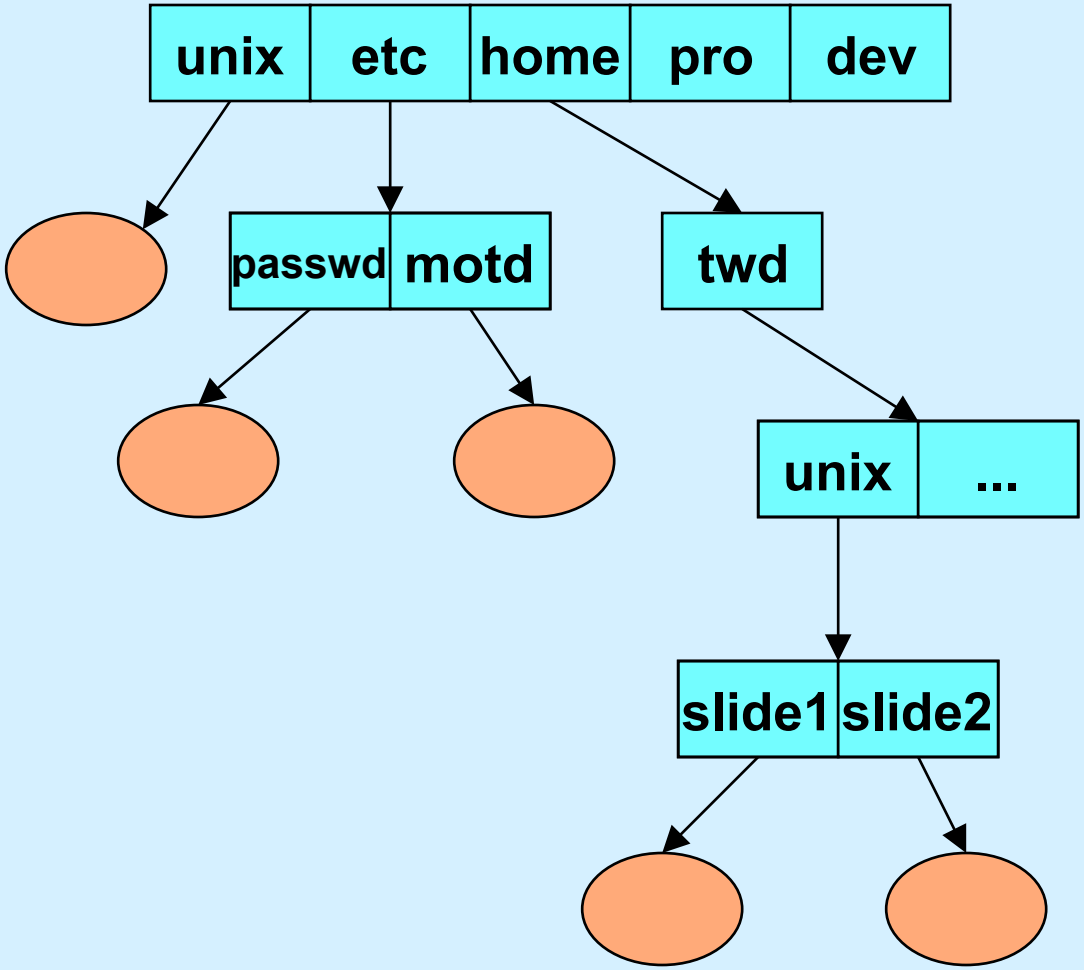
CS 33

Files Part 2

Directories



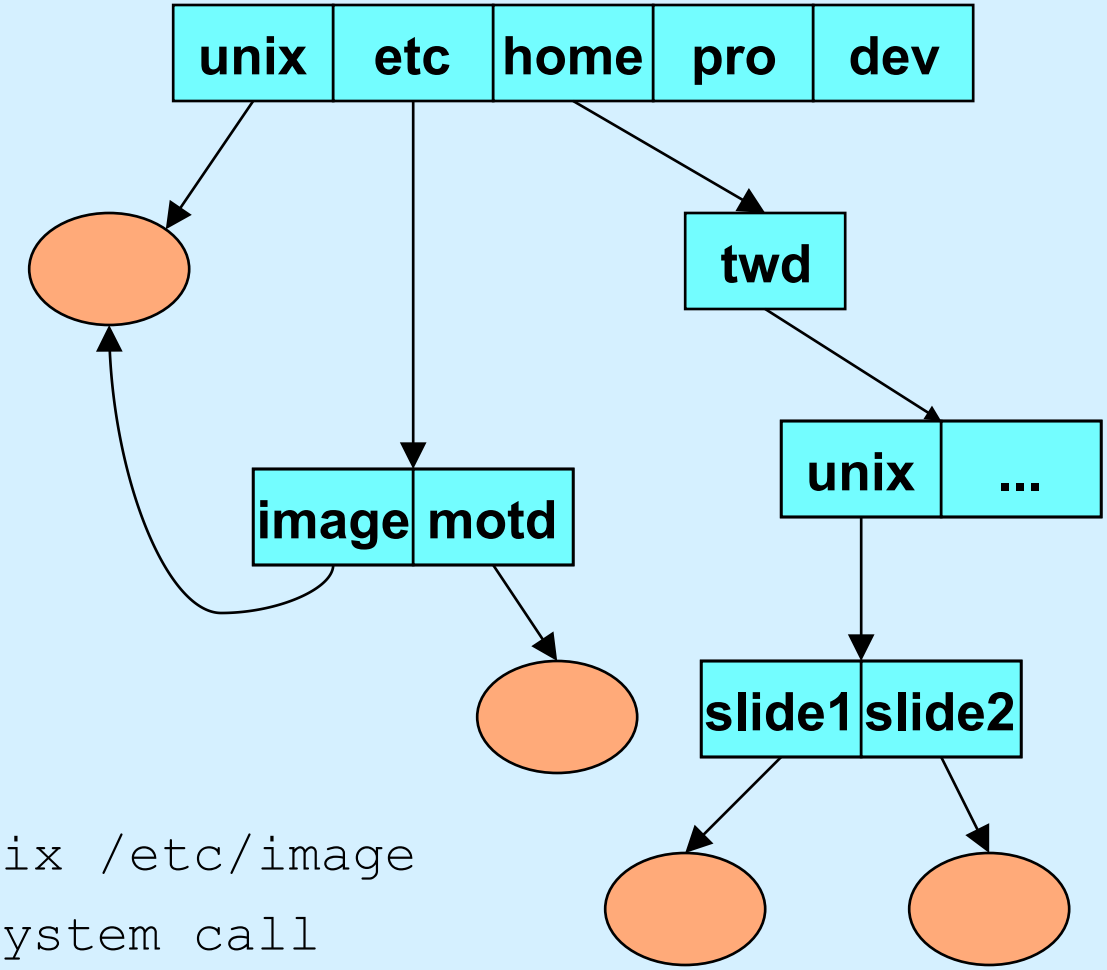
Directory Representation

Component Name	Inode Number

