“OS” Components

CSE 333 Autumn 2025

Instructors: Naomi Alterman, Chris Thachuk

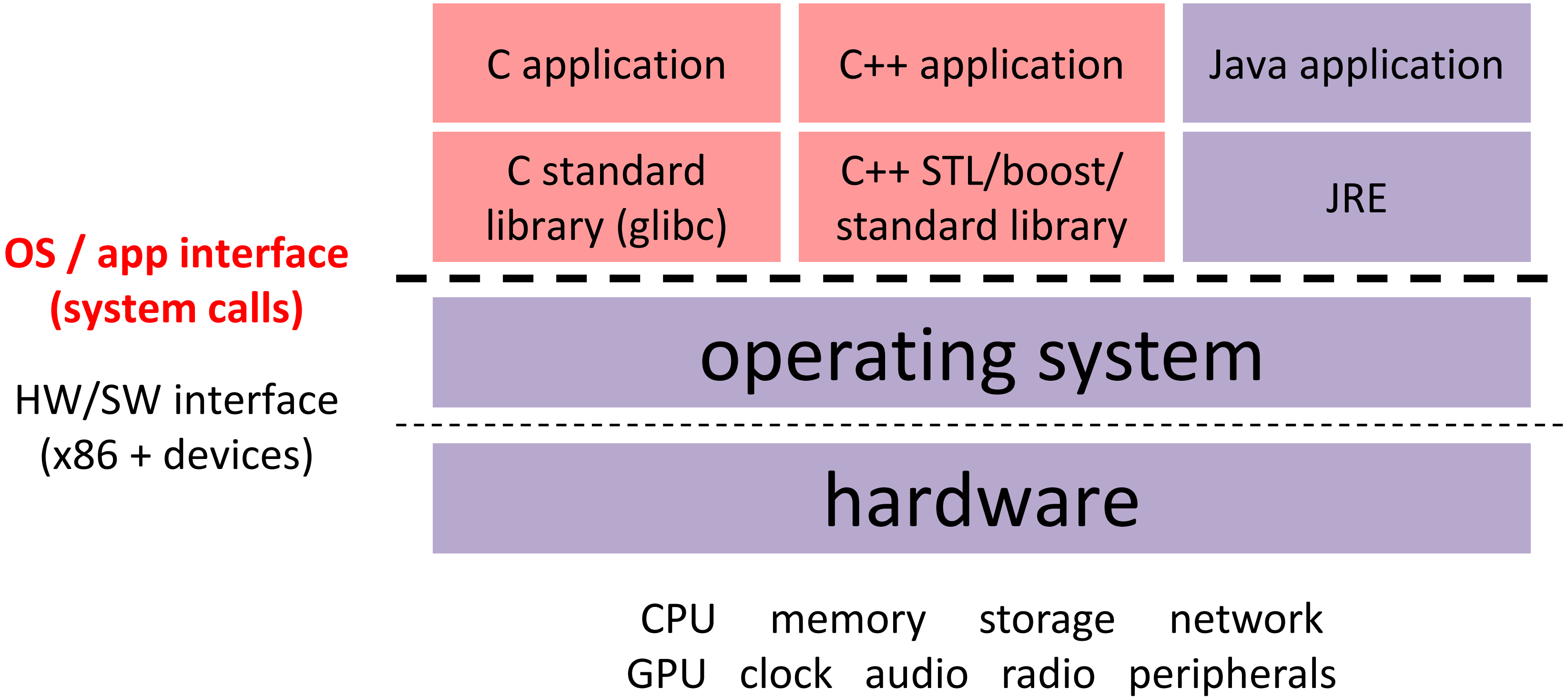
Teaching Assistants:

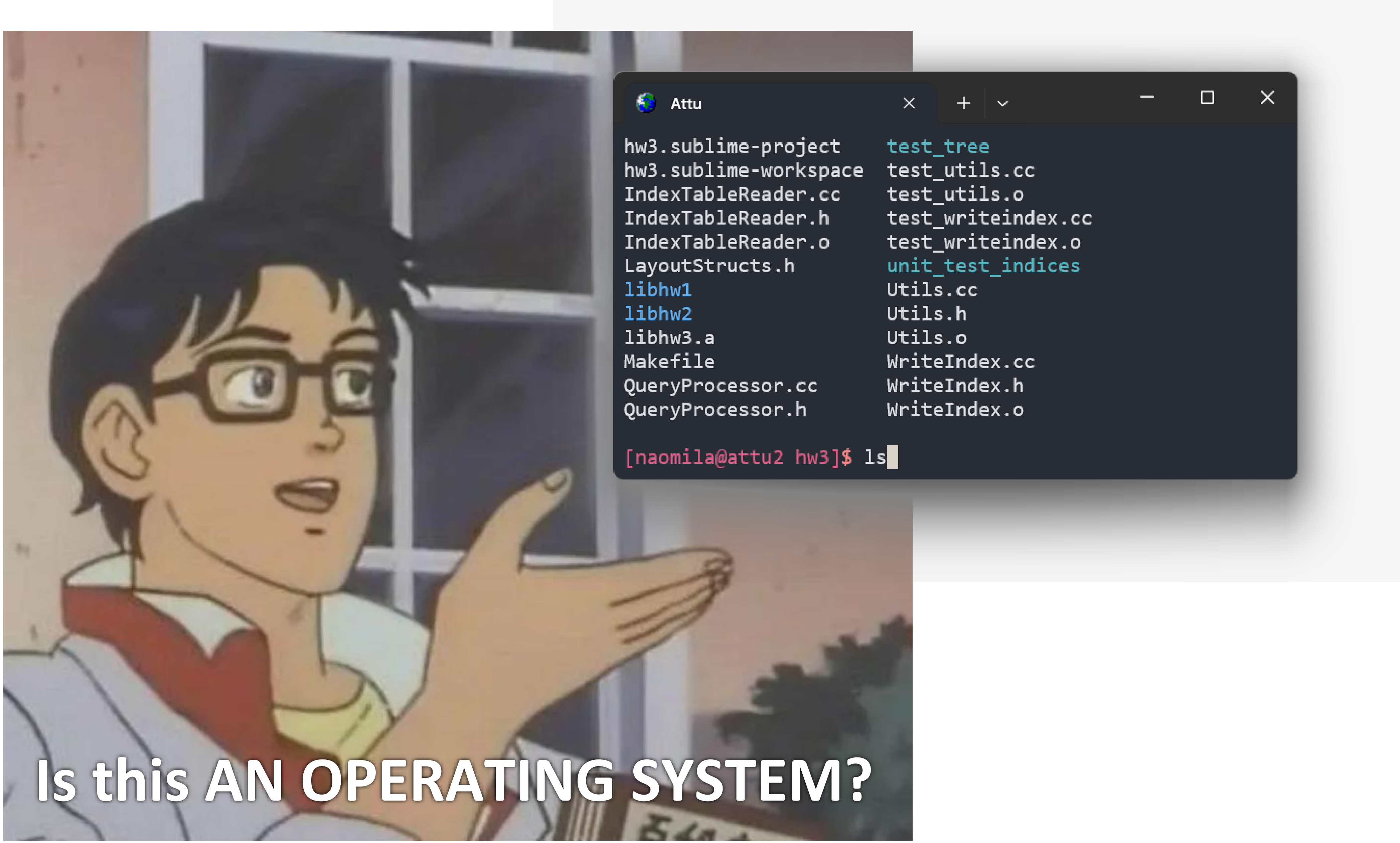
Ann Baturytski
Derek de Leuw
Blake Diaz
Rishabh Jain
Chendur Jel Jayavelu
Lucas Kwan
Irene Xin Jie Lau
Nathan Li
Maya Odenheim
Advay Patil
Selim Saridede
Deeksha Vatwani
Angela Wu
Jiexiao Xu

Today

- **Major OS components**
- Files and filesystems
- IPC, isolation and containers
- The OS Family Tree

That 333 view of the world again:


C application
C++ application
Java application
C standard library (glibc)
C++ STL/boost/ standard library
JRE
OS / app interface (system calls)
operating system
HW/SW interface (x86 + devices)
hardware
CPU memory storage network
GPU clock audio radio peripherals



```
hw3.sublime-project    test_tree
hw3.sublime-workspace  test_utils.cc
IndexTableReader.cc    test_utils.o
IndexTableReader.h     test_writeindex.cc
IndexTableReader.o     test_writeindex.o
LayoutStructs.h        unit_test_indices
libhw1                 Utils.cc
libhw2                 Utils.h
libhw3.a               Utils.o
Makefile               WriteIndex.cc
QueryProcessor.cc      WriteIndex.h
QueryProcessor.h       WriteIndex.o

[naomila@attu2 hw3]$ ls
```

