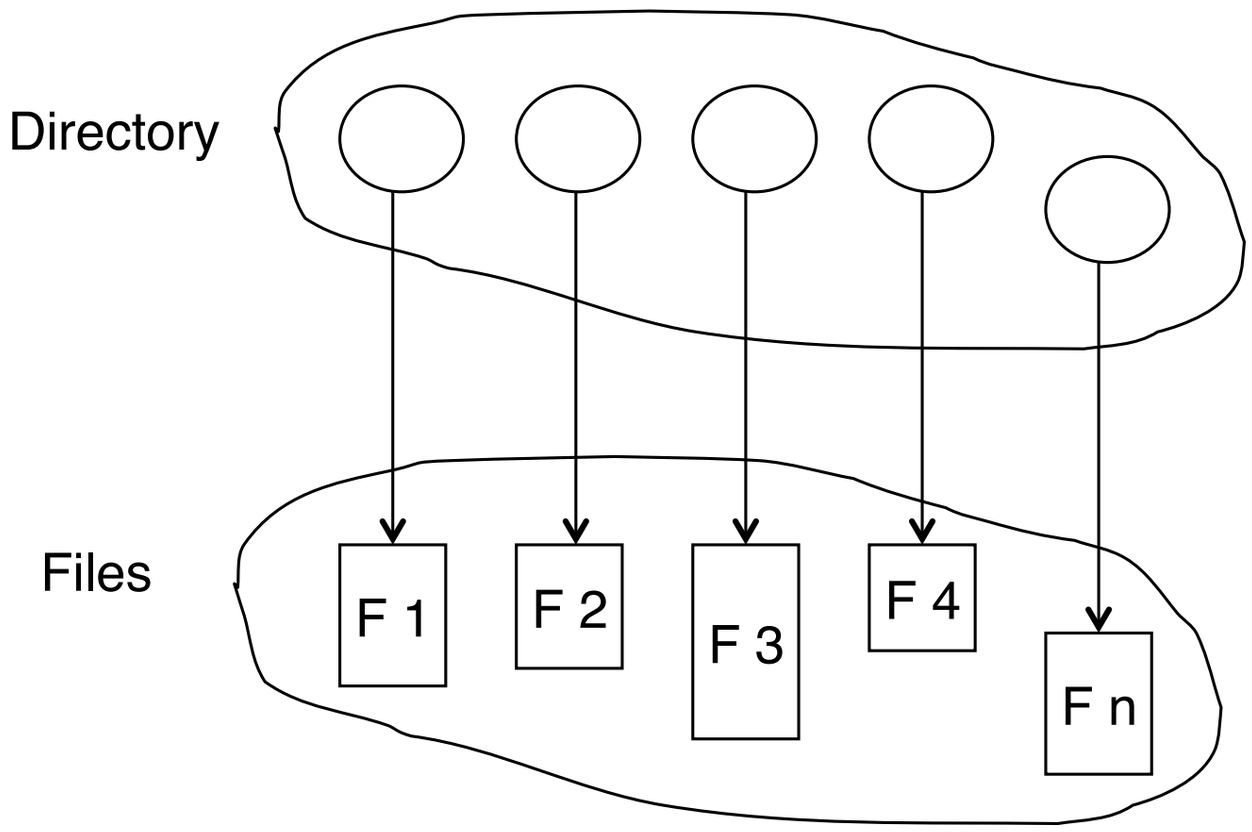
The Unix inode

Unix uses a combination of direct index blocks and indirect index blocks.



Directory structure

A directory is a collection of nodes containing information about all files (and other directories) in the directory.



Both the directory structure and the files reside on disk.

Backups of these two structures are kept on tapes.

