



BOXES

ideas worth spreading

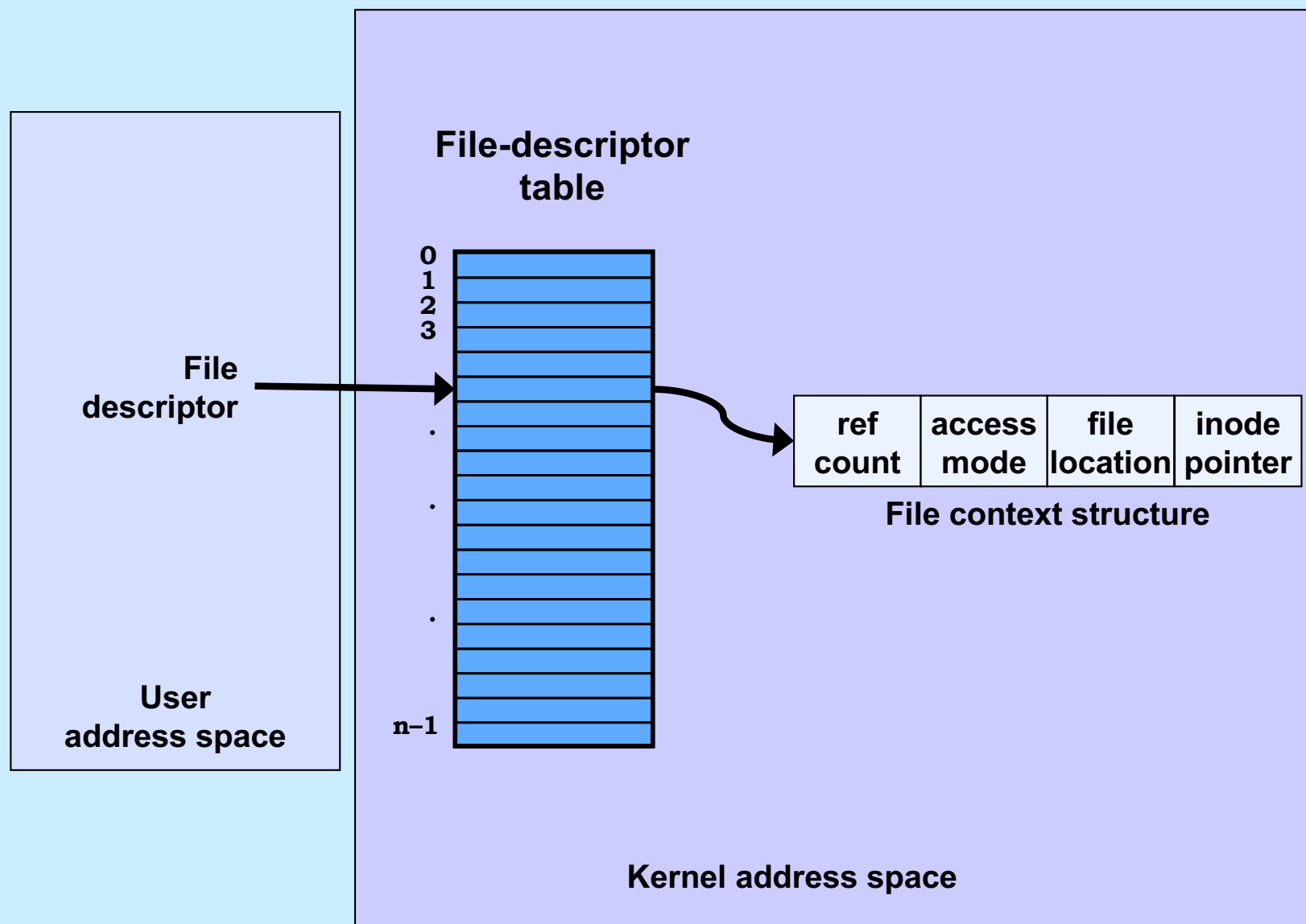
TED^x BrownU
x = independently organized TED event

october 25, 2019
7-9pm @ Salomon DECI

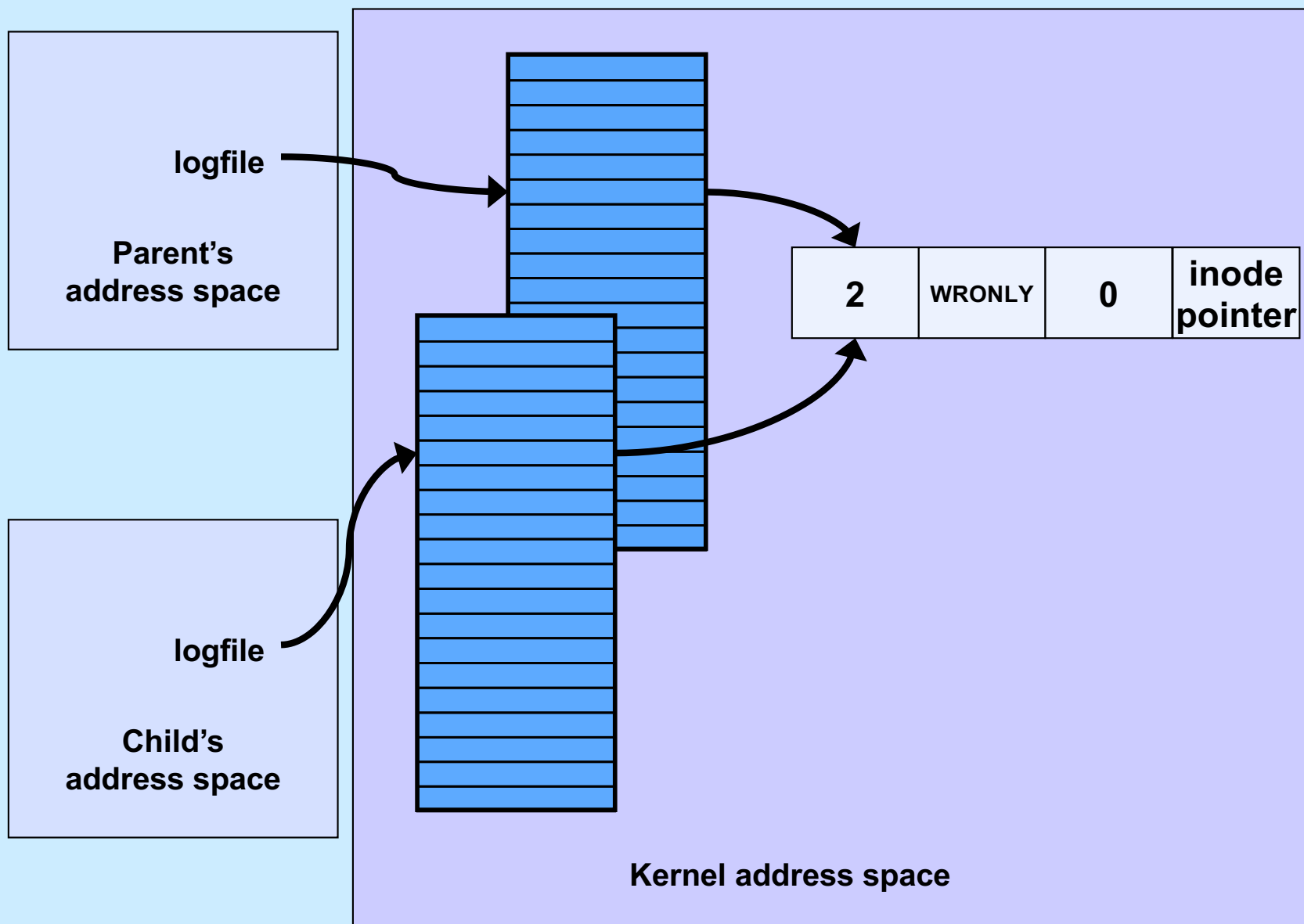
CS 33

Files Part 2

File-Descriptor Table



File Descriptors After Fork



Fork and File Descriptors

```
int logfile = open("log", O_WRONLY);
if (fork() == 0) {
    /* child process computes something, then does: */
    write(logfile, LogEntry, strlen(LogEntry));
    ...
    exit(0);
}

/* parent process computes something, then does: */

write(logfile, LogEntry, strlen(LogEntry));
...
```

Quiz 1

```
int main() {  
    if (fork() == 0) {  
        fprintf(stderr, "Child");  
        exit(0);  
    }  
    fprintf(stderr, "Parent");  
}
```

