CSE 421/521 - Operating Systems
Fall 2018

LECTURE - XIX

FILE SYSTEMS - II

Tevfik Koşar

University at Buffalo

November 8th, 2018

File Systems

- An interface between users and files
- Provides organized and efficient access to data on secondary storage:

  1. Organizes data into files and directories and supports primitives to manipulate them (create, delete, read, write etc)
  2. Improves I/O efficiency between disk and memory (perform I/O in units of blocks rather than bytes)
  3. Ensures confidentiality and integrity of data

File Control Block (inode in Unix)

The file system contains file structure via a File Control Block (FCB)

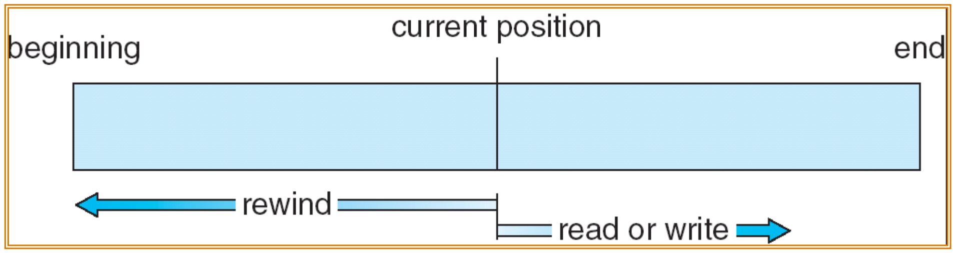
- Ownership, permissions, location..etc information

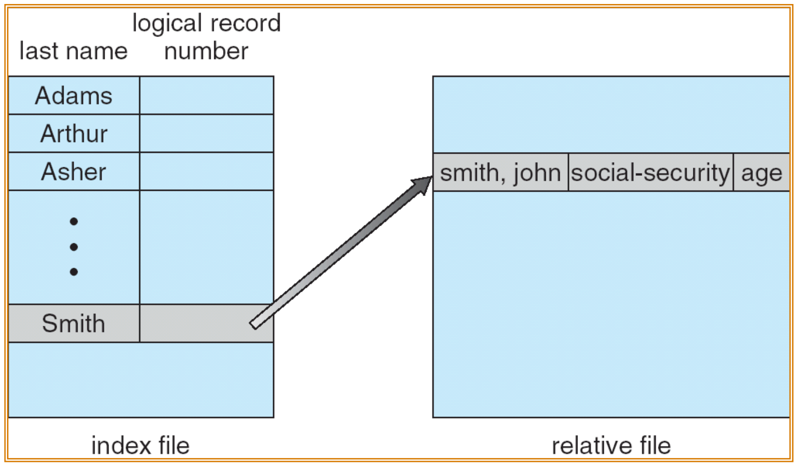
A typical File Control Block:

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

File access methods

- Some file systems provide different access methods that specify ways the application will access data
  - sequential access
    - read bytes one at a time, order
  - direct access
    - random access given a block/byte #
  - record access
    - file is array of fixed- or variable-sized records
  - indexed access
    - FS contains an index to a particular field of each record in a file, and applications can find a file based on value in that record (similar to DB)





Directories

Directories provide:

- a way for users to organize their files
- a convenient file name space for both users and FS's

Most file systems support multi-level directories

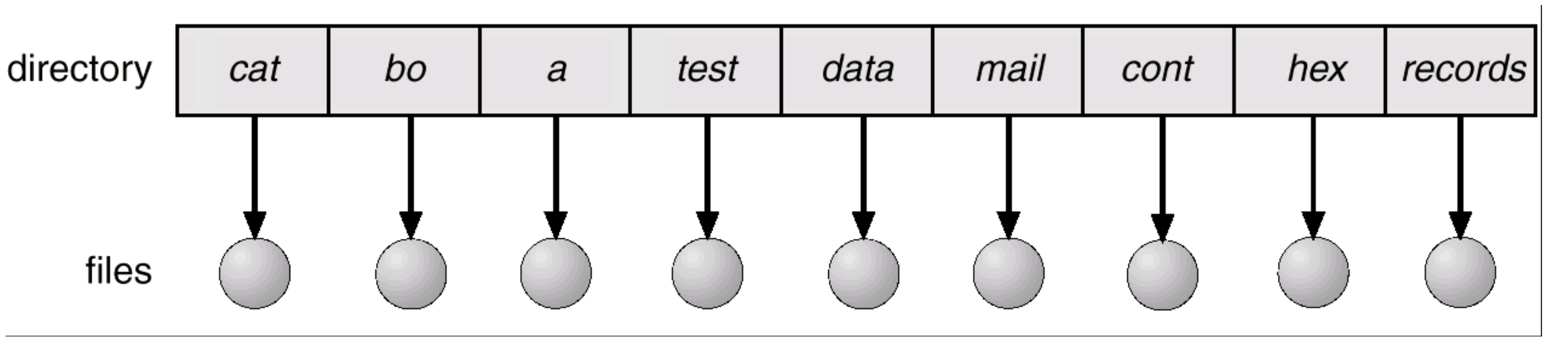
- naming hierarchies (/, /usr, /usr/local, /usr/local/bin, …)