In memory file system structures

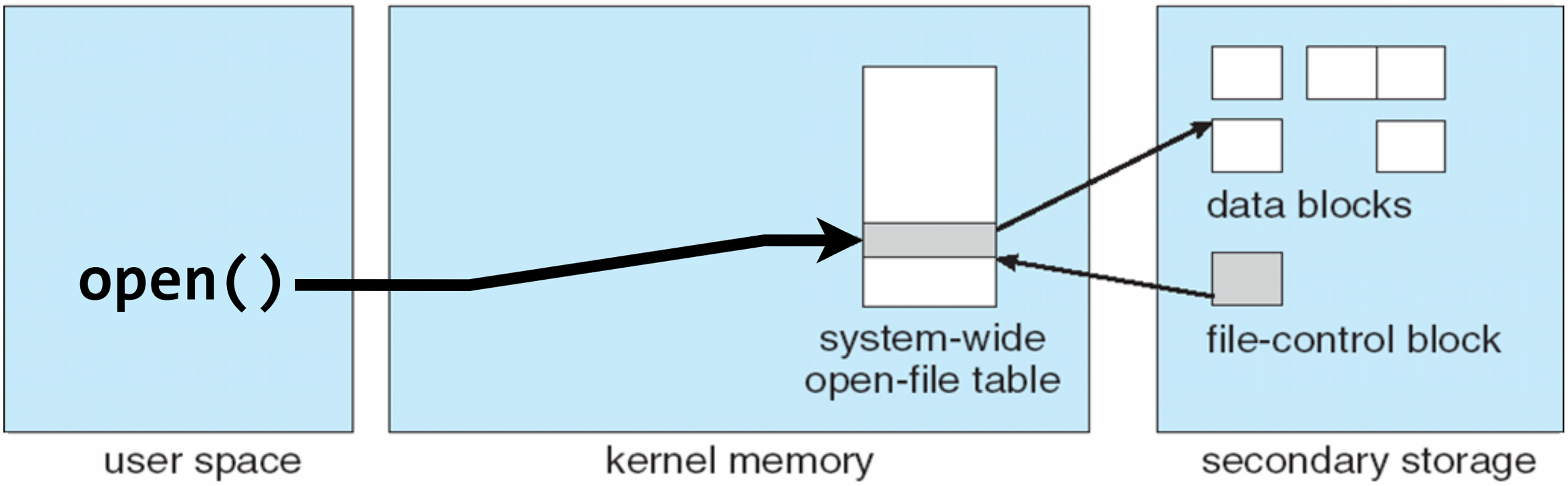
In memory file system structures

- directory structure
- system-wide open-file table
- per-process open-file

The **open()** system call first searches the **system-wide open-file table** to see if the file is already in use by another process.

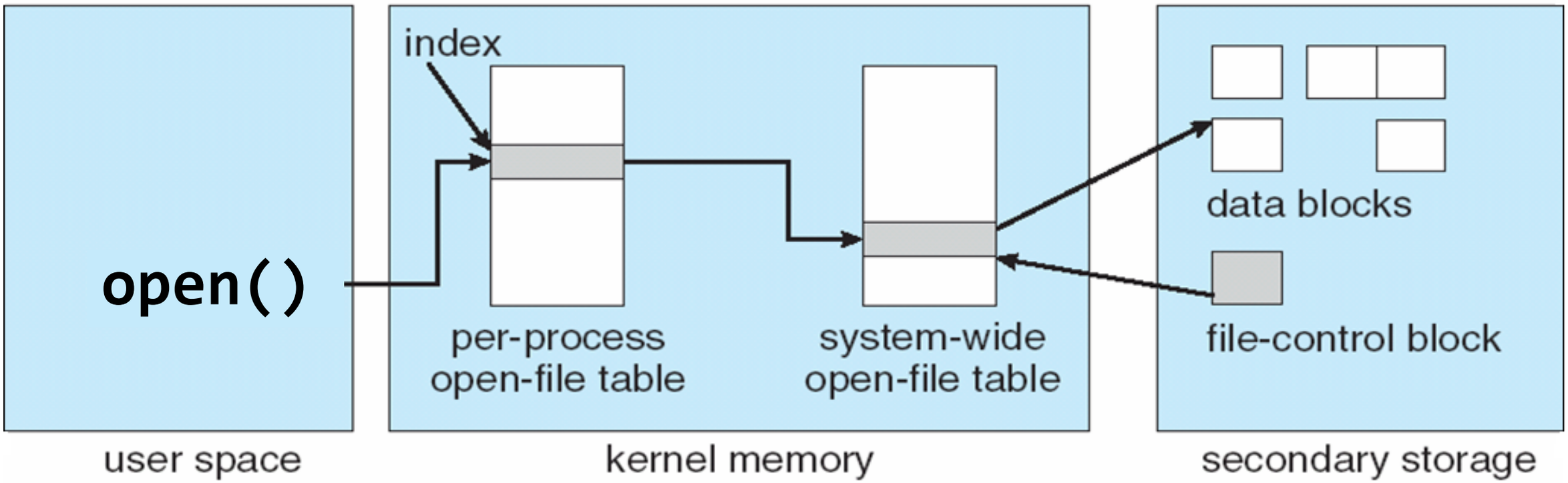
- **If it is**, a **per-process open-file** table **entry** is **created** pointing to the existing **system-wide open-file table**.
- **If not**, the **directory structure** is **searched** and once the file is found:
  - Copy the **FCB** into the **system-wide open-file table** in memory.
  - Create **per-process open file table entry** pointing to system-wide open file entry.

The **open()** system call first searches the **system-wide open-file table** to see if the file is already in use by another process.



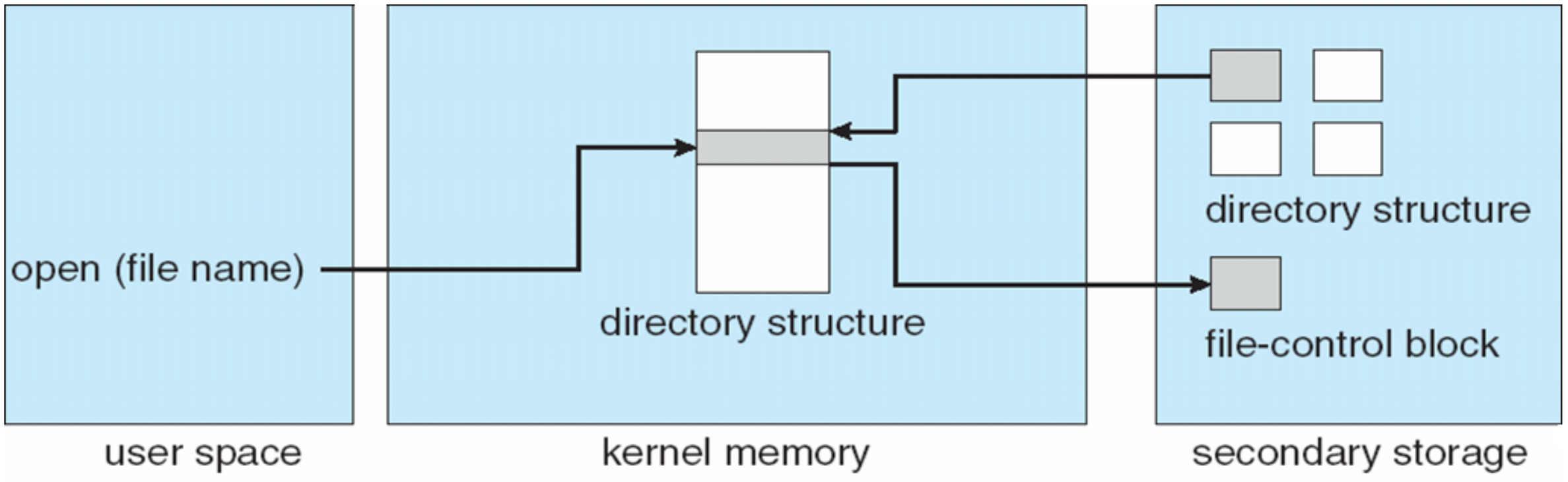
The **system-wide open-file table** contains a copy of the FCB of each open file.