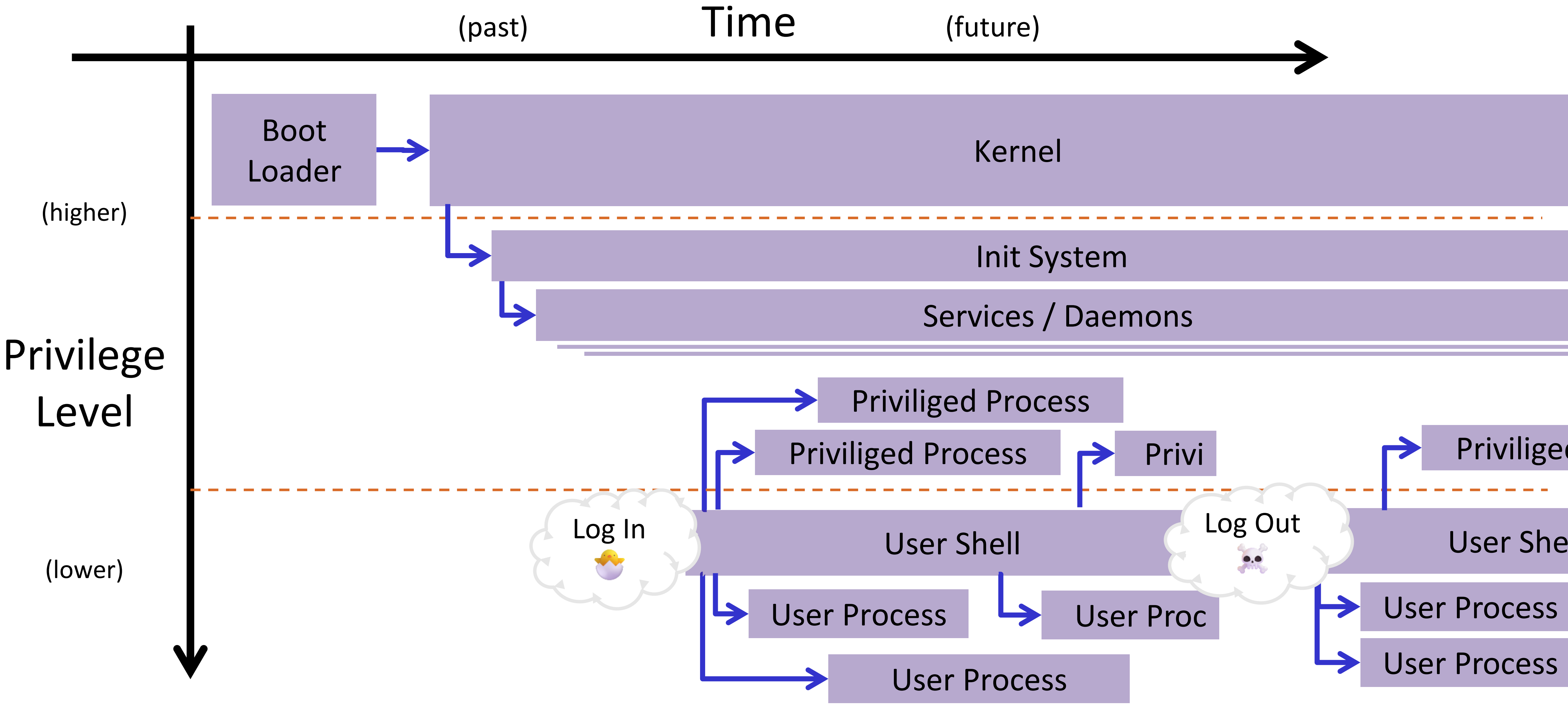
Let's brainstorm

What *are* the different components of an operating system like Linux?

Keep going until Naomi is satisfied 😈

https://pollev.com/naomila

"A Day In the Life of Your Computer"

