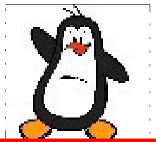


Video Lecture # 27
Programming UNIX
Named Pipes (FIFOs)

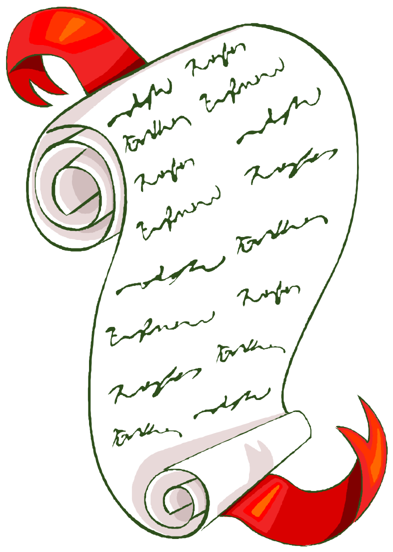
Course: SYSTEM PROGRAMMING

Instructor: Arif Butt

Punjab University College of Information Technology (PUCIT)
University of the Punjab

Today's Agenda

- Introduction to UNIX Named Pipes
- Illustration showing the working of FIFO
- Working with FIFOs on the Shell
- Communicating via FIFO in a C Program
- Bidirectional Communication using FIFOs



