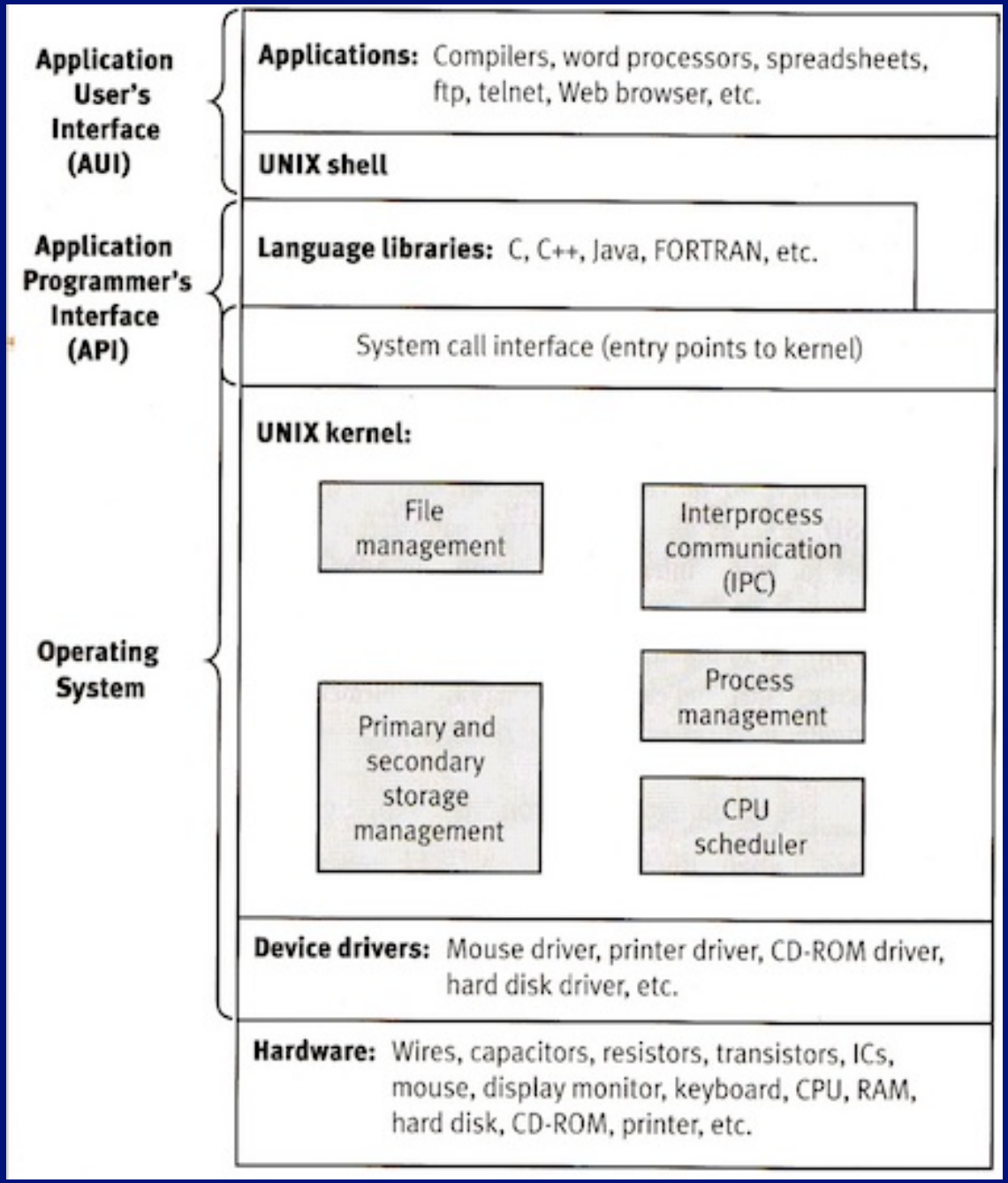
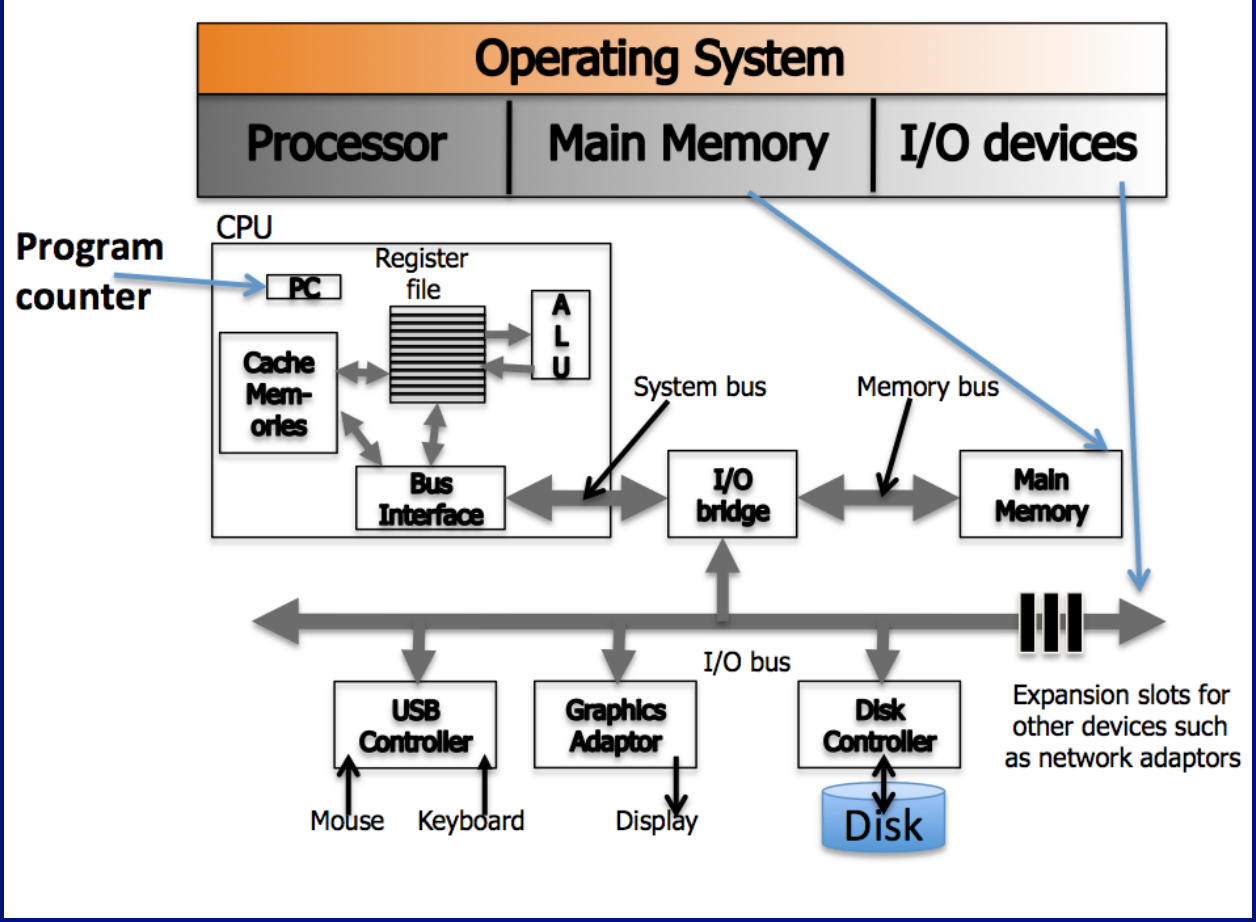
Breadth of CS32's subject matter (Reader p. 14)