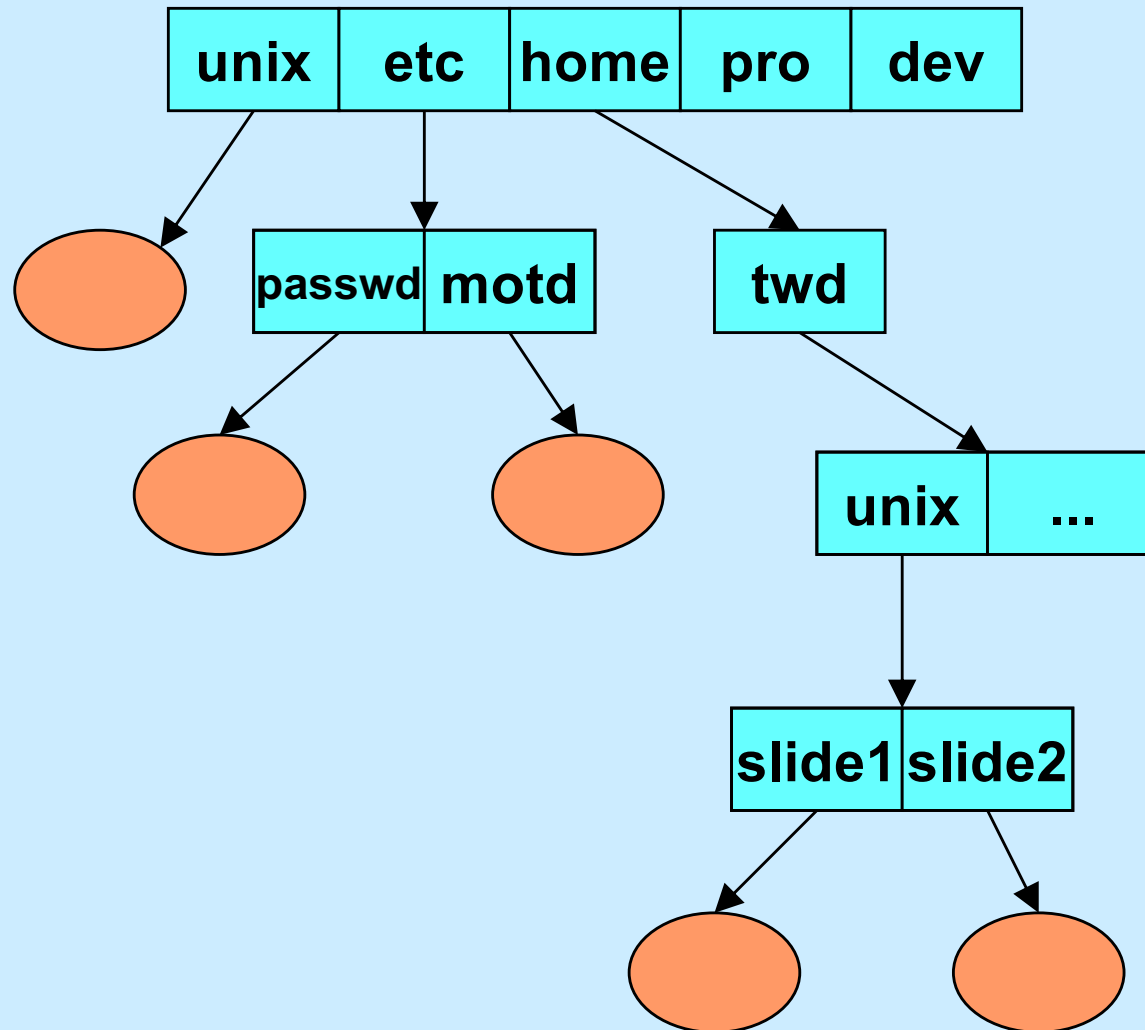
Suppose the program is run as:

```
$ prog >file 2>&1
```

What is the final content of file? (Assume writes are “atomic”.)

- a) either “ChildParent” or “ParentChild”
- b) either “Childt” or “Parent”
- c) either “Child” or “Parent”

