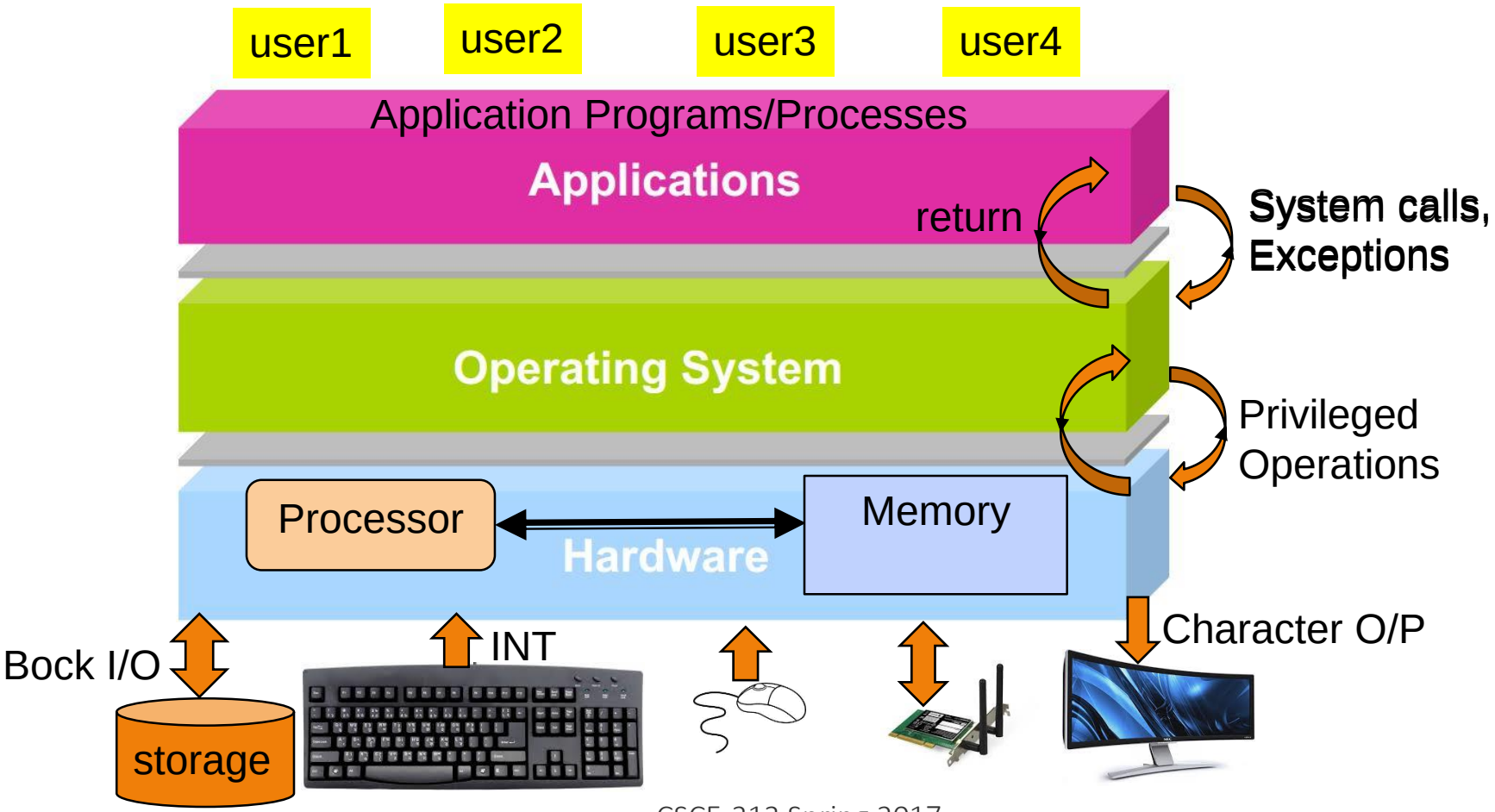
W7: FILE SYSTEMS

Aakash Tyagi
CSCE 313 Spring 2017

Theme of Today's Lecture

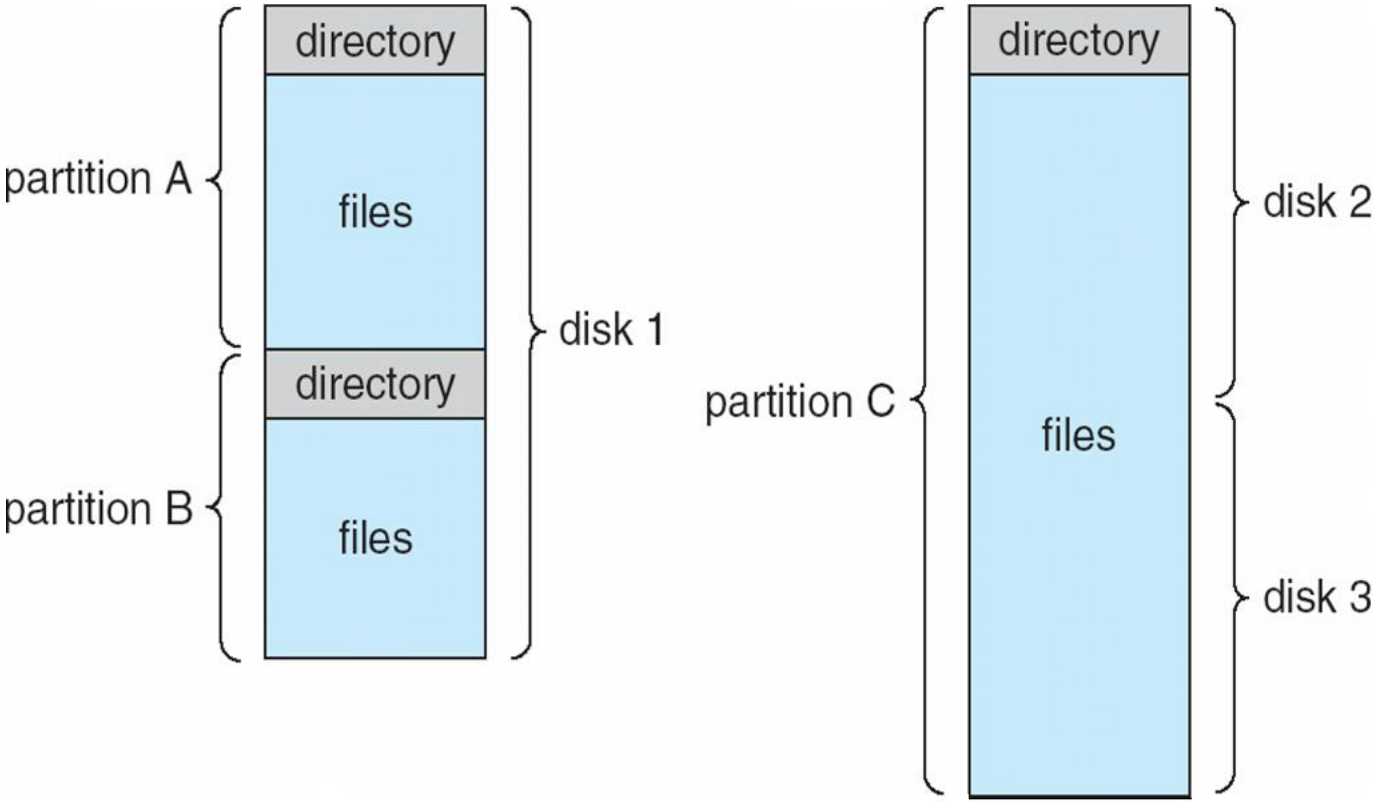
- Unix File Systems



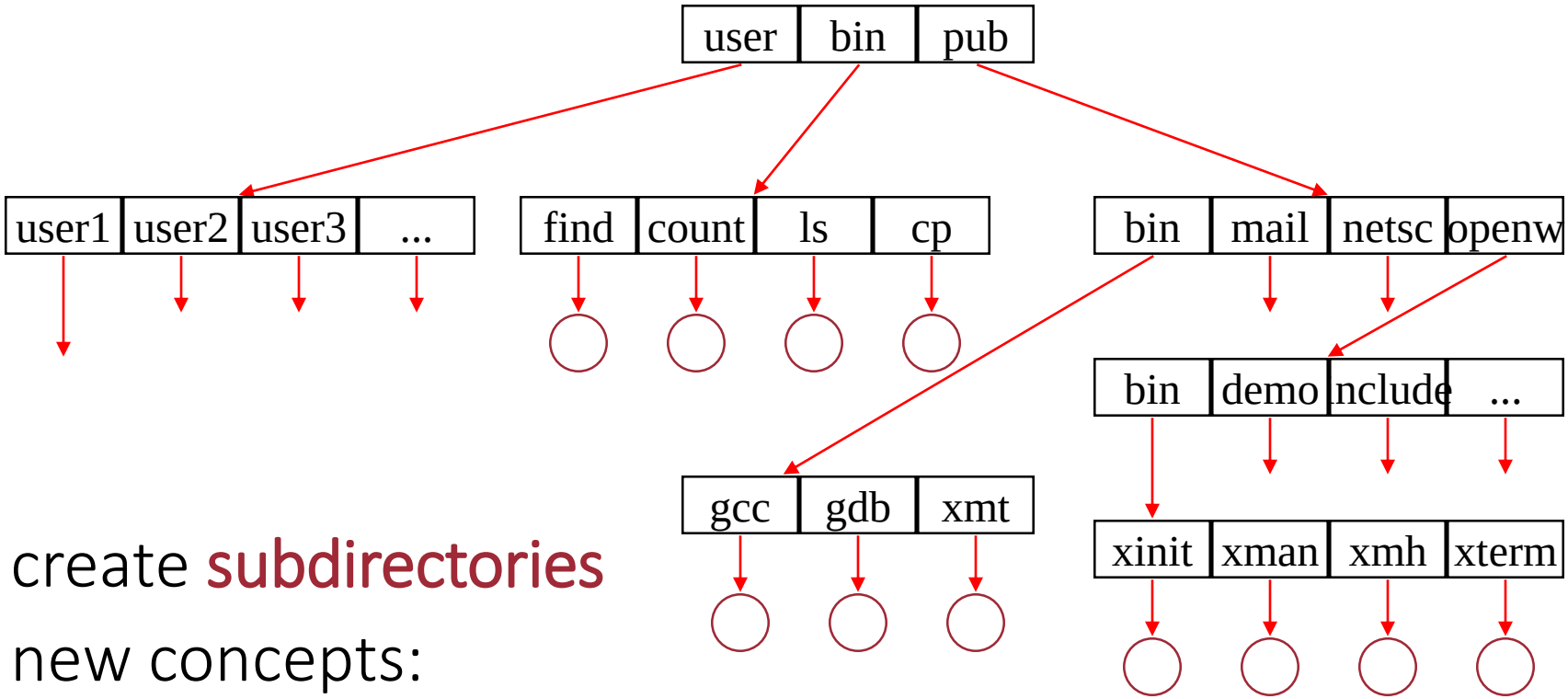
The UNIX File System

- File Systems and Directories
- Accessing directories
- UNIX inodes
- Understanding links in directories
- File protection/access control

Typical File System Organization



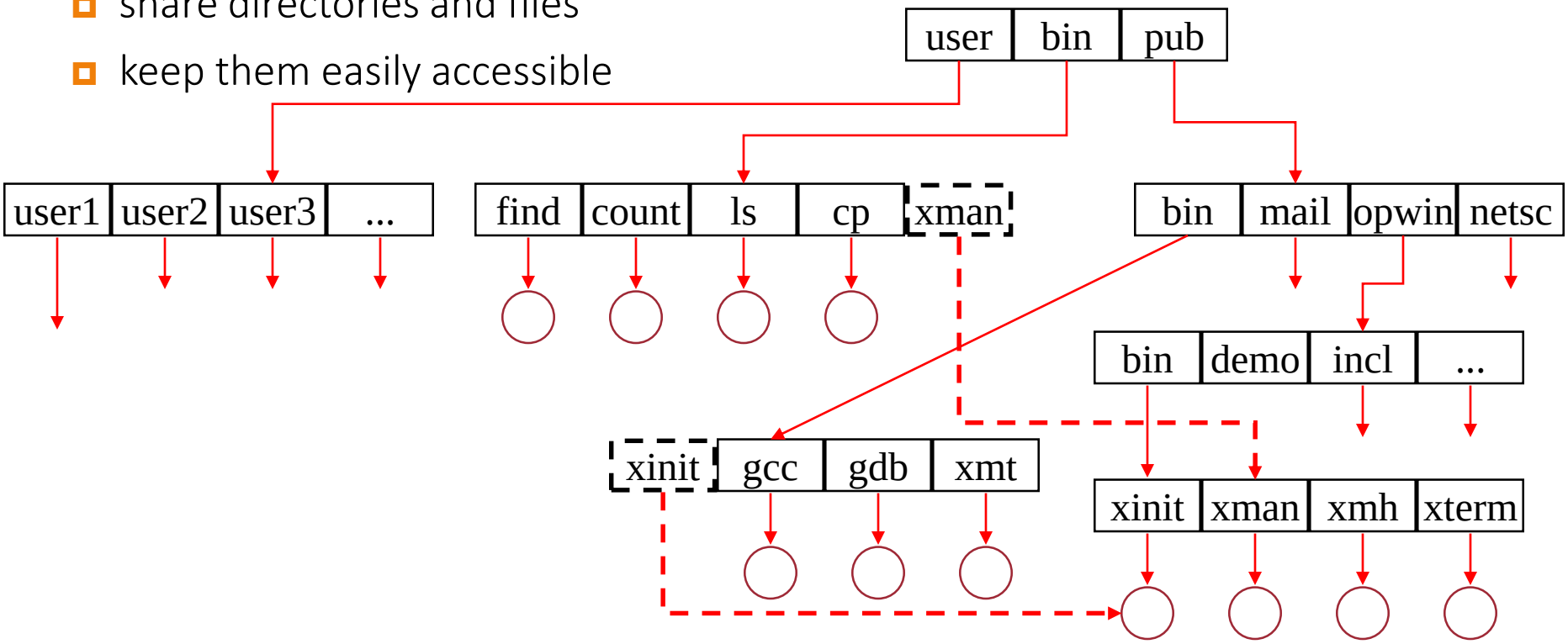
Tree-Structured Directories



- create subdirectories
- new concepts:
  - current directory
  - path names: complete vs. relative
  - search paths

Generalized Tree Structures

- share directories and files
- keep them easily accessible



- Links: File name that, when referred, affects file to which it was linked. (hard links, symbolic links)

UNIX Directory Navigation: Current Directory


```
#include <unistd.h>

char * getcwd(char * buf, size_t size);
/* get current working directory */
```

Example:

```
void main(void) {
    char mycwd[PATH_MAX];

    if (getcwd(mycwd, PATH_MAX) == NULL) {
        perror ("Failed to get current working directory");
        return 1;
    }
    printf("Current working directory: %s\n", mycwd);
    return 0;
}
```

UNIX Directory Navigation – Open, Read, Close


```
#include <dirent.h>

int main(int argc, char * argv[]) {
    struct dirent * direntp;
    DIR           * dirp;
    if (argc != 2) {
        fprintf(stderr, "Usage: %s directory_name\n",
argv[0]);
        return 1;
    }
    if ((dirp = opendir(argv[1])) == NULL) {
        perror("Failed to open directory");
        return 1;
    }
    while ((direntp = readdir(dirp)) != NULL)
        printf(%s\n", direntp->d_name);
    while((closedir(dirp) == -1) && (errno == EINTR));
    return 0;
}
```

UNIX Directory Navigation – Traversal


```
#include <dirent.h>

DIR           * opendir(const char * dirname);
        /* returns pointer to directory object          */
struct dirent * readdir(DIR * dirp);
        /* read successive entries in directory 'dirp'  */
int             closedir(DIR *dirp);
        /* close directory stream                       */
void            rewinddir(DIR * dirp);
        /* reposition pointer to beginning of directory */
```