A directory is typically just a file that happens to contain special metadata

- directory = list of (name of file, file attributes)
- attributes include such things as:
  - size, protection, location on disk, creation time, access time, …
- the directory list is usually unordered (effectively random)
  - when you type "ls", the "ls" command sorts the results for you

Directories

- Single-level directory structure
  - ✓ simplest form of logical organization: one global or **root** directory containing all the files
  - ✓ problems
    - global namespace: unpractical in multiuser systems
    - no systematic organization, no groups or logical categories of files that belong together

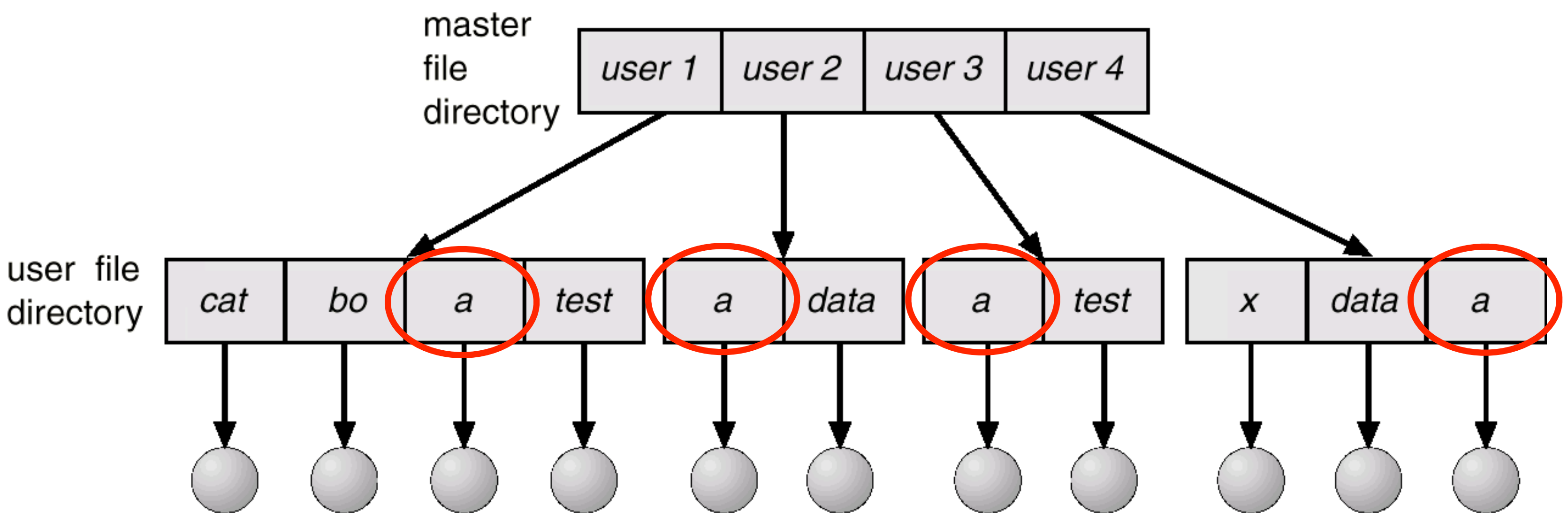


Single-level directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Directories

- **Two-level directory structure**
  - ✓ in multiuser systems, the next step is to give each user their own private directory
  - ✓ avoids filename confusion
  - ✓ however, still no grouping: not satisfactory for users with many files