Directories



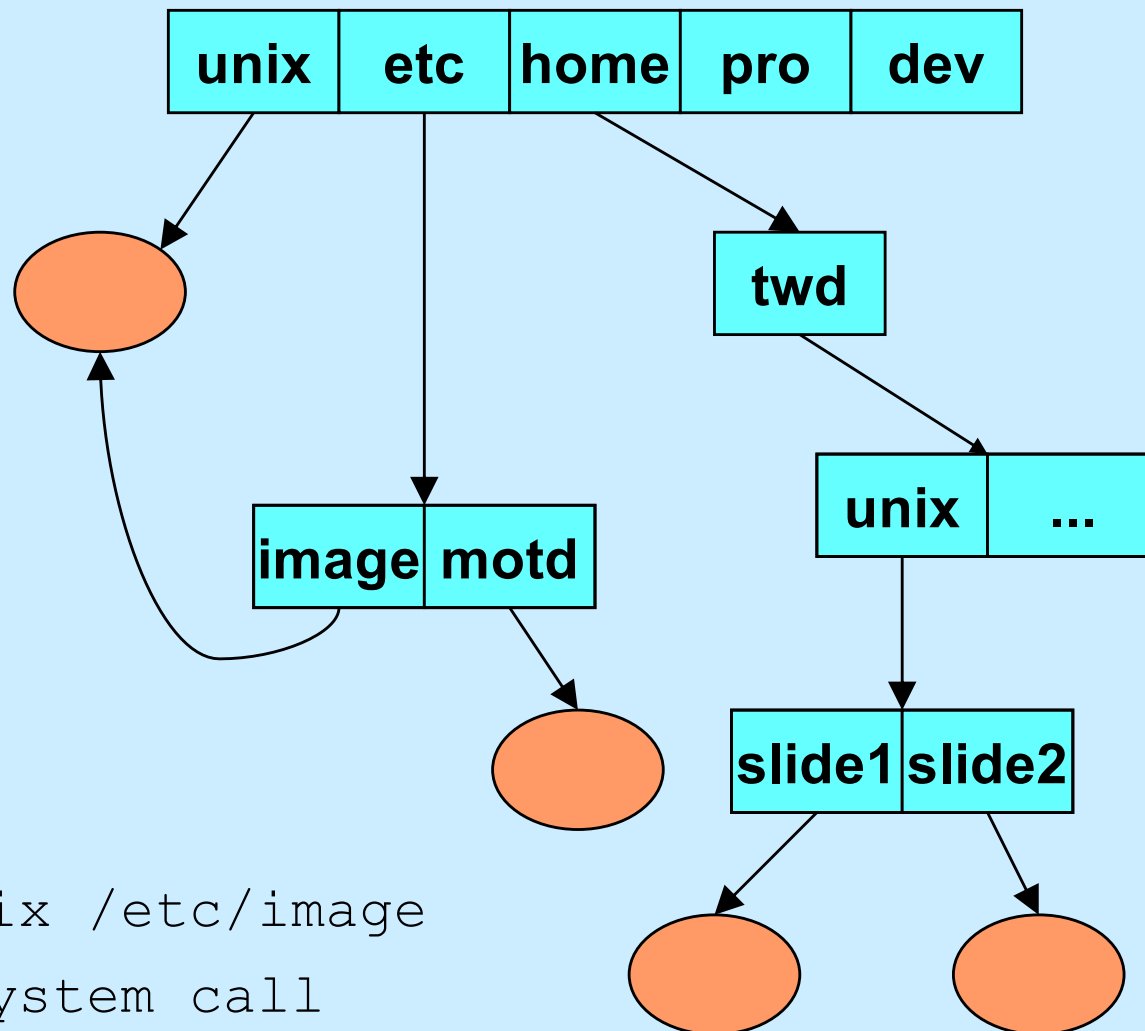
Directory Representation

Component Name	Inode Number
----------------	--------------

directory entry

.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

Hard Links



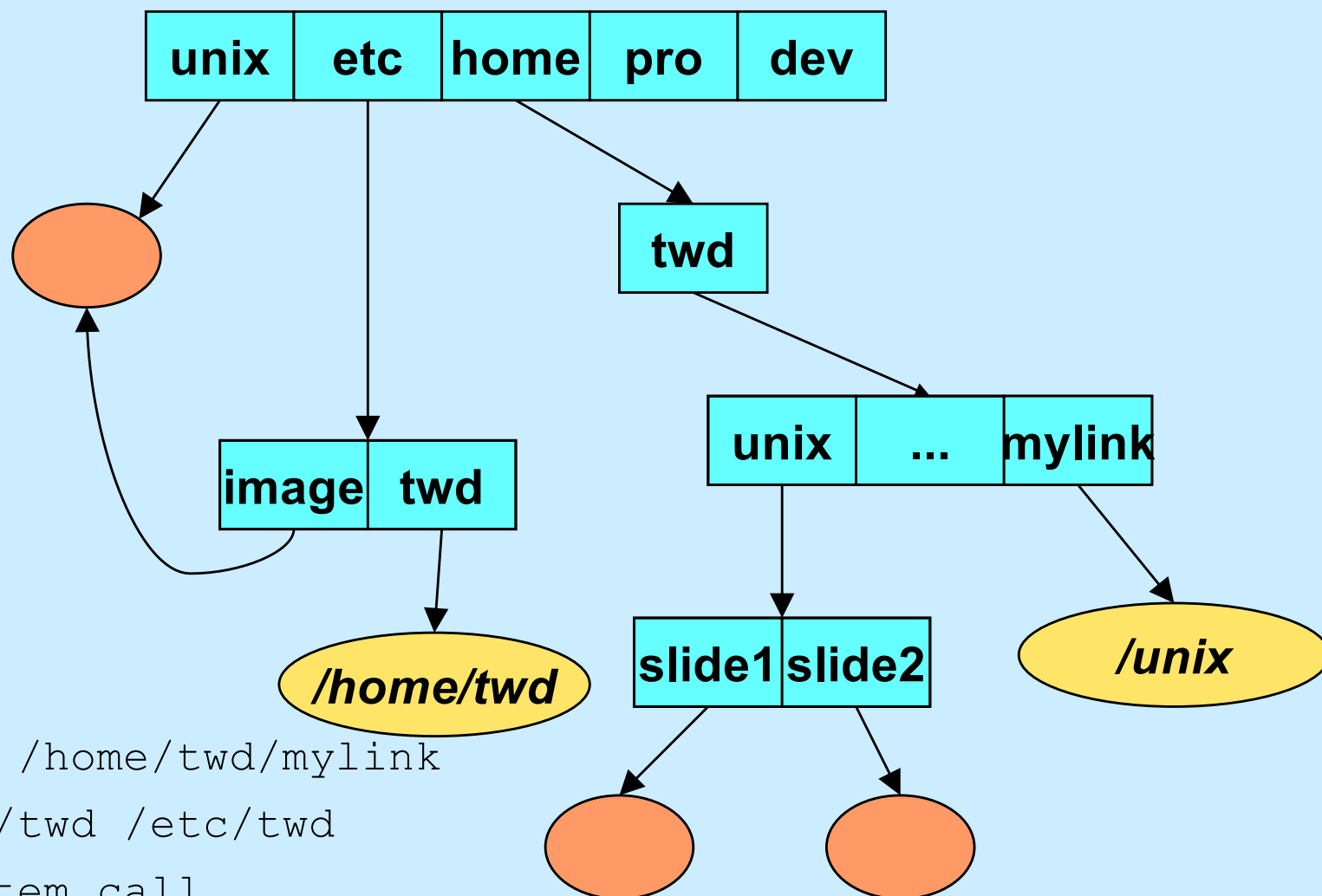
```
$ ln /unix /etc/image  
# link system call
```

Directory Representation

.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

.	4
..	1
image	117
motd	33

Symbolic Links

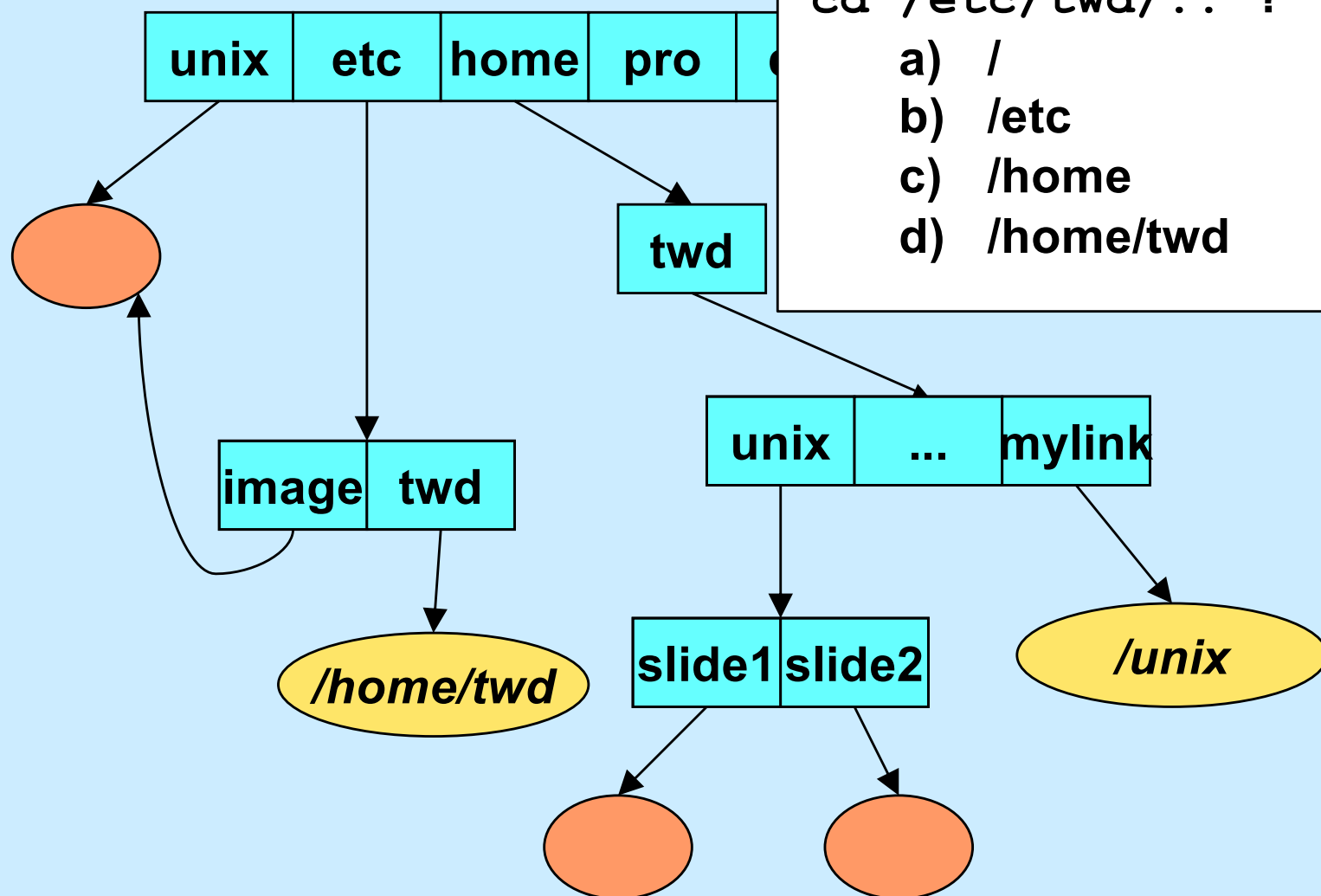


```
% ln -s /unix /home/twd/mylink
% ln -s /home/twd /etc/twd
# symlink system call
```

Working Directory

- **Maintained in kernel for each process**
 - paths not starting from “/” start with the working directory
 - changed by use of the *chdir* system call
 - » *cd* shell command
 - displayed (via shell) using “pwd”
 - » how is this done?

Symbolic Links



Quiz 2

What is the working directory after doing

`cd /etc/twd/.. ?`

- a) /
- b) /etc
- c) /home
- d) /home/twd

Open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int options [, mode_t mode])
```

– options

- » **O_RDONLY** open for reading only
- » **O_WRONLY** open for writing only
- » **O_RDWR** open for reading and writing
- » **O_APPEND** set the file offset to *end of file* prior to each *write*
- » **O_CREAT** if the file does not exist, then create it, setting its mode to *mode* adjusted by *umask*
- » **O_EXCL** if **O_EXCL** and **O_CREAT** are set, then *open* fails if the file exists
- » **O_TRUNC** delete any previous contents of the file
- » **O_NONBLOCK** don't wait if I/O can't be done immediately

File Access Permissions

- **Who's allowed to do what?**
 - **who**
 - » **user (owner)**
 - » **group**
 - » **others (rest of the world)**
 - **what**
 - » **read**
 - » **write**
 - » **execute**

Permissions Example

**adm group:
tom, trina**

```
$ ls -lR
```

```
..:
```

```
total 2
```

```
drwxr-x--x  2 tom      adm      1024 Dec 17 13:34 A
drwxr----- 2 tom      adm      1024 Dec 17 13:34 B
```

```
./A:
```

```
total 1
```

```
-rw-rw-rw-   1 tom      adm        593 Dec 17 13:34 x
```

```
./B:
```

```
total 2
```

```
-r--rw-rw-   1 tom      adm        446 Dec 17 13:34 x
-rw----rw-   1 trina    adm        446 Dec 17 13:45 y
```


Setting File Permissions