directory entry

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

Hard Links



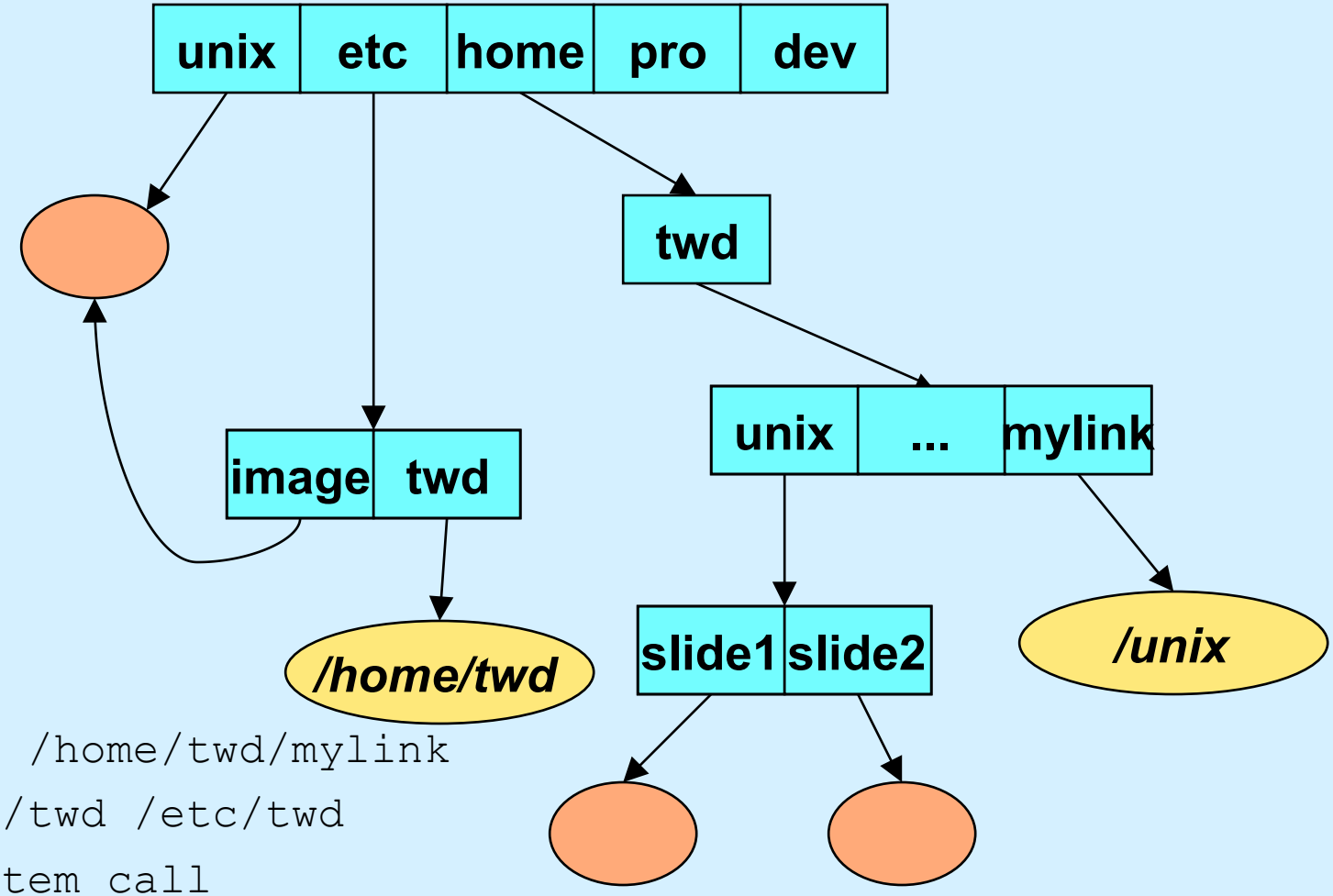

```
$ ln /unix /etc/image
# link system call
```

