Operating Systems!

An Overview of Operating Systems (Part 3)

Prof. Travis Peters

Montana State University

CS 460 - Operating Systems

Fall 2020

https://www.cs.montana.edu/cs460

Today

short/(quick) Video!

verify that reese & travis have access to your *PRIVATE* GitHub repo

- Announcements
  - *(Last) Reminder.... PA0 Due!*
    Sunday [09/06/2020] @ 11:59 PM (MST)

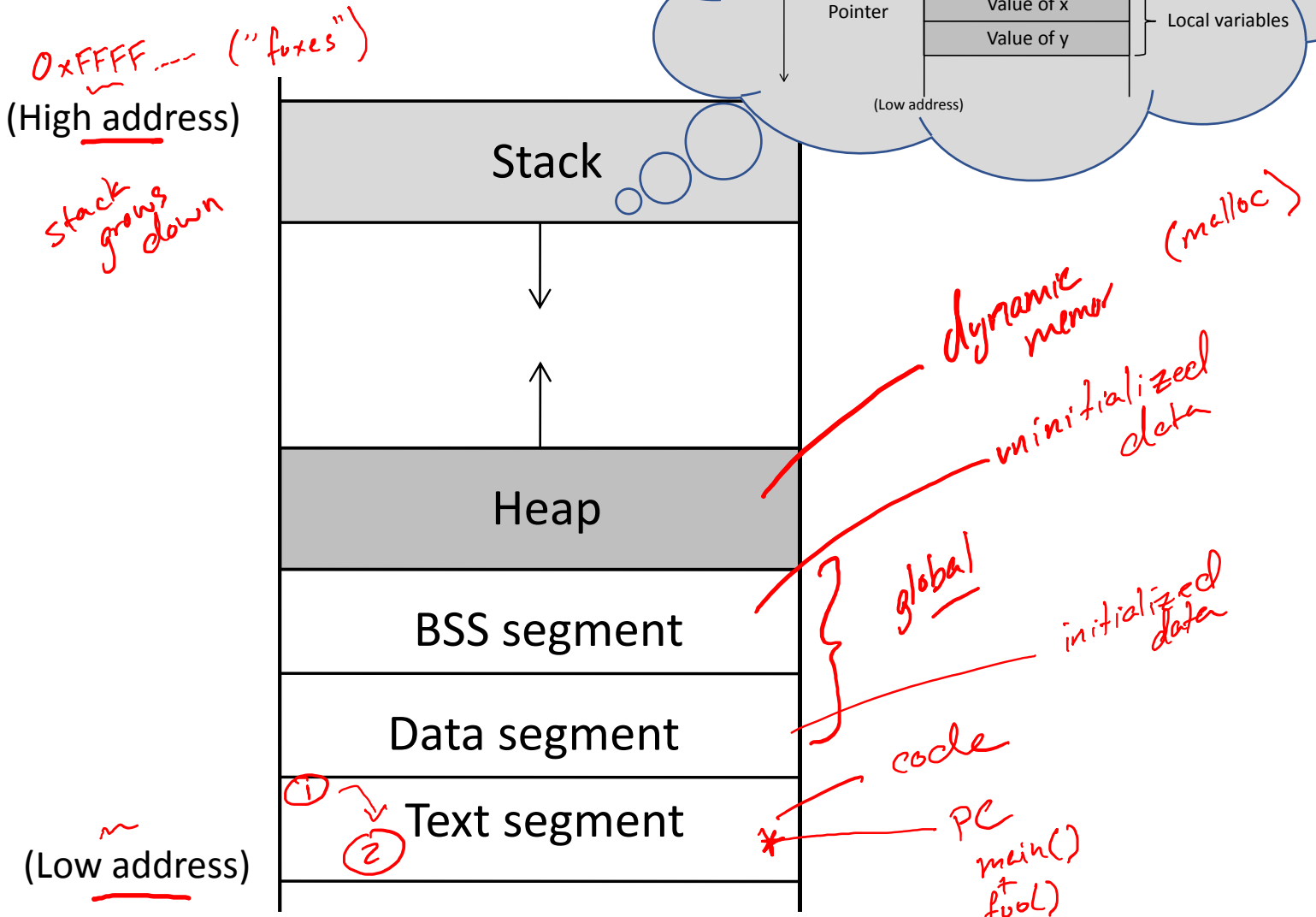
    NO D2L! *Code pushed to GitHub == submitted!*

  - *Heads up.... PA1 has been released!*
    Sunday [09/20/2020] @ 11:59 PM (MST)

- Learning Objectives
  - Understand the big ideas and constructs behind **operating systems**

Memory Layout

Q1: Have you seen diagrams like this before?!