```
#include <sys/types.h>
#include <sys/stat.h>
int chmod(const char *path, mode_t mode)
```

- sets the file permissions of the given file to those specified in *mode*
- only the owner of a file and the superuser may change its permissions
- nine combinable possibilities for *mode* (*read/write/execute* for *user*, *group*, and *others*)
 - » S_IRUSR (0400), S_IWUSR (0200), S_IXUSR (0100)
 - » S_IRGRP (040), S_IWGRP (020), S_IXGRP (010)
 - » S_IROTH (04), S_IWOTH (02), S_IXOTH (01)

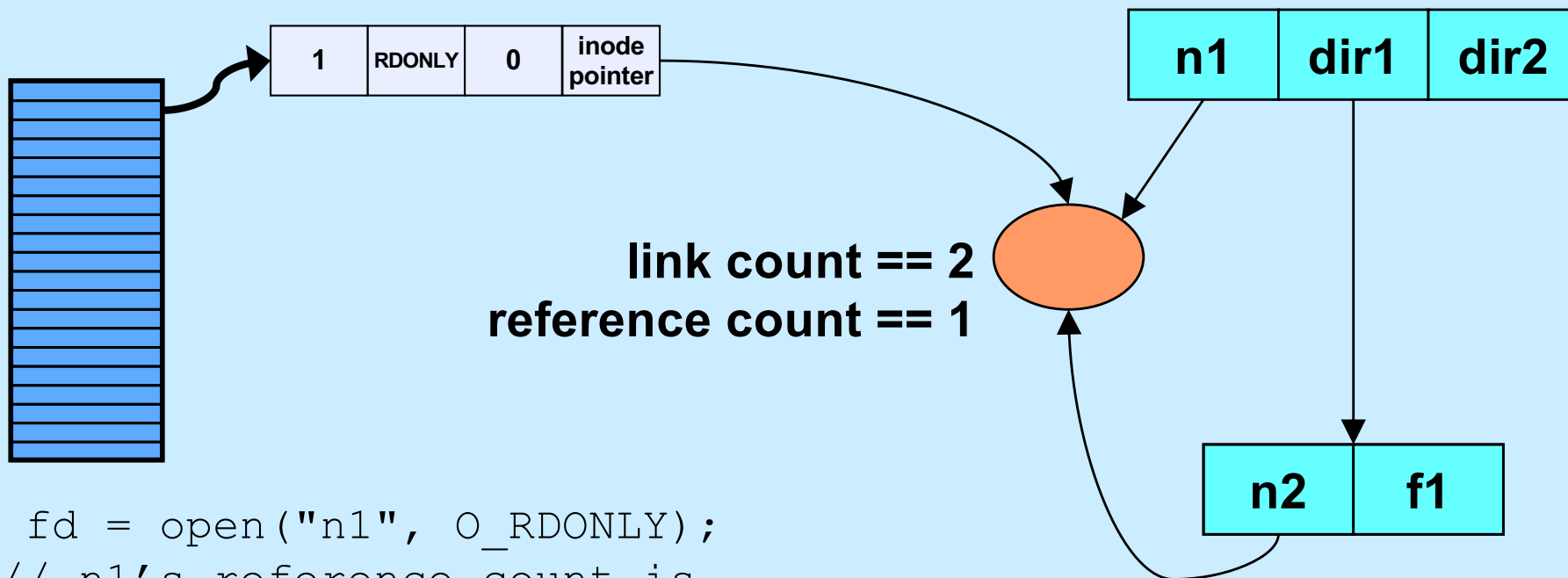
Umask

- **Standard programs create files with “maximum needed permissions” as mode**
 - compilers: 0777
 - editors: 0666
- **Per-process parameter, *umask*, used to turn off undesired permission bits**
 - e.g., turn off all permissions for others, write permission for group: set umask to 027
 - » compilers: permissions = $0777 \& \sim(027) = 0750$
 - » editors: permissions = $0666 \& \sim(027) = 0640$
 - set with *umask* system call or (usually) shell command

Creating a File

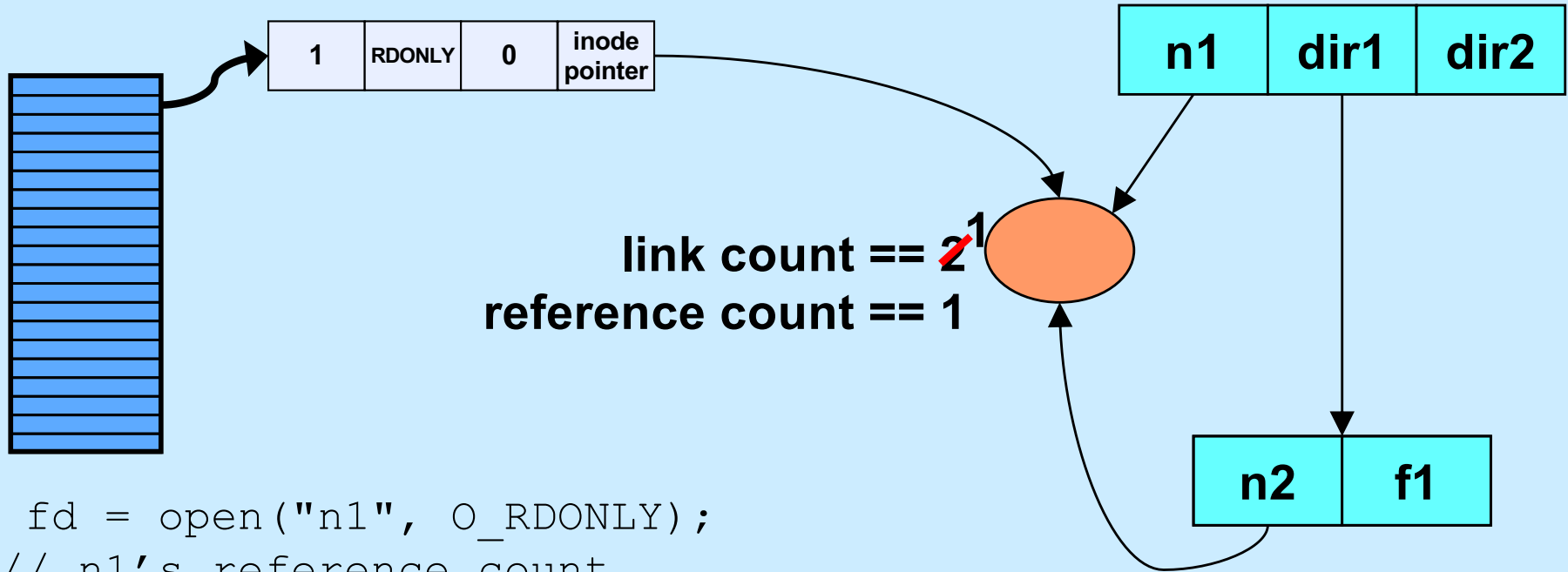
- Use either *open* or *creat*
 - `open(const char *pathname, int flags, mode_t mode)`
 - » flags must include `O_CREAT`
 - `creat(const char *pathname, mode_t mode)`
 - » `open` is preferred
- The *mode* parameter helps specify the permissions of the newly created file
 - `permissions = mode & ~umask`

Link and Reference Counts