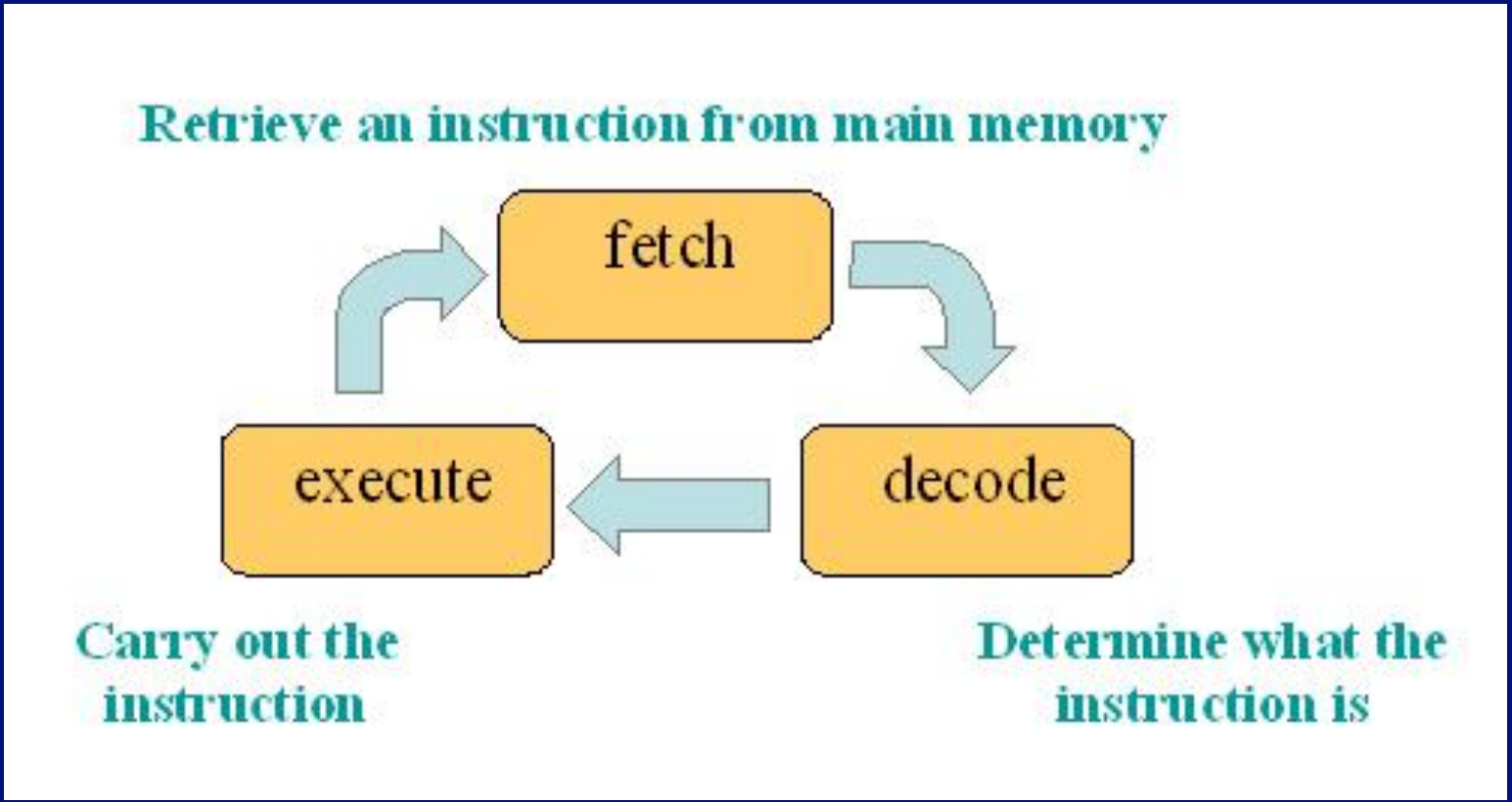


Underlying computer system = hardware + software



Thanks to Chandra Krintz and Kevin Sanft, for this figure and some other parts of these lecture notes.

Machine Cycle: What a CPU does … over and over again.



Processing data & instructions

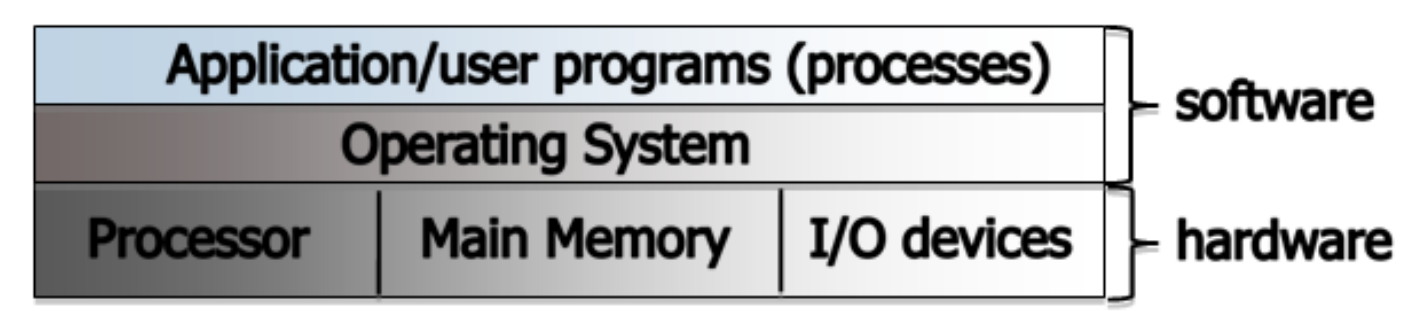
- Program instructions and data are in main memory
  - CPU loads next few instructions into a cache – for fast access – and similarly stores data used by the instructions in a data cache
- All CPU components (hardware registers, ALU, bus) use same data width – e.g., 32 bit or 64 bit
  - System bus (wires) = address bus + data bus + other signals
- CPU toggles pins to identify which devices (memory, IO) it wishes to access – and whether it wants to read or write
  - The CPU doesn't block after a request, it goes onto another task until the device "interrupts" it with the data.
  - Devices use special wires/pins to alert the CPU that the data that the CPU requested are ready

Things to ponder

- How are all of these computer operations managed effectively?
  - After all, the CPU just responds to the next instruction. So how are all the instructions managed, especially when there are many clients (users, processes)?
- And from a different perspective, how are we – and our simple programs – able to deal with such a complex system?
  - Don't we need an intermediary?
- Hmm … we need an operating system!