Directory Representation

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

.	4
..	1
image	117
motd	33

Symbolic Links

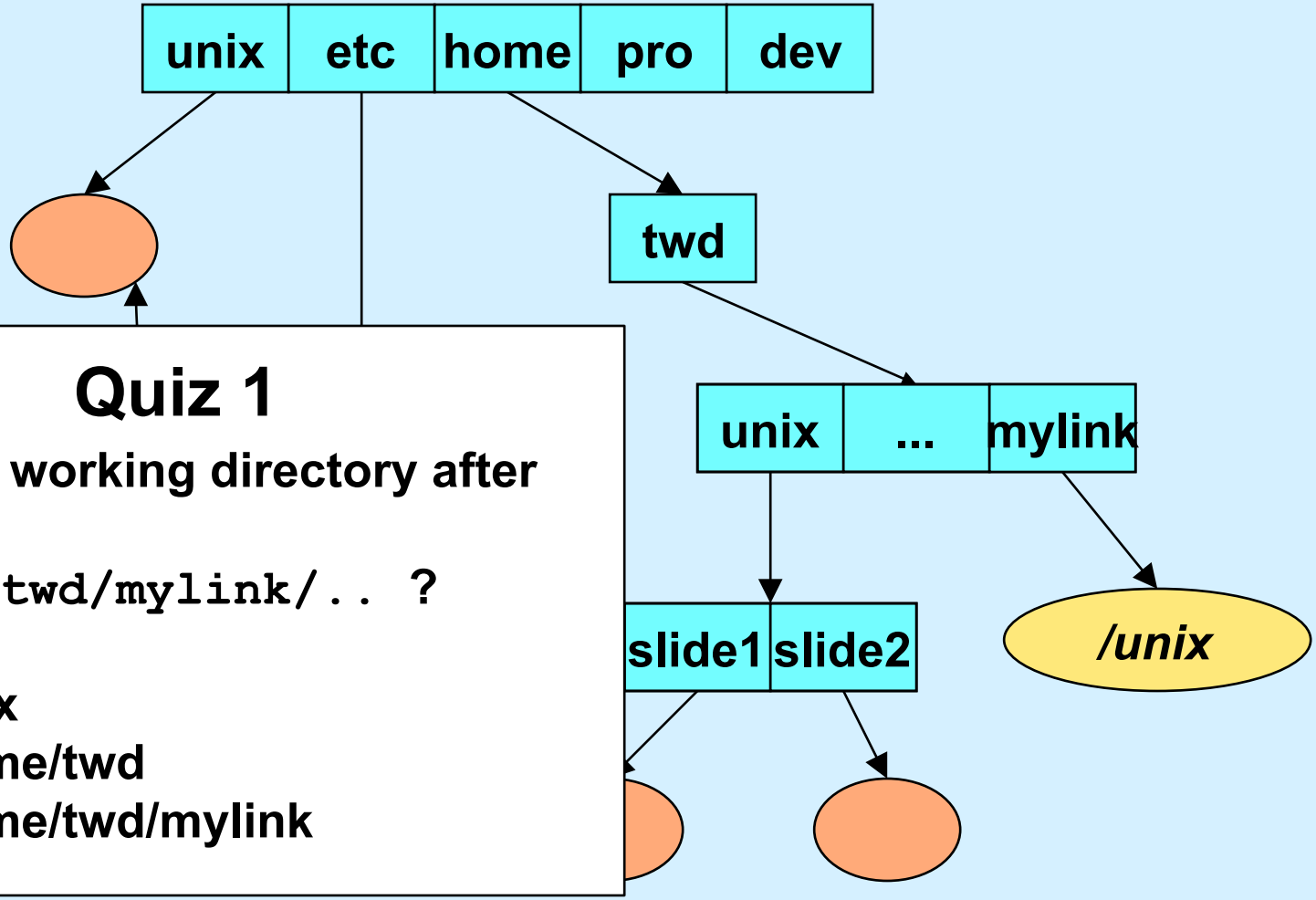



```
% ln -s /unix /home/twd/mylink
% ln -s /home/twd /etc/twd
# symlink system call
```