```
int fd = open("n1", O_RDONLY);  
// n1's reference count is  
// incremented by 1
```

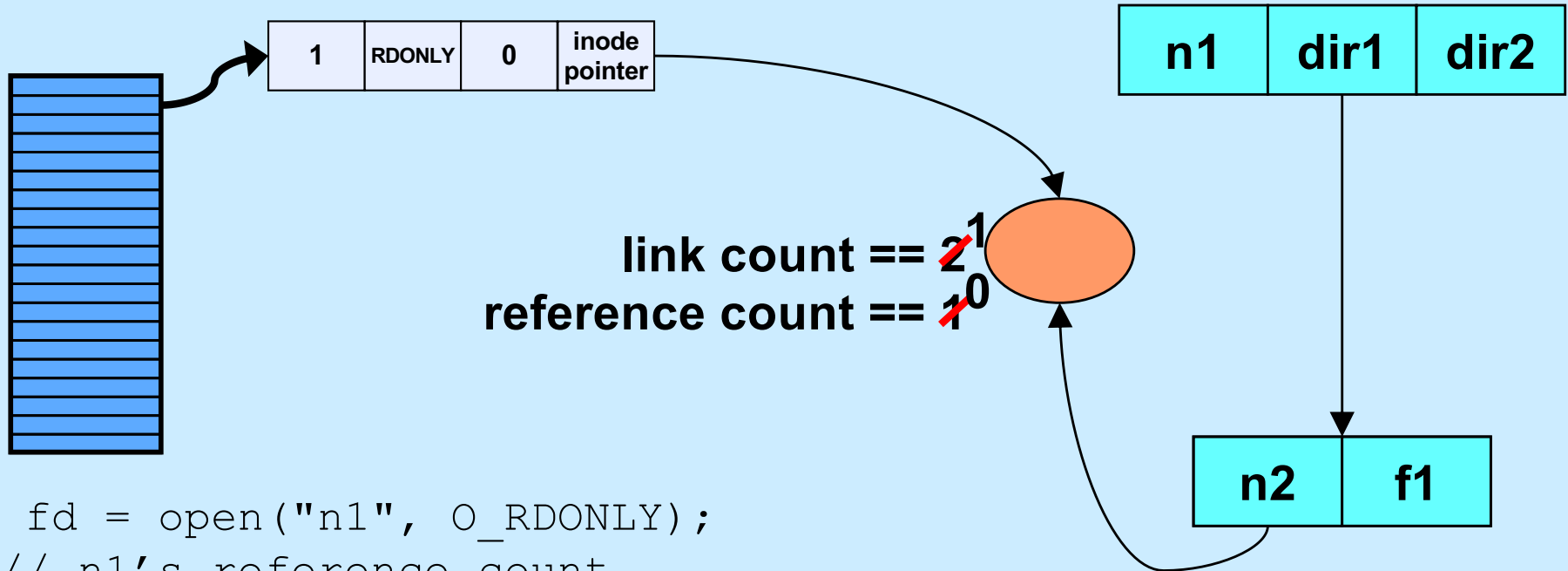
Link and Reference Counts



```
int fd = open("n1", O_RDONLY);  
// n1's reference count  
// incremented by 1
```

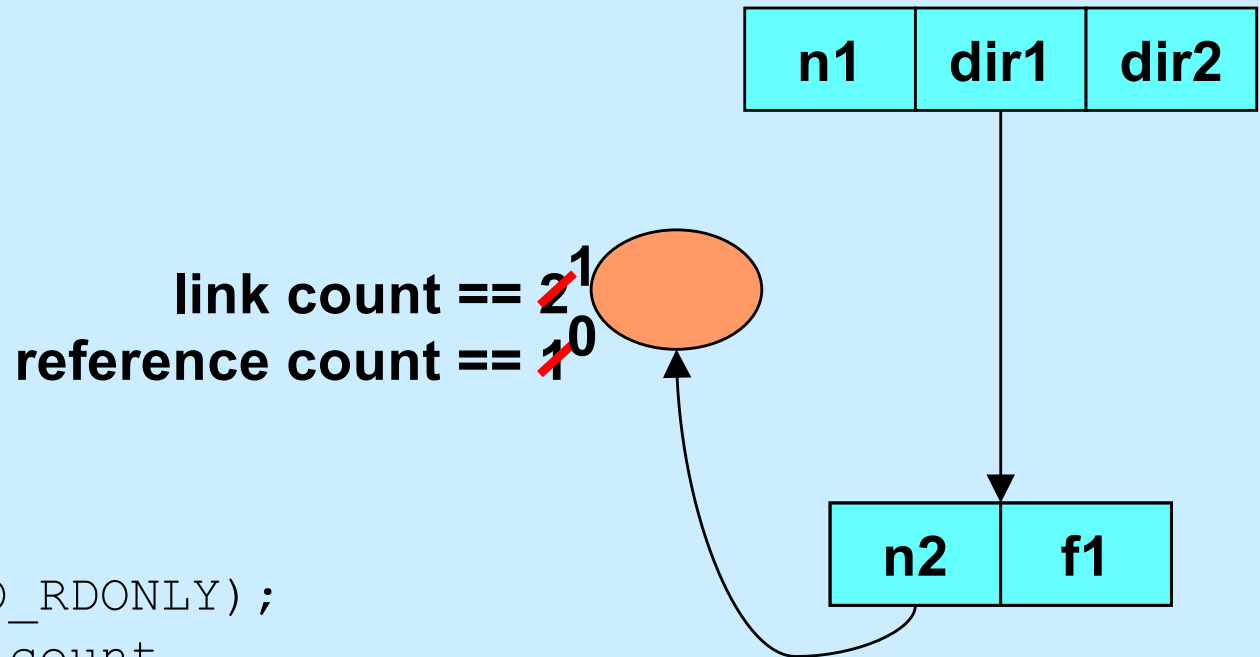
```
unlink("n1");  
// link count decremented by 1  
// same effect in shell via "rm n1"
```

Link and Reference Counts



```
int fd = open("n1", O_RDONLY);  
    // n1's reference count  
    // incremented by 1  
  
unlink("n1");  
    // link count decremented by 1  
  
close(fd);  
    // reference count decremented by 1
```

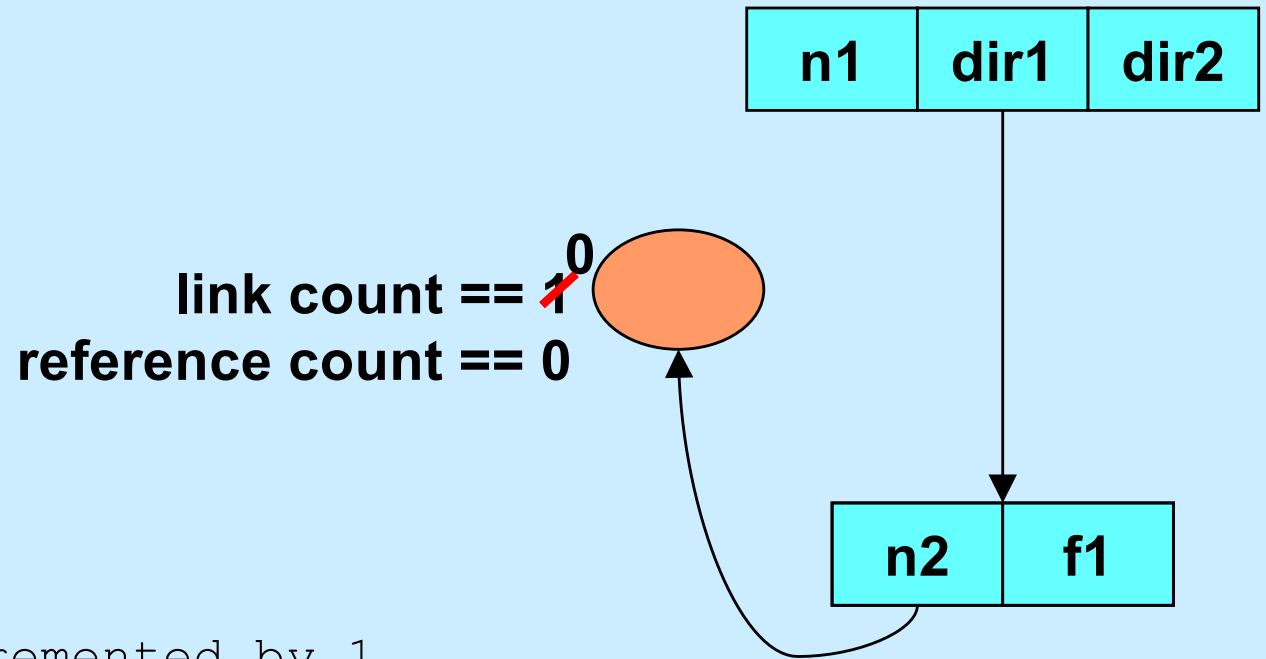
Link and Reference Counts



```
int fd = open("n1", O_RDONLY);  
    // n1's reference count  
    // incremented by 1  
  
unlink("n1");  
    // link count decremented by 1  
  
close(fd);  
    // reference count decremented by 1
```

Link and Reference Counts

```
unlink("dir1/n2");  
// link count decremented by 1
```



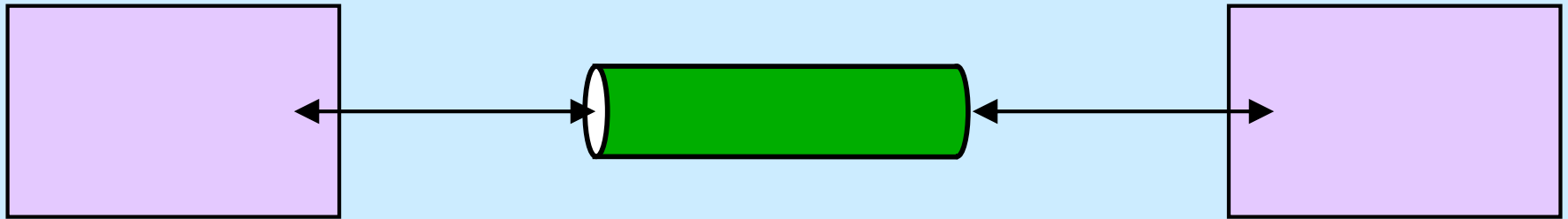
Quiz 3