If the **file** is found in the **system-wide** **open-file table** a **per-process** **open-file** table **entry** is **created** pointing to the existing system-wide open-file table.



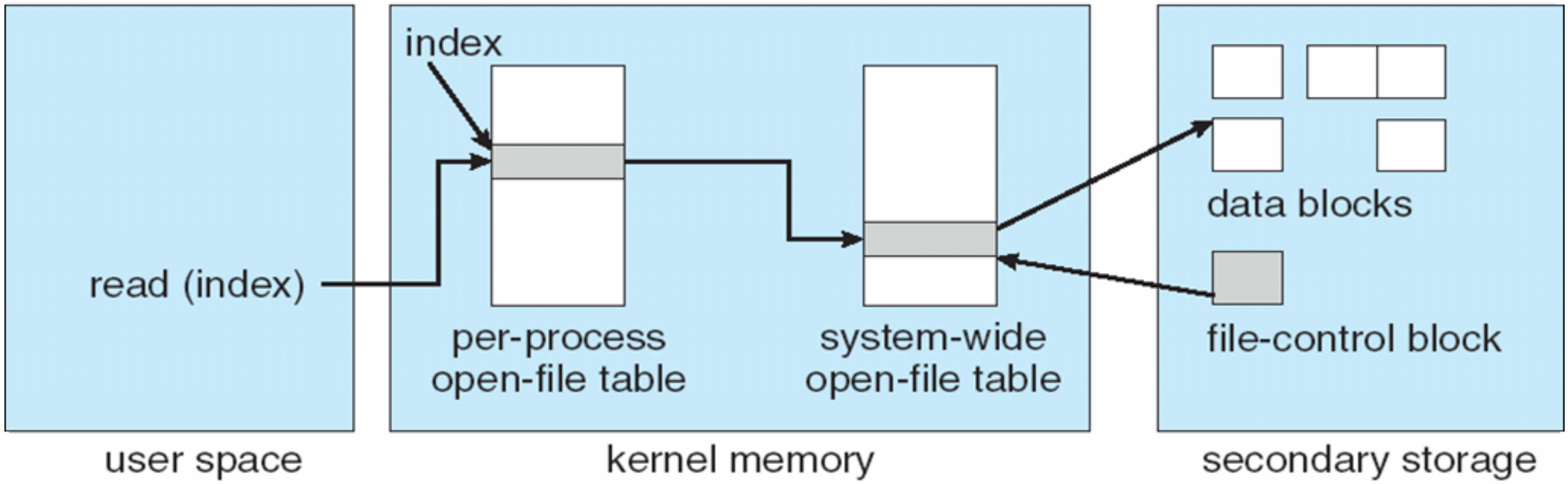
The **per-process** **open-file table** contains a pointer to the appropriate entry in the system-wide open-file table.

If the **file** is **not** found in the **system-wide** **open-file table** the **directory structure** is **searched** and once the file is found, the FCB is copied into the system-wide open-file table in memory.



The **directory structure** include file names and associated FCB numbers.

The `read()` system call searches the **per-process open-file table**.



The **per-process open-file table** contains a pointer to the appropriate entry in the system-wide open-file table.

The **system-wide open-file table** contains a copy of the FCB of each open file.

Directory

A directory is a file system **cataloging structure** which contains references to other computer files, and possibly other directories.

Source: https://en.wikipedia.org/wiki/Directory_(computing) 2018-02-24

Directories and file names

A file don't have a name.

A directory maps names to files.

Operations performed on directories

Search for a file

Create a file

Delete a file

List a directory

Rename a file

Traverse the file system

Linux/Unix commands

Command	Description
`ls`	List a directory directory
`find`	Search for a file
`ls -R`	List directory recursively (traverse)
`touch`	Create a new file or update access time
`mv`	Rename or move a file
`rm`	Delete a file

Terminal — bash — 22×29

bash

```
$> ls
a.c       b.txt
a.txt     dir
b.c
$> find . -name z*
./dir/z.txt
$> ls -R
a.c       b.txt
a.txt     dir
b.c

./dir:
z.txt
```

```
$> touch c.txt
$> ls
a.c       b.txt
a.txt     c.txt
b.c       dir
$> mv c.txt c.c
$> ls
a.c       b.txt
a.txt     c.c
b.c       dir
$> rm c.c
$> ls
a.c       b.txt
a.txt     dir
b.c
$>
```

Directory implementation

The directory structure can be implemented in various ways. Two examples are **linear list** and **hash table**.

Linear list

- Linear list of file names with pointer to the data blocks.
- Simple to program.
- Time-consuming to execute a search.