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 1

What is the working directory after doing

```
cd /home/twd/mylink/.. ?
```

a) /
b) /unix
c) /home/twd
d) /home/twd/mylink

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

```
$ 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)`

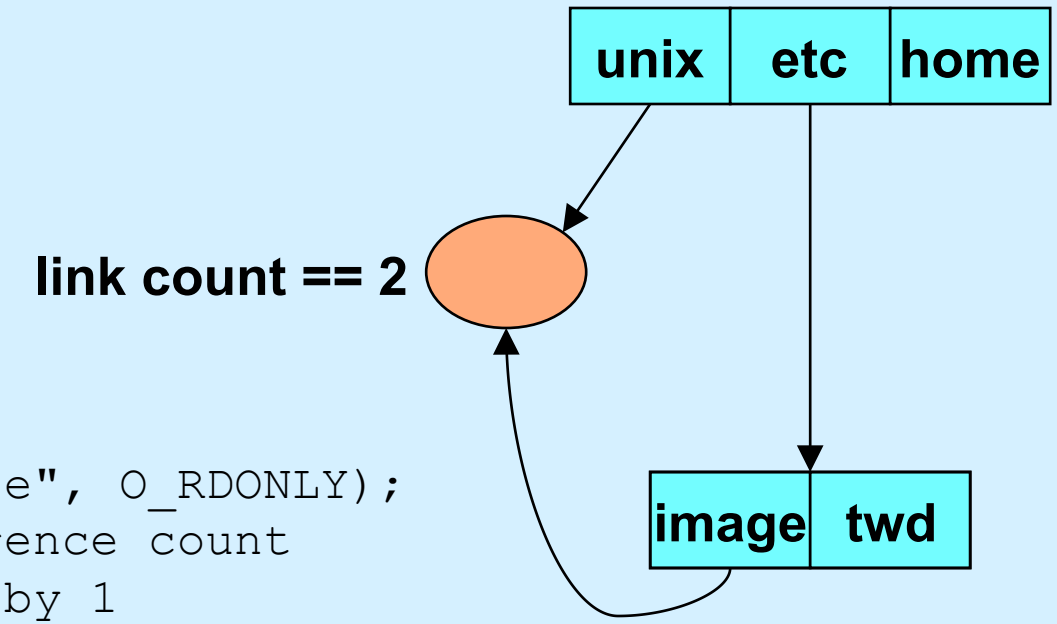
Umask

- 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 & ~(027) = 0750
    - » editors: permissions = 0666 & ~(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("file", O_RDONLY);
   // file's reference count
   // incremented by 1
unlink("file");
   // file's link count
   // decremented by 1
close(fd);
   // file's reference count
   // decremented by 1
```