Operating systems: two views

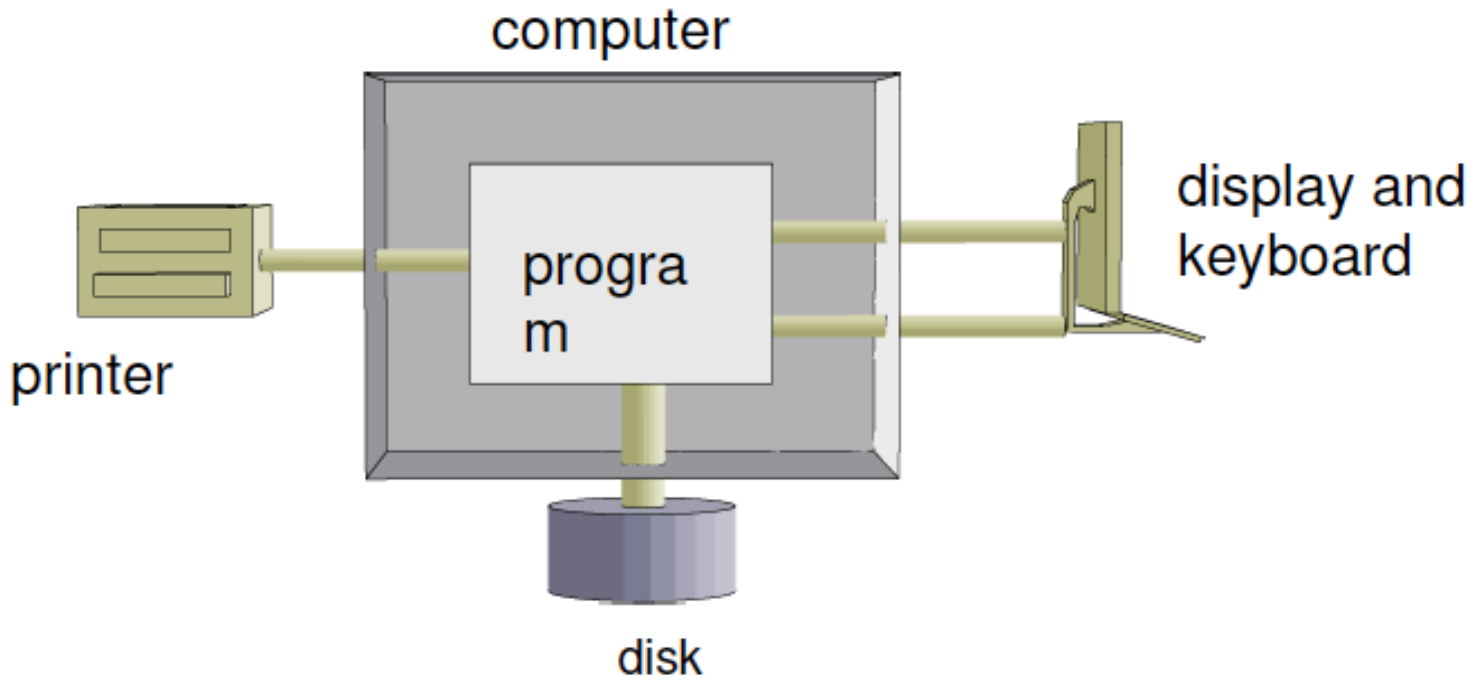
- Top-down view: an OS is software that isolates us from the complications of hardware resources
  - In other words, an OS is an application programmer's and a user's interface to computer operations



- Bottom-up view: an OS is software that allocates and de-allocates computer resources – efficiently, fairly, orderly and securely