Memory Layout

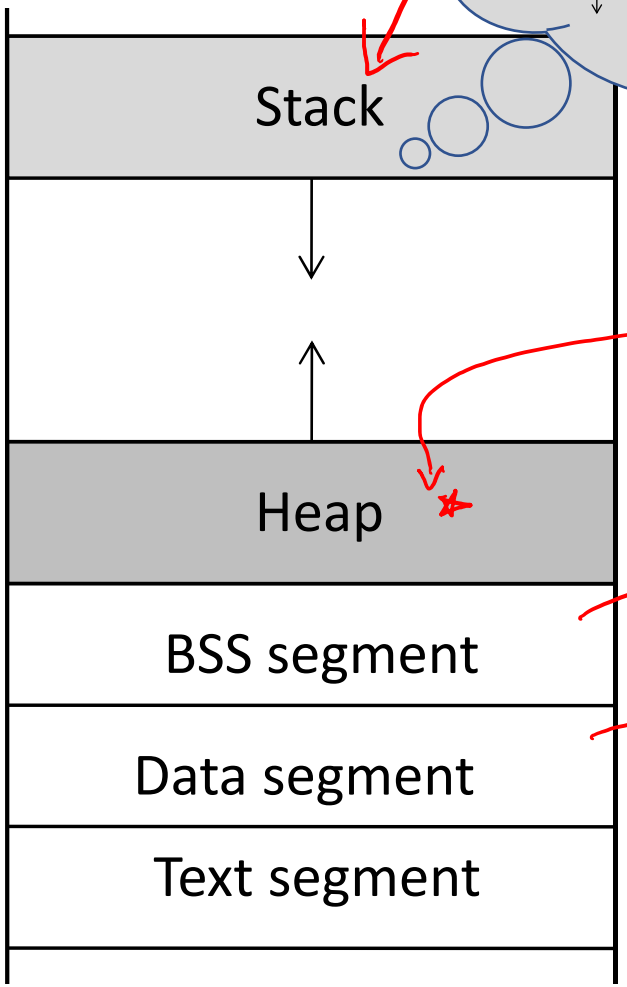
Q2:

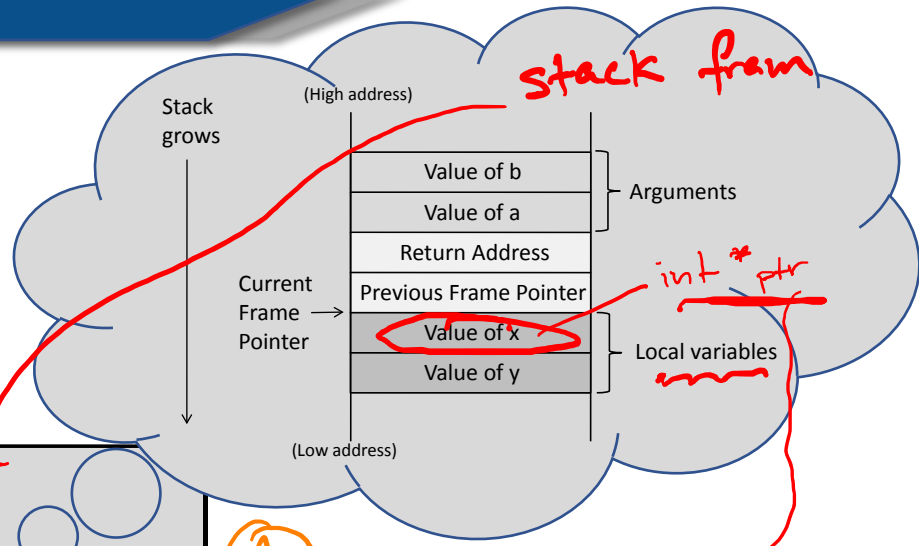
(High address)

```c
int x = 100;
int main()
{
    int a = 2;
    float b = 2.5;
    static int y;

    int *ptr = (int *) malloc(2*sizeof(int));
    ptr[0] = 5;
    ptr[1] = 6;

    free(ptr);
    return 0;
}
```

```
# Run prog. Where do variables go in memory?
$ gcc vars.c -o vars && ./vars
```