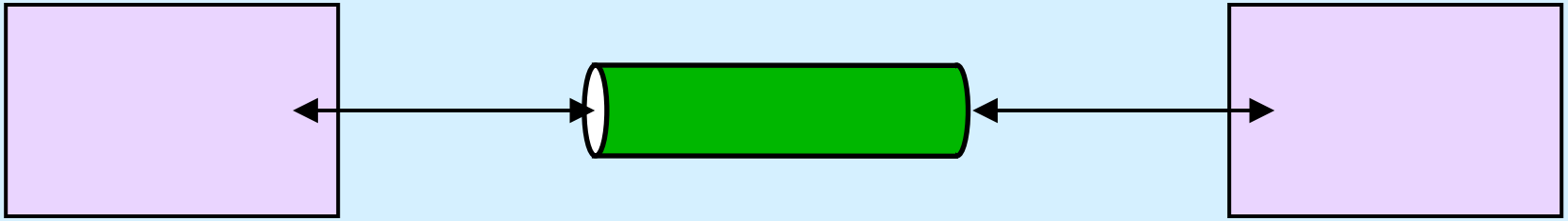
Quiz 2

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

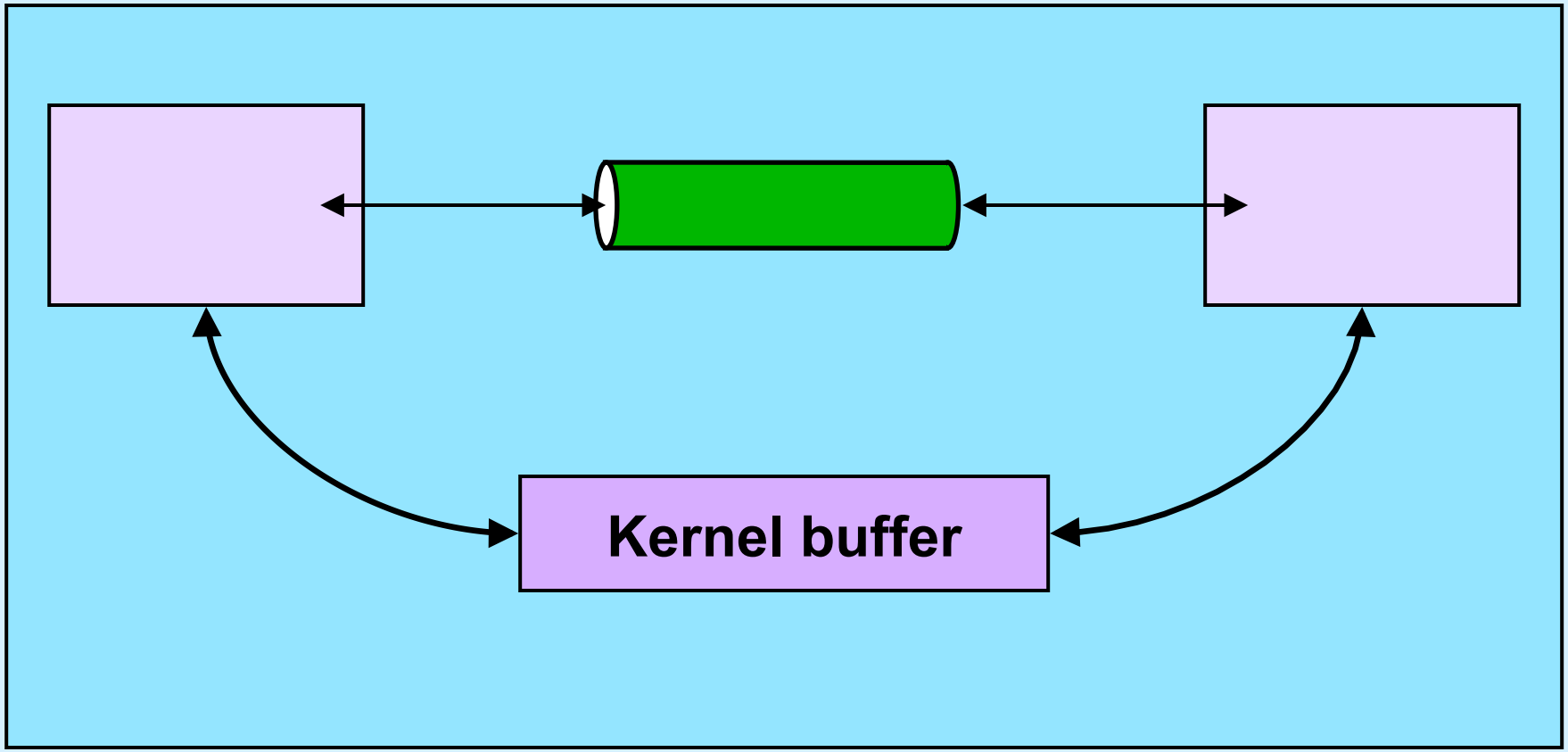
Assume that *PutStuffInFile* writes to the given file, and *ReadStuffFromFile* reads from the file.