Some "big picture" ideas: user's point of view

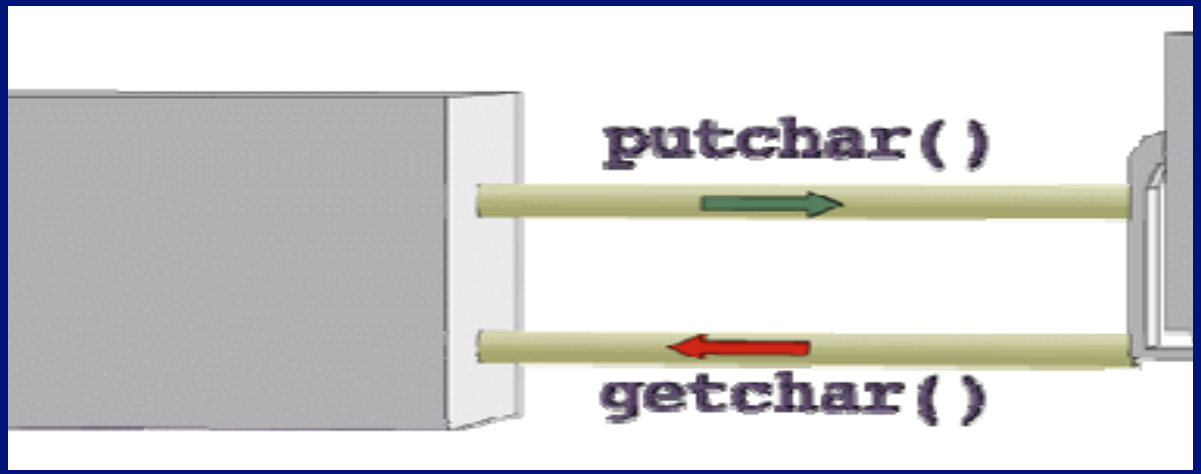
A *simple* computer model



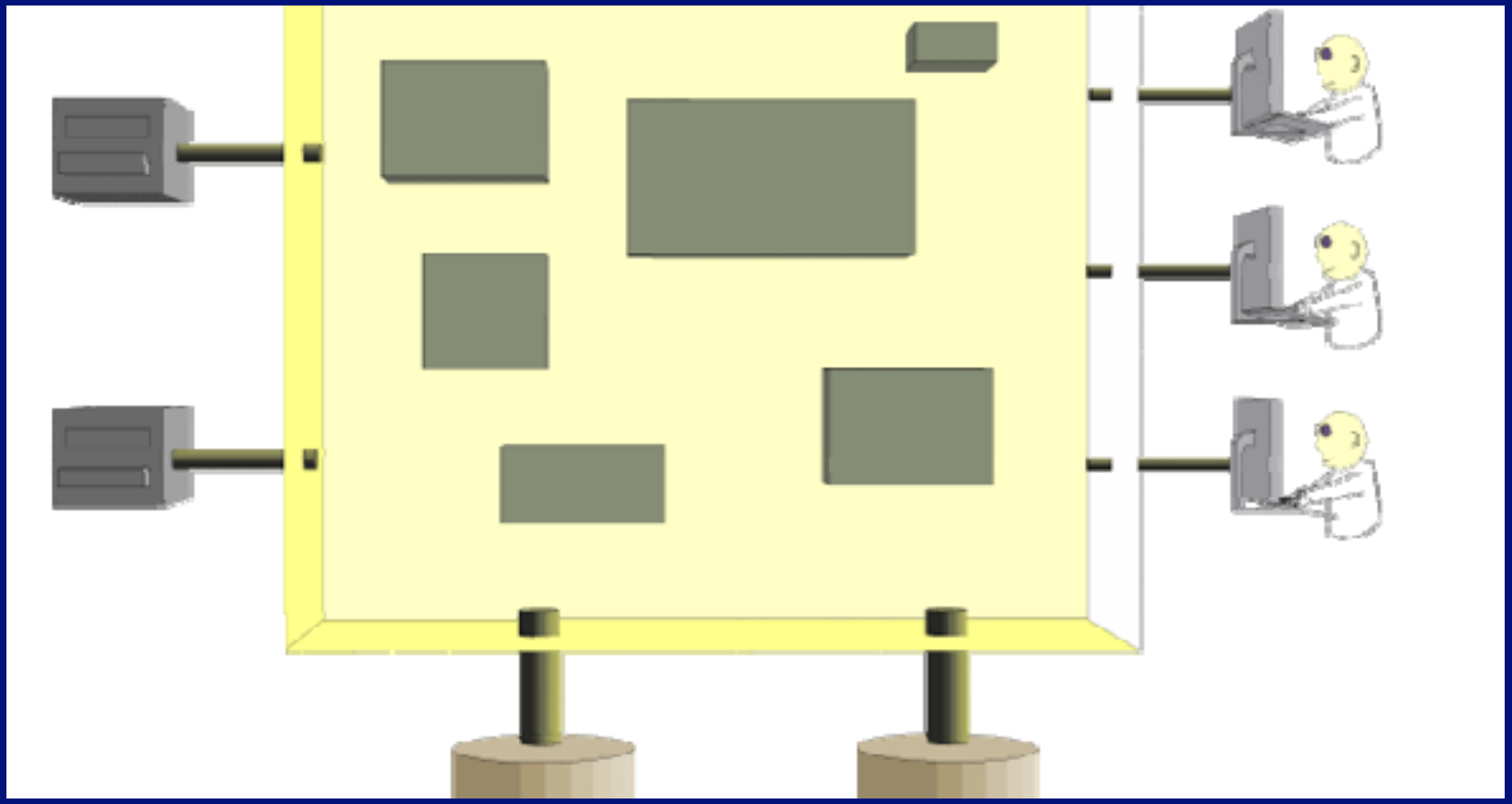
This and the next several figures derived from B. Molay's *Understanding Unix/Linux Programming*, Pearson 2003.

An example program

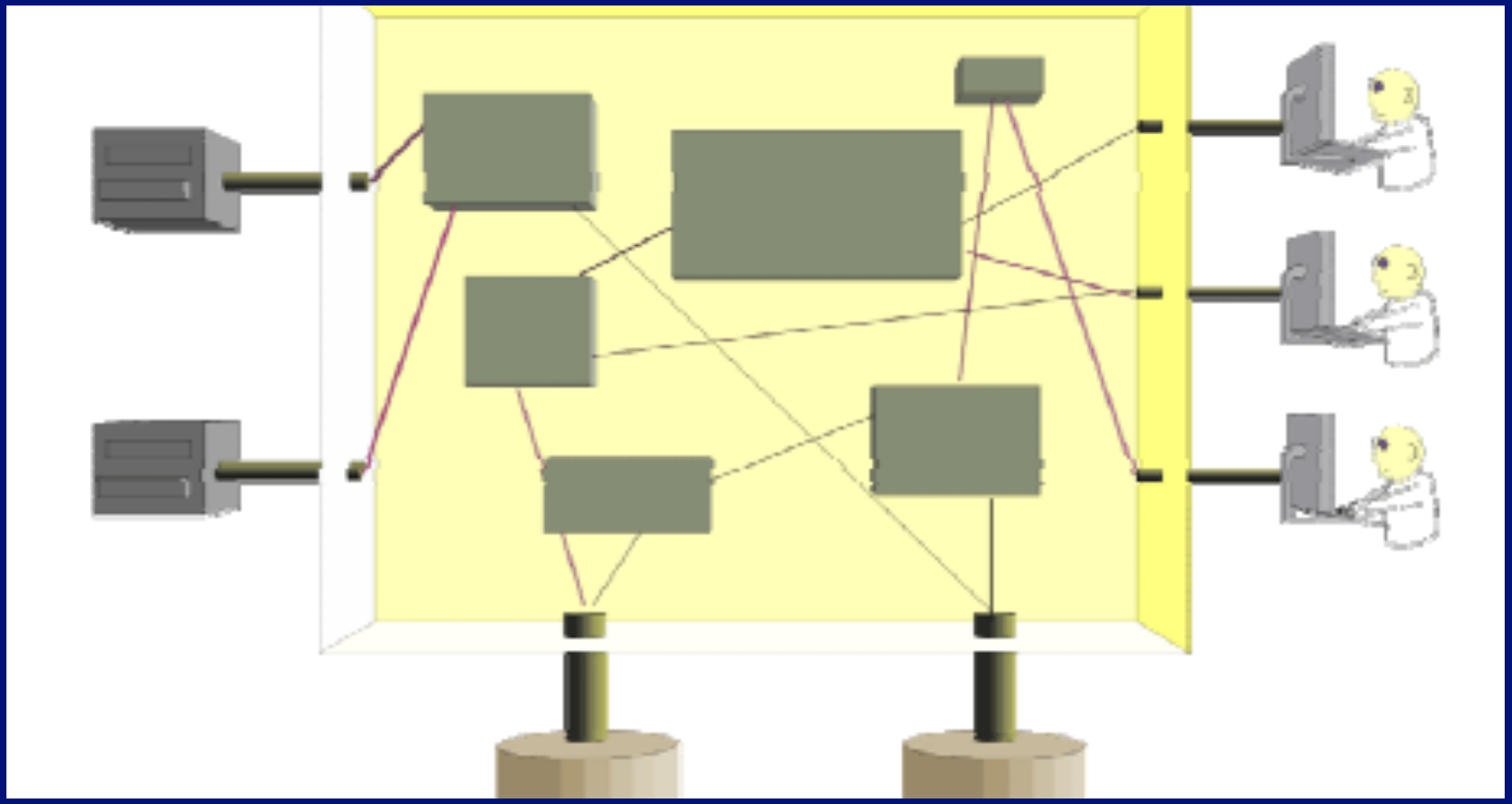
```
#include <stdio.h>
int main(void) {
    int c;
    while ( (c = getchar()) != EOF )
        putchar(c);
}
```