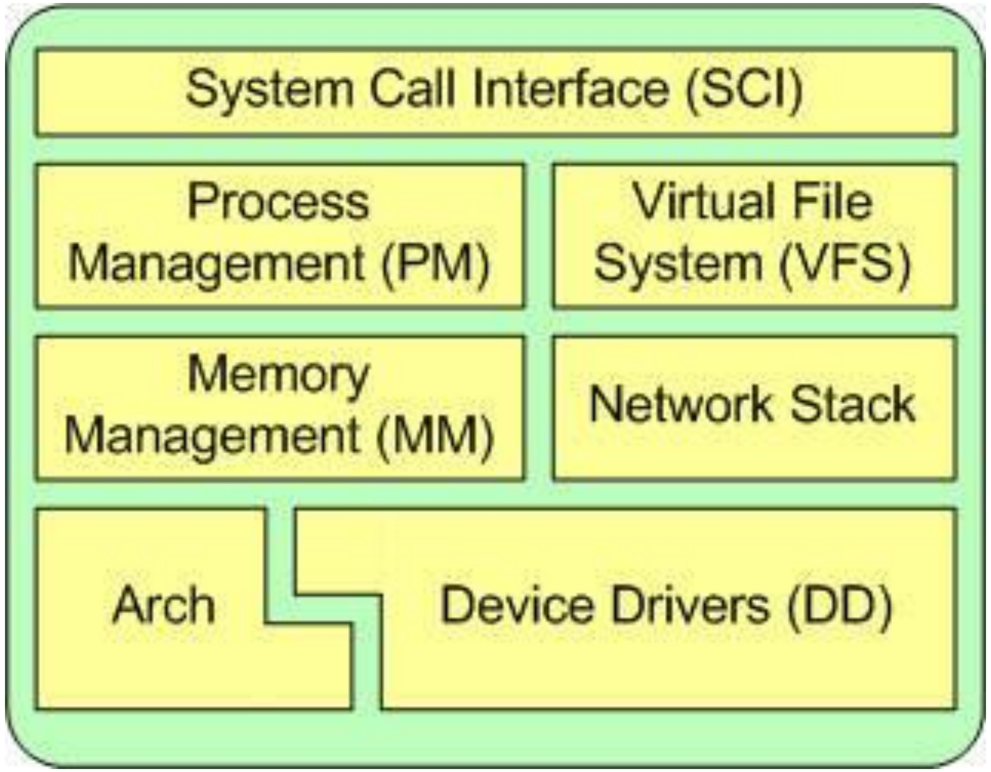


Boot loaders

- First software that runs when the CPU is powered on
- Separated into multiple stages:
  - First stage (“BIOS” or “UEFI”, among others) –
    - Hardware-specific
    - Power on all system components and check they’re operating properly.
    - Running when the manufacturer logo flashes up on the screen and your Macbook goes “🎶 DUMMMMMMM 🎶 .”
  - Second stage –
    - OS-specific
    - Set up conditions for the OS to be happy and pass control off to it
- Even systems software engineers try not to think about boot loaders
  - Once they’re written, no need to worry about them

Kernel

- Code that has complete control over system hardware and software
- Responsible for:
  - hardware interaction (drivers)
  - process scheduling
  - memory management (virtual memory)
  - shared resource management (files, networking)
- Focus of CSE 451



https://developer.ibm.com/articles/l-linux-kernel/

Daemons

- Lots of software needs to run non-interactively
  - The SSH server
  - Anti-virus software
  - The window manager which composites all the running graphical apps into one image

- In UNIX-land, long-running non-interactive processes are called **daemons** (pronounced “demon”)
  - On Windows, they’re called “services”

- Daemons are user-space programs, but often have ***elevated* permissions** beyond normal applications

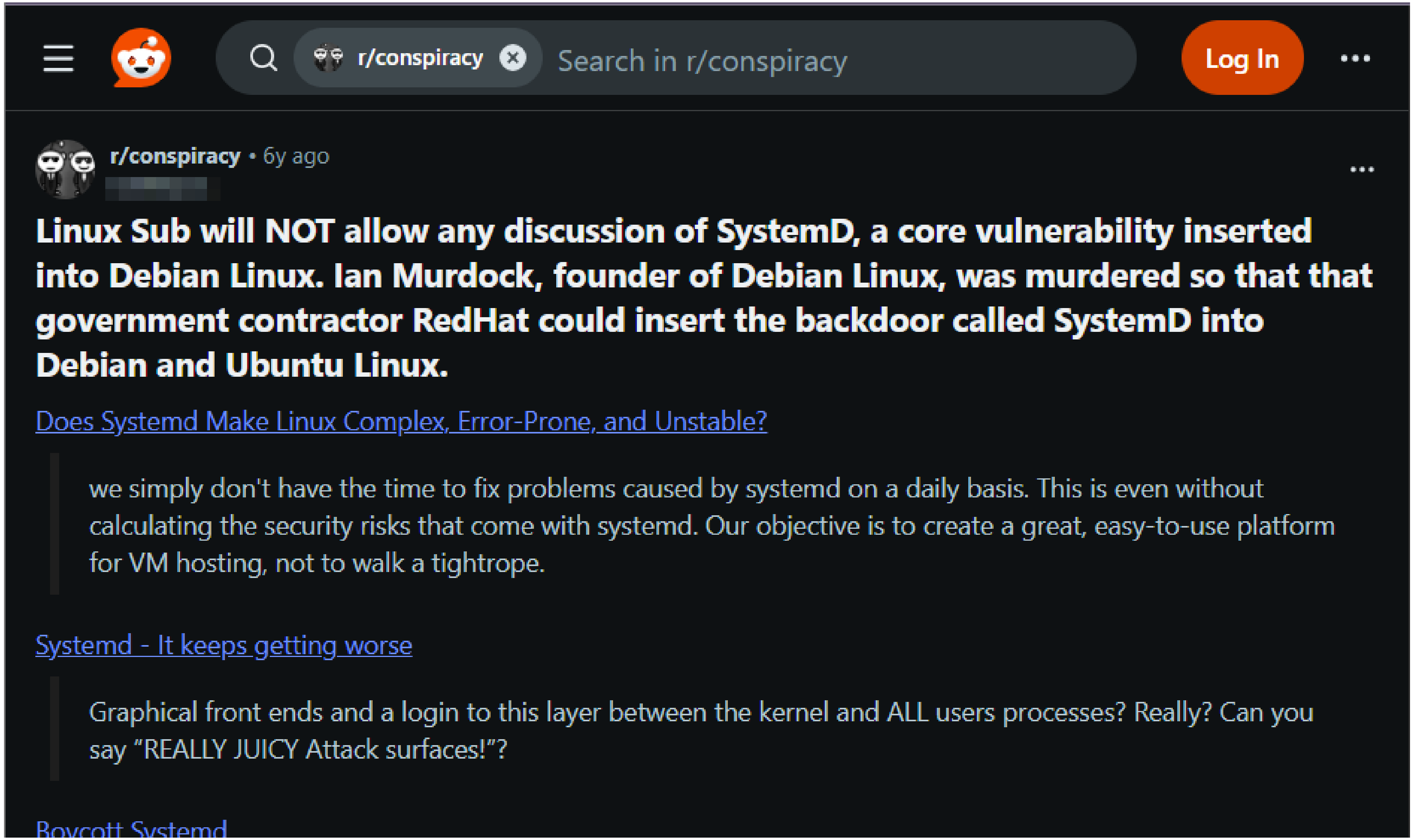
- Take a look at the running daemons on attu: systemctl list-units *.service

The init system

- The init system is userspace software responsible for managing the lifecycle of all system daeomns
  - The first process, started automatically when the OS boots
  - Starts all other system daemons
  - Monitors them and restarts them if they crash
  - Runs with **root-level permissions** (the highest non-kernel permission level)

- On Linux:
  - Formerly: “SysV Init” – launches an ordered list of shell scripts
  - Presently: “systemd” – Does its own fancy thing (interact with `` `systemctl` `` command)

People care (too) much about init systems



r/conspiracy • 6y ago

[redacted]

Linux Sub will NOT allow any discussion of SystemD, a core vulnerability inserted into Debian Linux. Ian Murdock, founder of Debian Linux, was murdered so that that government contractor RedHat could insert the backdoor called SystemD into Debian and Ubuntu Linux.

Does Systemd Make Linux Complex, Error-Prone, and Unstable?

> we simply don't have the time to fix problems caused by systemd on a daily basis. This is even without calculating the security risks that come with systemd. Our objective is to create a great, easy-to-use platform for VM hosting, not to walk a tightrope.

Systemd - It keeps getting worse

> Graphical front ends and a login to this layer between the kernel and ALL users processes? Really? Can you say “REALLY JUICY Attack surfaces!”?

Boycott Systemd

Process permissions

- From Bash, can launch processes with elevated permissions using “**`sudo`**” command (“super-user do”)

- From C, can query or change the owner of your code’s process using these syscalls:
  - **`uid_t getuid()`**
  - **`int setuid(uid_t uid)`**

- “uid_t” is an integer type that uniquely identifies a give user account within the running operating system
  - UID 0 always represents the “root user,” or administrator, will full permissions over the entire system.

Shells

- Shells are software systems that provide a human interface to the computer
  - **Command-line shells** like Bash, Zsh, powershell and nushell use ***text streams*** (stdin, stdout, stderr) as their interaction paradigm
  - **Graphical shells** use collections (“windows”) of interactive widgets (buttons, text boxes, scrollbars, etc) as their interaction paradigm

- These are not required OS components!
  - Attu doesn’t have a graphical shell
  - Windows largely doesn’t use command-line shells
  - The chip embedded in your husky card doesn’t have *any* shell (but it does run Java 😉)
    - *…lol wait what?*

Today

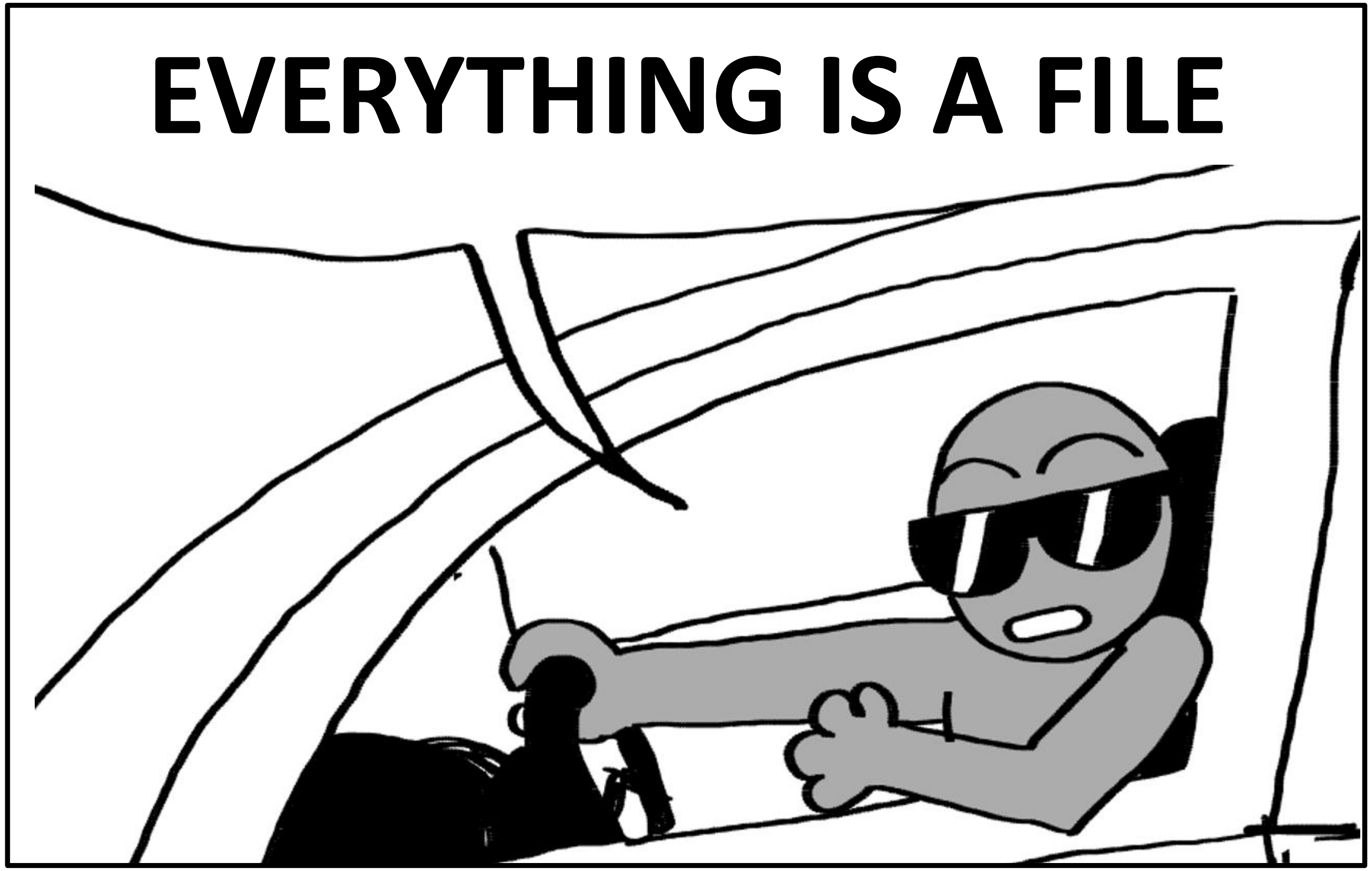
- Major OS components
- **Files and filesystems**
- IPC, isolation and containers
- The OS Family Tree

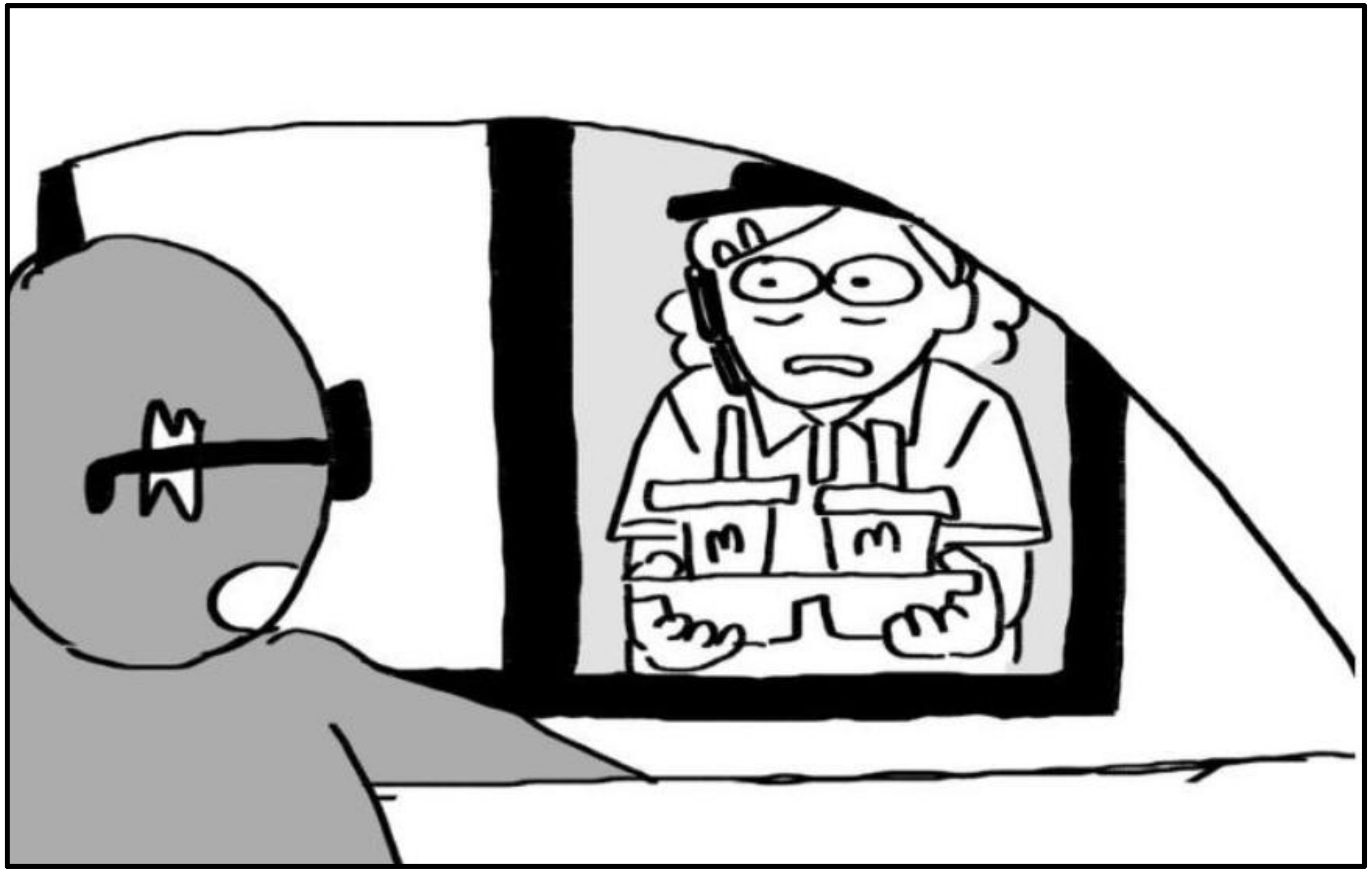
Files, to a normal person

- Discrete documents with names and types
  - Executables (machine code)
  - JPEGs of our favorite cats
  - Rich text documents full of writing
  - Database stores
  - Folders/directories, which contain more files

- **Dear normie, what are some things that are *not* files?**

Files, to a UNIX person





m
m

The UNIX file philosophy

- Represent ***every single thing*** in the computer as a file
  - Documents
  - Folders
  - Network connections
  - Hardware components
  - Running processes
  - Abstract concepts like 'zero' and 'random number'

- All files exist in one unified directory structure
  - Starting at the root folder: "/"
  - Similar to how we use URLs to identify content on the internet

ls /

- `bin` - Core system executables like ls and cat
- `lib` - Core shared libraries like glibc
- `boot` - Bootloader files
- `usr` - User-installed binaries and libraries
- `dev` - Files representing hardware components
- `proc` - Files representing running processes
- `etc` - Configuration files and "other stuff"
- `home` - User home directories ("~")
- `media` and `mnt` - External storage like USB thumb drives
- `var` - Working files that can rapidly grow/shrink (like logs)
- `tmp` - "Scratch space" stored in RAM (disappears on reboot)

Run:
`man hier`
to learn more!

File operations and permissions

- We can **read** files ("R"), **write** files ("W"), and **execute** files ("X")
  - For text files like a Python script, R W and X mean what you'd expect
  - For *device* files, reading and writing might cause a device driver to literally retrieve data or send data to a hardware device

- In general, we don't want *everyone* to be able to R, W or X *everything*. Instead, we grant **permission** to R, W or X based on three coarse categories:
  - "Owner" – A single user account which has the final say over permissions
  - "Group" – A group of user accounts which might have privileged access
  - "Everyone" – Everyone else

- See terminal commands `` `chmod` ``, `` `chown` `` and `` `ll` `` for working with permissions

Filesystems

- The **code** and **low-level structures** which are responsible for organizing, reading and writing files are called a **filesystem** (or "fs")
  - Only some filesystems store bytes on hard disks
  - Others, like "/tmp", are **RAM filesystems** that store bytes in RAM
  - Others still, like "/proc", are **virtual filesystems** that don't store bytes at all

- The "root fs" of your computer (responsible for the "/" folder) delegates control of *different* folders to *different* filesystems
  - The process of adding a filesystem to the root tree is called **mounting**
  - The path the filesystem is mounted at is called the **mount point**

Today

- Major OS components
- Files and filesystems
- **IPC, isolation and containers**
- The OS Family Tree

Inter-process Communication

- Processes often need to communicate with each other ("IPC")
- Use a number of different mechanisms:
  - Normal files ("lockfiles")
  - Sockets (TCP/UDP sockets connected to localhost, "UNIX domain sockets")
  - **Remote procedure calls (RPC)**: calling functions from code in another process's virtual memory
    - Many RPC management systems, usually called "**system busses**" ("**dbus**" in Linux-land, "**COM**" in Windows-land)

Security and isolation

- We can see everything running on attu!
  - That's creepy!

- Most OSes offer some form of resource isolation through **namespacing**
  - Similar to C++ namespaces
  - Namespaces for sockets, namespaces for processes, namespaces for users, …
  - The namespaces a process runs in dictate what users/sockets/etc it can be aware of. Everything outside of those namespaces is invisible to it.

- Desirable for running software you don't entirely trust, or that you need to upgrade often

Container systems

- A collection of namespaces that completely hides the rest of the computer from a process is called a **container**

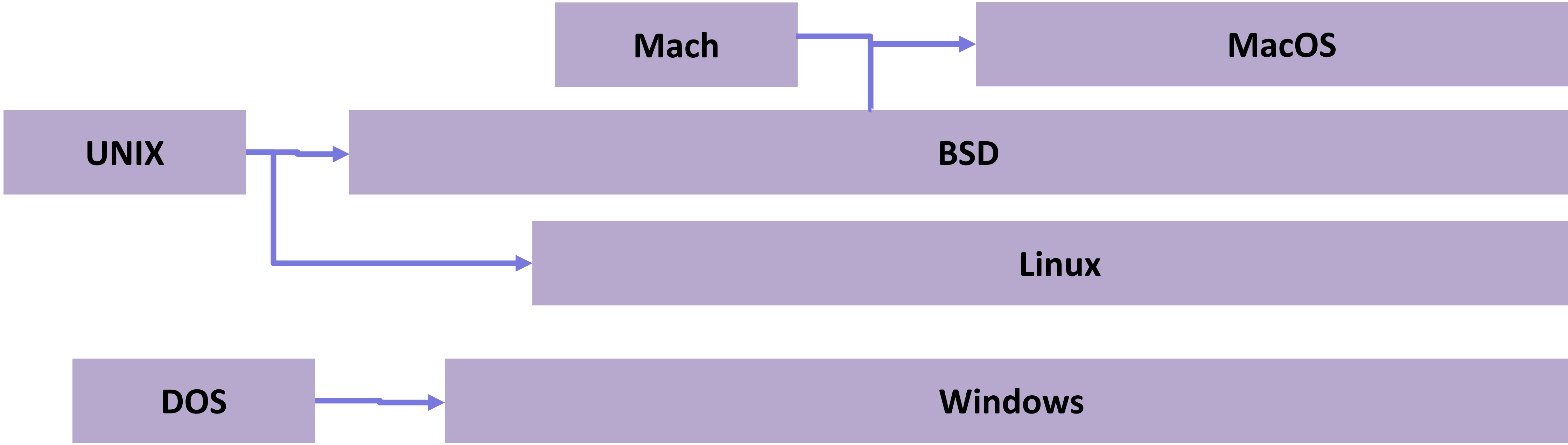
- Folks have built lots of tools to create and manage containers
  - Linux containers ("LXC")
  - Docker (big commercial player)
  - Apptainer/Singularity (open source, common in scientific computing)

Today

- Major OS components
- Files and filesystems
- IPC, isolation and containers
- **The OS Family Tree**

OS Landscape

- Uncountably many OSes, but broadly speaking:



- In grueling detail: https://eylenburg.github.io/os_familytree.htm