Directory Traversal – Example 1

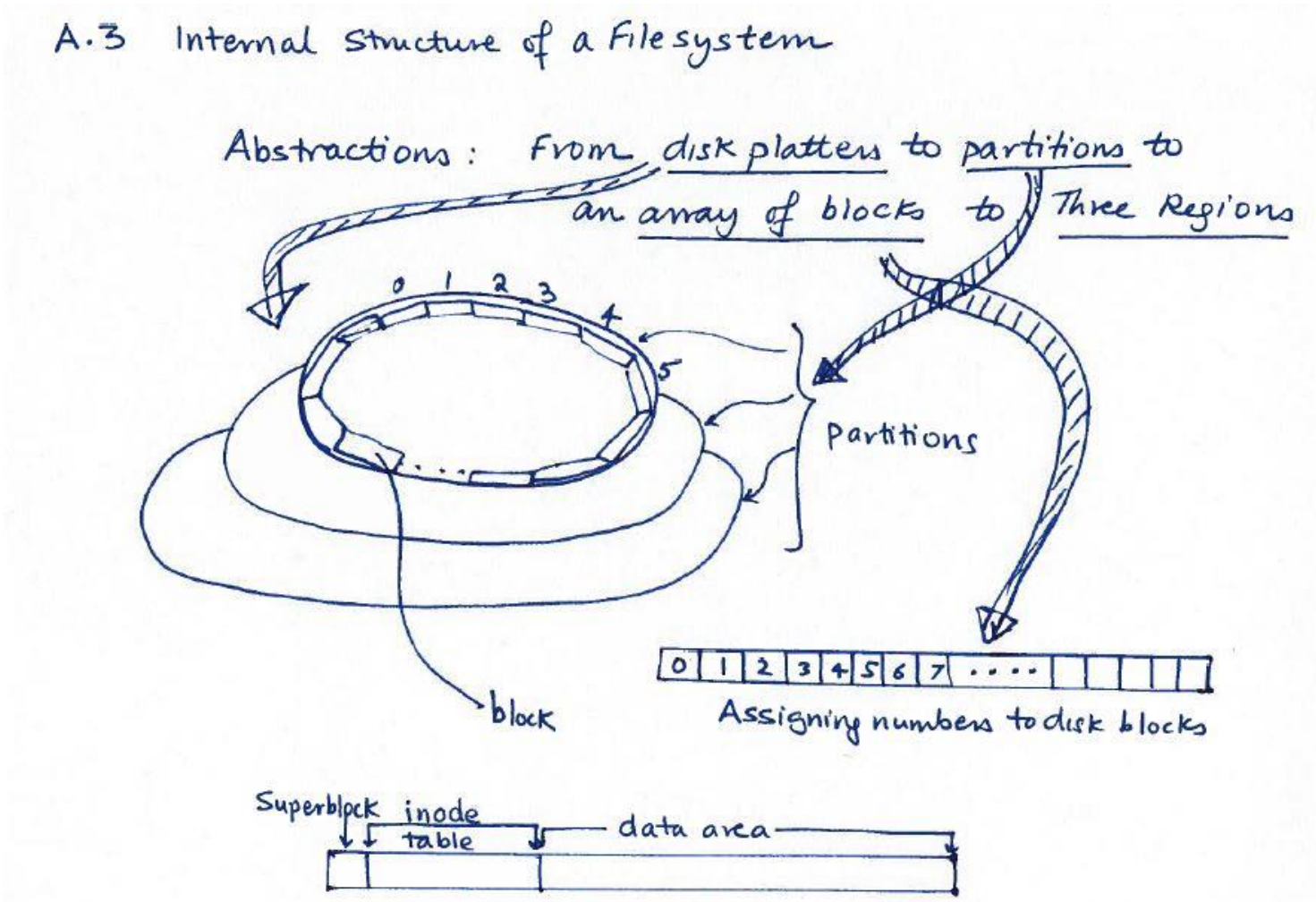
mkdir: creates a directory with the name supplied in the argument
rmdir: removes a directory with the name supplied in the argument
mv: moves a file or directory to new location (or name)
cd: change directory

Eg.1 User View of a FileSystem - Working example

```
% mkdir demodir
% cd demodir
% mkdir b oops
% mv b c
% rmdir oops
% cd c
% mkdir d1 d2
% cd ../..
% mkdir demodir/a
```

<space for drawing directory structure>

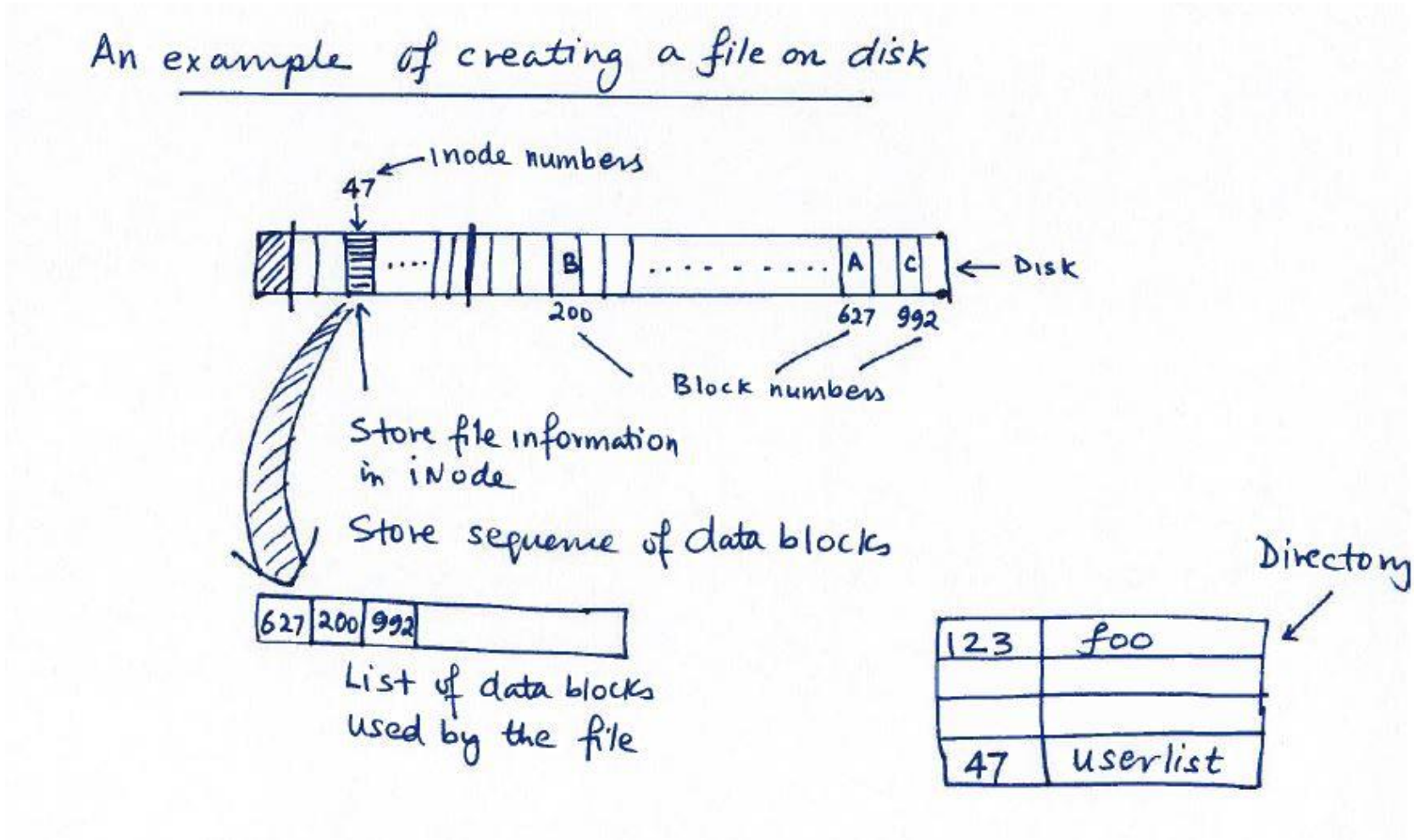
Internal Structure of a File System



Sections of a File System

- Super Block : Contains information about the organization of the file system. Size of the super block is Unix Versi dependent.
- inode table : Struct containing properties of files is an iNo
  - Size, owner id, time of mod etc.
  - All inodes are the same size
  - inode table is an array of inode structs
  - Each inode has a position id in the inode table
    - For example, inode 2 is the 3rd struct in the inode table of the file system
- Data area : Contains the file content. A file can be ≥ 1 block

Creating a File



Steps to Create a File

Four main operations of creating a file

1) Store properties : Kernel looks for a free inode Here it finds inode47. The kernel records file information in this inode.

2) Store data : This file requires 3 blocks of storage. The kernel finds 3 disk blocks from its list of free blocks : # 627, 200, 992. The content is stored in that order.