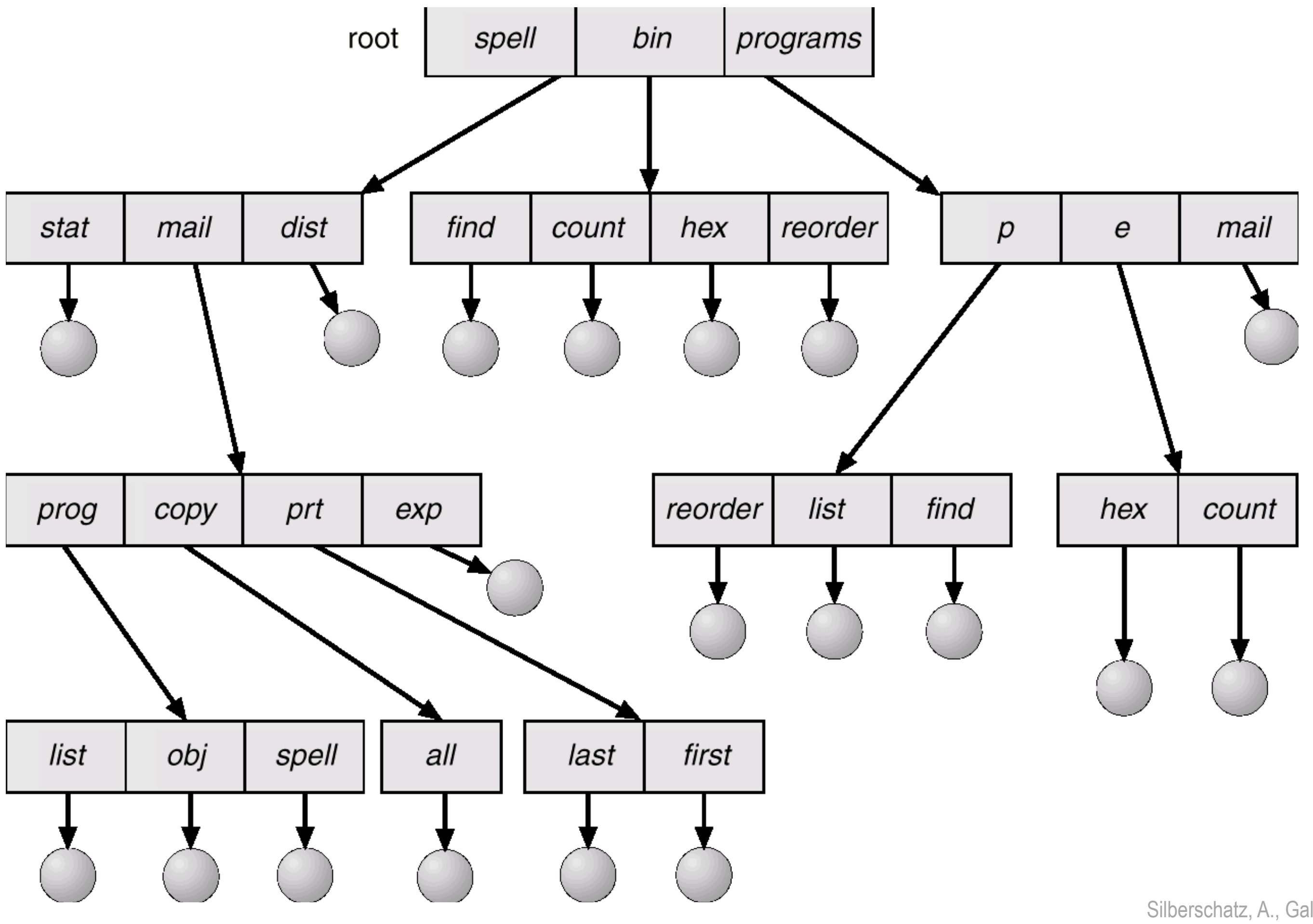


Two-level directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Directories

- **Tree-structured directory structure**



Tree-structured directory

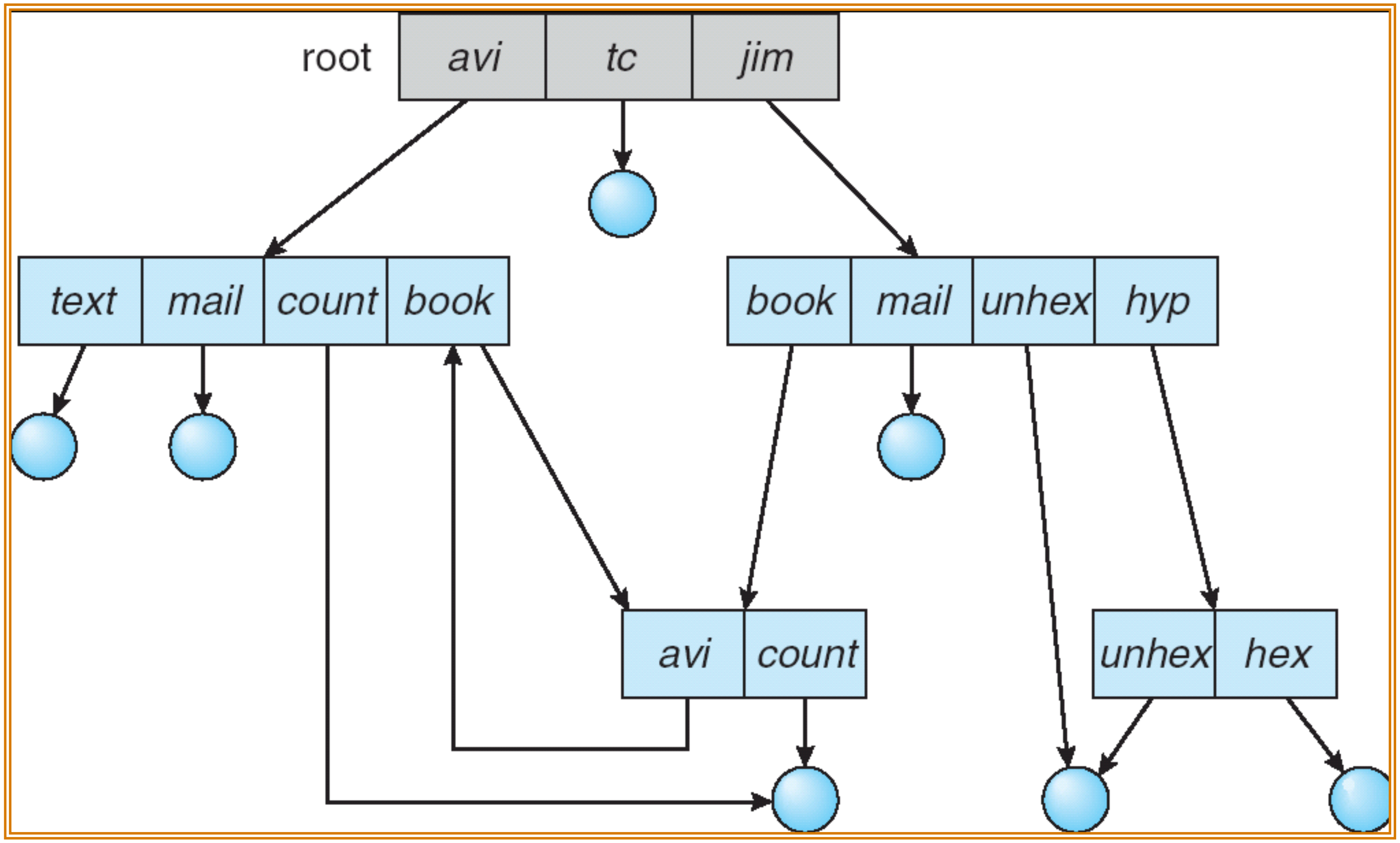
Silberschatz, A., Galvin, P. B. and Gagne. G. (2003)
Operating Systems Concepts with Java (6th Edition).