Hash table

- Decreases directory search time.
- **Collisions** – situations where two file names hash to the same location.
  - Enlarge the hash table.
  - Or use fixed size with overflow chains.

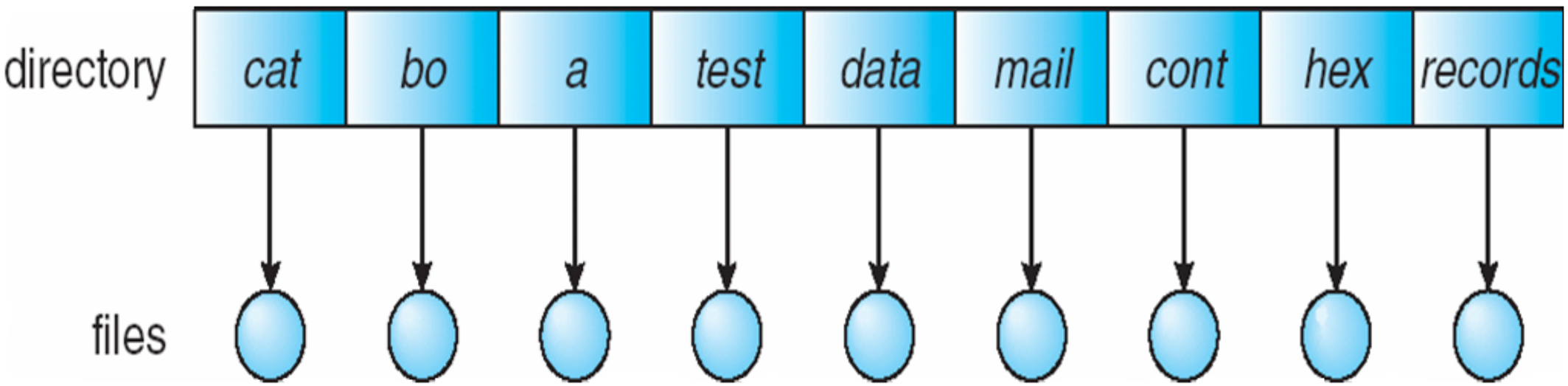
Directory hierarchy

Alternatives for organising directories

- single level
- two level
- tree
- acyclic graph

Single level directory

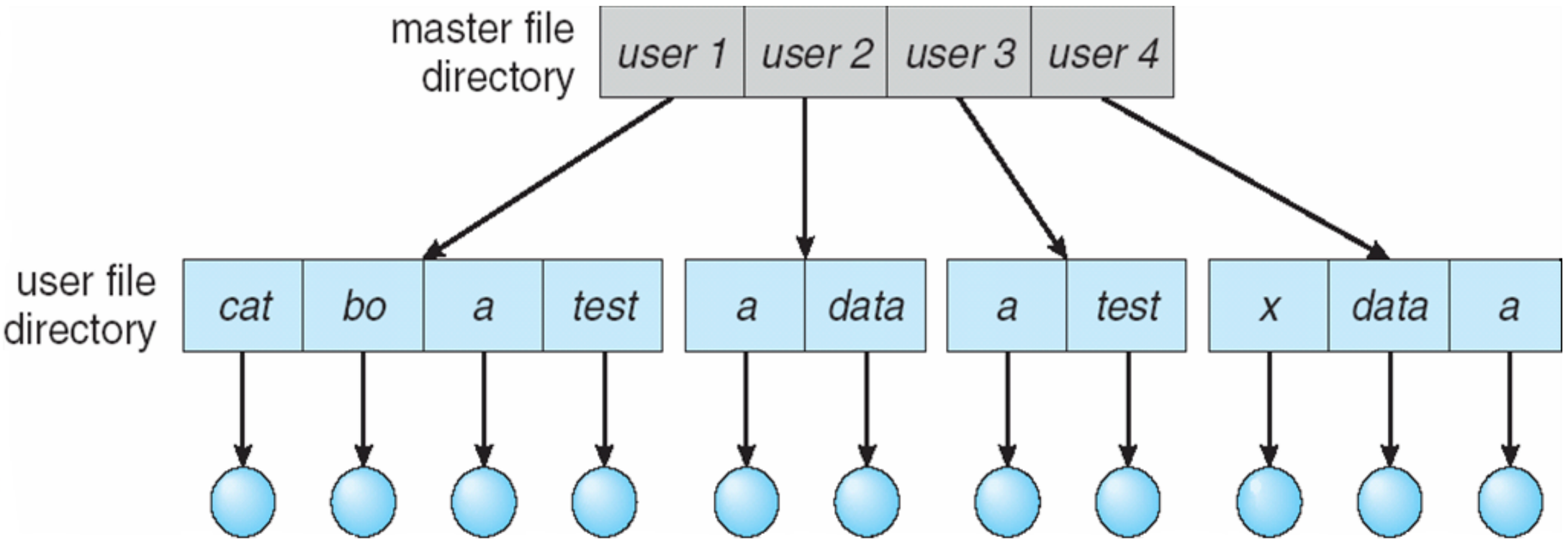
A single directory for all users.