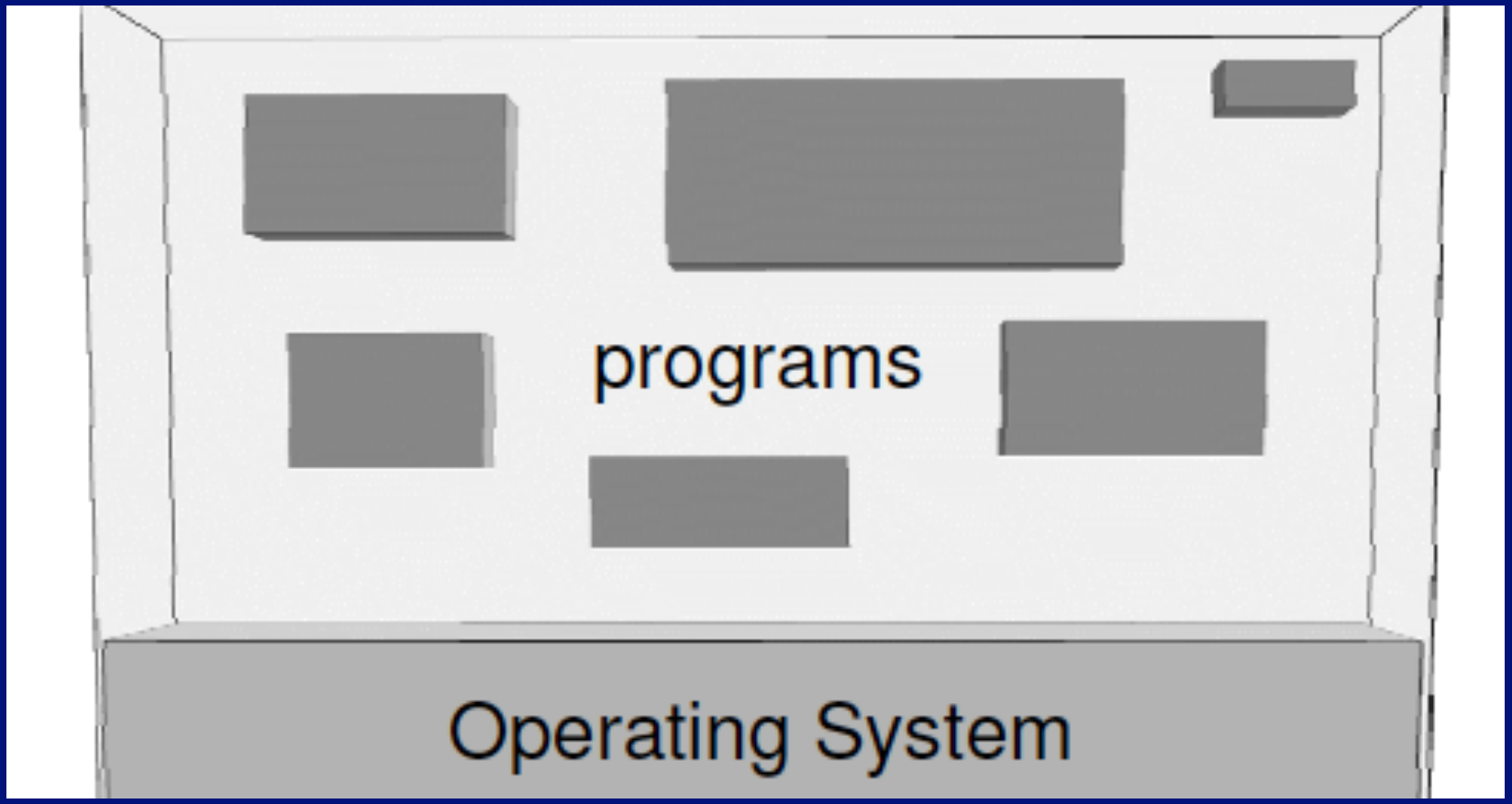
More realistic computer model



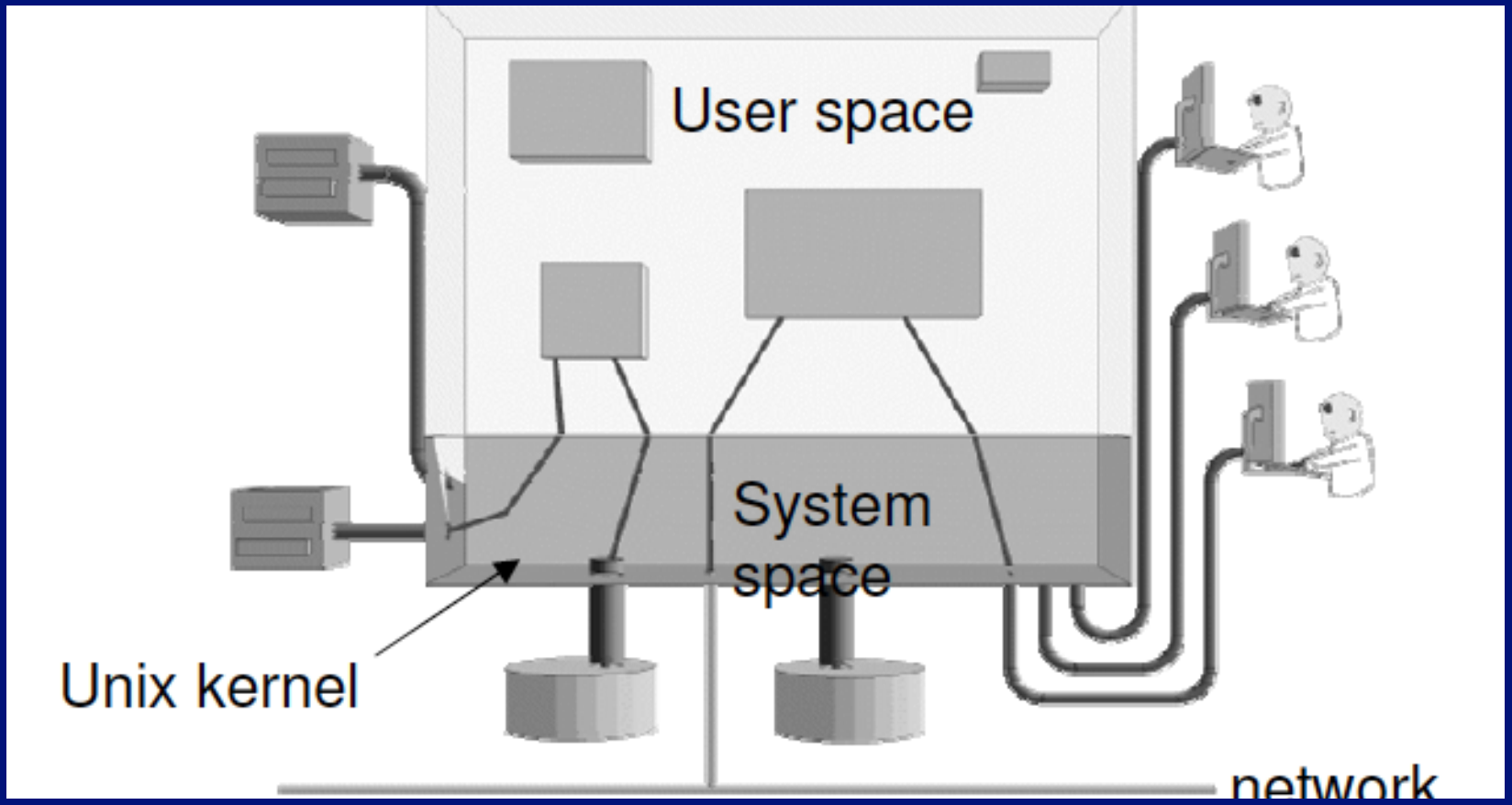
How connected? Not like this!