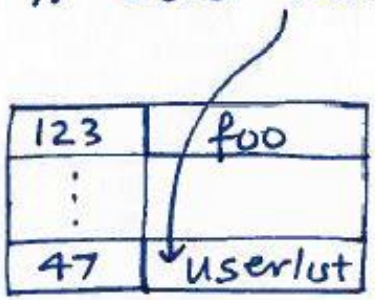
3) Record Allocation : The kernel records the sequence of block numbers in the disk allocation section of the inode.

4) Add Filename to the Directory : Kernel adds the filename and inode# association

How does “cat userlist” work

1) Search the Directory for the filename
2) Locate and Read inode
3) Go to the data blocks one by one in sequence

% cat userlist



From filename to file content
- Search the directory
- find the associated inum
- use the inum to locate inode
- read inode to access data blocks

Characteristics of Files

A Five-Year Study of File-System Metadata

NITIN AGRAWAL
University of Wisconsin, Madison
and
WILLIAM J. BOLOSKY, JOHN R. DOUCEUR, and JACOB R. LORCH
Microsoft Research

- Most files are small
- Most of the space is occupied by the rare big ones

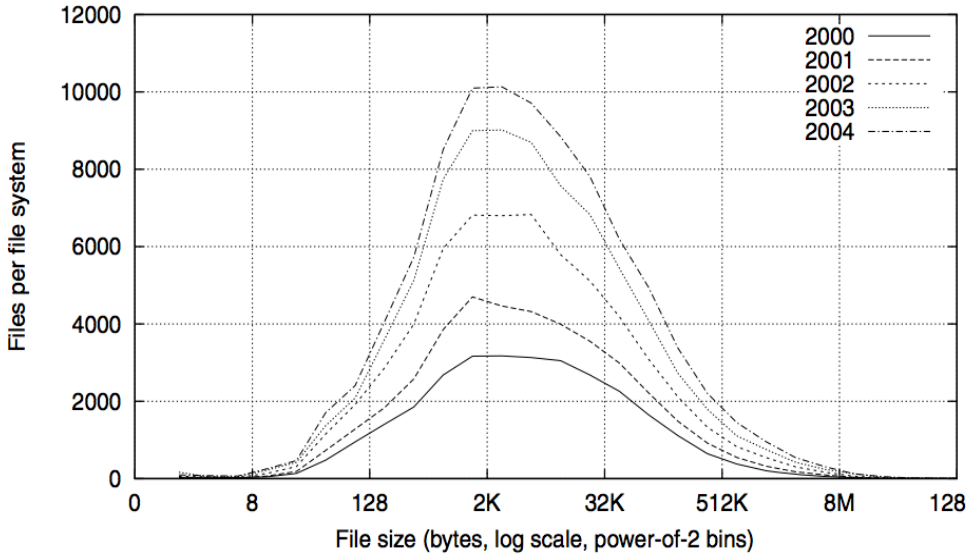


Fig. 2. Histograms of files by size.

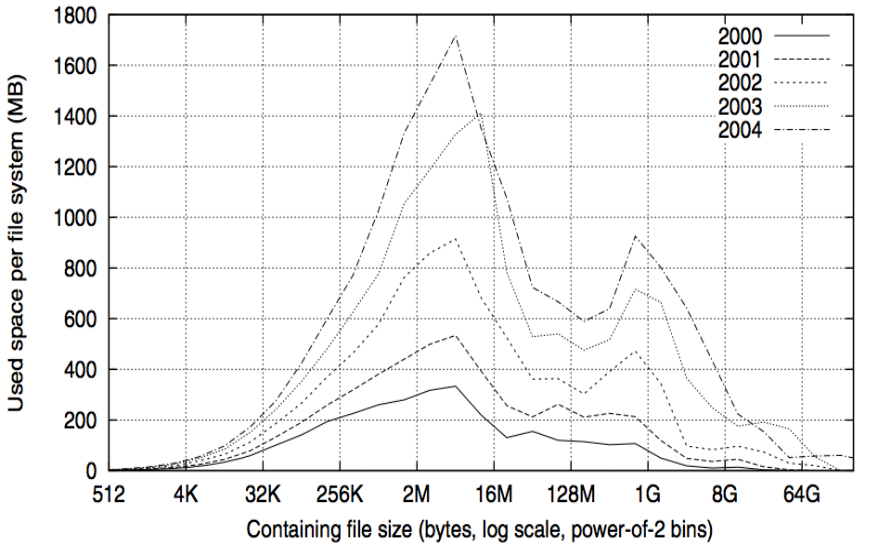
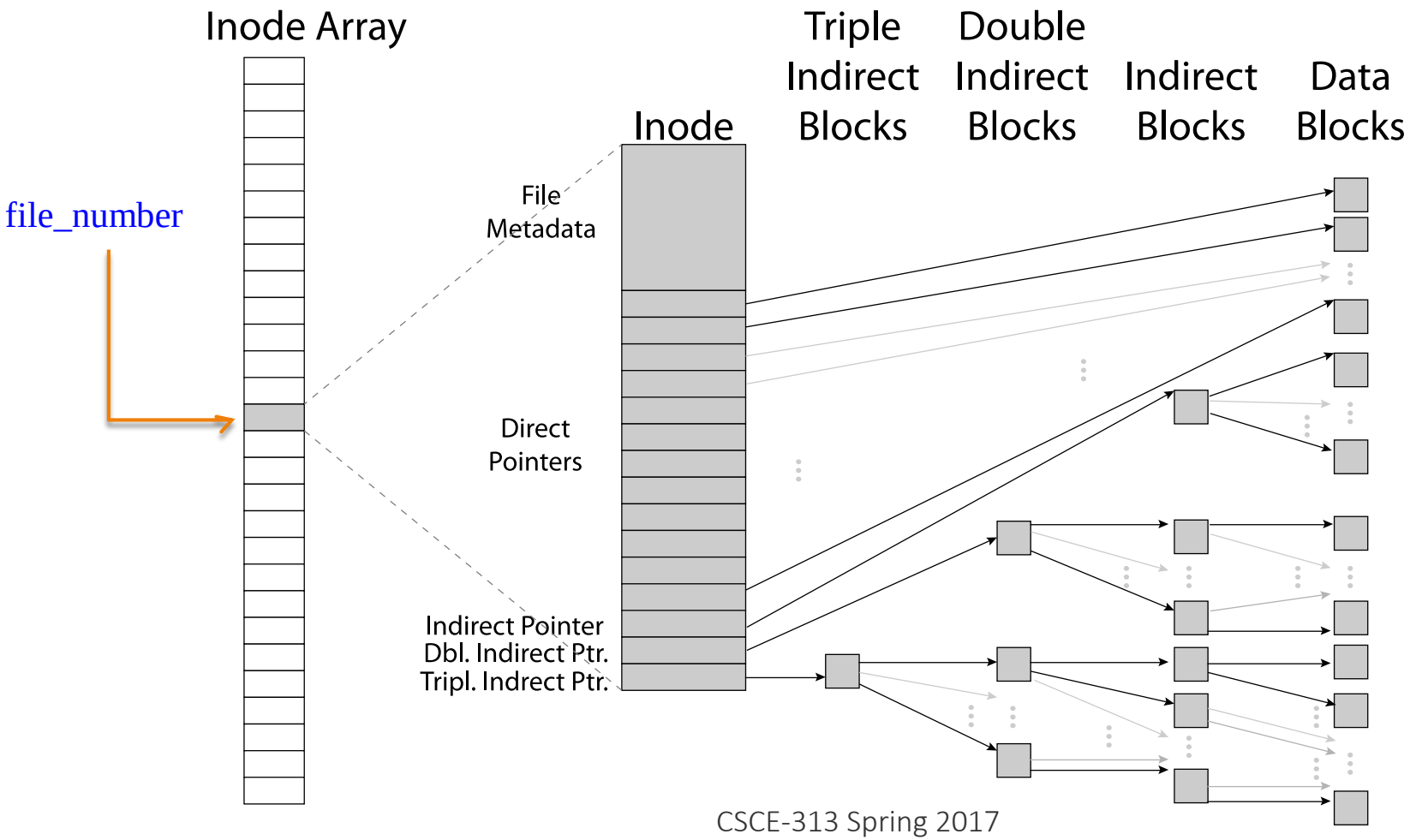


Fig. 4. Histograms of bytes by containing file size.