- Naming problem
- Grouping problem

Two level directory

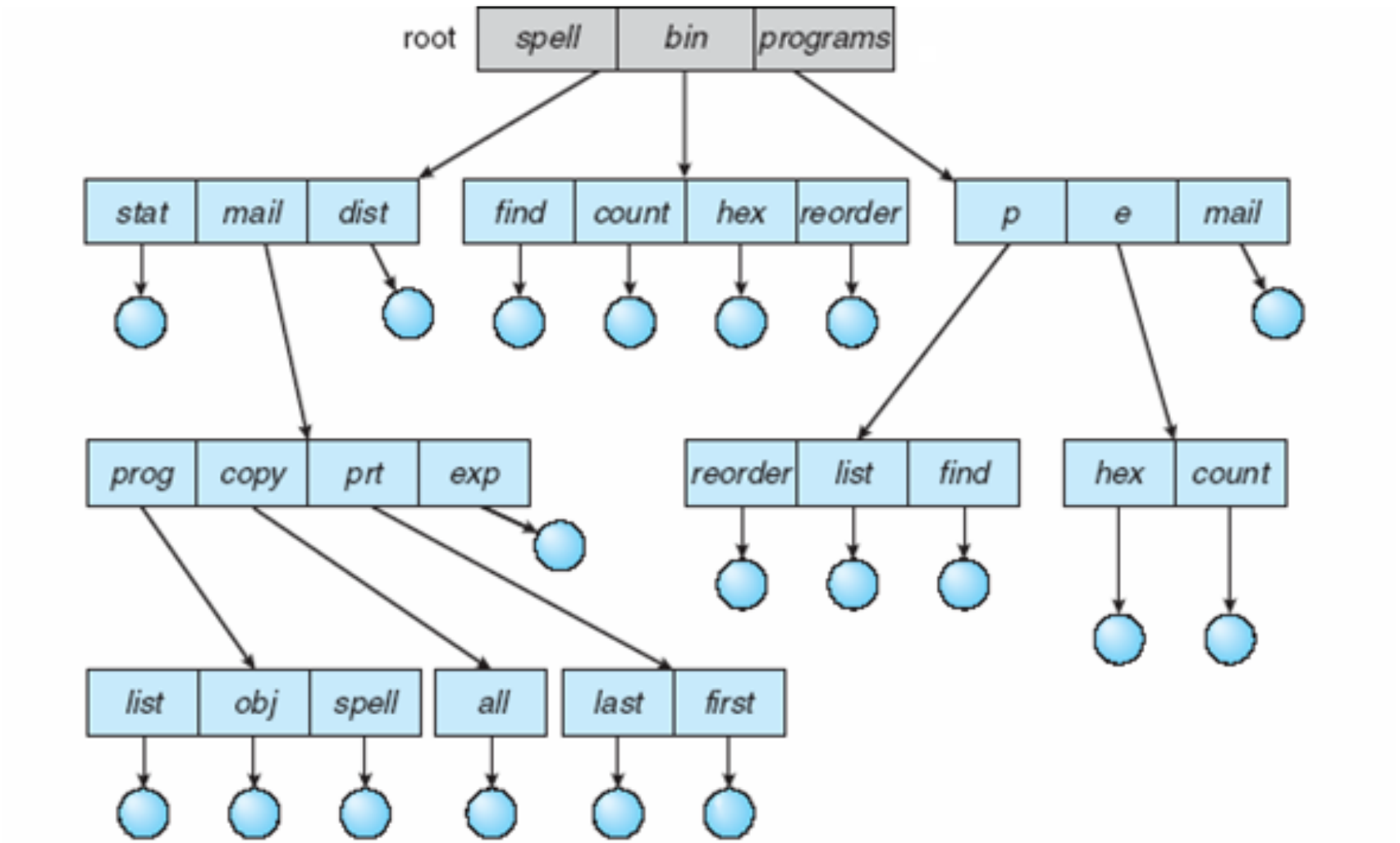
A single-level directory often leads to confusion of file names among different users. The standard solution is to create a separate directory for each user.



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability besides separating users