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**

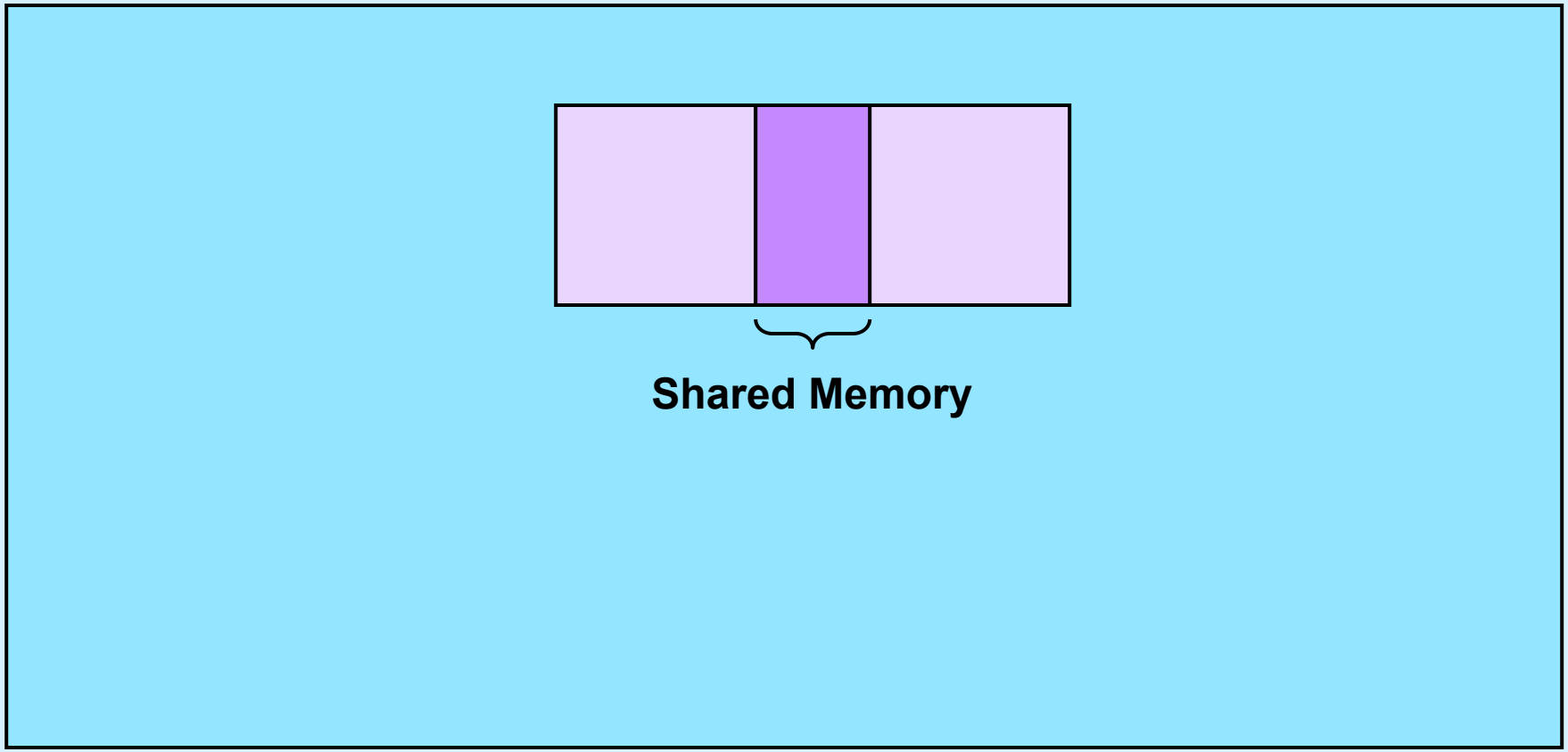
Interprocess Communication (IPC)