So what about a “real” file system

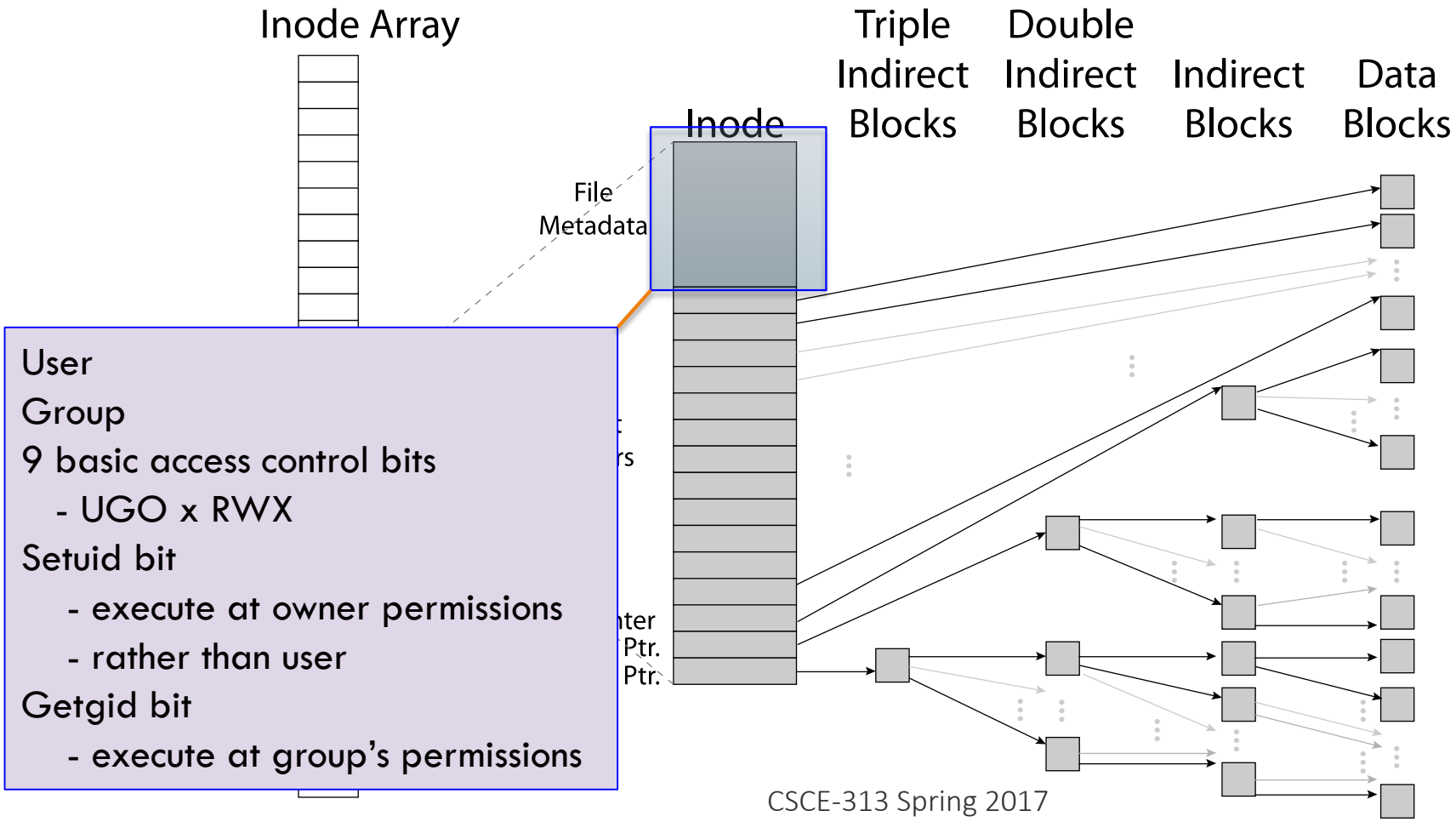


Unix Fast File System

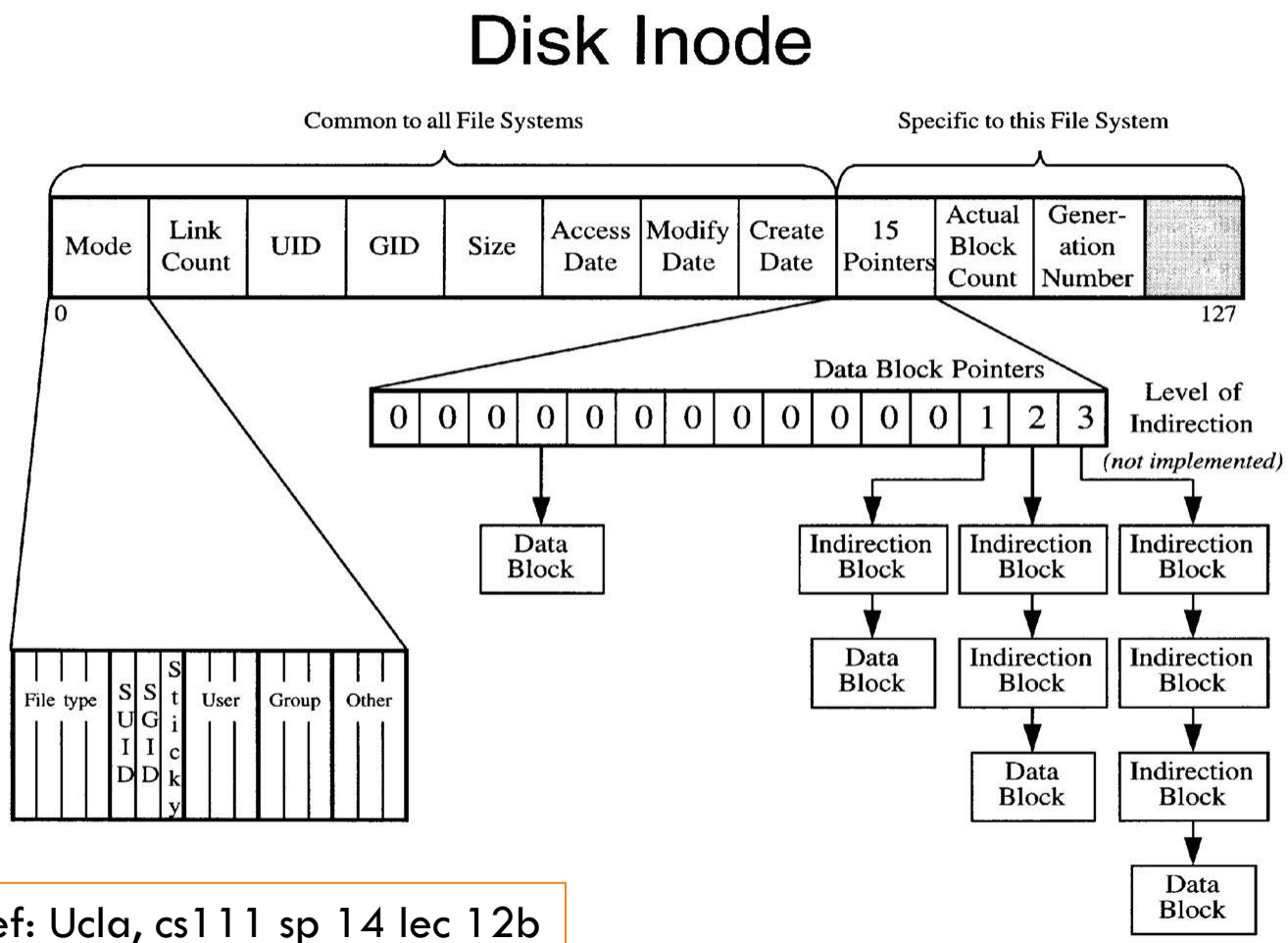
- File or iNode Number is index into inode arrays
- Multi-level index structure
  - Great for little to large
  - Asymmetric tree with fixed sized blocks
- Metadata associated with the file
  - Rather than in the directory that points to it
- Locality Heuristics
  - Block group placement
  - Reserve space
- Scalable directory structure