Directories

- <u>Tree-structured directory structure</u>
  - ✓ natural extension of the two-level scheme
  - ✓ provides a general hierarchy, in which files can be grouped in natural ways
  - ✓ good match with human cognitive organization: tendency to categorize objects in embedded sets and subsets
  - ✓ navigation through the tree relies on **pathnames**
    - absolute pathnames start from the root, example: /jsmith/academic/teaching/cs446/assignment4/grades
    - relative pathnames start at from a current **working directory**, example: assignment4/grades
    - the current and parent directory are referred to as . and ..

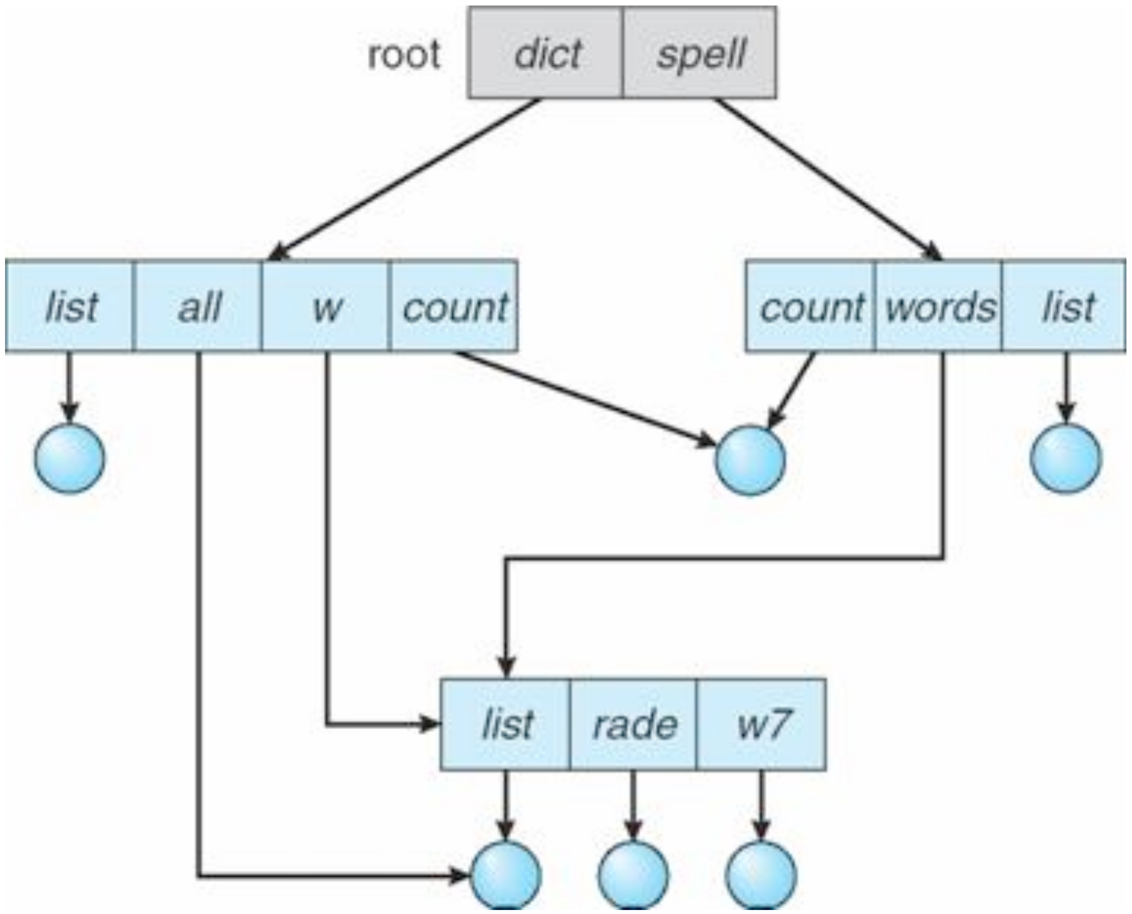
Directories

- <u>General Graph directory structure</u>



Directories

- <u>A-cyclic Graph directory structure</u>



A-cyclic graph structured directory