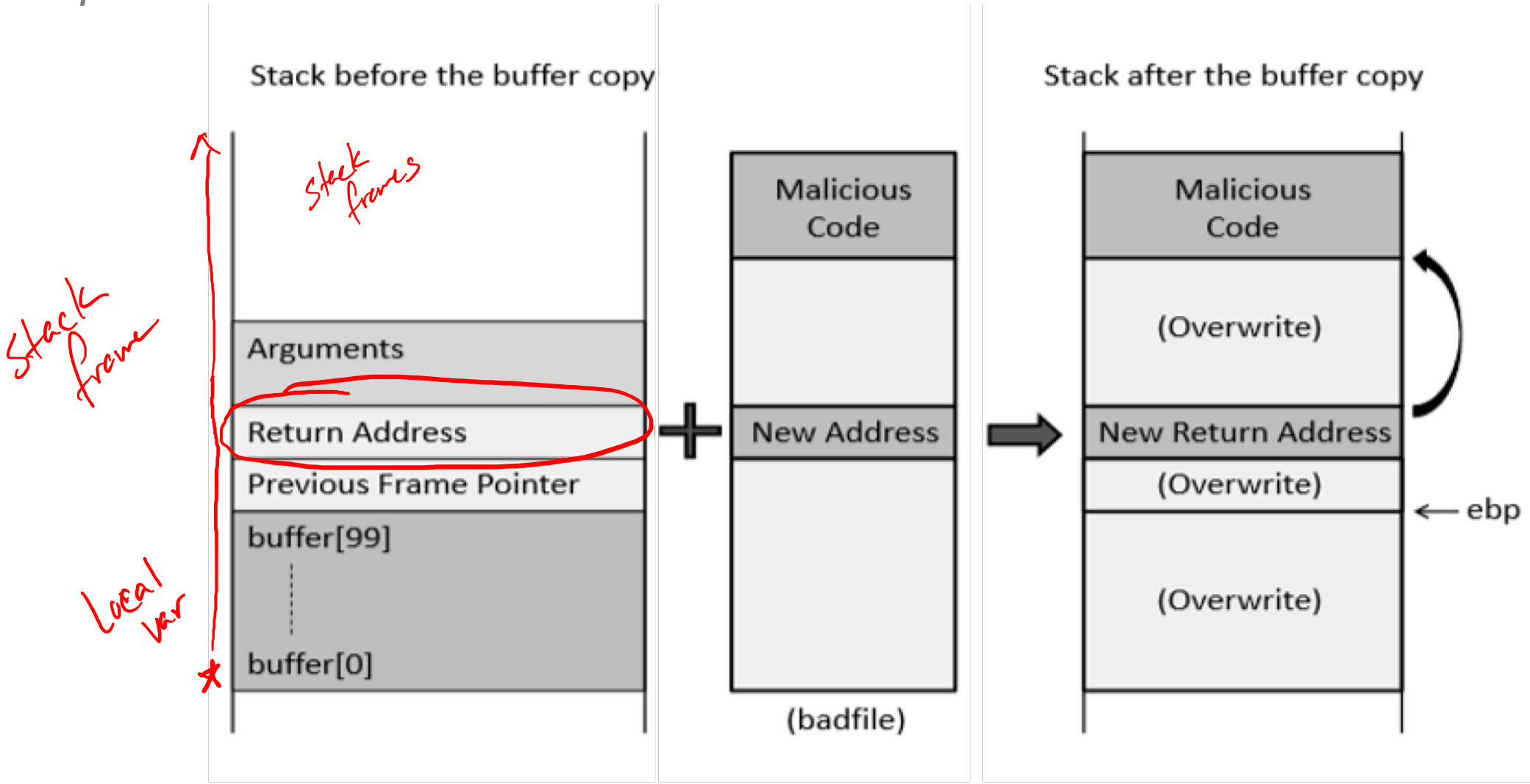
(Low address)