FFS: File Attributes

- Inode metadata



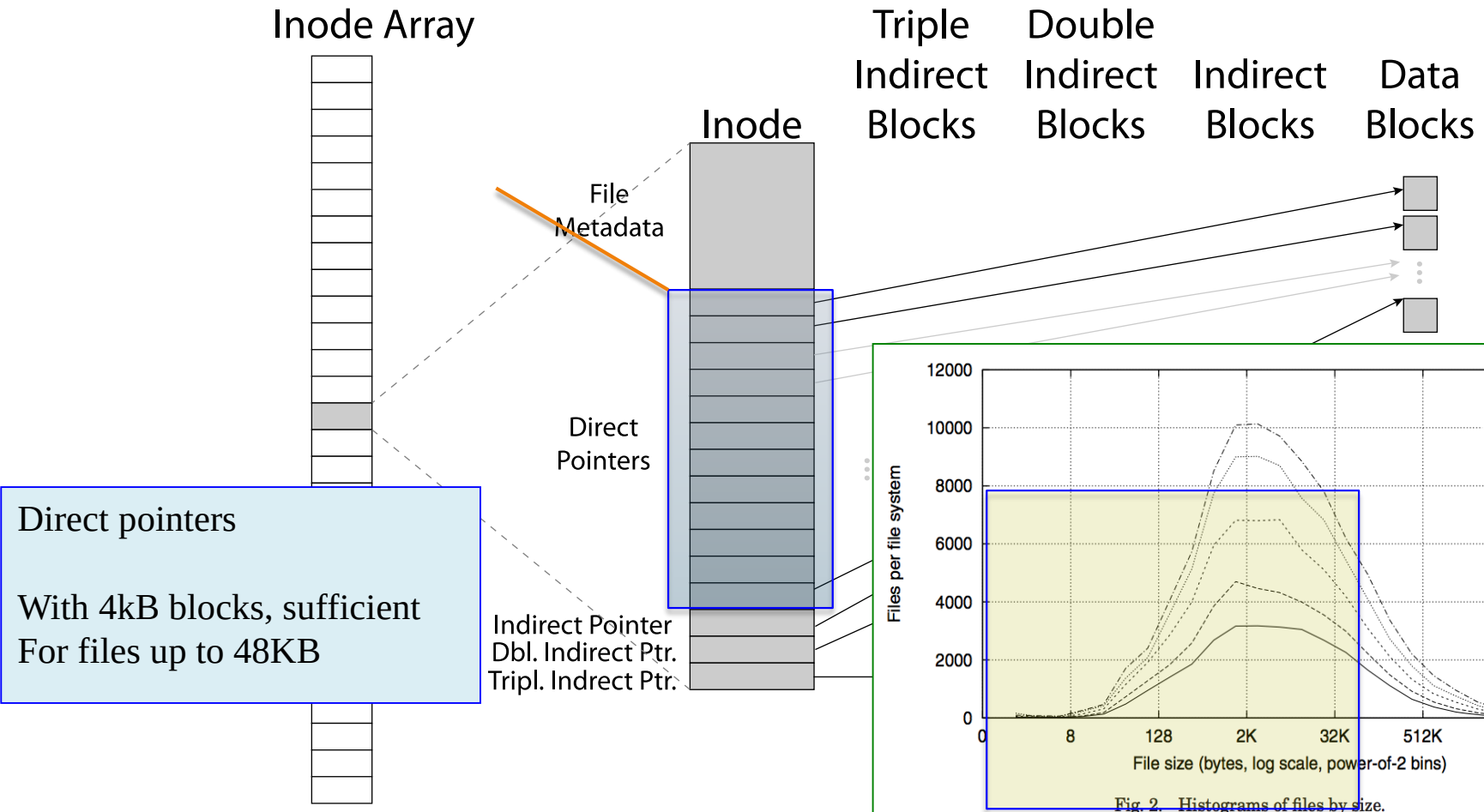
Disk Inode



Ref: Ucla, cs111 sp 14 lec 12b

FFS: Data Storage

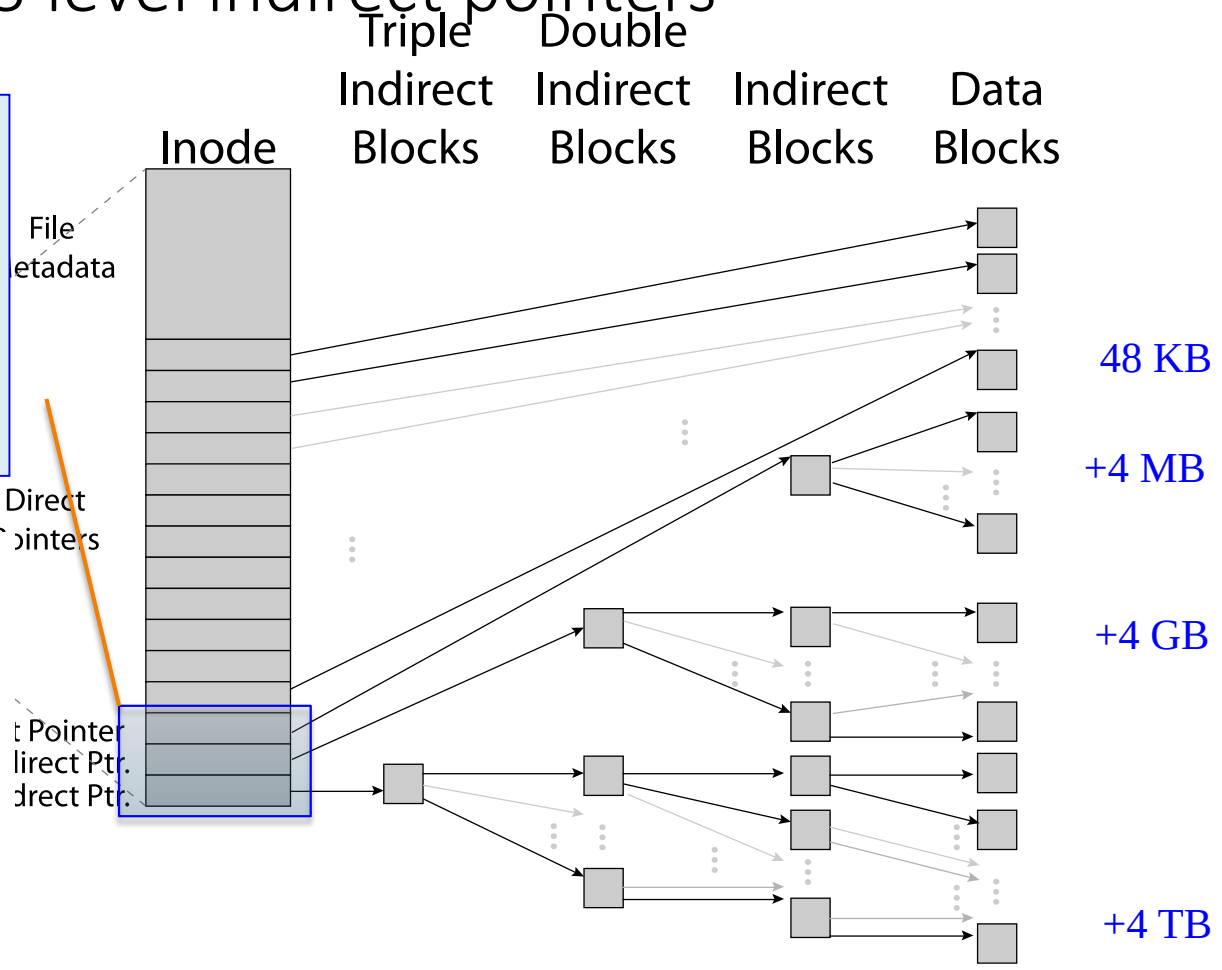
- Small files: 12 pointers direct to data blocks



FFS: Data Storage

- Large files: 1,2,3 level indirect pointers

Indirect pointers
- point to a disk block containing only pointers
- eg. 4 KB blocks => 1024 pointers
  => 4 MB @ level 2
  => 4 GB @ level 3
  => 4 TB @ level 4