Silberschatz, A., Galvin, P. B. and Gagne. G. (2003) *Operating Systems Concepts with Java (6th Edition).*

Pathname Translation

- Let’s say you want to open “/one/two/three”

```
fd = open(“/one/two/three”, O_RDWR);
```

What goes on inside the file system?

open directory “/” (well known, can always find)
search the directory for “one”, get location of “one”
open directory “one”, search for “two”, get location of “two”
open directory “two”, search for “three”, get loc. of “three”
open file “three”
(of course, permissions are checked at each step)

FS spends lots of time walking down directory paths

this is why open is separate from read/write (session state)
OS will cache prefix lookups to enhance performance

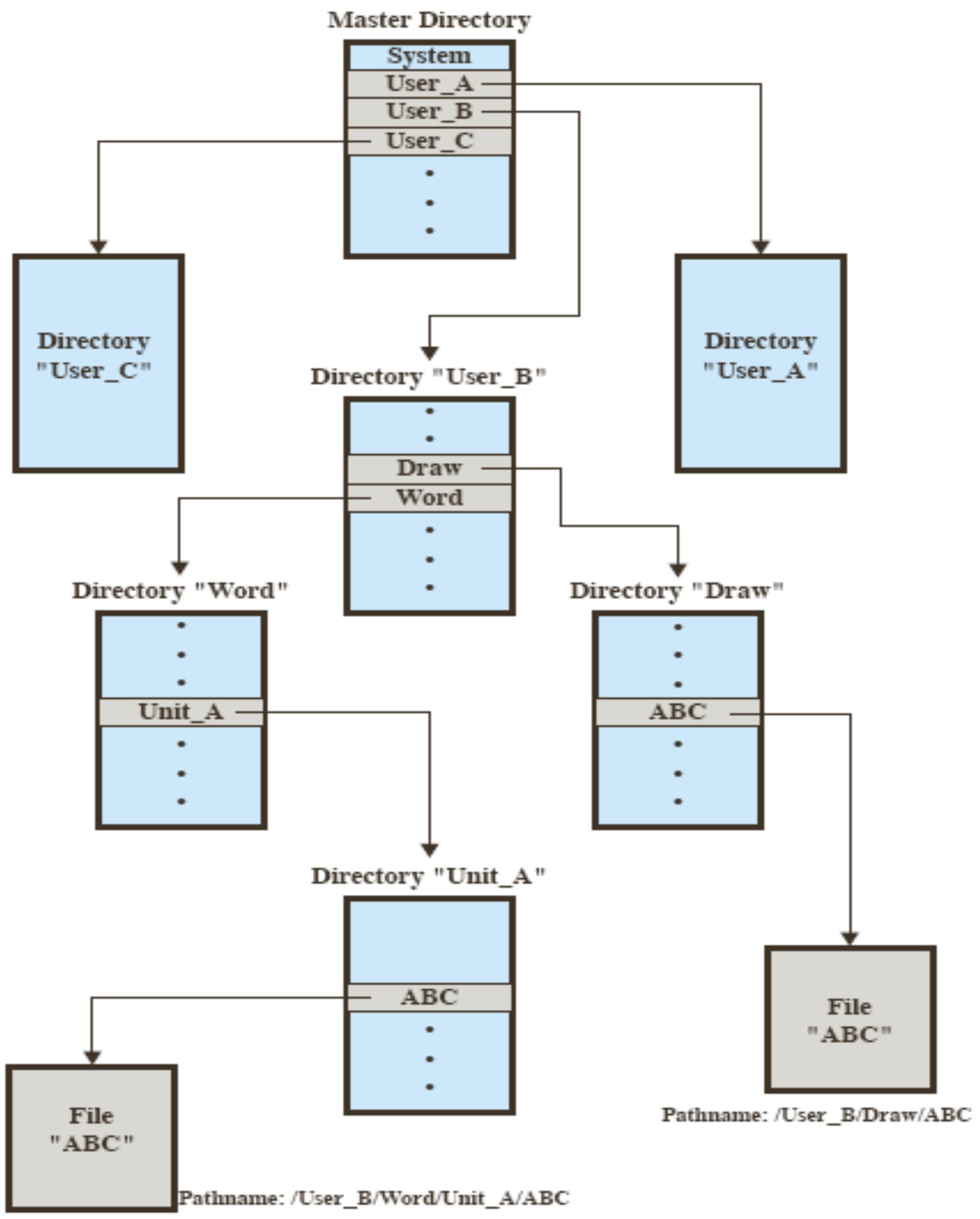
/a/b, /a/bb, /a/bbb all share the “/a” prefix

Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute

- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

Directory Implementation



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

UNIX Directories

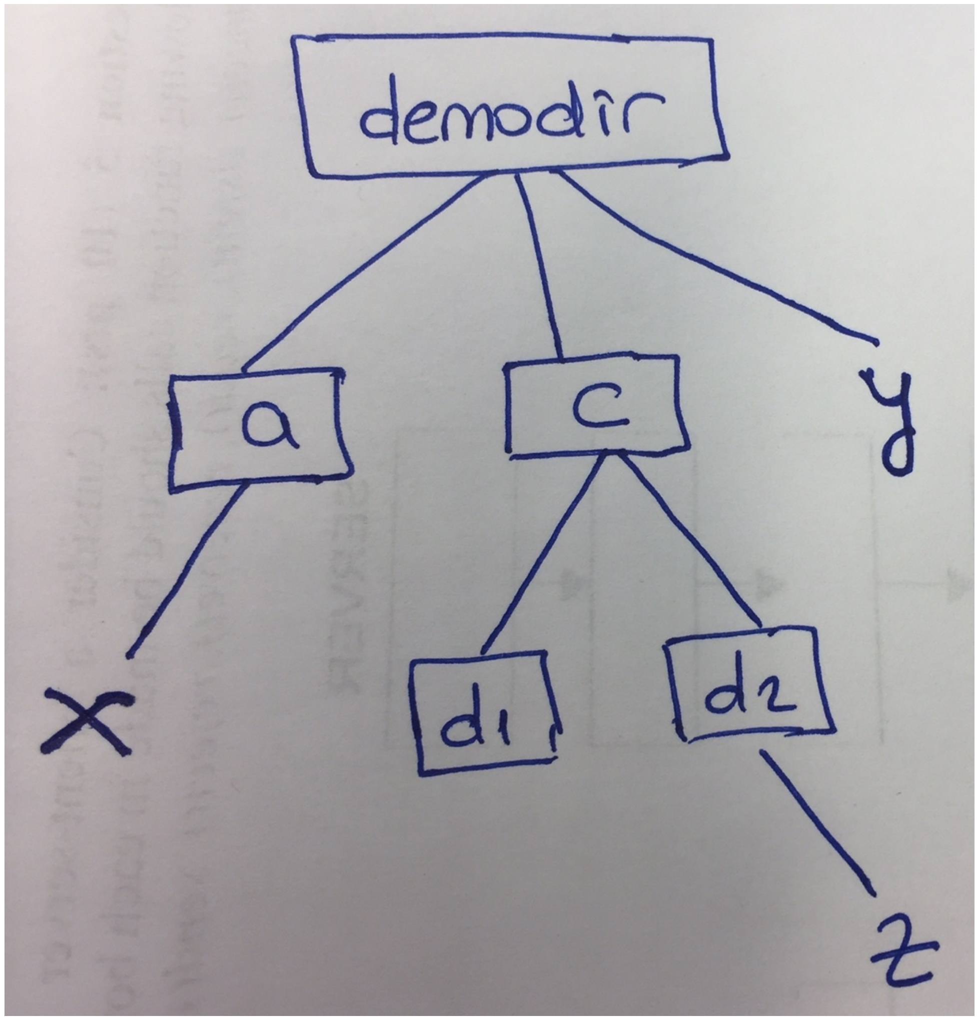
- Directory is a special file that contains list of names of files and their inode numbers
- to see contents of a directory:

```
$ls -1ia .
9535554 .
9535489 ..
9535574 .bash_history
9535555 bin
9535584 .emacs.d
9535560 grading
9535803 hw1
9535571 test
9535801 .viminfo
```

Directories - System View

- user view vs system view of directory tree
  - representation with “dirlists (directory files)”
- The real meaning of “A file is in a directory”
  - directory has a link to the inode of the file
- The real meaning of “A directory contains a subdirectory”
  - directory has a link to the inode of the subdirectory
- The real meaning of “A directory has a parent directory”
  - “..” entry of the directory has a link to the inode of the parent directory