```
int main() {  
    int fd = creat("file", 0666);  
    unlink("file");  
    PutStuffInFile(fd);  
    ReadStuffFromFile(fd);  
    return 0;  
}
```

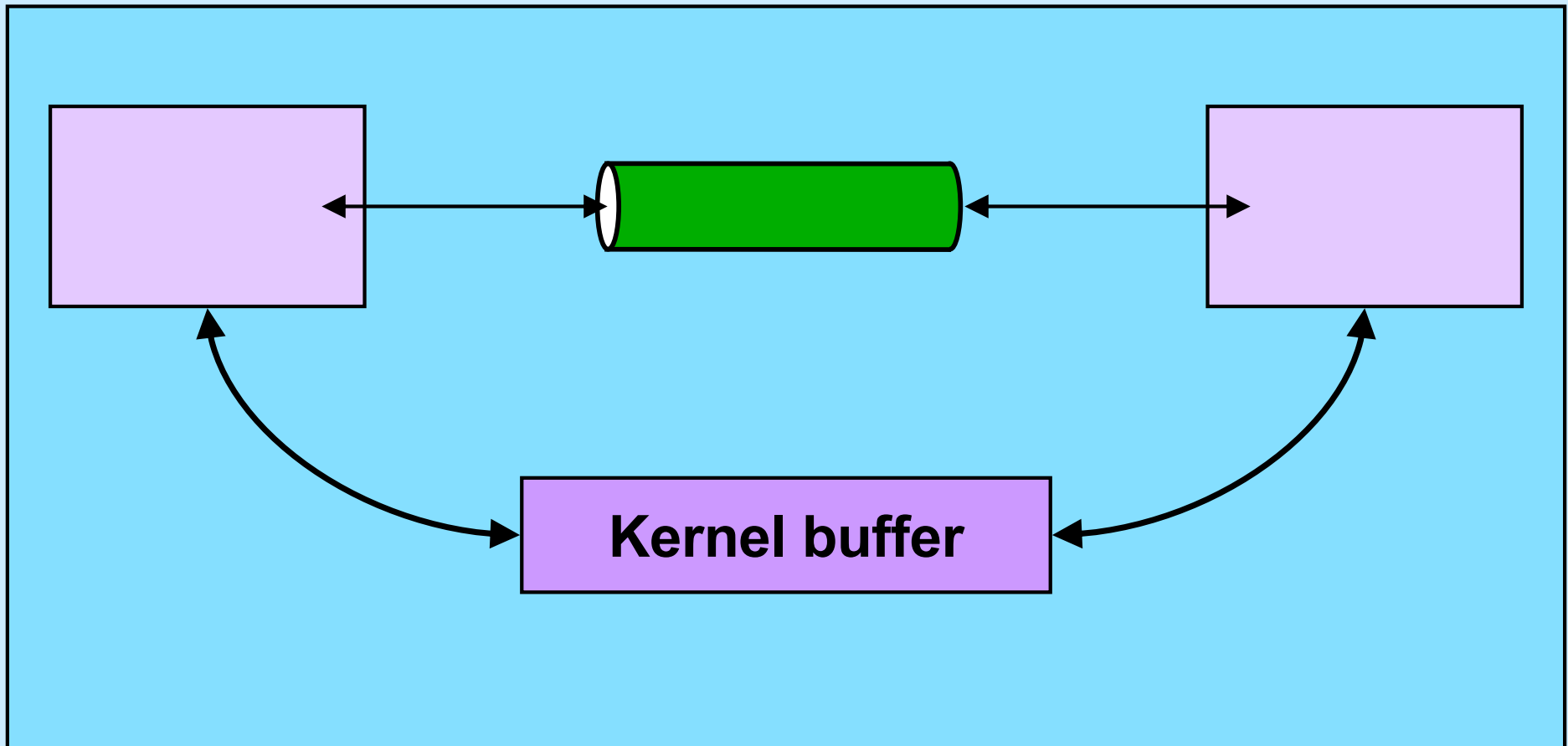
Assume that *PutStuffInFile* writes to the given file, and *ReadStuffFromFile* reads from the file.

- a) This program is doomed to failure, since the file is deleted before it's used**
- b) Because the file is used after the unlink call, it won't be deleted**
- c) The file will be deleted when the program terminates**

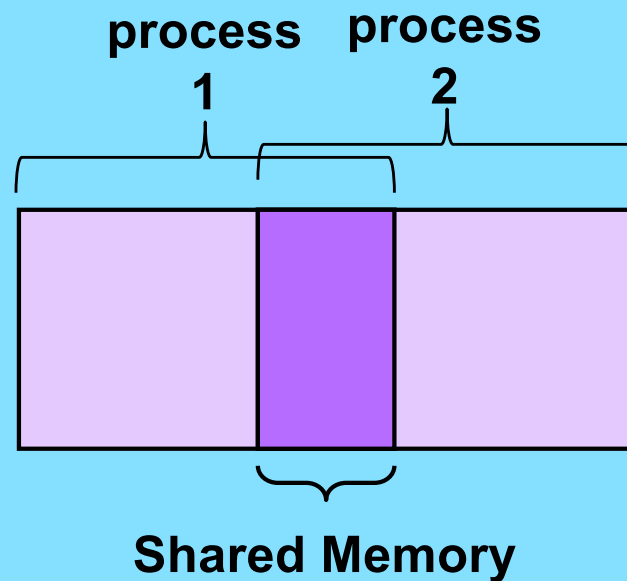
Interprocess Communication (IPC): Pipes



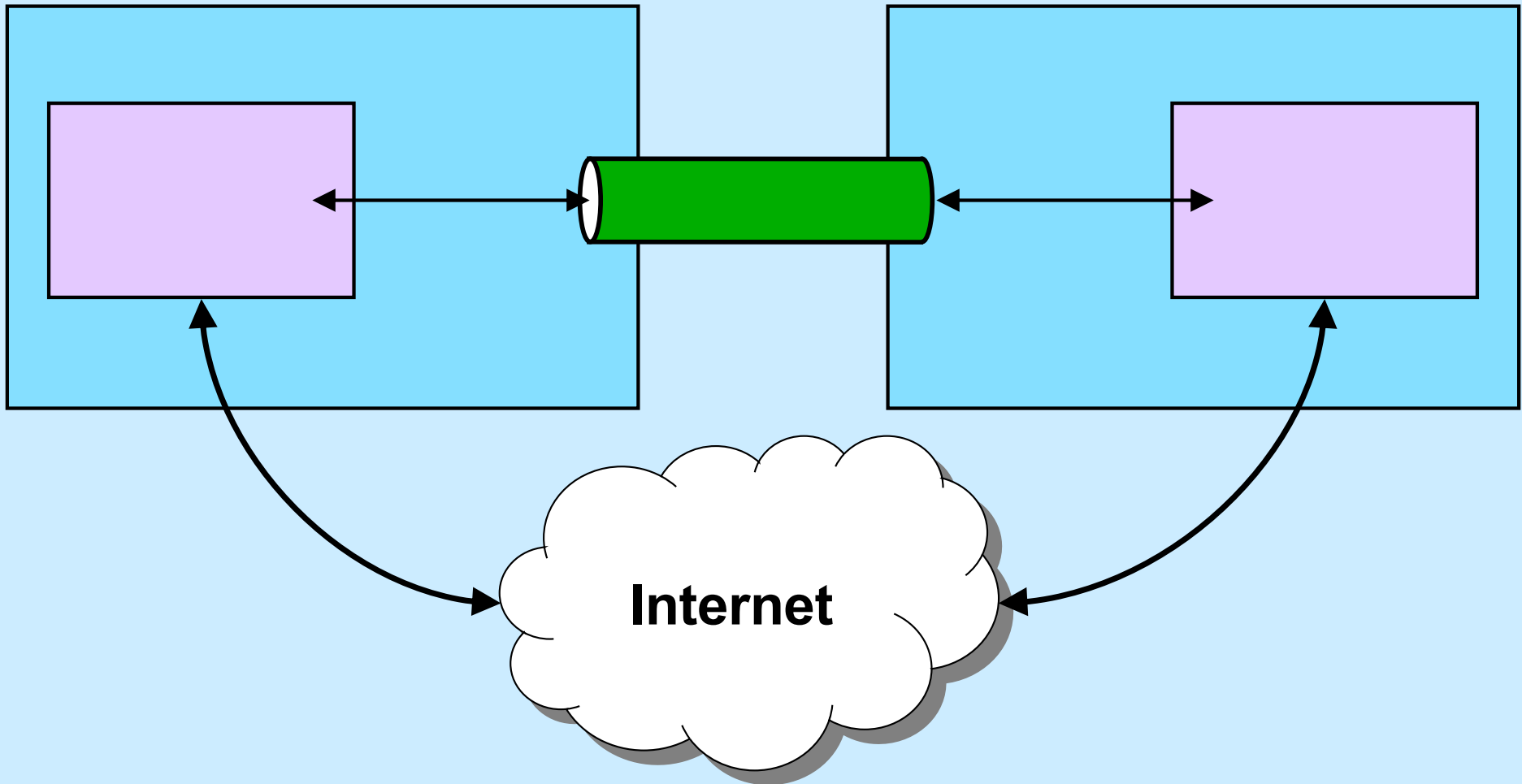
Interprocess Communication: Same Machine I



Interprocess Communication: Same Machine II

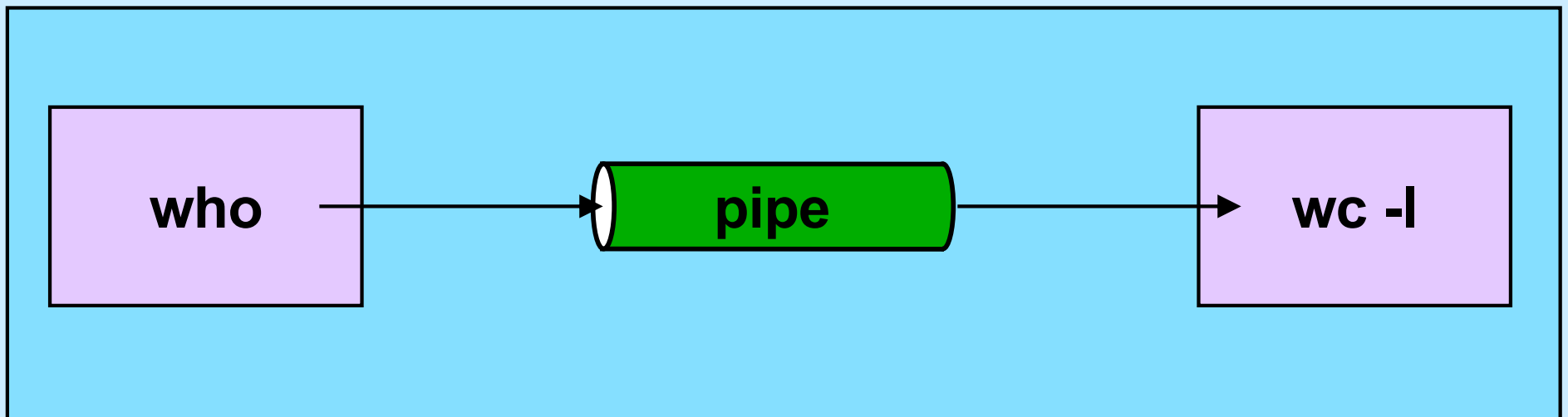


Interprocess Communication: Different Machines



Pipes