A Five-Year Study of File-System Metadata • 9:9

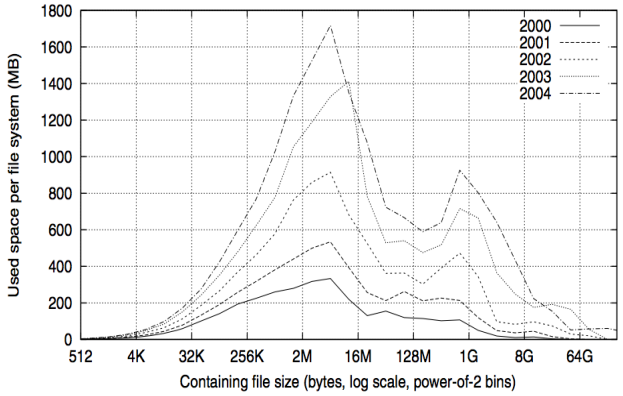
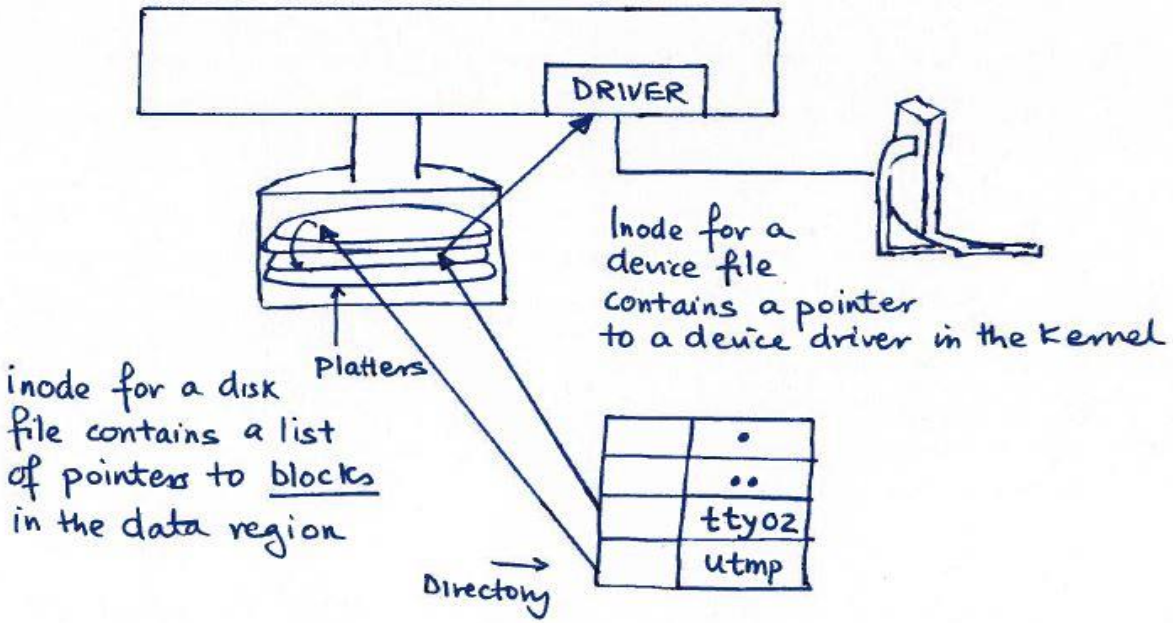


Fig. 4. Histograms of bytes by containing file size.

Device Files and iNodes



- The distinction between type of files occurs at inode level
- An inode can be a disk file inode or a device file inode

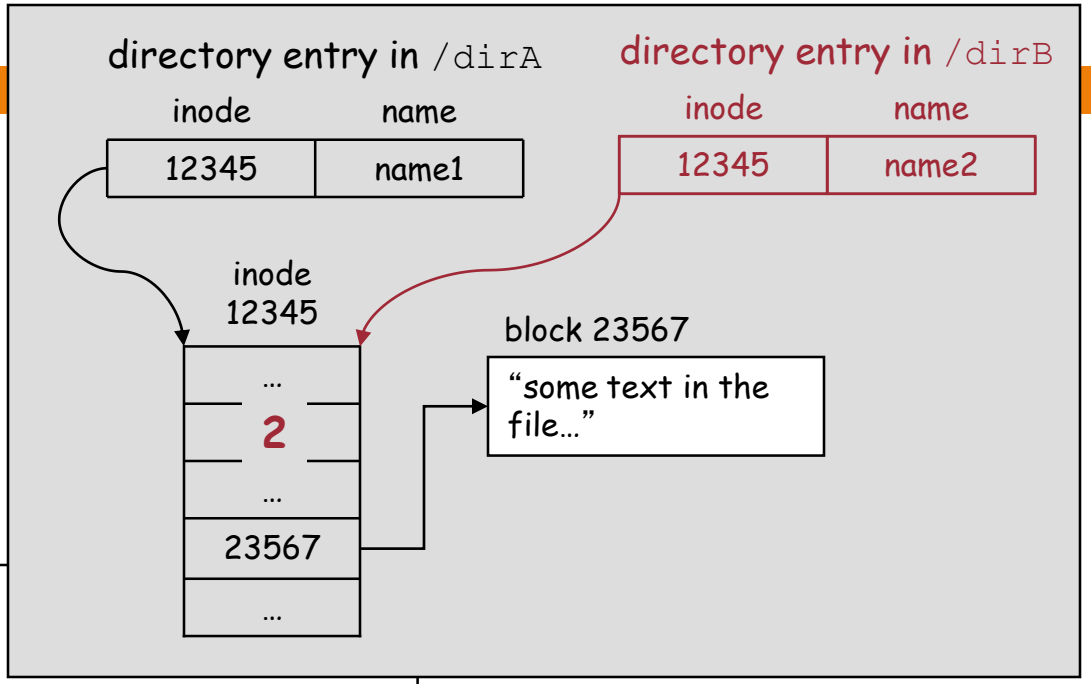
points to the blocks on the disk holding data

contains a pointer to a table of kernel subroutines

Links

- Hard link
  - Sets another directory entry to contain the file number for the file
  - Creates another name (path) for the file
  - Each is “first class”
- Soft link or Symbolic Link
  - Directory entry contains the name of the file
  - Map one name to another name

Hard Links



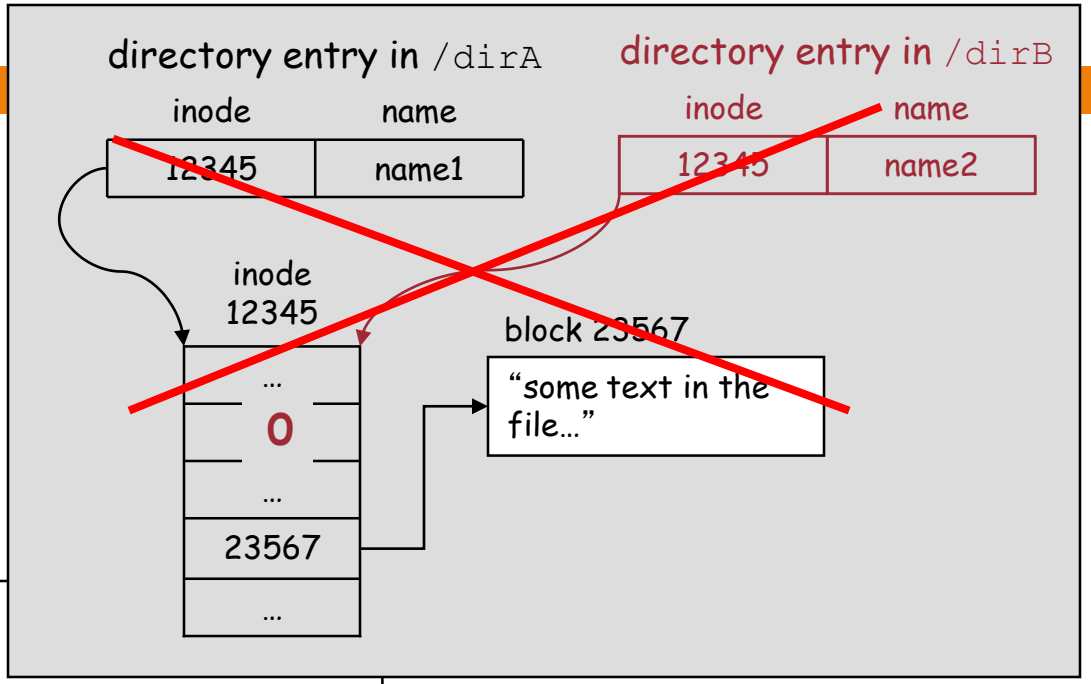
shell command

```
ln /dirA/name1 /dirB/name2
```

is typically implemented using the link system call:

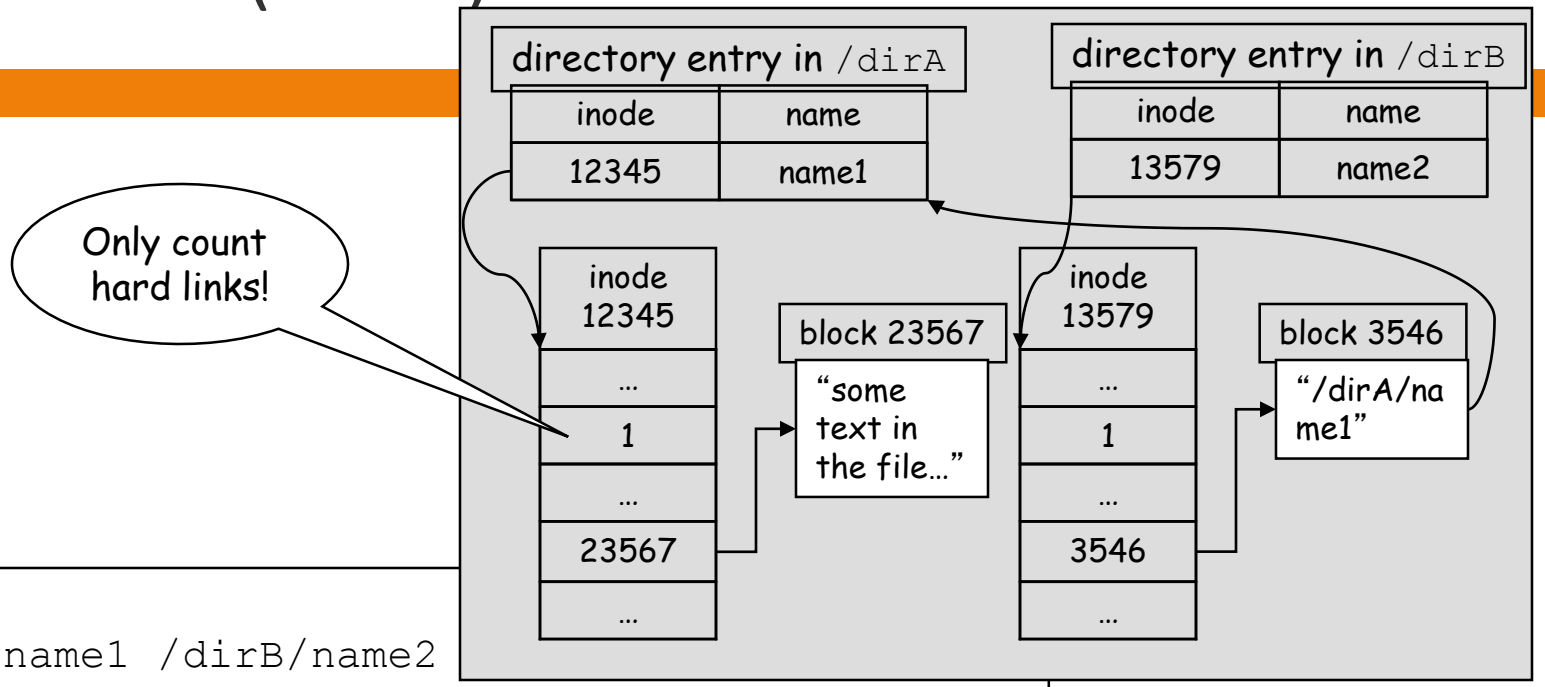
```
#include <stdio.h>
#include<unistd.h>

if (link("/dirA/name1", "/dirB/name2") == -1)
  perror("failed to make new link in /dirB");
```

Hard Links: `unlink`




```
#include <stdio.h>
#include<unistd.h>

if (unlink("/dirA/name1") == -1)
  perror("failed to delete link in /dirA");

if (unlink("/dirB/name2") == -1)
  perror("failed to delete link in /dirB");
```

Symbolic (Soft) Links



shell command

```
ln -s /dirA/name1 /dirB/name2
```

is typically implemented using the link system call:

```
#include <stdio.h>
#include<unistd.h>

if (symlink("/dirA/name1", "/dirB/name2") == -1)
  perror("failed to create symbolic link in /dirB");
```

Directory Traversal – Example 2

File Commands - Working example

```
% cd demodir
% cp /etc/group x
% cat x
root :: 0 :
bin :: 1 : bin, daemon
users :: 200 :
% cp x copy.of.x
% mv copy.of.x y
% mv x a
% cd c
% cp ../a/x d2/xcopy
% ln ../a/x d1/xlink
% ls > d1/xlink
% cp d1/xlink z
% rm ../../demodir/c/d2/../z
% cd ../..
% cat demodir/a/x
```

(what appears here?)

<space for working the directory structure

cp: copies file or directory
cat: list the contents of a file
ln: hard link
ls: list the contents of directory

Simple shell code

```
while true
do
    mkdir deep-well
    cd deep-well
done
```

what would this produce

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom

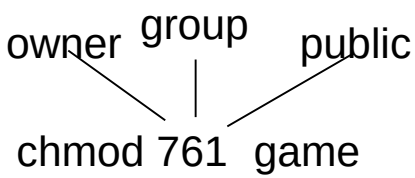
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

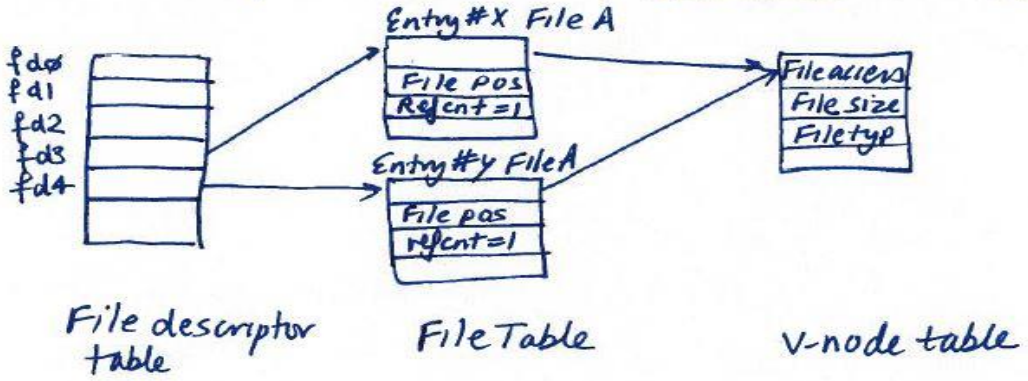
chgrp G game

A Sample UNIX Directory Listing


```
[tyagi]@linux2 ~/csce313/fa14> (17:01:25 02/13/15)
:: ls -lrt
total 14
drwxr-xr-x 2 tyagi games 7 Nov  5 18:17 pipes
drwxr-xr-x 2 tyagi games 9 Nov 12 14:59 shmem
drwxr-xr-x 4 tyagi games 4 Nov 18 09:28 demodir
drwxr-xr-x 5 tyagi games 5 Dec  3 17:48 network
drwxr-xr-x 2 tyagi games 3 Dec 10 16:20 fifo
drwxr-xr-x 2 tyagi games 7 Dec 11 18:28 signals
drwxr-xr-x 6 tyagi games 6 Dec 12 15:56 unixio

[tyagi]@linux2 ~/csce313/fa14> (17:01:28 02/13/15)
:: ln -s unixio/f3/foo .

[tyagi]@linux2 ~/csce313/fa14> (17:02:02 02/13/15)
:: ls -lrt
total 15
drwxr-xr-x 2 tyagi games  7 Nov  5 18:17 pipes
drwxr-xr-x 2 tyagi games  9 Nov 12 14:59 shmem
drwxr-xr-x 4 tyagi games  4 Nov 18 09:28 demodir
drwxr-xr-x 5 tyagi games  5 Dec  3 17:48 network
drwxr-xr-x 2 tyagi games  3 Dec 10 16:20 fifo
drwxr-xr-x 2 tyagi games  7 Dec 11 18:28 signals
drwxr-xr-x 6 tyagi games  6 Dec 12 15:56 unixio
lrwxrwxrwx 1 tyagi games 13 Feb 13 17:02 foo -> unixio/f3/foo

[tyagi]@linux2 ~/csce313/fa14> (17:02:06 02/13/15)
::
```

Recap: File Connections

- Each open connection is represented by a File Descriptor
- Two open connections to the same file may exist at two different locations in the file
- Each process has its own File Descriptor table representing all of its open "file" connections
- File descriptor table establishes connection to the Open File Table data structure inside the kernel.
- Each entry inside the Open file table represents a connection to a file, # of file descriptors pointing to it, and pointer to the Vnode table that contains details.



Big Picture tying User Space, Kernel Space, and Disk File