System Vulnerabilities

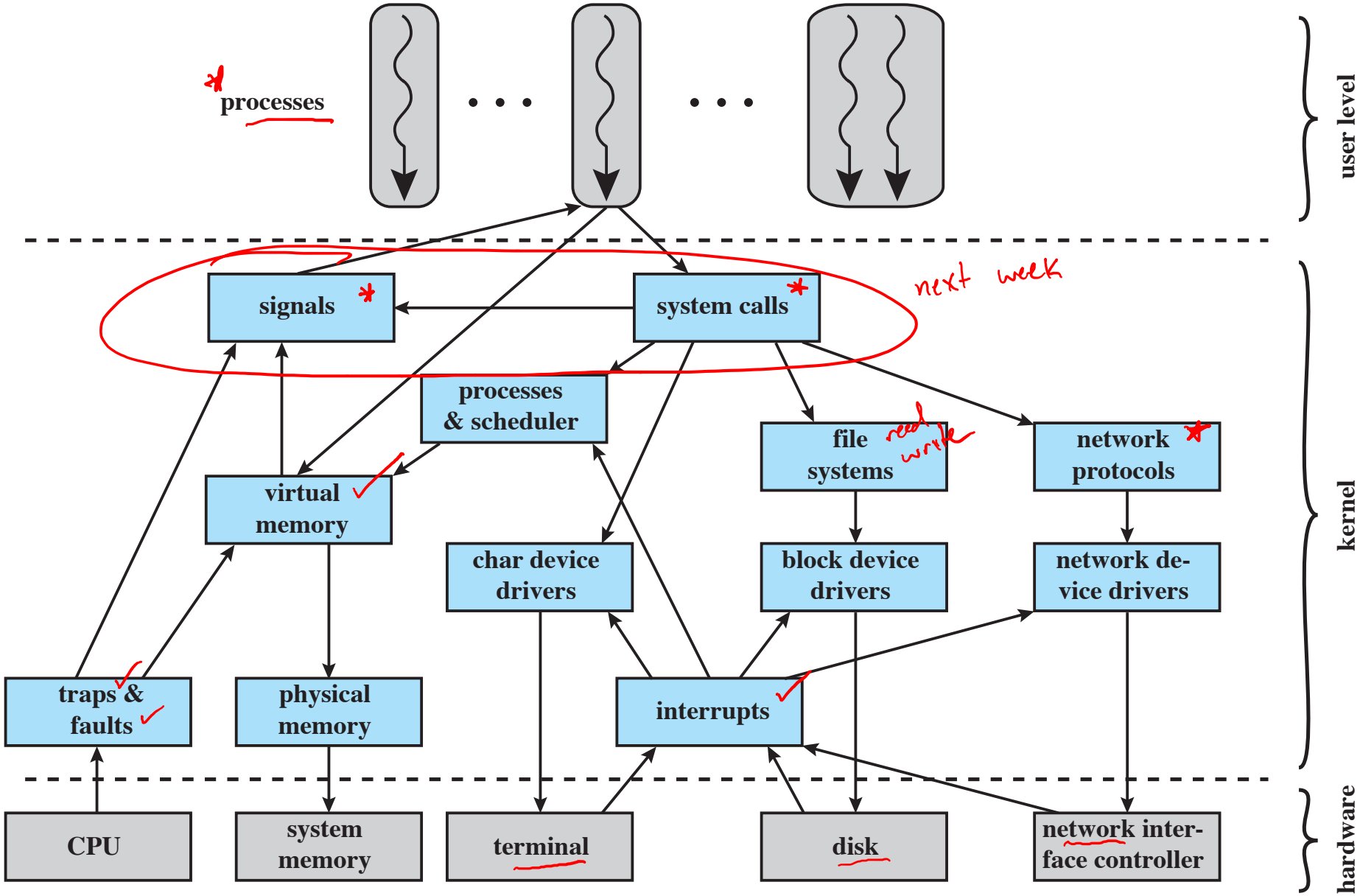
Example: Buffer Overflows & Malicious Code Execution



OS IRL

What follows are the big ideas,
which gives us context for the role and purpose of the OS.

Linux Kernel Components



A Glimpse At What You'll Do *(The important part!)*

Yalnix! (≈6 checkpoints)

1. sketch/pseudocode kernel data structures, traps, syscalls, major functions
2. get yalnix to boot
   (setup virtual memory + interrupt vector)
   & run "Idle" process
3. get a true "Init" process to run
4. implement fork(), exec(), wait()
5. implement some I/O
6. implement remaining syscalls
   (e.g., Process Coordination, I/O, IPC, Synchronization)
7. *extras for grad students?! ☺*

Yalnix is probably a lot closer to a traditional UNIX architecture than a modern version of Linux... but now you can probably understand a lot more about the Yalnix overview diagram!