Tree structured directories

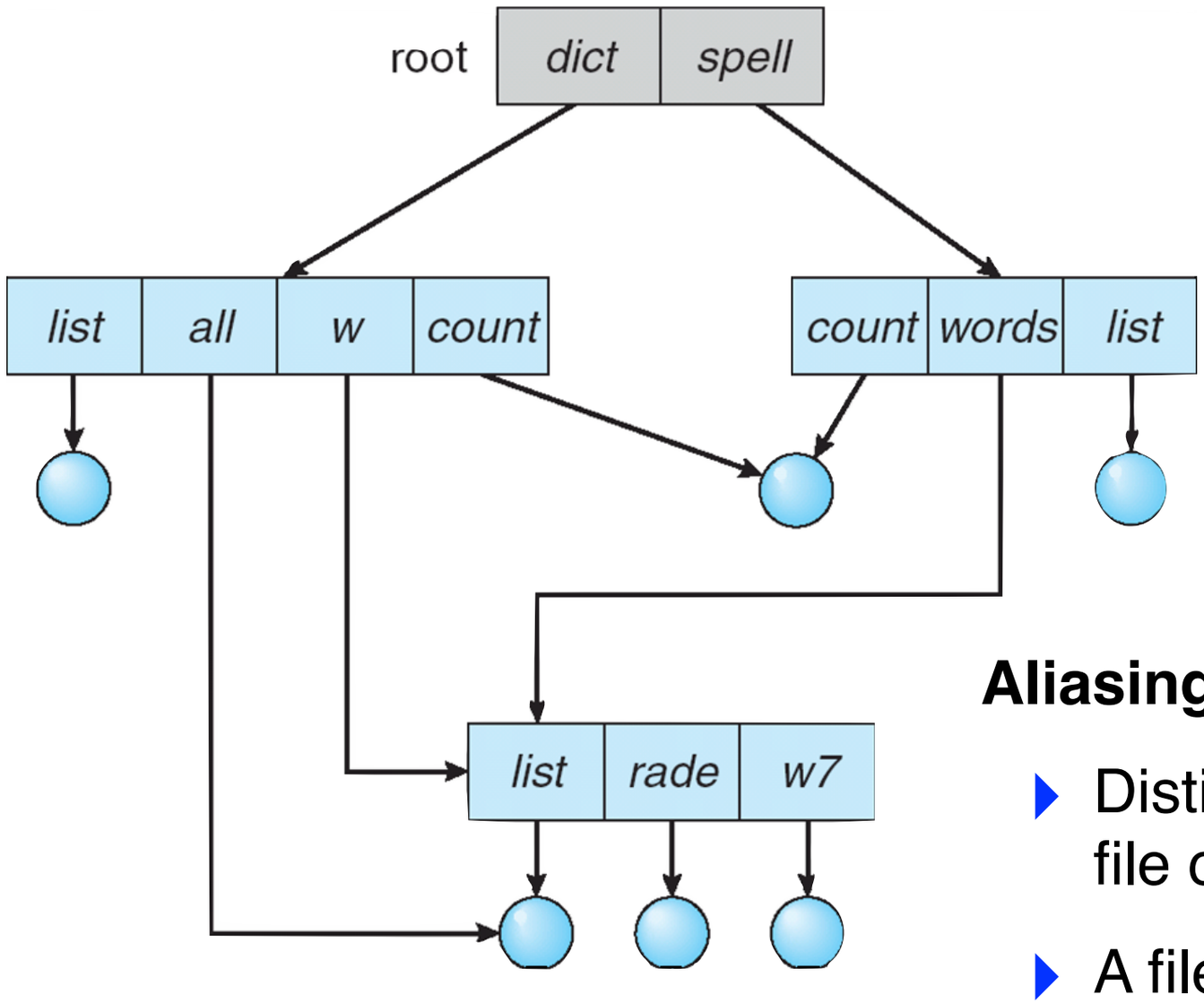
A tree structure allows users to create their own subdirectories.



- Path name
- Can have the same file name for different user
- Efficient searching
- Grouping capability
- Current directory (working directory)

Acyclic graph directories

An acyclic graph structure allows directories to **share** subdirectories and files.



Aliasing

- Distinct file names may refer to the same file or directory.
- A file may have multiple absolute path names.

Deletion

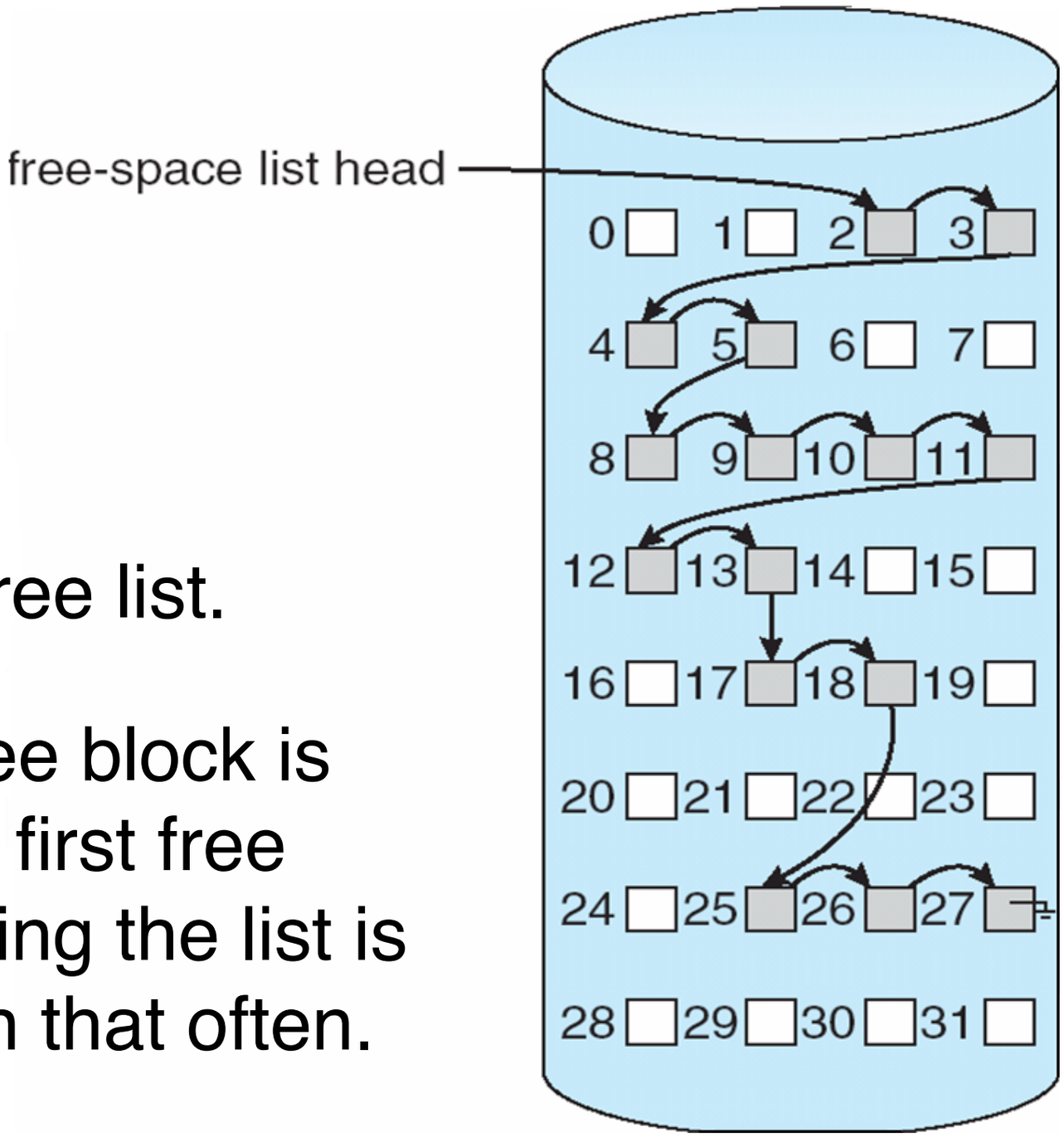
- How do deal with deletion of shared files.