Example: User view of directory tree

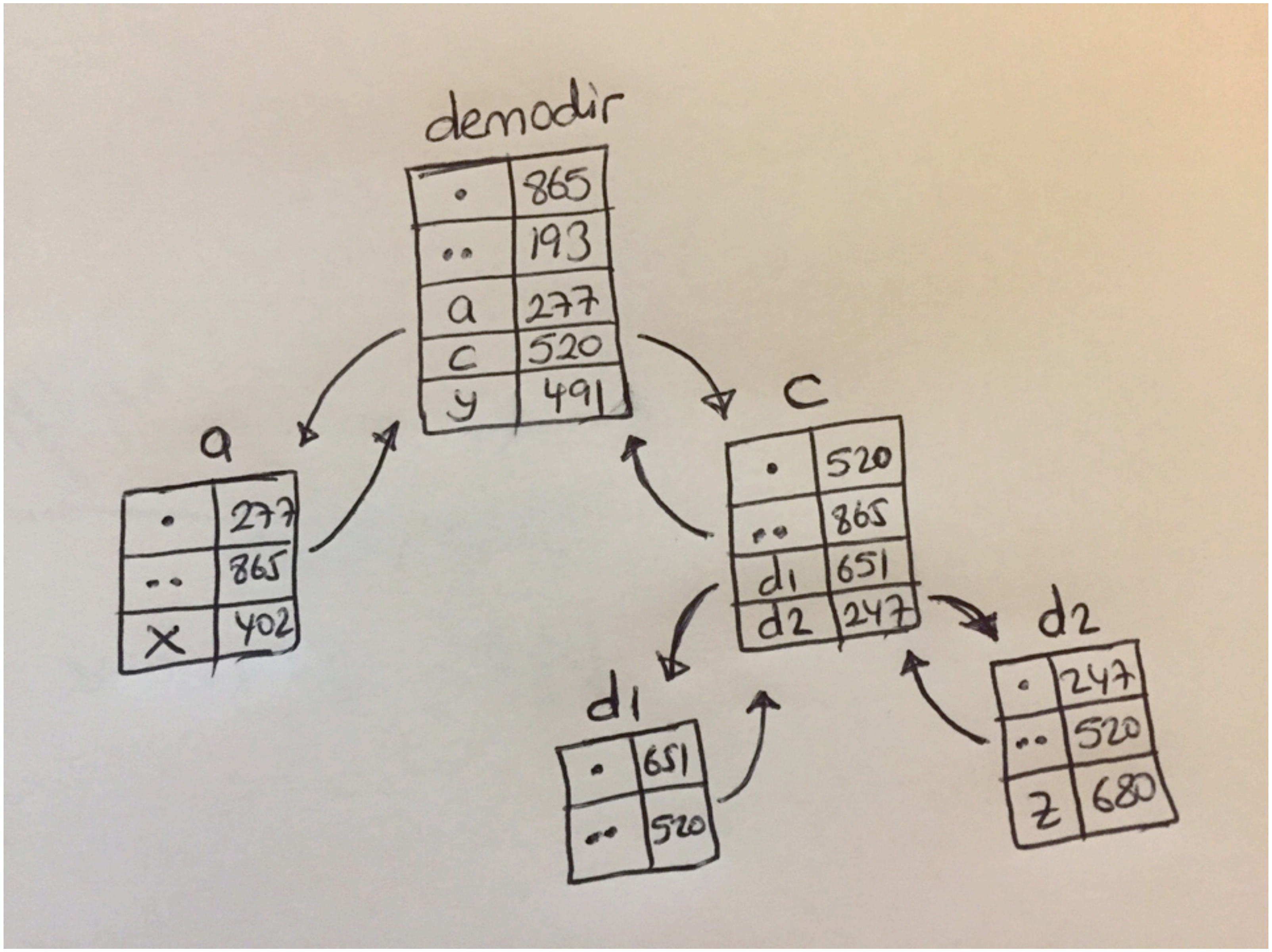


Example: inode listing of directory tree

```
$ ls -iaFR demodir:
865 .       193 ..      277 a/      520 c/      491 y

demodir/a:
277 .       865 ..      402 x

demodir/c:
520 .       865 ..      651 d1/         247 d2/

demodir/c/d1:
651 .       520 ..

demodir/c/d2:
247 .       520 ..      680 z
```

Please show the system representation (system view) of this directory tree.

Example: System view of directory tree



inode

- Inode = structure maintaining all metadata about a file (or directory), *except for name*
- Inode number = unique ID of inode
- One or more file *names* can point (link) to the same inode
- Why do we need inode numbers? Can’t we use absolute paths as IDs?
- Where do we store file names?

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

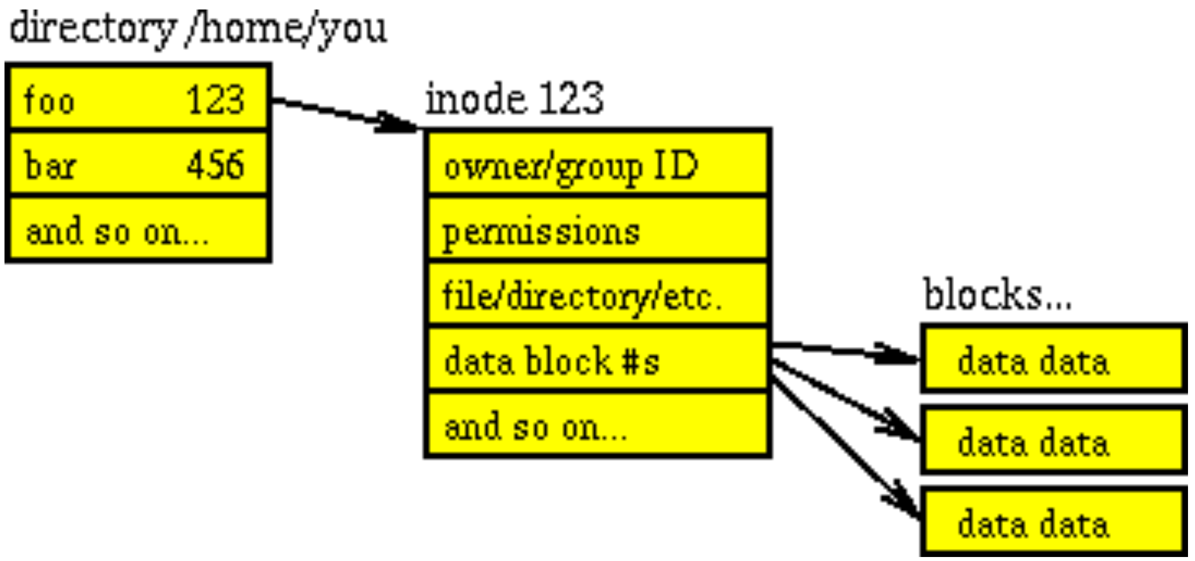
Where to store file names?