```
$cs1ab2e who | wc -l
```



Intramachine IPC

```
$cs1ab2e who | wc -l
```

```
int fd[2];
pipe(fd);
if (fork() == 0) {
    close(fd[0]);
    close(1);
    dup(fd[1]); close(fd[1]);
    execl("/usr/bin/who", "who", 0); // who sends output to pipe
}
if (fork() == 0) {
    close(fd[1]);
    close(0);
    dup(fd[0]); close(fd[0]);
    execl("/usr/bin/wc", "wc", "-l", 0); // wc's input is from pipe
}
close(fd[1]); close(fd[0]);
// ...
```



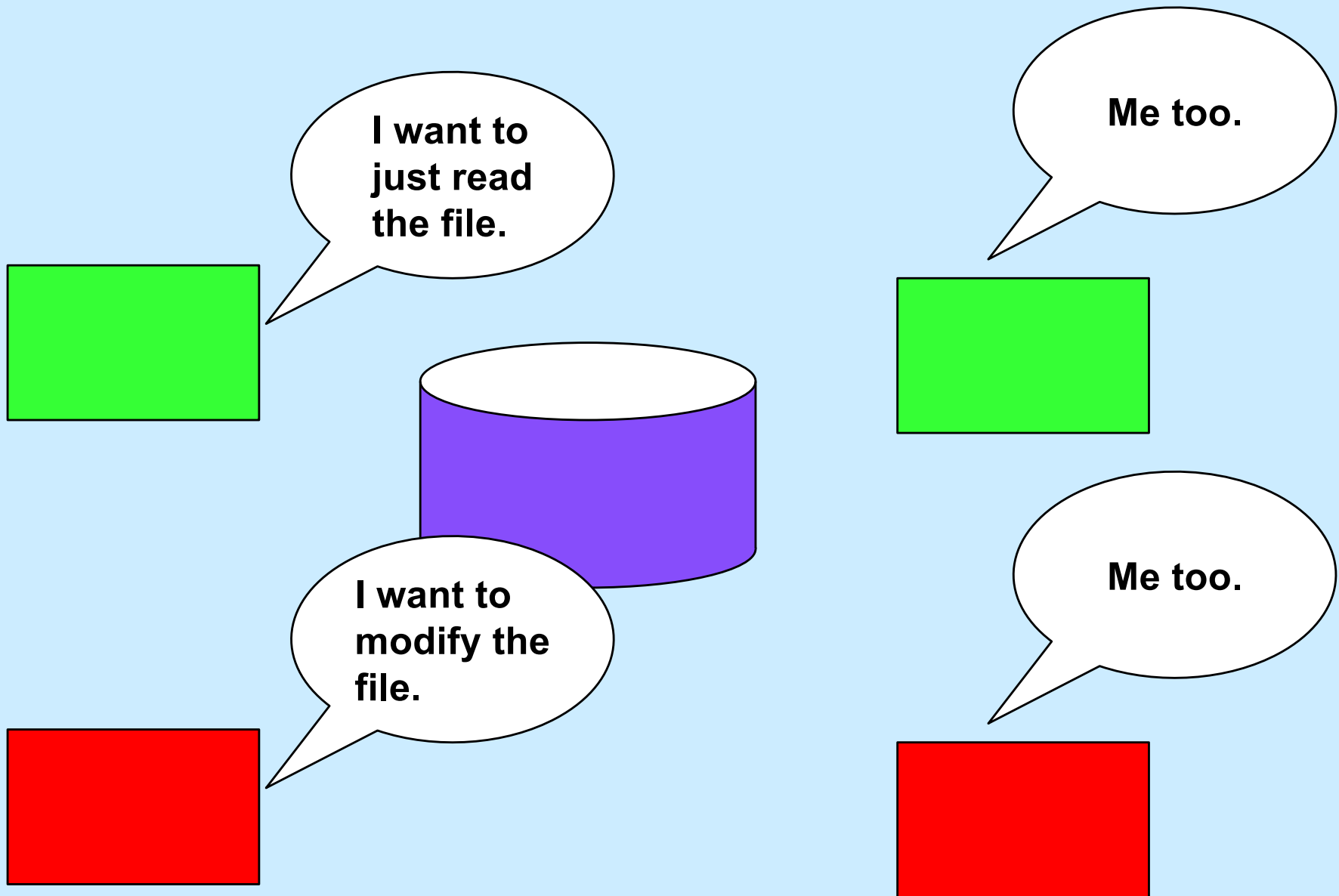
Sharing Files

- **You're doing a project with a partner**
- **You code it as one 15,000-line file**
 - the first 7,500 lines are yours
 - the second 7,500 lines are your partner's
- **You edit the file, changing 6,000 lines**
 - it's now 5am
- **Your partner completes her changes at 5:01am**
- **At 5:02am you look at the file**
 - your partner's changes are there
 - yours are not

Lessons

- **Never work with a partner**
- **Use more than one file**
- **Read up on git**
- **Use an editor and file system that support file locking**

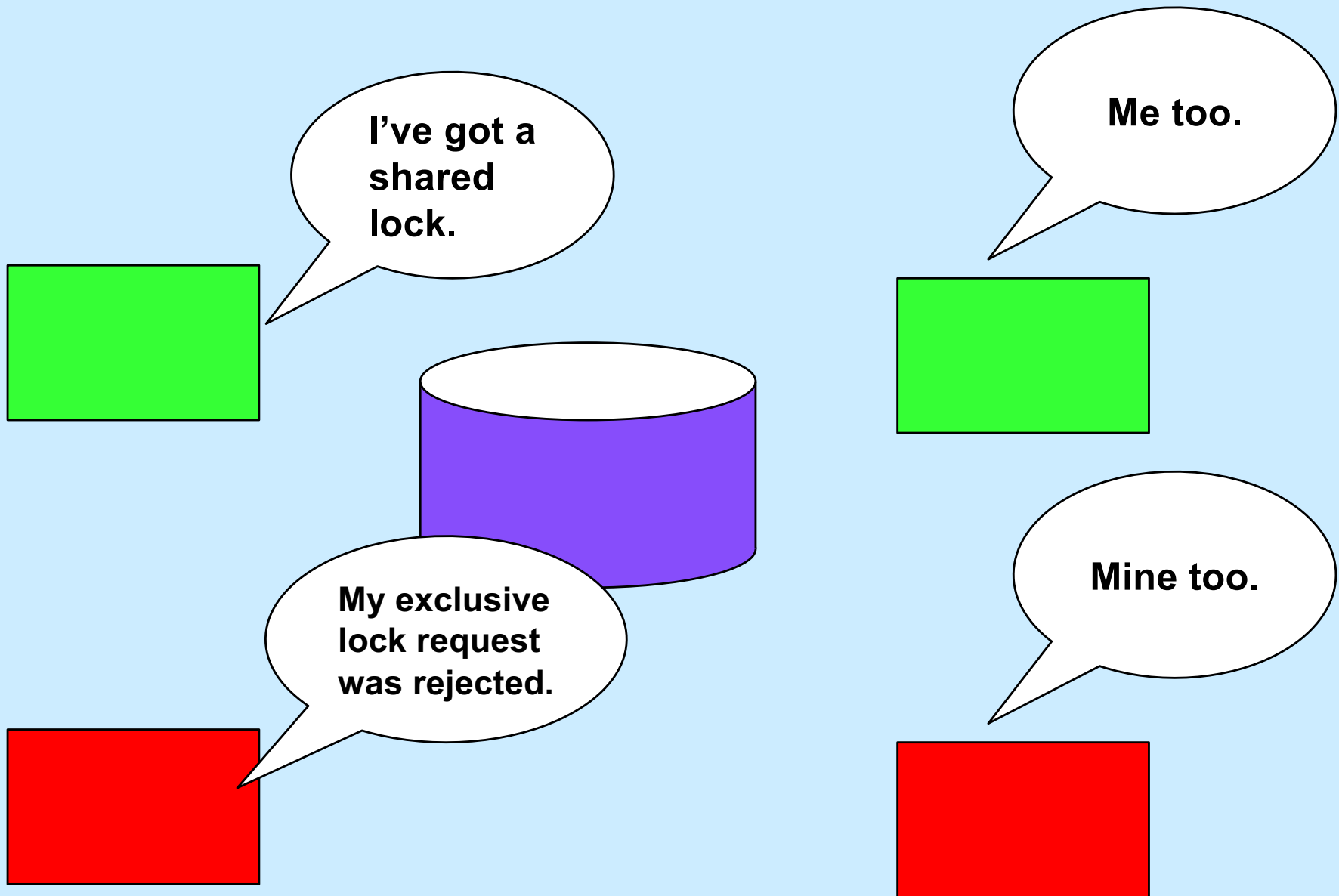
What We Want ...



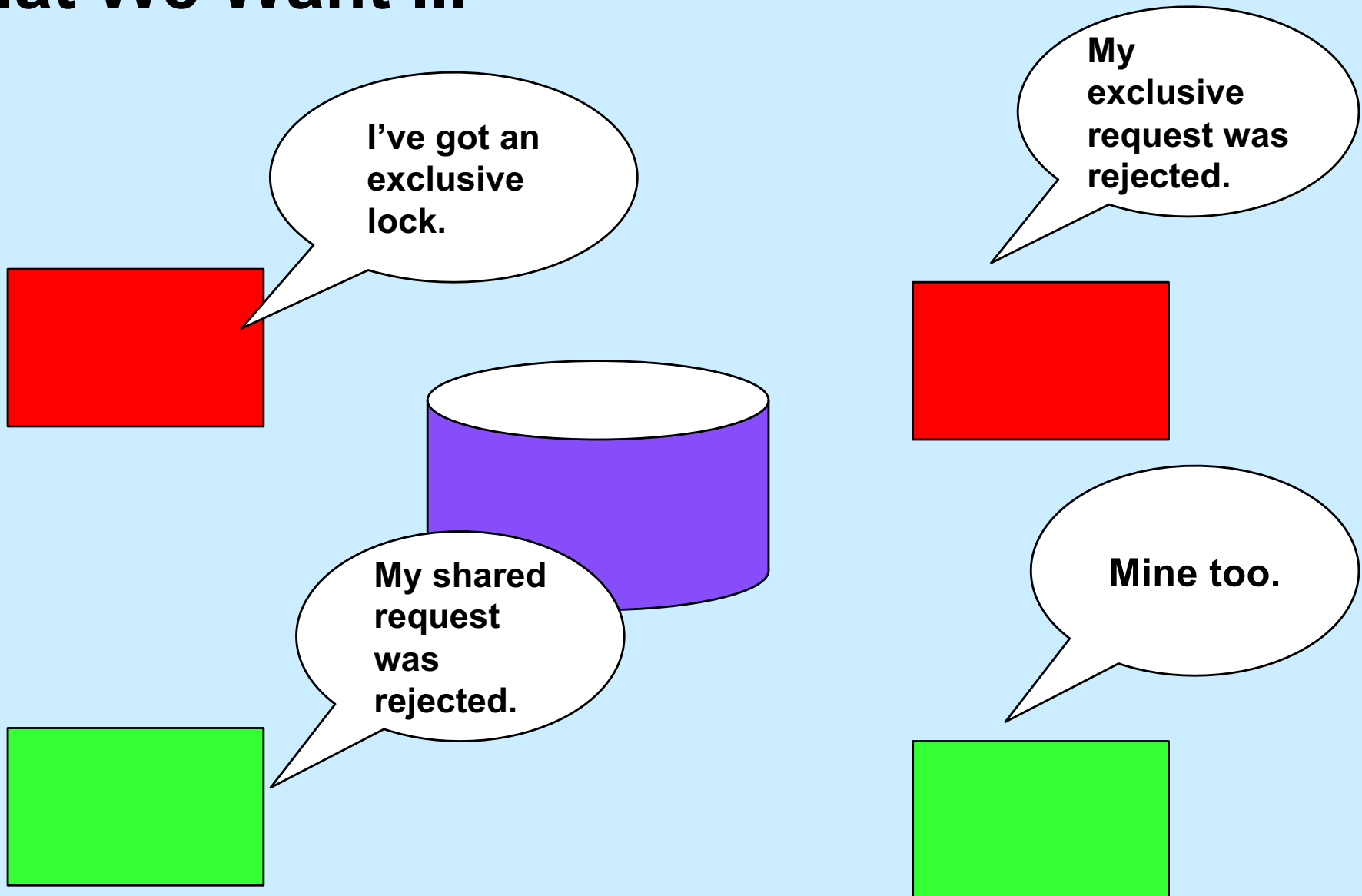
Types of Locks

- **Shared (readers) locks**
 - any number may have them at same time
 - may not be held when an exclusive lock is held
- **Exclusive (writers) locks**
 - only one at a time
 - may not be held when a shared lock is held

What We Want ...



What We Want ...



Locking Files

- Early Unix didn't support file locking
- How did people survive?

- `open("file.lck", O_RDWR|O_CREAT|O_EXCL, 0666);`
 - » operation fails if *file.lck* exists, succeeds (and creates *file.lck*) otherwise
 - » requires cooperative programs

Locking Files (continued)

- How it's done in “modern” Unix
 - “advisory locks” may be placed on files
 - » may request shared (readers) or exclusive (writers) lock
 - *fcntl* system call
 - » either succeeds or fails
 - » *open*, *read*, *write* always work, regardless of locks
 - » a lock applies to a specified range of bytes, not necessarily the whole file
 - » requires cooperative programs

Locking Files (still continued)

- **How to:**

```
struct flock fl;
fl.l_type = F_RDLCK;           // read lock
// fl.l_type = F_WRLCK;       // write lock
// fl.l_type = F_UNLCK;       // unlock
fl.l_whence = SEEK_SET;        // starting where
fl.l_start = 0;                // offset
fl.l_len = 0;                  // how much? (0 = whole file)
fd = open("file", O_RDWR);
if (fcntl(fd, F_SETLK, &fl) == -1)
    if ((errno == EACCES) || (errno == EAGAIN))
        // didn't get lock
    else
        // something else is wrong
else
    // got the lock!
```