Free-space management

Space from deleted files must be reused.

Linked free space list on disk

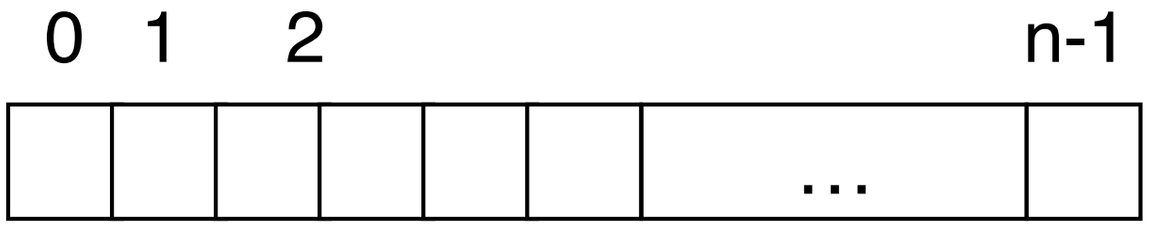
Link together all the free disk blocks.



- **Slow** traversing of the free list.
- Usually, only a single free block is needed, simply pick the first free block in the list. Traversing the list is slow but doesn't happen that often.

Bit vector

A common technique to implement the free-space list is to use a bit vector (bit map).



$$\text{bit}[i] = \begin{cases} 1 \Leftrightarrow \text{block}[i] \text{ free} \\ 0 \Leftrightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Hardware support

CPU instructions that return the offset in a word of the first bit with value 1.

Find a free block: Scan the bit vector sequentially, looking for the first non-0 word (at least one free block). Next, scan the first non-0 word for the first 1 bit.

Block number calculation: (number of bits per word) x (number of 0-value words) + offset of first 1 bit.

Hard links

A hard link is a directory entry that **associates** a **name** with a file on a file system.

All directory-based file systems **must** have **at least one hard link** giving the original name for each file.

The term hard link is usually only used in file systems that allow more than one hard link for the same file.

Source: https://en.wikipedia.org/wiki/Hard_link 2018-02-24

Symbolic links

A symbolic link (also symlink or soft link) is a term for any file that contains a **reference** to another file or directory in the form of an **absolute or relative path** and that affects pathname resolution.

Hard links

- A hard link is a **pointer to the inode** of a file.
- Established using the `ln` command.
- The **link count** of the file is **incremented**.
- Both the original file and the new entry point to the **same inode**.
- When deleted, the link count is decremented, and the file is **only deleted** if the resulting **link count** is **zero**.

Symbolic links

- Established using the `ln -s` command.
- The **link count** of the file is **not incremented**.
- The created file is of the special type “link” denoted by “l” in directory listings.
- The linked file is an actual file that contains the **path to the original file**.
- Symbolic links can be created **across file systems**.
- Symbolic links to directories can be created by any user.
- Symbolic links can be left **dangling** when the original file is moved or deleted.

The `ln` command

The `ln` command is a standard Unix command utility used to create a **hard link** or a **symbolic link** (symlink) to an existing file.

Source: https://en.wikipedia.org/wiki/Ln_(Unix) 2018-02-24

The `ln` command

In Unix, links can be used to share files and directories.