- Inode numbers provide location independence

- File names are stored in directory entries, in the form of

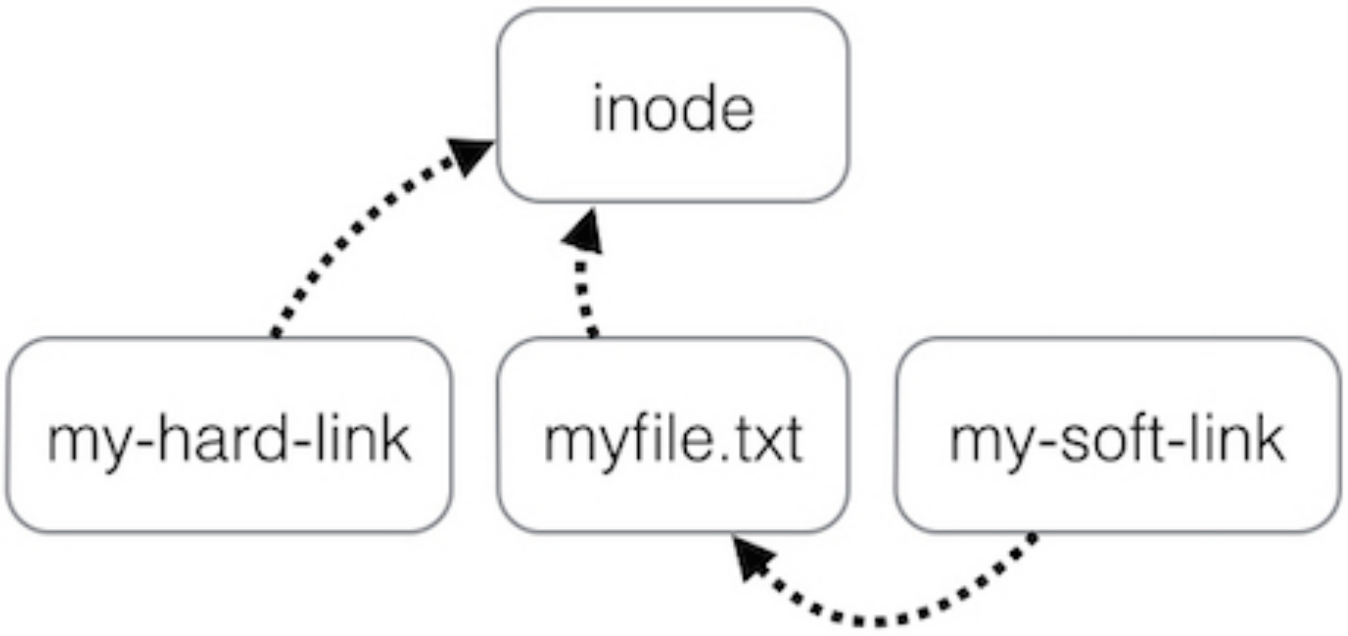
  => pair (name, inode number)

- A directory is *just a file*, whose contents is a list of directory entries



Soft Link vs Hard Link

- A **hard link** then creates another file with a link to the same underlying inode.
- A **soft link (symbolic link)** is a link to another name in the file system.



Link Counts

- The kernel records the number of links to any file/ directory.
- The *link count* is stored in the inode.
- The *link count* is a member of *struct stat* returned by the *stat* system call.

Change Links

- **What will be the resulting changes in directory tree?**

```
$ cp demodir/a/x demodir/c/xcopy

$ ln demodir/a/x demodir/c/d1/xlink

$ ln -s demodir/a/x demodir/c/d1/slink

$ mv demodir/y  demodir/a/y
```

Implementing “pwd”

1. “.” is 247

   chdir ..

2. 247 is called “d2”

   “.” is 520

   chdir ..

3. 520 is called “c”

   “.” is 865

   chdir ..

4. 865 is called “demodir”

   “.” is 193

   chdir ..

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR