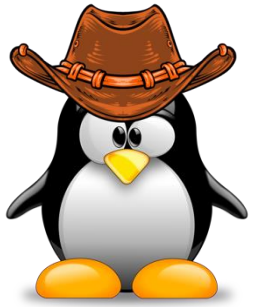


# CSE 330:
# Operating Systems

Adil Ahmad

**Lecture #1:** Introductions and course details

# Welcome to CSE 330!

A little bit about your instructor

Undergrad
(2012-2016)

PhD
(2016-2022)

Asst. Professor
(2022- now)

Research interests: **security,** **systems,** and **architecture**

# Let’s get to know **you!**

Please submit the following survey

**Allowed time:** 10 minutes

# Source material acknowledgements

ARM education initiative

The xv6 operating system handbook

Pedro Fonseca (Purdue University)

Xiaokuan Zhang (George Mason University)

Franz Koshoek, Russ Cox, Robert Morris,  Adam Belay (MIT)

Zhicao Cao, Ming Zhao (Arizona State University)



Massachusetts
Institute of
Technology

FreeBSD

Windows 11

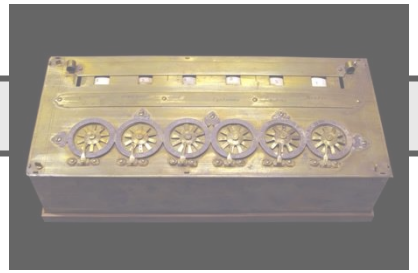
# What is an Operating System (OS)?

# A historical perspective on Computers

If you've ever wondered if anything good has come of taxes!

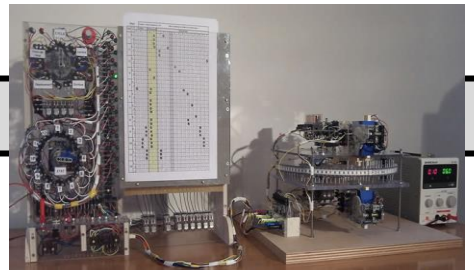
Abacus
(ancient times)

Addition, subtraction, even squares and cubes



Pascal's calc.
(~1600s)

Complex tax collection



Turing machine
(1936)

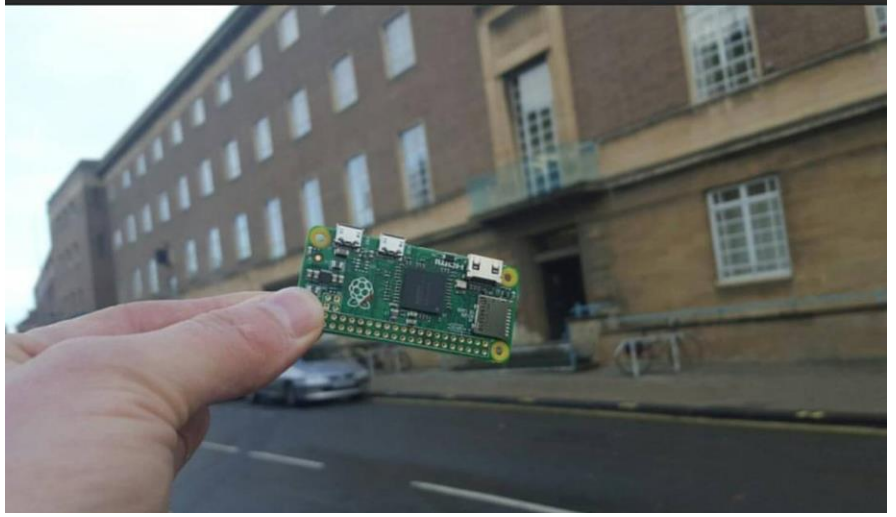
Break WW2 ciphers, gen. computational model

Other important contributions from Khwarizmi, Lovelace, Napier, Babbage, etc.

# A fun view of how computers have changed

**Elliott 405 Computer: 1957**
- $850,000
- 1KHz Single-Core CPU
- 16 KB RAM (Memory)
- 1 MB Storage (Storage)
- 21 Cabinets (1 is shown in the Picture)
- 5 Tons for the Full Computer


**Raspberry Pi Zero 2W: 2022**
- $15
- 1.5 GHz Quad-Core CPU
- 1/2/4GB RAM (Memory)
- 256GB microSD (Storage)
- 1 Stick (Shown)
- 46 Grams (0.00005 Tons)



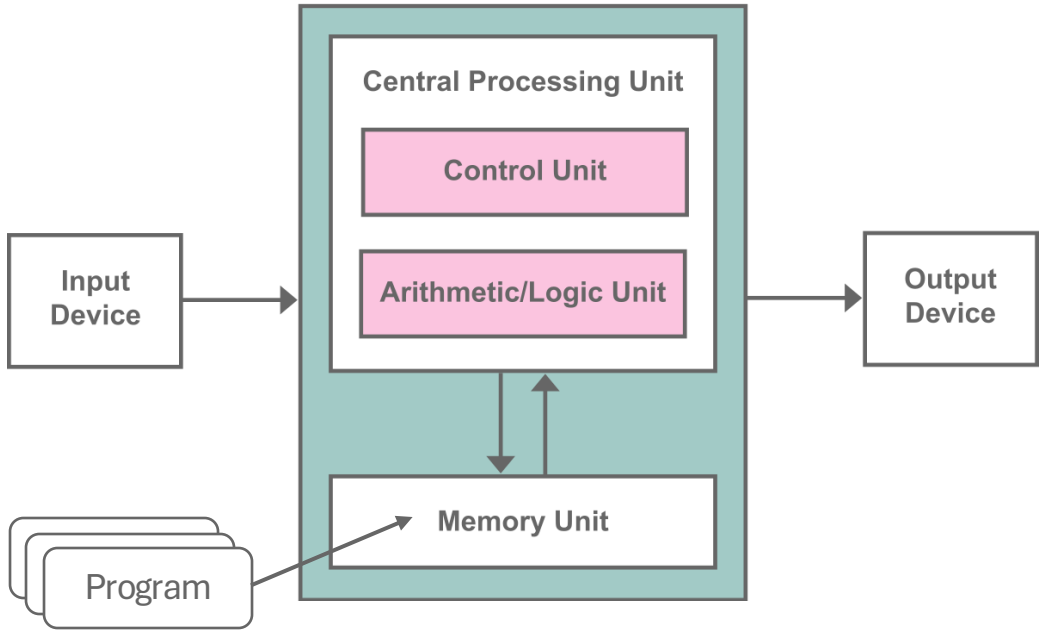
# From special-purpose to **general-purpose** computing

Hard to redesign hardware again

- Turing's machine was designed for programmable computations

**Von-Neumann architecture** made arbitrary computation even simpler

- Programs sent to a memory unit which is acted upon by a CPU




**Who manages these programs?**

# A historical perspective: Operating systems

## Era of no operating systems (sad!)

- Transfer programs using simple switches; programmer must set-up everything

## Then, came the human operator

- Human operator would take punch cards; manually run them and return results

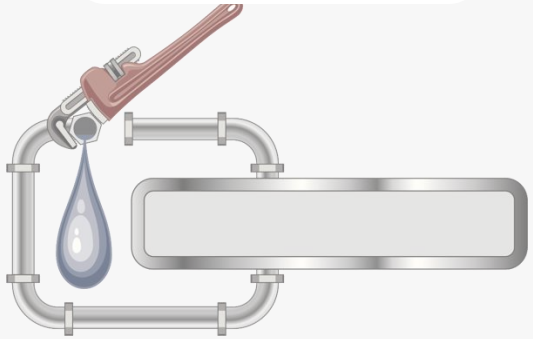
## And finally, the early OS

- Takes a batch of punch cards; automatically runs them one at a time

A punch-card machine

# **Three important duties** of the modern OS

## Plumbing



Bootloader/BIOS

Privilege management

Device drivers

MMIO/DMA interface

## Managing


Memory management

Process loading

Process scheduling

Virtualization

## Policing


System auditing

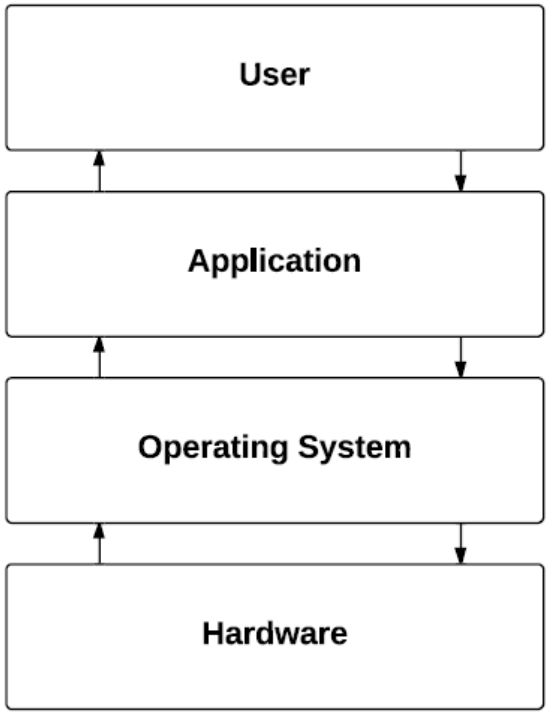
Enclave computation

Defences, e.g., KASLR

Software isolation

# OS does the **plumbing** in modern computers

Provides an abstraction for the messy hardware




Manipulating hardware requires: (a) programming knowledge and (b) understanding of the hardware

With an abstraction, users do not care about “pipes”, rather they can simply “turn the taps on or off”.

# Consider writing to a disk without an OS

Copy data from process memory to device buffer

Memory and device buffer management

Find the correct position of a write on a disk

Deep knowledge of all devices and tracking

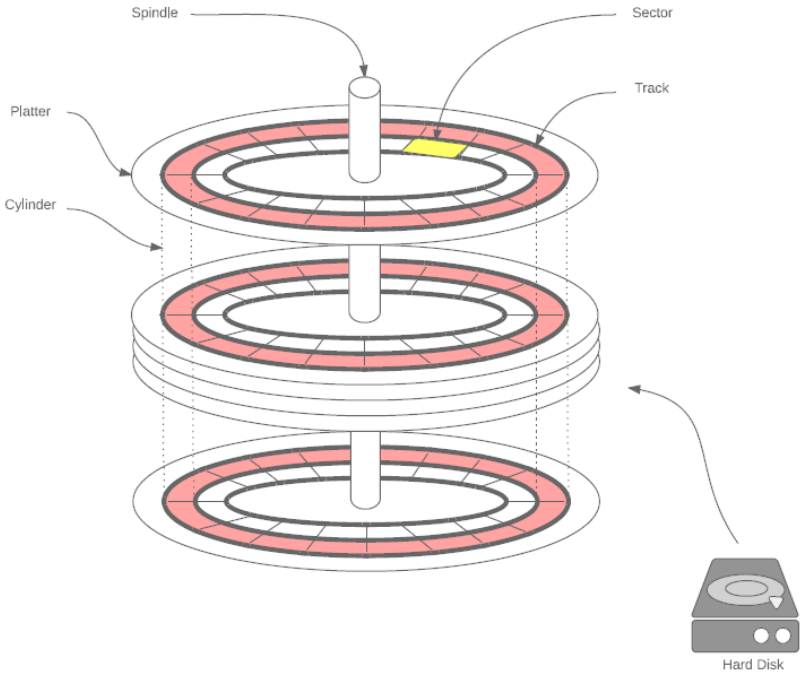
Send a command to the device for a write

Device-specific commands and IO capabilities

Receive response and send it to the user

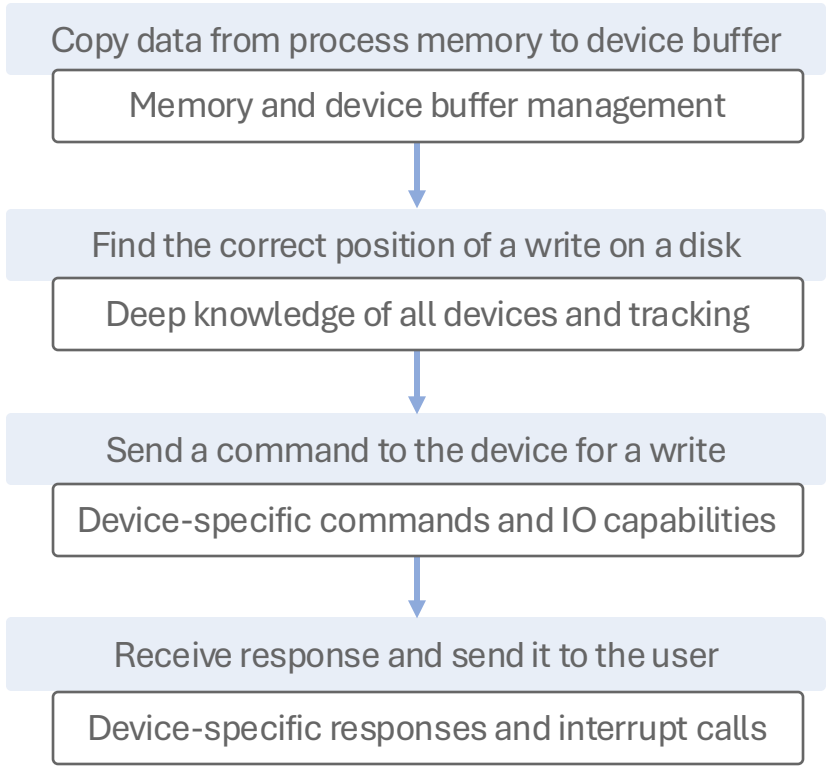
Device-specific responses and interrupt calls

A now-ancient spinning hard drive




Even this is an abstracted view of all the different things that happen

# OS provides a simplified abstraction for disk writes




write(device, data, size) system call for these tasks

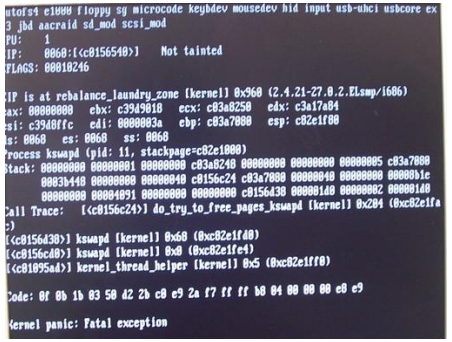
An even more abstracted function:

- A unified (memory/disk) file "*FileId*"
- And then use a library such as "C stdio"
- Write an integer variable, *datum*, onto the disk
  at an implicit file offset
- *fprintf*(*FileId*, *"%d"*, *datum);*

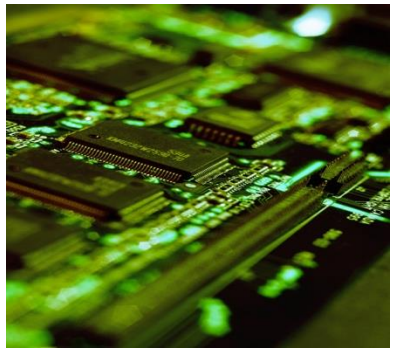
# Other examples of simplified-by-the-OS interfaces

A copy-paste example on modern machines


User applications – system call –




OS – hardware instructions –

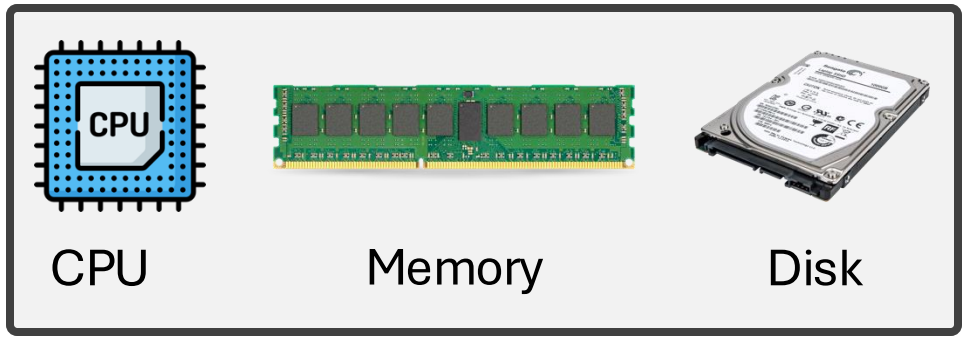


hardware

**“All system complexities can be handled by another level of abstraction”**

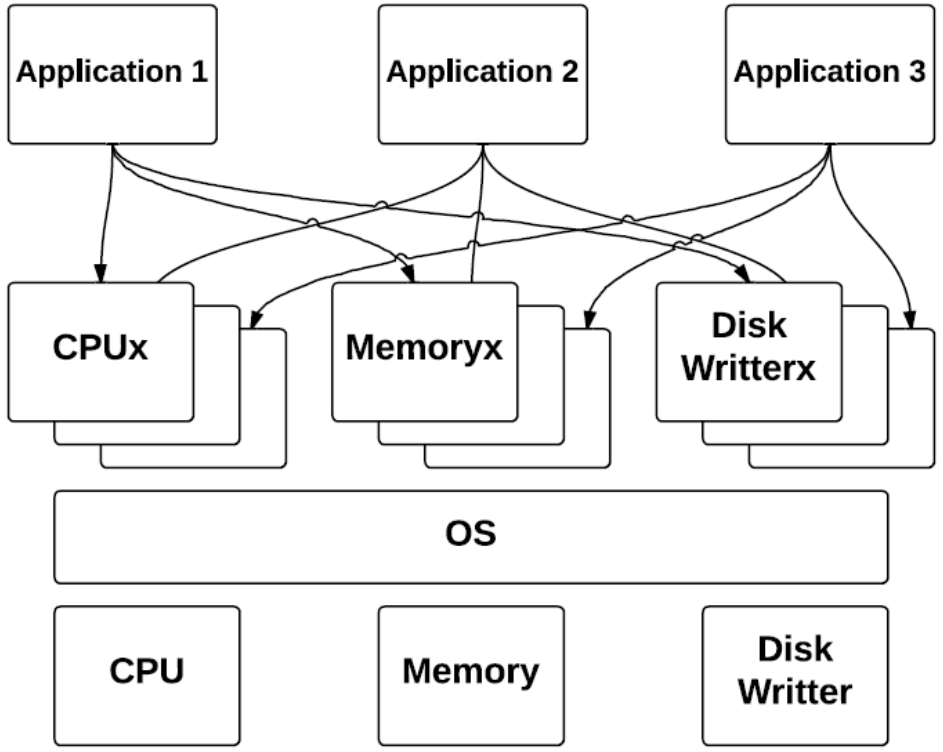
# OS is also responsible for **managing resources**

What are the 3 main computer resources?



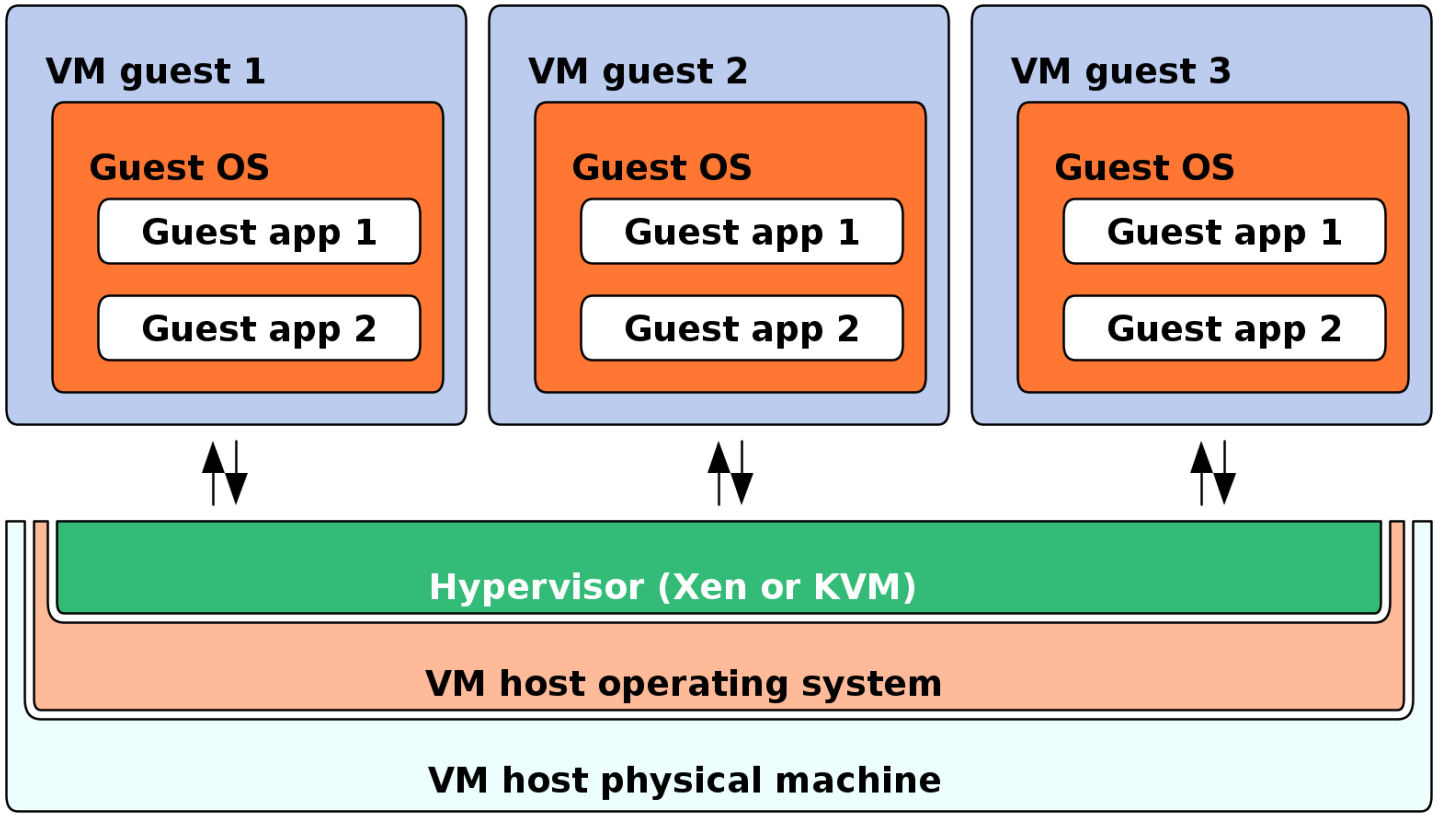

Almost every process you can think of requires access to these resources

Multiplex resources to applications



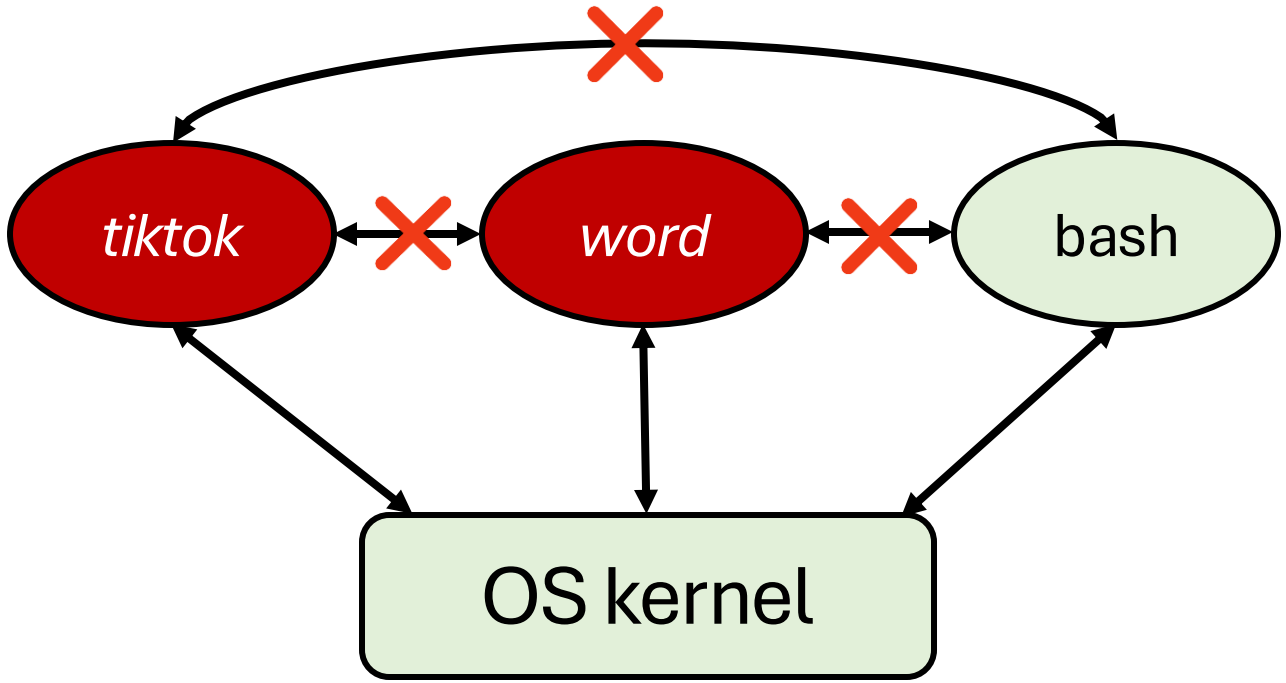
# Resource management can go beyond applications

Create *virtualized* instances of different OSs and run them on a machine

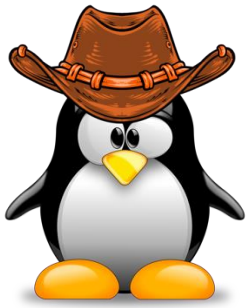


# Finally, the OS **enforces security** and other policies

Cannot *always* trust processes that run on your machine!




A simple example of OS-ensured process isolation for security



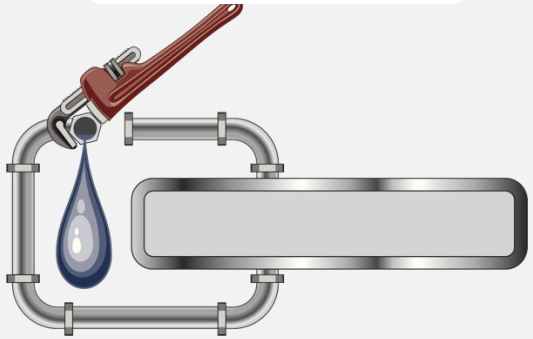
# What is this course about?

# Not about understanding a “specific” OS

- Understanding a specific OS implementation (e.g., Linux) is not useful
  - All OS implementations are very similar to each other.
- There are standards to each OS, but those you’re expected to learn on your own after you complete this course
- You will for all your projects use “Linux” as an example OS

# Understand “principles” behind all three roles of the OS

## Plumbing



Bootloader/BIOS

Privilege management

Device drivers

MMIO/DMA interface

## Managing


Memory management

Process loading

Process scheduling

Virtualization

## Policing


System auditing

Enclave computation

Defences, e.g., KASLR

Process isolation

**Why are aspects are implemented in a certain way?**

# Detailed **(*tentative)** breakdown of topics in this class

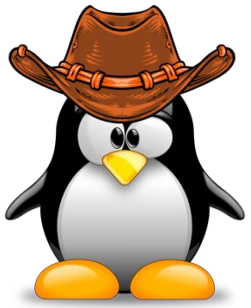
**Pre-midterm**

| | |
|---|---|
| Topic #1: Introduction, OS design, and boot | (~3 lectures) |
| Topic #2: Memory management | (~5 lectures) |
| Topic #3: Processes and threads | (~5 lectures) |

**Post-midterm**

| | |
|---|---|
| Topic #4: Block and filesystem abstractions | (~4 lectures) |
| Topic #5: Device drivers | (~3 lectures) |
| Topic #6: Storage (e.g., SSDs) | (~2 lectures) |
| Topic #7: Virtualization | (~4 lectures) |

I know everything and I’m just
here to fill course requirements!



# Why should I care about learning OS?

# OS concepts are relevant in *every* CS field today

**Machine learning/AI**

- Optimizing training/inference time → Smart memory management (by the OS) is critical for very large models
- Running models in low resource settings → Transparent memory compression mechanisms supported by the OS

**Cybersecurity**

- Preventing malicious USBs from stealing data → Hardware device isolation enabled by the OS

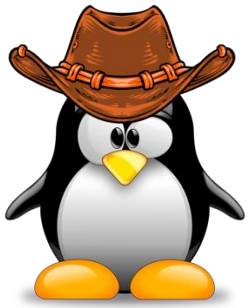
OS concepts are important to *improve (not just use)* every large system today!

# And the OS is a “living” thing

- 5+ million lines of code being tried to incorporate in the Linux codebase every year!
  - Fun fact: Only ~1 million is incorporated due to code issues with rest

- While we might not design new OSs *(for the most part)*, we design many new components all the time. These components:
  - Provide an interface with new hardware features
  - Provide new abstractions to improve security or performance of existing programs (e.g., web browsers)
  - Help increase system stability and debugging
  - … many more!

Believe me, I’m completely objective!



# How will I get an A+ in this class?

# Come to class on your *assigned* days!

-  Mondays and Wednesdays (1.30 – 2.45p)
-  TEMPE – ARM101 / Zoom

**Please note:**

- In-person attendance (in your session) is highly-encouraged
- Zoom and recordings will be available for all lectures

# Come to recitations and office hours!

We will have a total of 9 hours of office hours each week. Use them!

- **Instructor (every Friday from 11a – noon)**
- **Graduate Teaching Assistants (2-hour office hours each)**
  - Vikram Ramaswamy (vramasw3@asu.edu)
  - Qishen Li (qishenli@asu.edu)
- **Undergraduate Teaching Assistants (2 recitation sections each)**
  - Gia Phi
  - Abhirup Vijay Gunakar

* Exact G/UGTA office hours weekly times will be posted on Canvas

** No office hours or recitations in the first week (and later during breaks)

# Course grade decision criteria

- Taken from the course syllabus **(*subject to change)**

This course will have a grade (A+, A, A-, B+, B, B-, C+, C, C-, D, E), assigned through an *absolute* grading scheme. A tentative version of that scheme is provided below.

| Score (%) | Grade | Score (%) | Grade |
|---|---|---|---|
| ≥95% | A+ | 90 - <95% | A |
| 85 - <90% | A- | 80 - <85% | B+ |
| 75 - <80% | B | 70 - <75% | B- |
| 65 - <70% | C+ | 60 - <65% | C |
| 55 - <60% | C- | 50 - <55% | D |
| <50% | E | | |

# Course grading breakdown by components

**Three (3) components in this course**

1. Course projects (47.5%)
2. Midterm and final exams (47.5%)
3. Class participation (5%)

➢In addition to the above, we ****may*** provide you bonus points (e.g., 4– 8% to recover lost points)


# Midterm and final examinations

Consist of MCQs, true/false, and short answers

~80 – 90 minutes of exam time

**Contesting** your grade:

- 2 weeks from midterms to contest in office hours
- Less time for finals, but we can always fix your final grade (even next semester) if an error is spotted

# Course projects

- **Most Valuable Player** of this course's grading
  - This will very likely decide your grade!

- 5 **individual** projects in the course
  - ➢Project 1 ➔ 6%
  - ➢Project 2 ➔ 8.5%
  - ➢Projects 3/4/5 ➔ 11% each

➢The exact deadlines and dates will be announced shortly

# Course projects (tentative overview)

| 1. VM, kernels, and system calls | 2/3. Process/thread scheduling | 4. Memory management | 5. Block abstraction for USB |
|---|---|---|---|

What is expected?

1. Understanding different OS principles
2. Writing C code in the kernel

➢You may be provided sample code(s)

We will try to go over the project details in class whenever they are released

# Course projects (details)

- Projects will be challenging, and they will take lots of time (e.g., 15+ hours in many cases)

- Start early and get feedback during office hours!

- **Do not be afraid to ask questions but** always ask clarifying questions. Do not ask:
  - “Can you help me write this code?”


# Academic integrity

- Please do not plagiarize!
  - Ask for help if you need it; **we are here to help!**
- **Strict** on plagiarism cases (e.g., run our code checker)
  - Automatic fail grade depending on severity
  - 0% on assignments/exams
- Every case will be reported to the ASU academic integrity office (AIO)


# Academic integrity (details on GenAI)

- You are not allowed to copy code from Generative AI (e.g., ChatGPT)
  - ➢ Feel free to understand and then **write your own code!**

- ChatGPT is quite easy to flag:
  - ➢ Your code will be very similar to a different student in class
  - ➢ There are online tools to check as well

➢ **Even if you claim to understand the code you copied, SCAI still considers it plagiarism**


# Academic integrity (fun statistics!)

- Recently taught CSE 536 (Advanced Operating Systems) in Fall 2024. **What percentage do you think I reported?**

- Out of 56 total students, we caught and reported 21 students **(i.e., 37.5% of the class)**
  - ➢15 of these students were reported twice!

- **Not bluffing** when I say we will be tough on academic integrity violations ;)

# Class participation and pop quizzes

- Coming to class will help understand many new concepts

- Feel free to ask **questions** during the lecture
  - Do not ask for lab assignment clarifications
  - Ask those at the end or during office hours
  - Do not be disruptive!

- I will hold 3 – 5 pop quizzes **in-class** over the course of the semester, which will count for 5% of the grade

# Join our **slack** channel for discussion

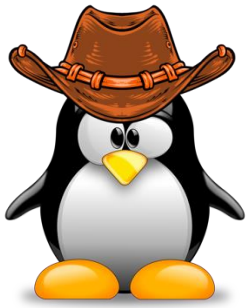
- Post questions, memes, just **not your freaking solutions!**
  - **Posting your own code on Slack will count as plagiarism**
- Goes without saying: **be nice, be respectful**
  - Harassment of *any kind* will get you banned from the channel and reported to the Office of Student Affairs
  - Please check course/ASU policies in the syllabus
- We will add you to the slack channel

# Finally, reaching out to us outside of class/office hours

- **Instructor is not the first point-of-contact for most things**
  - I will likely not respond very rapidly in most cases

- *Grading matters:* send email to TA and your grader
  - TA(s) ➔
    Vikram Ramaswamy (vramasw3@asu.edu)
    Qishen Li (qishenli@asu.edu)
  - Graders ➔ Publish emails on Canvas

- Other matters like missing classes, etc: send an email to Student Services (don't send me private info!)
  - They can only verify your emergency, and they will let us know



**Questions?** Otherwise, see you in the next class!