OS manages everything!



OOP idea: OS provides *services*



Types of operating systems

- Single-user, single-process – i.e., one customer, and one job at a time
- Single-user, multi-process – one workstation, but lots of stuff running
  - Actually the CPU handles just one process at any moment – jobs are swapped in/out in "time slices"
- Multi-user, multi-process – e.g., Unix/Linux
  - Same idea, but much more swapping to do
  - And added fairness, efficiency and security concerns

Unix history (Linux prequel)

- AT&T Bell Labs – System V standard
  - 1969-70: Ken Thompson wrote Unix in "B"
  - 1972: Dennis Ritchie developed C – a better B
  - Unix rewritten in C, 1973
  - … eventually System V, 1983
- UC Berkeley – BSD standard
  - Started with a copy of System IV, late 1970s
  - Lots of changes/additions in 1980s
  - Now FreeBSD
- Open source – Linux, since early 1990s

Unix philosophy (same as C)

- Small is beautiful
  - Each program does just one thing
  - Pipe commands (or use successive functions in C) to accomplish more complicated things
  - Less typing is best (using 1970s computers)
    - That's why so many commands are short (ls, cp, mv, …)
- Users/programmers know what they are doing
  - That's what makes the brevity sufficient
  - Means very few restrictions (or safety nets) apply

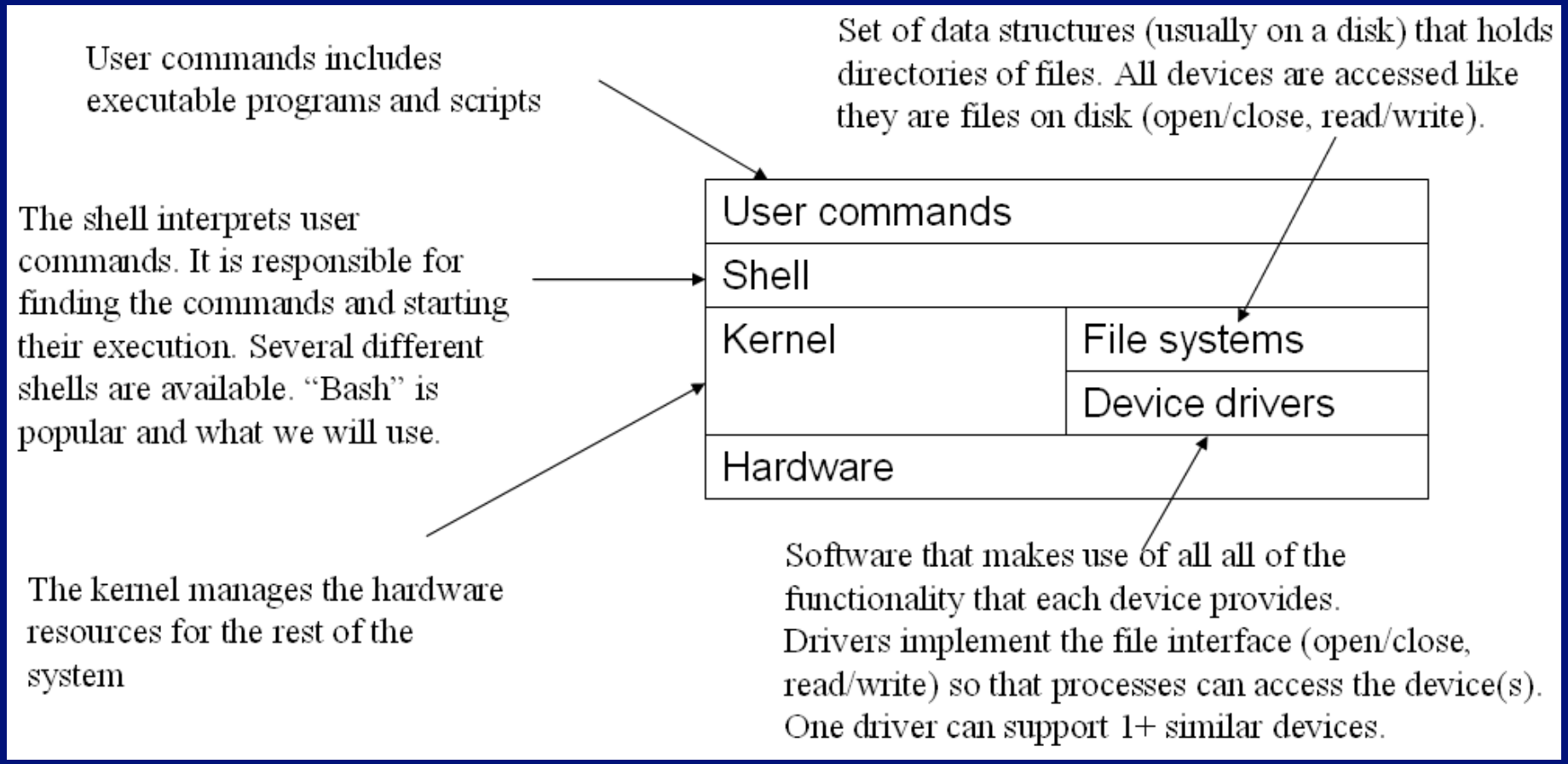
Linux

- Linus Torvalds created it as a Finnish *undergraduate* student
- Posted on Internet in 1991
  - Open source – licensed under GPL
  - Version 1.0 in 1994; version 2.2 in 1999; version currently at CSIL is Linux 3.11.10 (Fedora release 18)
- 1000s of programmers worldwide can read, modify, and redistribute its source code, so *it evolves*.
  - People improve it, adapt it, fix bugs, …

What is Linux?

- A fully-networked Unix-like operating system
- Multi-user, multitasking, multiprocessor system
  - Fundamental in the system's design and implementation
- Both command-line and graphical interfaces
- Coexists with other operating systems
- Runs on multiple platforms
- Distribution includes the source code!