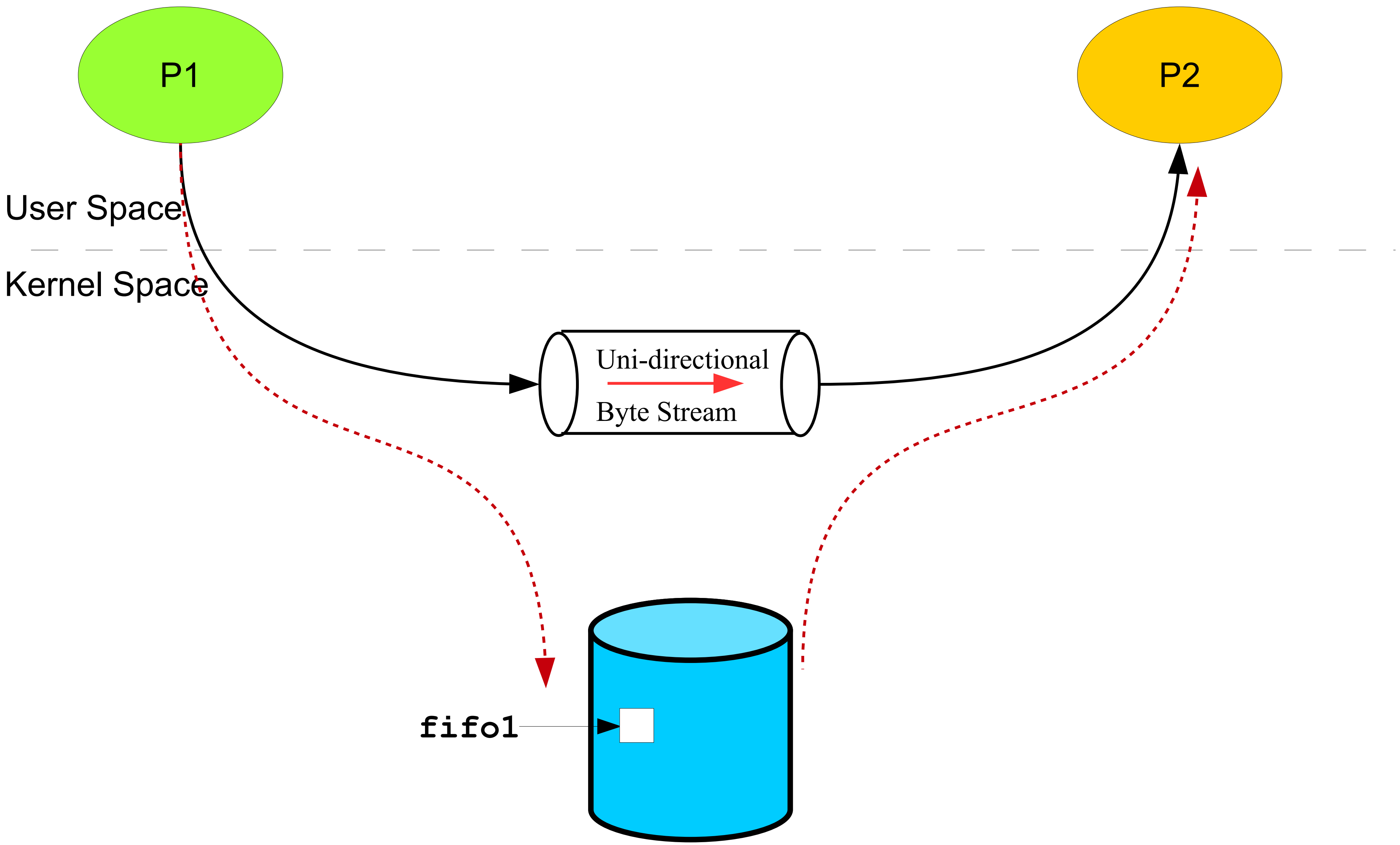
Introduction to FIFOs

- Pipes have no names, and their biggest disadvantage is that they can be only used between processes that have a parent process in common (ignoring descriptor passing)
- UNIX FIFO is similar to a pipe, as it is a one way (half duplex) flow of data. But unlike pipes a FIFO has a path name associated with it allowing unrelated processes to access a single pipe
- FIFOs/named pipes are used for communication between related or unrelated processes executing on the same machine
- A FIFO is created by one process and can be opened by multiple processes for reading or writing. When processes are reading or writing data via FIFO, kernel passes all data internally without writing it to the file system. Thus a FIFO file has no contents on the file system; the file system entry merely serves as a reference point so that processes can access the pipe using a name in the file system

Use of FIFO Between Unrelated Processes

```
$ echo "Hello PUCIT" 1> fifo1
```

```
$ cat fifo1
```



UNIX Named Pipes (FIFOs)
On the Shell
Proof of Concept

`mkfifo()` Library Call

```
int mkfifo(const char*pathname, mode_t mode);
```

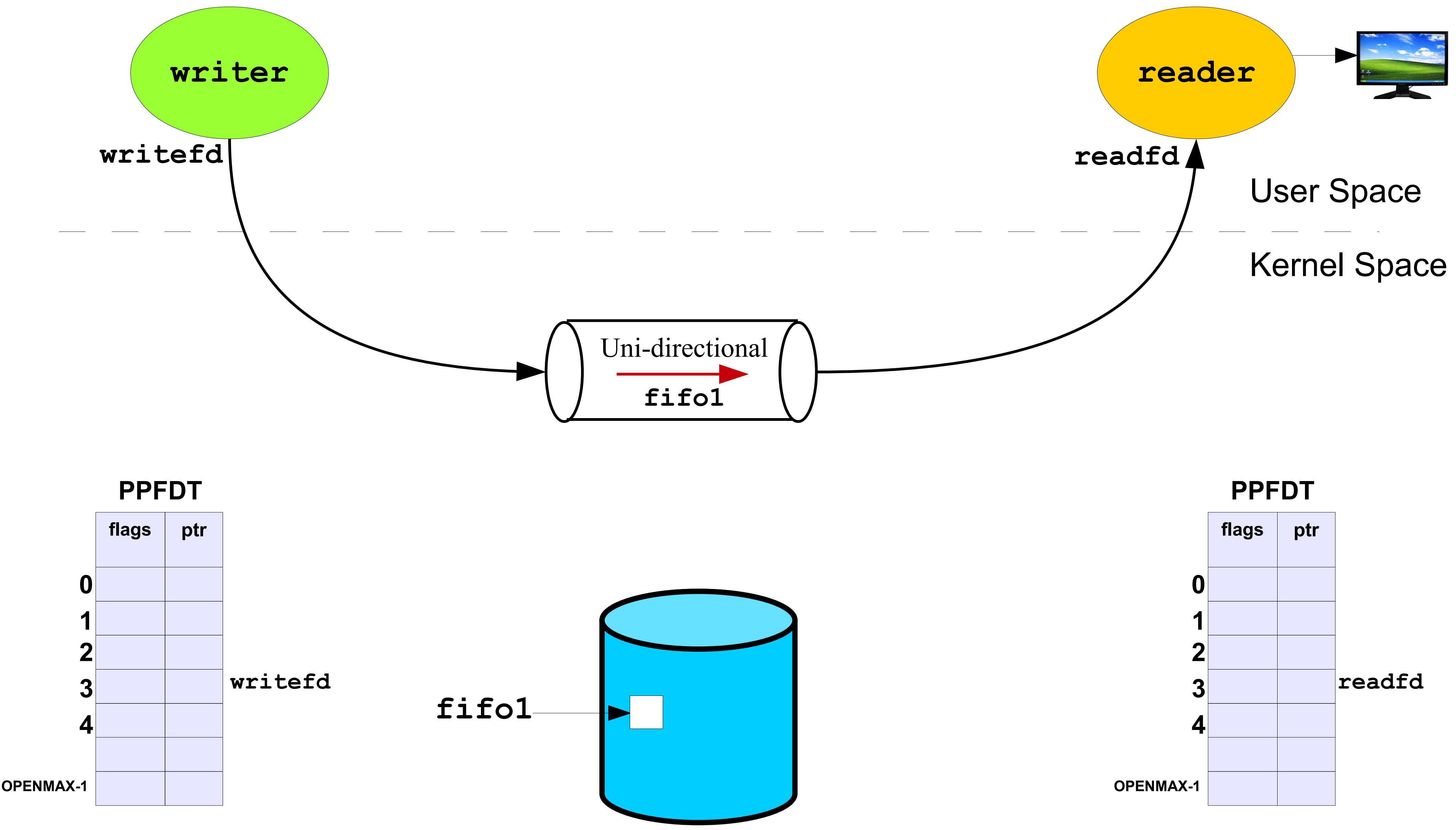
- Makes a FIFO special file with name `pathname`. The second argument `mode` specifies the FIFO's permissions. It is modified by the process's umask in the usual way: (`mode & ~umask`)
- Once you have created a FIFO, any process can open it for reading or writing, in the same way as an ordinary file. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa
- Call Fails when:
  - Parent directory does not allow write permission
  - Path name already exists
  - Path name points outside accessible address space
  - Path name too long
  - Insufficient kernel memory

mknod() System Call

```
int mknod(const char*name, mode_t mode, dev_t device);
```