Syntax

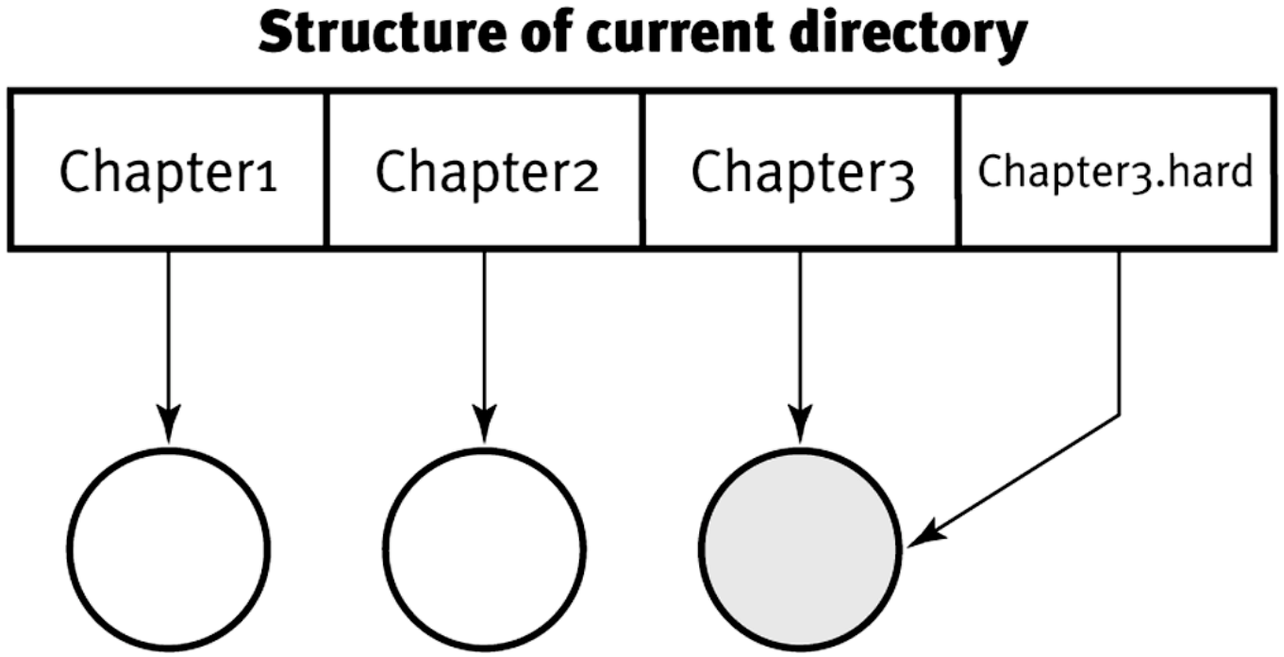
```
ln [options] oldfile          newfile
ln [options] old-file-list    directory
```

Options

```
-f   force creation (overwrite existing
     file)
-n   don’t force
-s   create soft (symbolic) link
```

Example - hard link

```
$ ln Chapter3 Chapter3.hard
```



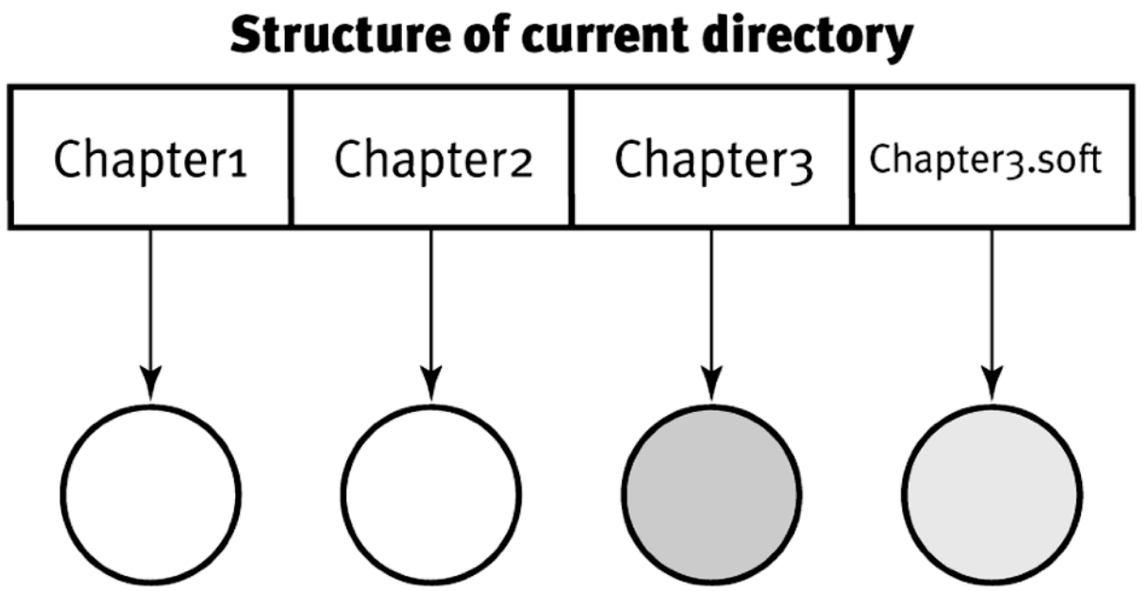
Contents of current directory

Inode #	File
1076	.
2083	..
13059	Chapter1
17488	Chapter2
52473	Chapter3
52473	Chapter3.hard

The hard link points to the same inode as the target

Example - soft link

```
$ ln -s Chapter3 Chapter3.hard
```



Contents of current directory

Inode #	File
1076	.
2083	..
13059	Chapter1
17488	Chapter2
52473	Chapter3
52479	Chapter3.soft

Creating a soft link creates a new inode.

Soft links in directory listing

```
$> ln  -s  Chapter3  Chapter3.soft
$> ls  - il
52473  -rwxr--r--   1 sarwar  faculty  9352  May 28  23:09  Chapter3
52479  lrwxr--r--   2 sarwar  faculty     8  Oct 13  14:24  Chapter3.soft  --> Chapter3
```

Soft links are shown using an arrow `-->` in the output from `ls -l`

Using the `-i` option `ls` shows the inode number of files and directories.

```
$ touch a
$ ln -s a link1
$ ls -l
total 8
-rw-r--r--  1 karl  staff  0  6 Mar 21:46 a
lrwxr-xr-x  1 karl  staff  1  6 Mar 21:47 link1 -> a
$ echo "abc" >> a
$ cat link1
abc
$ rm a
$ cat link1
cat: link1: No such file or directory
$ ls -l
total 8
lrwxr-xr-x  1 karl  staff  1  6 Mar 21:47 link1 -> a
$
```

Symbolic links can be left dangling when the original file is moved or deleted.