The Linux System



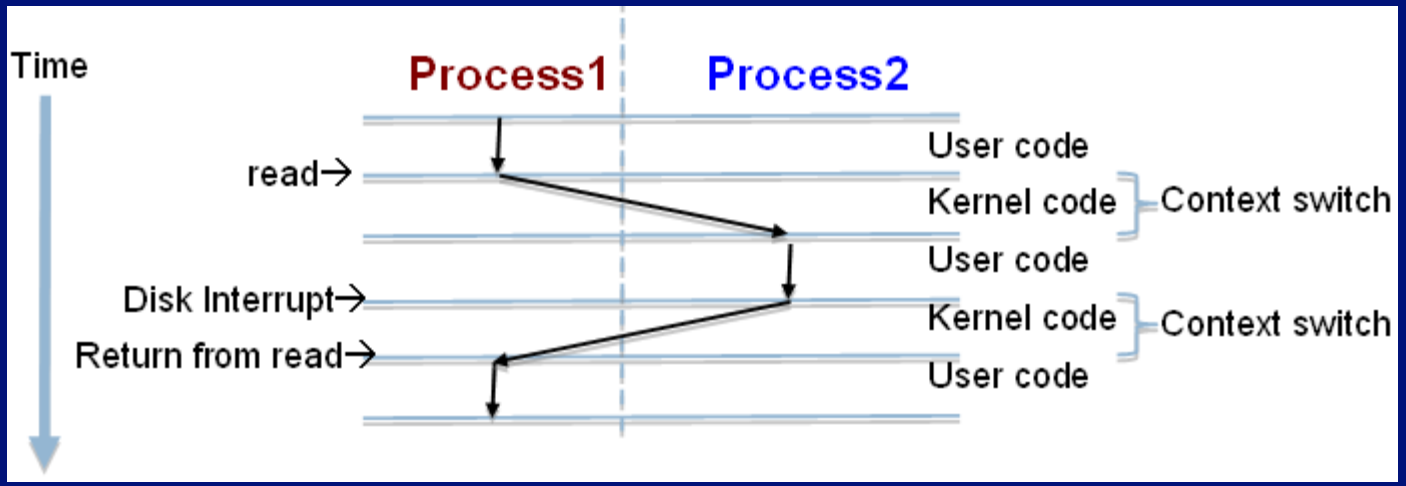
Thanks again to Chandra Krintz and Kevin Sanft.

Linux kernel – the actual OS

- Manages processes:
  - Starts, stops, suspends, swaps, manages inter-process communication, …
  - Maintains their state
- Manages files (and directories)
- Manages main memory
- Manages disk operations

CPU scheduling

- Kernel sends interrupt to a process to give another process a turn to use the CPU
- Processes can give up CPU when they don't need it (e.g. waiting on I/O device)

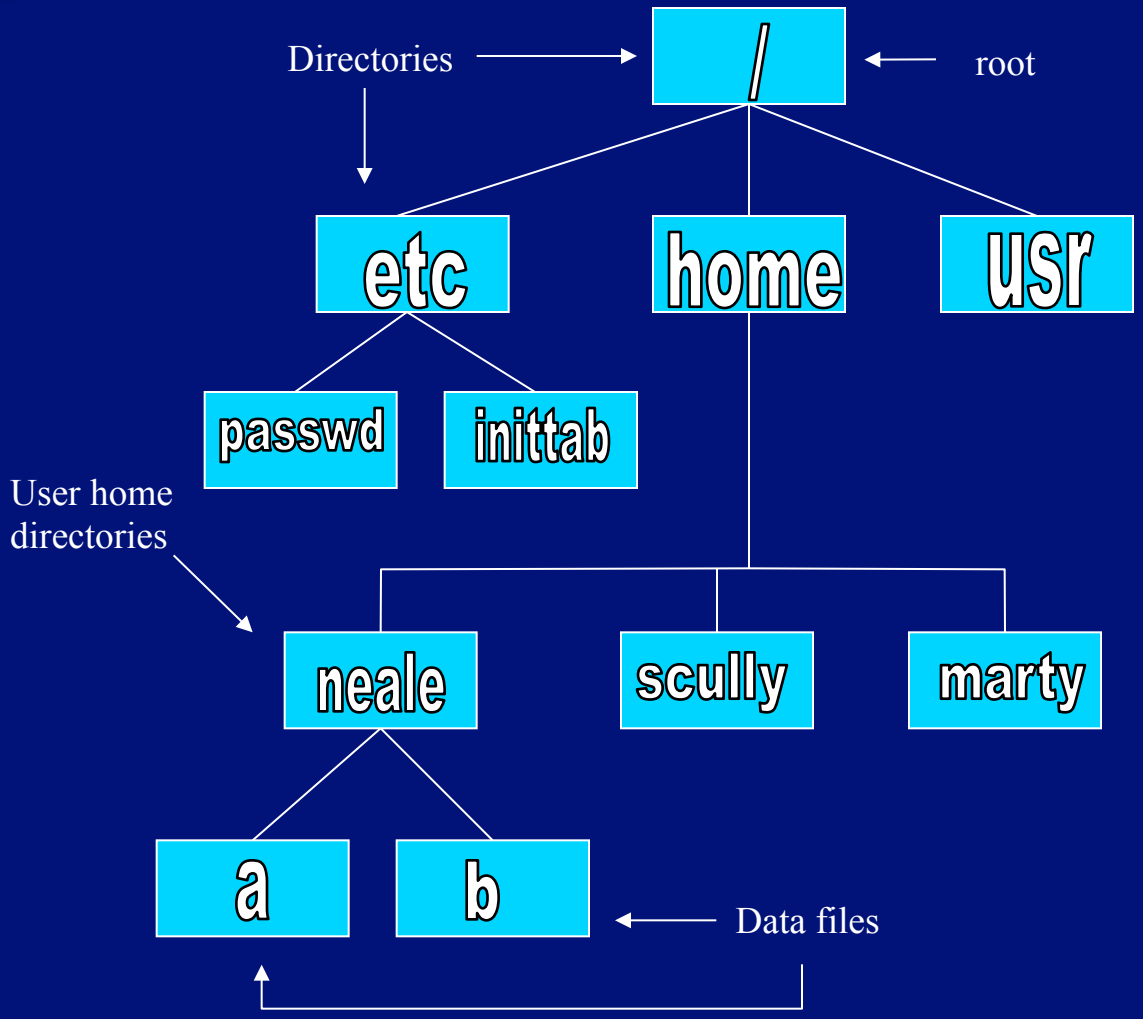


Processes *request* services from the kernel in two ways

- 1. Using system calls (read, write, fork, …)
  - OOP idea: these are the kernel's interface
  - Btw, processes access devices just like files – that's how they are represented by the kernel, and they occupy places in the file system
    - Use open, close, read, write, release, seek, …
- 2. Or indirectly, through shell commands (including programs) or library functions that, in turn make use of system calls