- The mknod() system call creates a FIFO file with name mentioned as the first argument
- The second argument mode specifies both the permissions to use and the type of node to be created. It should be a combination (using bitwise OR) of one of the file types (S_IFREG, S_IFIFO, S_IFCHR, S_IFBLK, S_IFSOCK) and the permissions for the file
- For creating a named pipe the third argument is set to zero. However, to create a character or block special file we need to mention the major and minor numbers of the newly created device special file

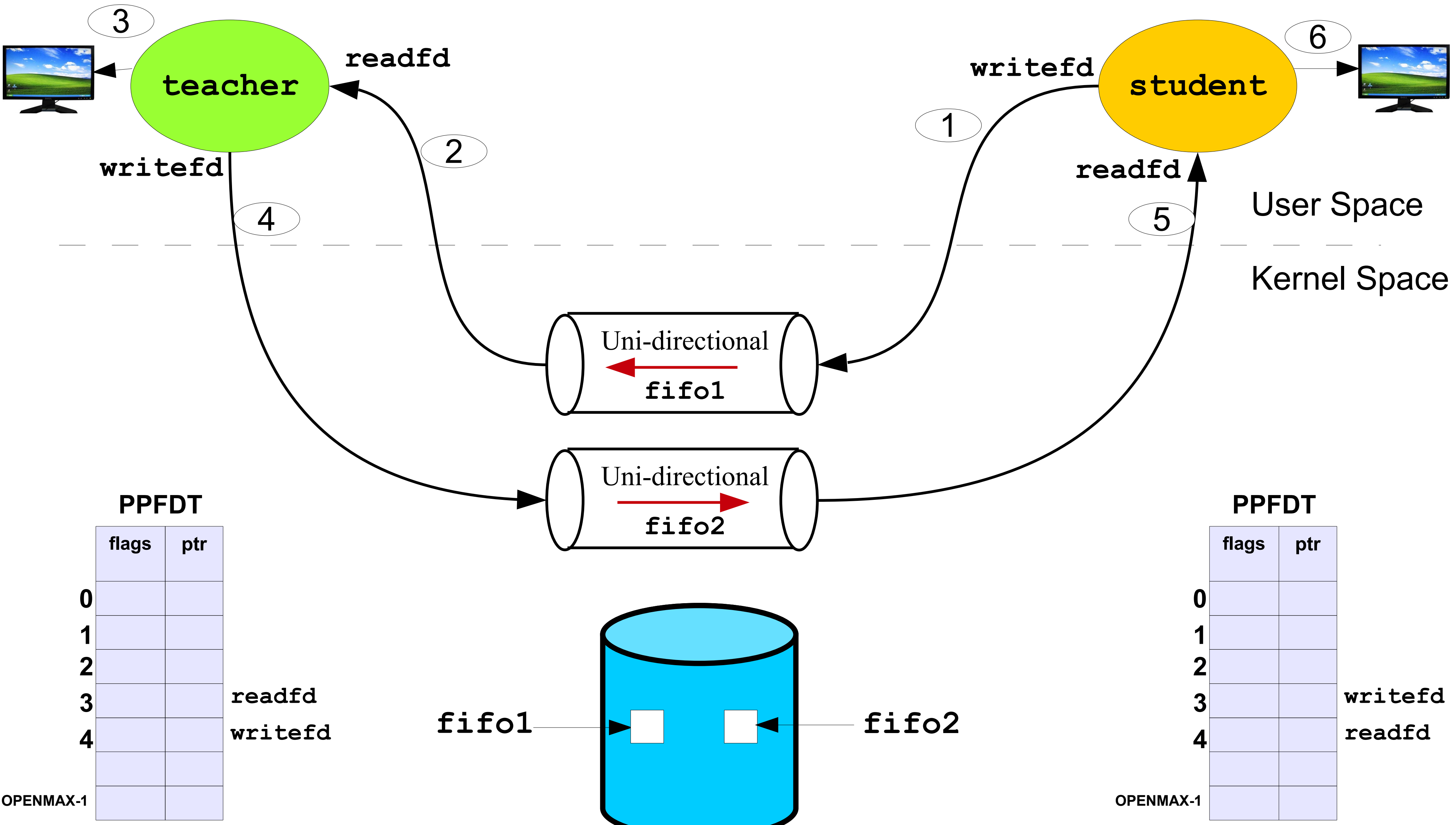
Communication Using FIFO



Communication using FIFO Proof of Concept

`ex2/writer.c – ex2/reader.c`

Bidirectional Comm Using FIFOs



Bidirectional Comm using FIFOs Proof of Concept

`ex3/teacher.c - ex3/student.c`

Things To Do

If you have problems visit me in counseling hours. . . .