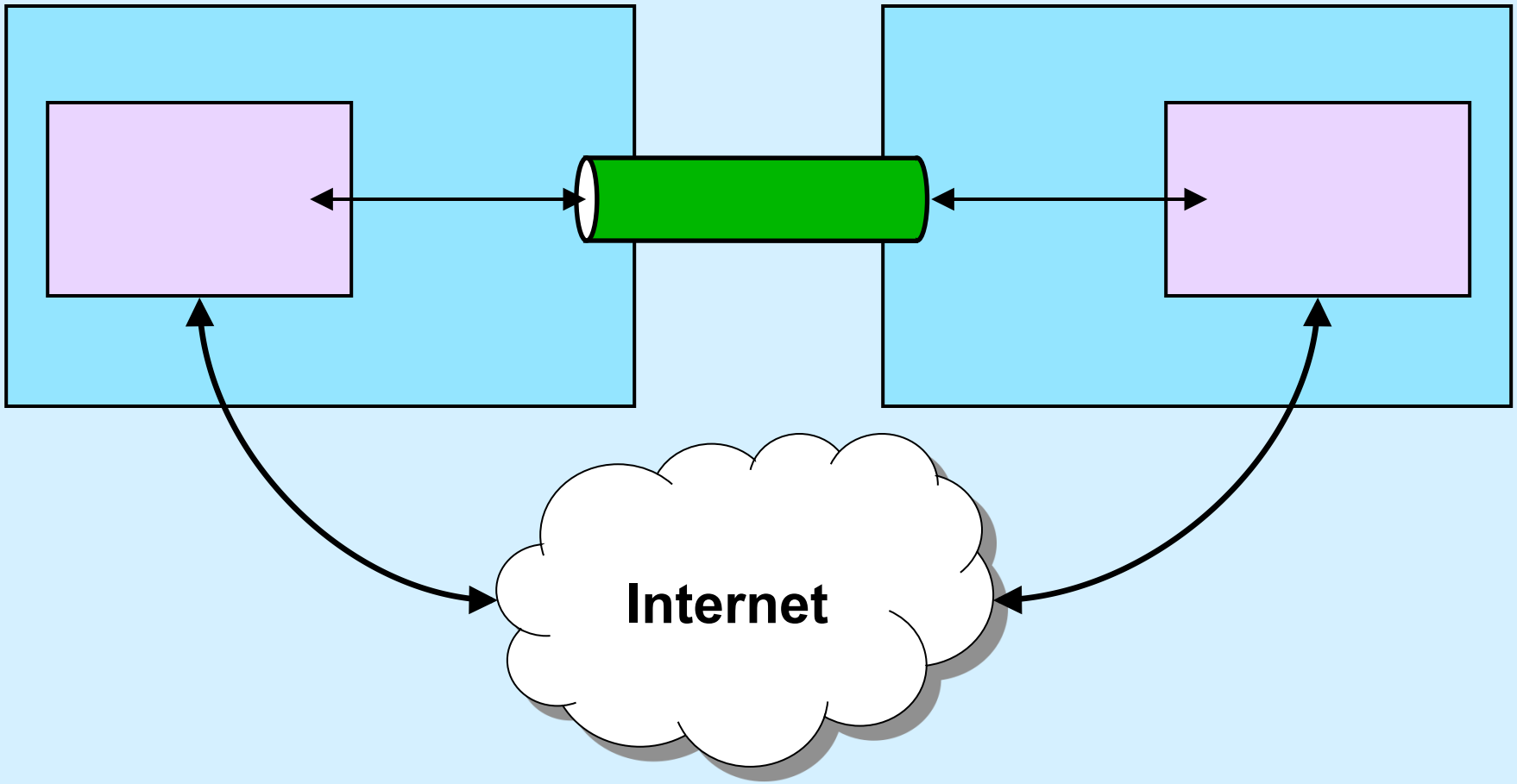
Interprocess Communication: Same Machine I