Linux file system

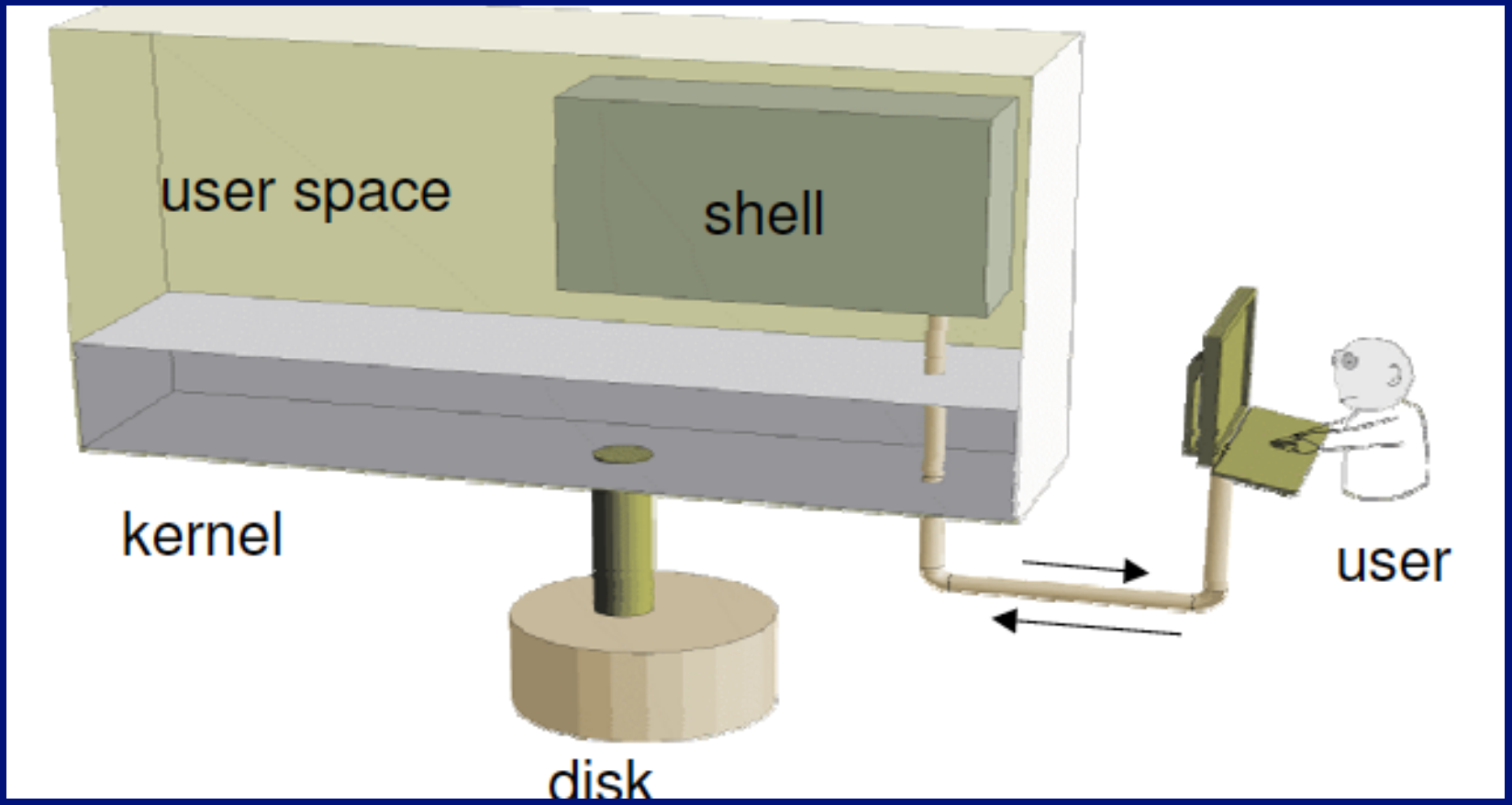
- Rooted, hierarchical
  - Data files are stored in *directories*
- A file's (full) *pathname* starts at the root
  - /etc/passwd
  - /home/neale/b



Special file names

- `.` (by itself) The current directory
  - `./a` is the same as a
- `..` The parent (toward root) directory
  - `../jane/x` go up one level then look in directory named jane for x
- `~` Your home directory
  - `~harvey` Username harvey's home directory
- Have to "escape" spaces with a backslash
  - `my\ file\ name\ with\ spaces`
  - Moral: don't use spaces in file or directory names!

Basic user interface is the shell



Shell

- A program that runs in a terminal and provides a command-line interface for user
- Also an interpreter that executes user commands
- And a powerful programming language
  - Shell script – a sequence of commands in a file
- Lots of different shells to choose from
  - sh, csh, tcsh, bash …
  - We'll focus on bash (and sh scripts) in this course

Shell scripts

Not covered in Reader (#1 just mentions)

This is just an introduction – learn much more doing lab work

Bourne shell (`sh`) programs

- Are text files with `sh` commands – e.g., `myScript`
  - To execute, can do `sh myScript`
    - The program runs in a new shell – called a child shell
  - Or `chmod u+x myScript` – then just `./myScript`
    - Requires compatible default shell (`sh` and usually `bash` okay)
- `#` – normally identifies a comment
  - Special case if line 1 – `#!/bin/sh` – identifies shell
    - Means use `sh` as child shell for this script – works in all shells
- Can access command line arguments: `$1` to `$#`
  - e.g., `cp $1 $2` # copies first to second (if files)
  - e.g., `echo $#` # prints number of arguments

`sh` variables and assignment

- `name="Jack Sprat"` # note *no spaces*
- `echo "The name is $name"` # need '`$`'
- `workdir=\`pwd\`` # use `` `...` `` to assign result of …
  - Or can use `$(pwd)` instead of `` `pwd` ``
- Similarly, `echo "date and time is \`date\`"`
- Can read from standard input and calculate too
  - `echo "enter value"`
  - `read val`
  - `doubleval=\`expr $val + $val\``
    - Or: `doubleval=$((val + val))` # "c-style expr."
  - Or just: `echo "doubled: \`expr $val + $val\`"`

`sh` control structures, and FYIs

- An `if-then-elif-else-fi` statement
  - Expression is a test: `test $# -gt 0`
  - Or simpler: `[ $# -gt 0 ]` # spaces mandatory
  - Can test file attributes too: `-d`, `-f`, `-e`, `-r`, `-w`, `-x`, …
- A `while-do-done` statement: same expressions
- A `for-do-done` statement: `for variable in list`
  - List is command line arguments if `in` clause omitted
- FYI: can program *any* shell, but different syntax
  - Also "scripting languages" (e.g., Perl, Python, …)
- Examples at `~mikec/cs32/demos/scripts/`