Interprocess Communication: Same Machine II

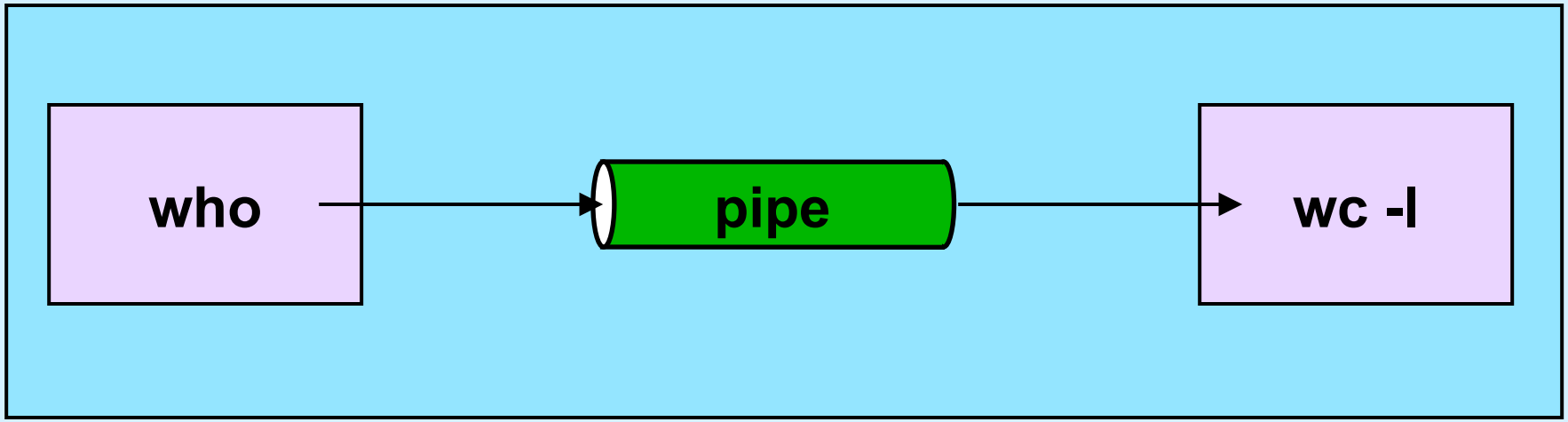


Interprocess Communication: Different Machines



Intramachine IPC

```
$cslab2e who | wc -l
```



Intramachine IPC

```
$cslab2e who | wc -l
```


```
int fd[2];
pipe(fd);
if (fork() == 0) {
   close(fd[0]);
   close(1);
   dup(fd[1]); close(fd[1]);
   execlp("who", "who", 0); // who sends output to pipe
}
if (fork() == 0) {
   close(fd[1]);
   close(0);
   dup(fd[0]); close(fd[0]);
   execlp("wc", "wc", "-l", 0); // wc gets input from pipe
}
close(fd[1]); close(fd[0]);
// …
```

Pipes

- Pro
  - really easy to use
  - anonymous: no names to worry about
- Con
  - anonymous: can’t give them names
    - » communicating processes must be related

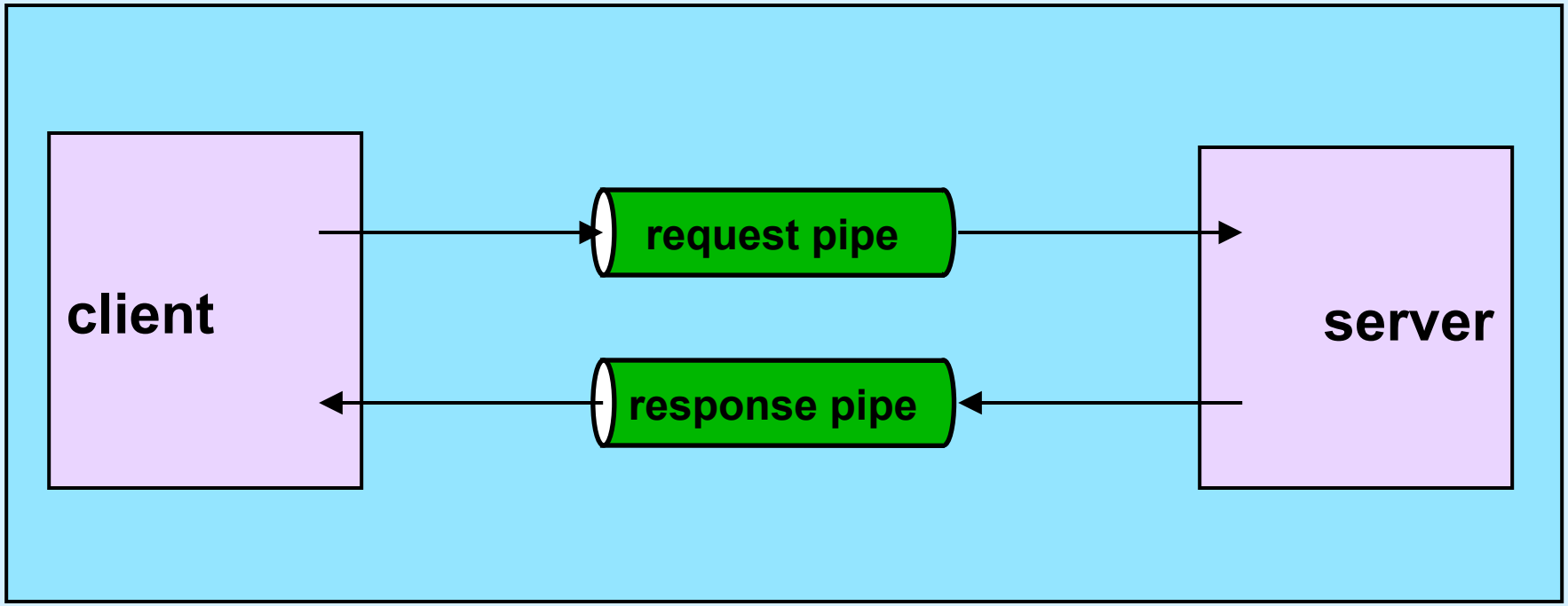
Named Pipes

```
mkfifo("/u/twd/service", 0622);
  // creates a named pipe (FIFO) that
  // anyone may write to but only whose
  // owner may read from

int wfd = open("/u/twd/service", O_WRONLY);
write(wfd, request, sizeof(request));
  // send request in one process

int rfd = open("/u/twd/service", O_RDONLY);
read(rfd, request, sizeof(request));
  // receive request in another process
```

Client/Server



Intermachine Communication

- **Can pipes and named pipes be made to work across multiple machines?**
  - **covered soon ...**
    - » **what happens when you type**

```
who | ssh cslab3a wc -l
```

?