Welcome!
CS 355: Operating Systems

Instructor: Aline Normoyle

TA: Rebecca Lassman

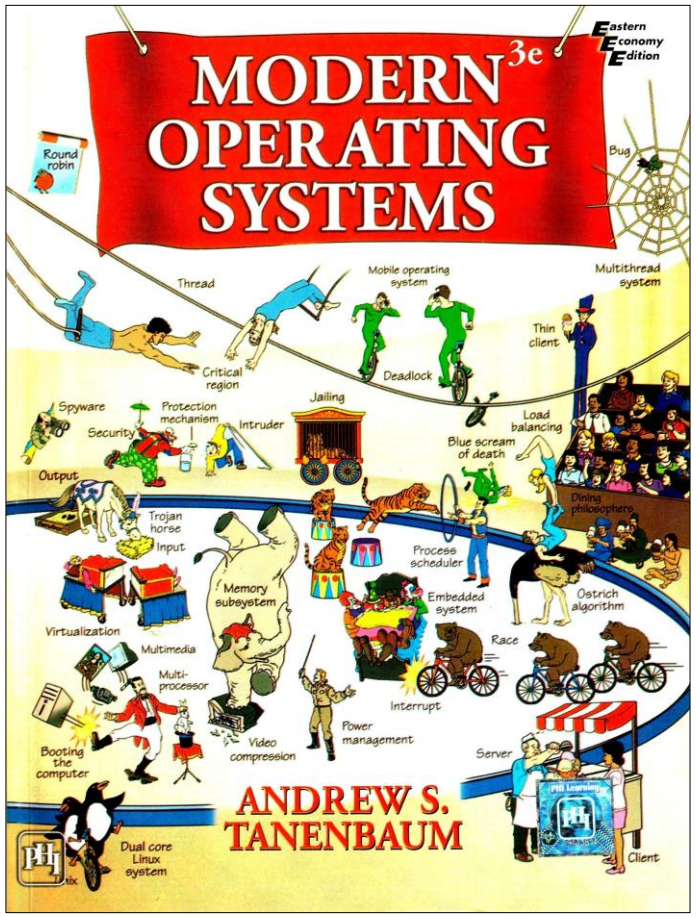
Textbook: Modern Operating Systems

Slack: Announcements, links, etc

Website: Policies, syllabus, etc

Github: Code repository

Lab: Park 231



Agenda: Week 01

- Administrivia
- OS Overview
- History (Chapter 1)
- Review
  - Bash
  - Files/directories
  - C++/C
  - Makefiles and building

Course Resources

Webpage

https://brynmawr-cs355-s26.github.io/website/

Github

https://github.com/BrynMawr-CS355-S26/

Slack

https://BrynMawr-CS355-S26.slack.com

Learning Goals

- Understanding fundamental OS components and structure.
- Understanding OS design challenges and algorithms
- Knowledge of the services provided by the OS and details of major OS concepts.
- Experience in developing low-level operating system code.
- Understanding the performance and design trade-offs in complex software systems
- Experience in developing benchmarks and test suites to evaluate the performance and robustness of system

Lecture/Lab Format

- Lectures
  - Slides/activities
  - 2 Midterm Exams (90 minutes, closed book, 1 cheat sheet)
- Labs
  - Coding assignments in pairs
- Assignments/Projects
  - Weekly or Bi-Weekly (due on Tuesdays)
- Budget 10-15 hrs/week for this class!

Policies

- Accommodations
  - Need at least 2 weeks prior notice for extensions on quizzes/exams
- Covid policy: mask friendly
- Late policy depends on assignment
  - Short assignments: up to 1 day late
  - Projects: no late submissions, resubmission opportunities

Why study operating systems?

"A systems programmer will know what to do when society breaks down, because the systems programmer already lives in a world without law."

James Mickens

"When you create an operating system, you're creating the world in which all programs running the computer live. Basically, you're making up the rules of what's acceptable and can be done and what can't be done."

Linus Torvalds

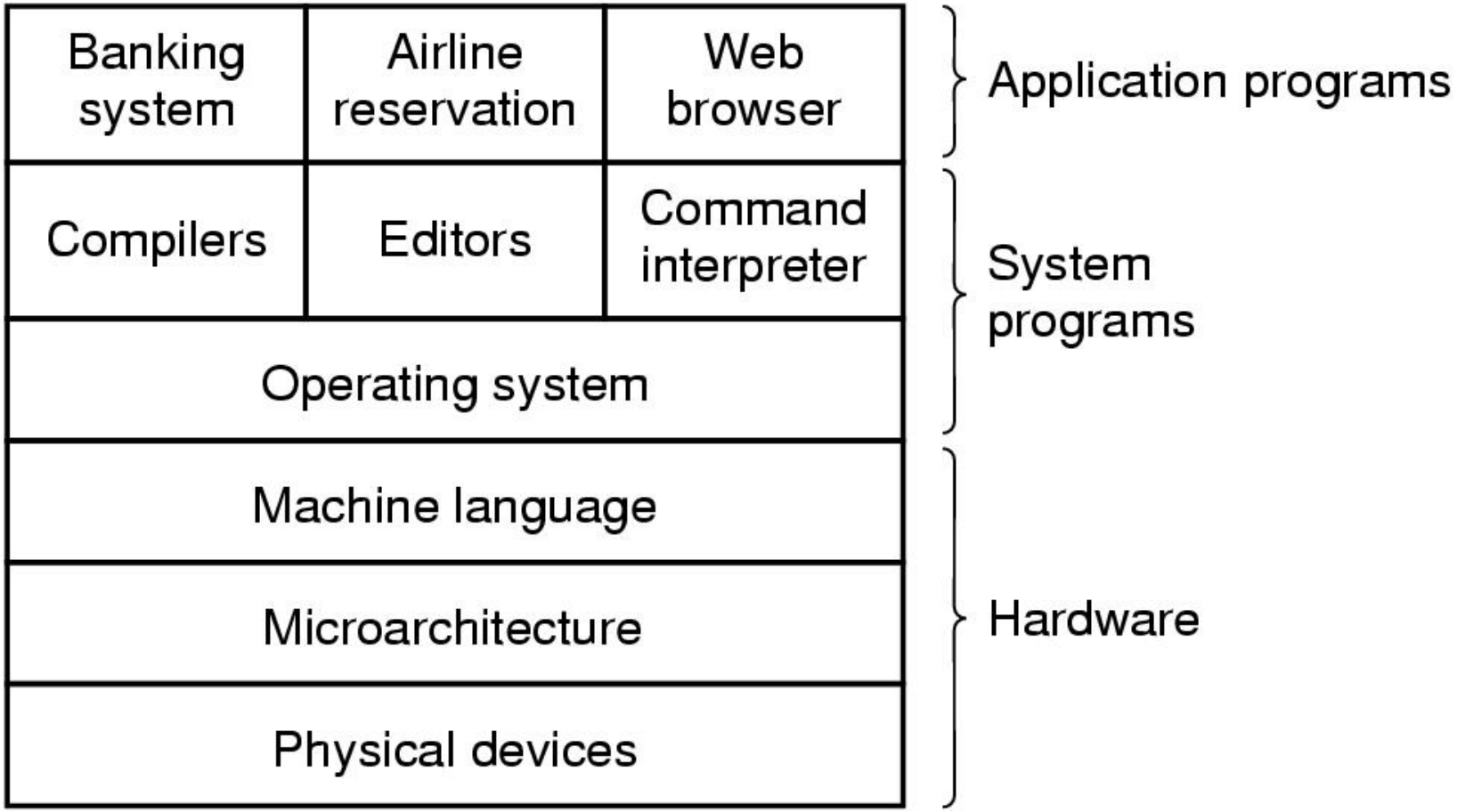
Discuss: What does an operating system do?

Example: Suppose we want to print "Hello World" on a printer?

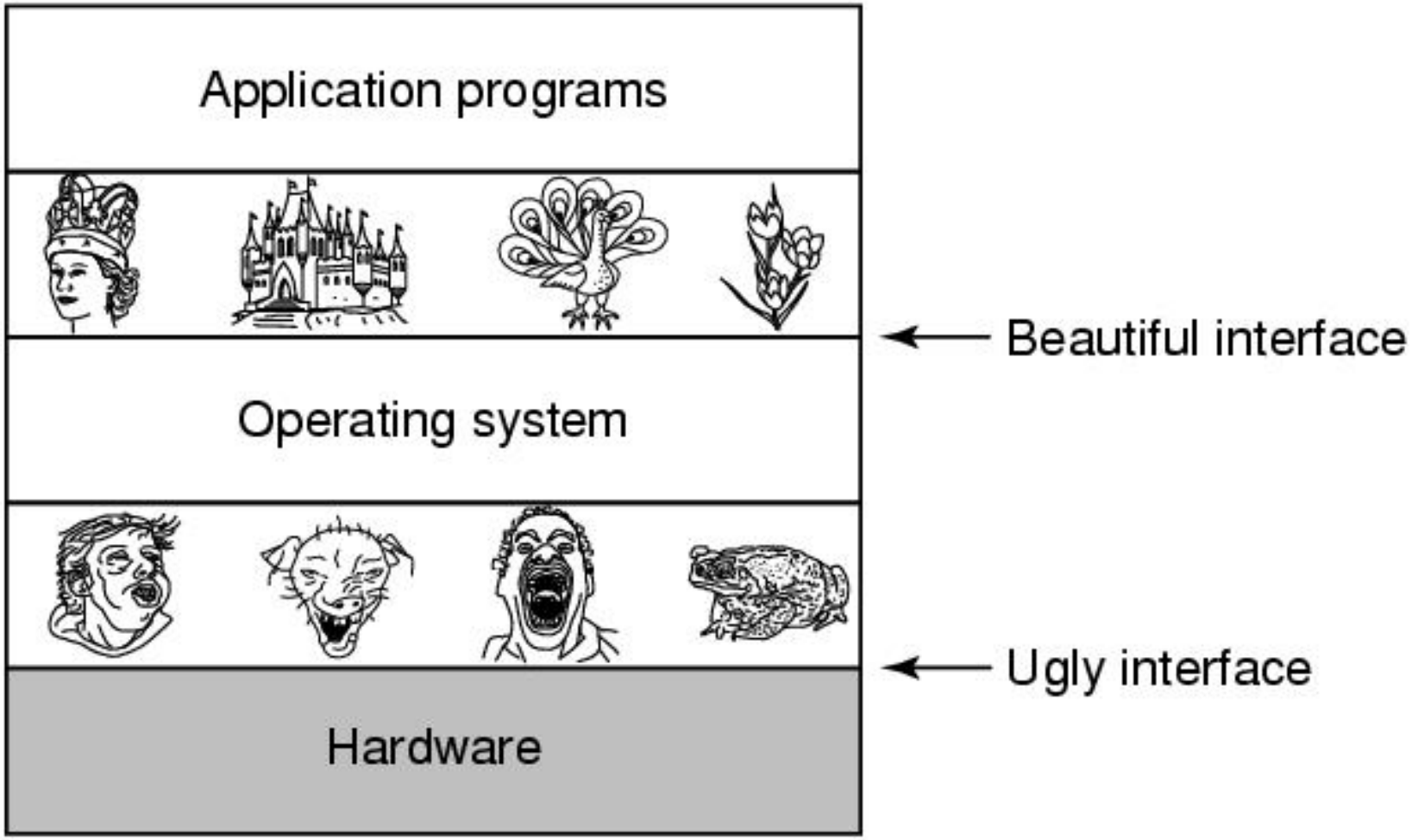
TY OS!

- Get printer manual
  - find out how to send messages to it
- Write program
  - put string "Hello World!" in memory buffer
  - do what printer requires to send buffer to it
  - go into waiting loop
- Find a Computer
- Translate your program into machine code
- Figure out a way to load the program into memory
- Start the program (somehow)

OS: Interface between hw and apps



OS: User-friendly control over hw



The OS has two primary jobs

Abstractions for working with hardware

- Hides the messy details (e.g. writing to a disk)
- Presents user with a virtual machine, easier to use

Resource management

- sharing resources among multiple programs/users
- **Time multiplexing**: “taking turns” (e.g. printer)
- **Space multiplexing**: “sharing part of a resource” (e.g. each process uses part of memory)

Exercise: Match abstractions to hw

Hardware

- Disks
- Running program
- Network
- Monitor
- Keyboard
- Mouse

OS abstraction

- Connection
- Locator
- Windows and GUI
- Input
- Files
- Processes

Challenges in OS

- Performance is critical
  - how to reduce the memory and time overhead due to OS
- Synchronization and deadlocks due to shared resources
- Scheduling of multiple programs
  - Fairness, response time, real-time applications

Challenges in OS

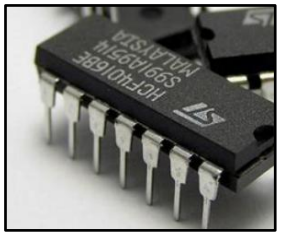
- Memory management
  - Virtual memory, paging, segmentation
- Resource management
- Security and protection
  - Authorization, authentication and viruses
- Interrupt management and error handling
- Marketability and backwards compatibility

Brief History of Operating Systems

First generation 1945 - 1955
- vacuum tubes, plug boards

Second generation 1955 - 1965
- transistors, batch systems

Third generation 1965 – 1980
- Integrated circuits (ICs)
- Multiprogramming



Fourth generation 1980 – present
- personal computers

1940's - First Computers

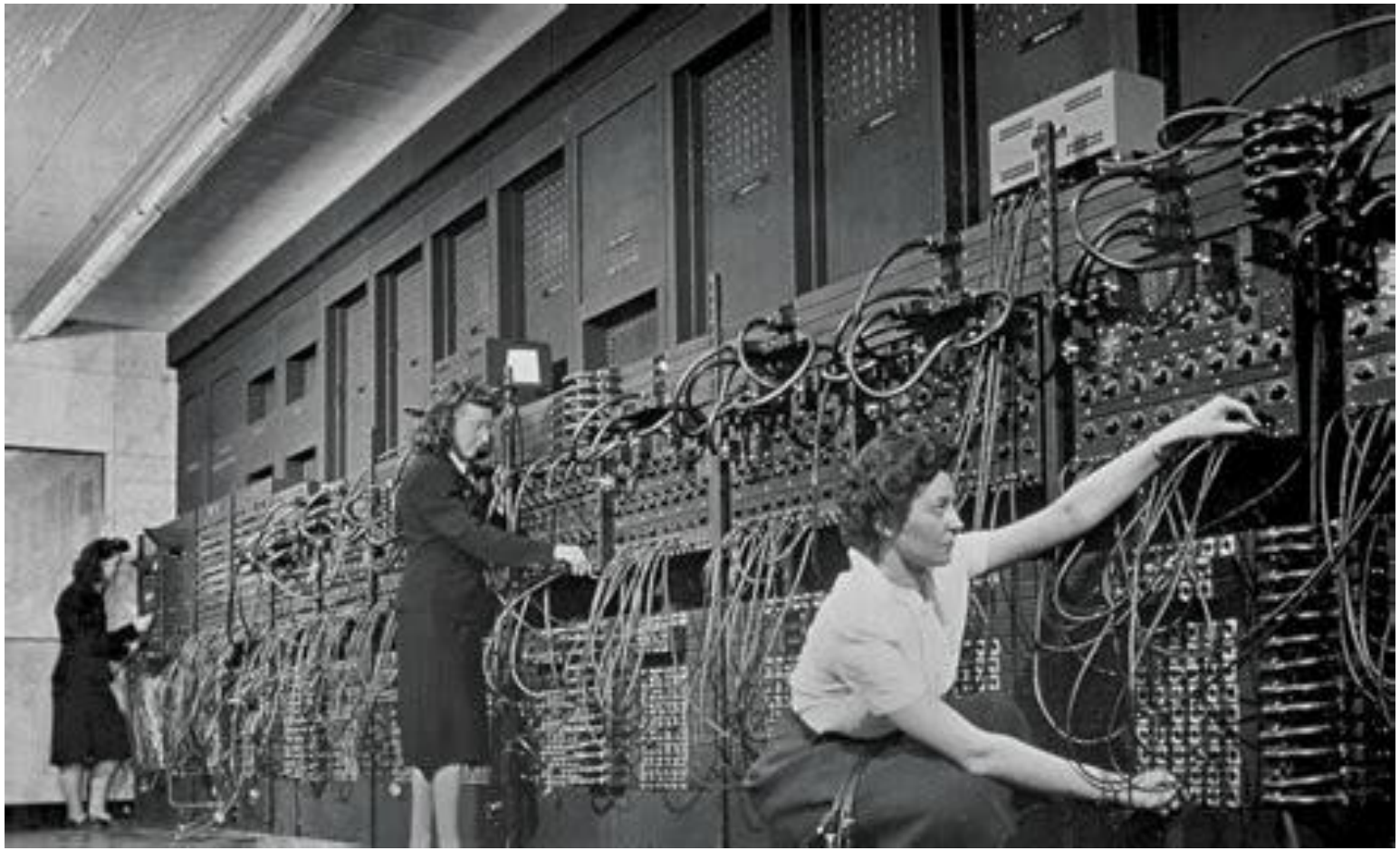
One user/programmer at a time.

- Program loaded manually using console switches.
- Debugging using console lights.

Expensive machine idle most of time, because people are slow.

Each program must include code to operate peripherals - error prone, device dependencies.

ENIAC, 1945



Used to compute ballistic trajectories by the US in WWII

1950's – Batch Processing

User/programmer submits a deck of cards that describes a job to be executed.

Jobs submitted by various users are sequenced automatically by a resident monitor.

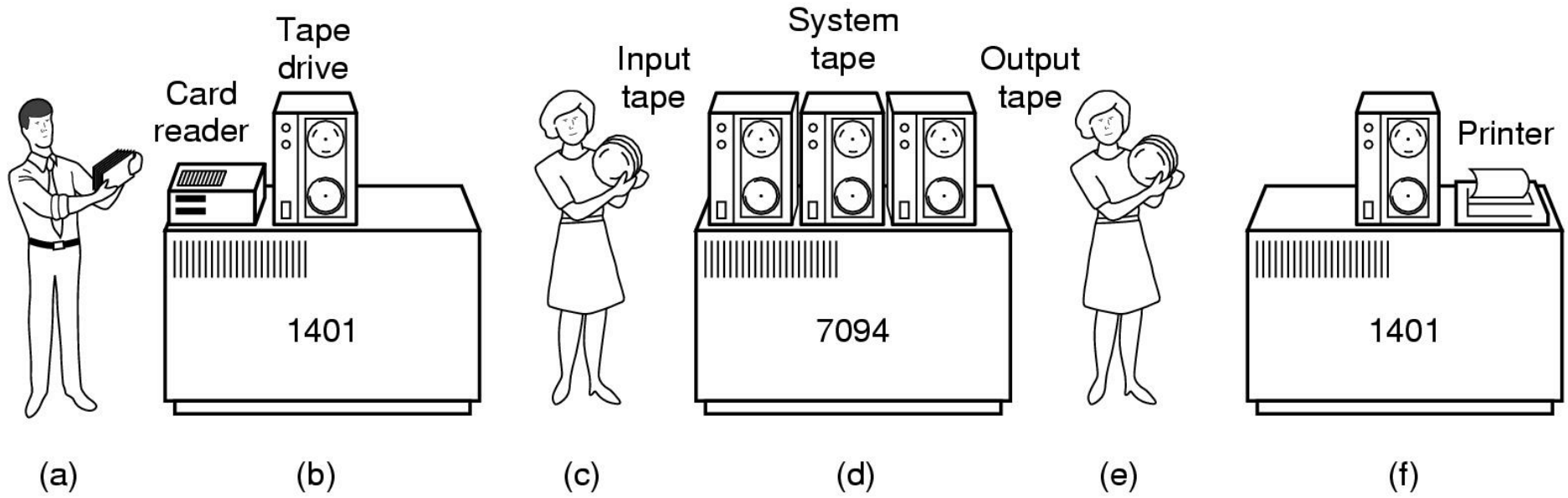
Tape drives available for batching of input and spooling of output.

Computer system is kept busier but no longer interactive. Turnaround times are longer.

IBM 7094

Photo: Columbia University Archive.

Typical Batch System



Early batch system

- bring cards to 1401
- read cards to tape
- put tape on 7094 which does computing
- put tape on 1401 which prints output

1960's –Multiprogramming/Timesharing

Multiprogramming: the ability to run more than one program at once

Timesharing: multiple users can share the same computer system

Simultaneous I/O and CPU processing possible.

CPU is multiplexed (shared) among a number of jobs. Usage is again interactive.

Hardware and OS significantly more complex

- New OS issues from timesharing – CPU scheduling, deadlock, protection, memory management, etc

IBM 360

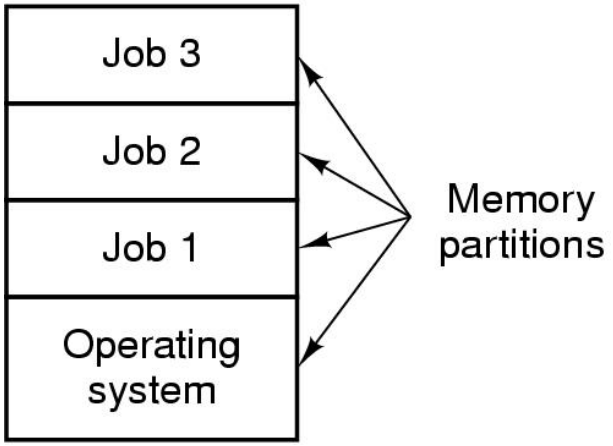

System 360
READ ONLY STORAGE
COUNT REGISTER
CHANNEL NUMBER ONE
DATA REGISTER
KEY
COMMAND
FLAGS
TAGS
CHECKS

Multiprogramming System

The user(s) start any number of jobs and the OS handles the execution to maximize the usage of the hardware

From the user perspective, the OS appears to run all jobs at once.

From the job perspective, the OS makes it appear like they are the only job running.



A multiprogramming system with 3 jobs in memory

1970's - Minicomputers & Microprocessors (UNIX, DOS)

Personal computers, rather than a single mainframe.

Early minicomputers and microprocessors were small, so there was some regression to earlier OS ideas.

This trend changing rapidly because of powerful new microprocessors.

New OS Issues: User interfaces (GUI)

MULTICS

MULTICS and related Honeywell 6000 Series Front Panels
(mercury.lcs.mit.edu)

1980's - Networking

Powerful workstations (e.g., PDP, VAX, Sun work stations, etc.)

Local area networks (e.g., Ethernet, Token ring) and long-distance network (Arpanet)

Decentralized computing requires more communication (e.g., resource sharing)

New OS issues – network communication protocols, data encryption, security, reliability, consistency of distributed data

Personal Computers

Apple IIe

IBM

1990's and Beyond

Parallel Computing (tera-flops)

Powerful PCs, Multimedia computers, embedded computers: medical devices, cars, smartphones

High-speed, long-distance communication links to send large amounts of data, including graphical, audio and video, World Wide Web

New OS issues – Large heterogeneous systems, power, security, etc.

OS have evolved along with hw and business

1950-1970s: Hardware manufacturers such as IBM gave software away for free – necessary for their products to have value

→ All software was open-source

1970s – Now: Rise of proprietary software accompanied with expanded copyright laws and lobbying by entrepreneurs such as Bill Gates.

From Future histories, *Collaborative Work is Liberating and Effective*, Lizzie O'Shea

Closed or open source?

“As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if people who worked on it get paid? Is that fair? ... More directly, the thing you do is theft.” – Bill Gates, 1976

Bill Gates

“Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. ” – Richard Stallman, GNU (GNU Not UNIX), 1983

Richard Stallman

Rise of Linux

1979: UNIX changed its licensing to prohibit users from reading or changing the code (AT&T)

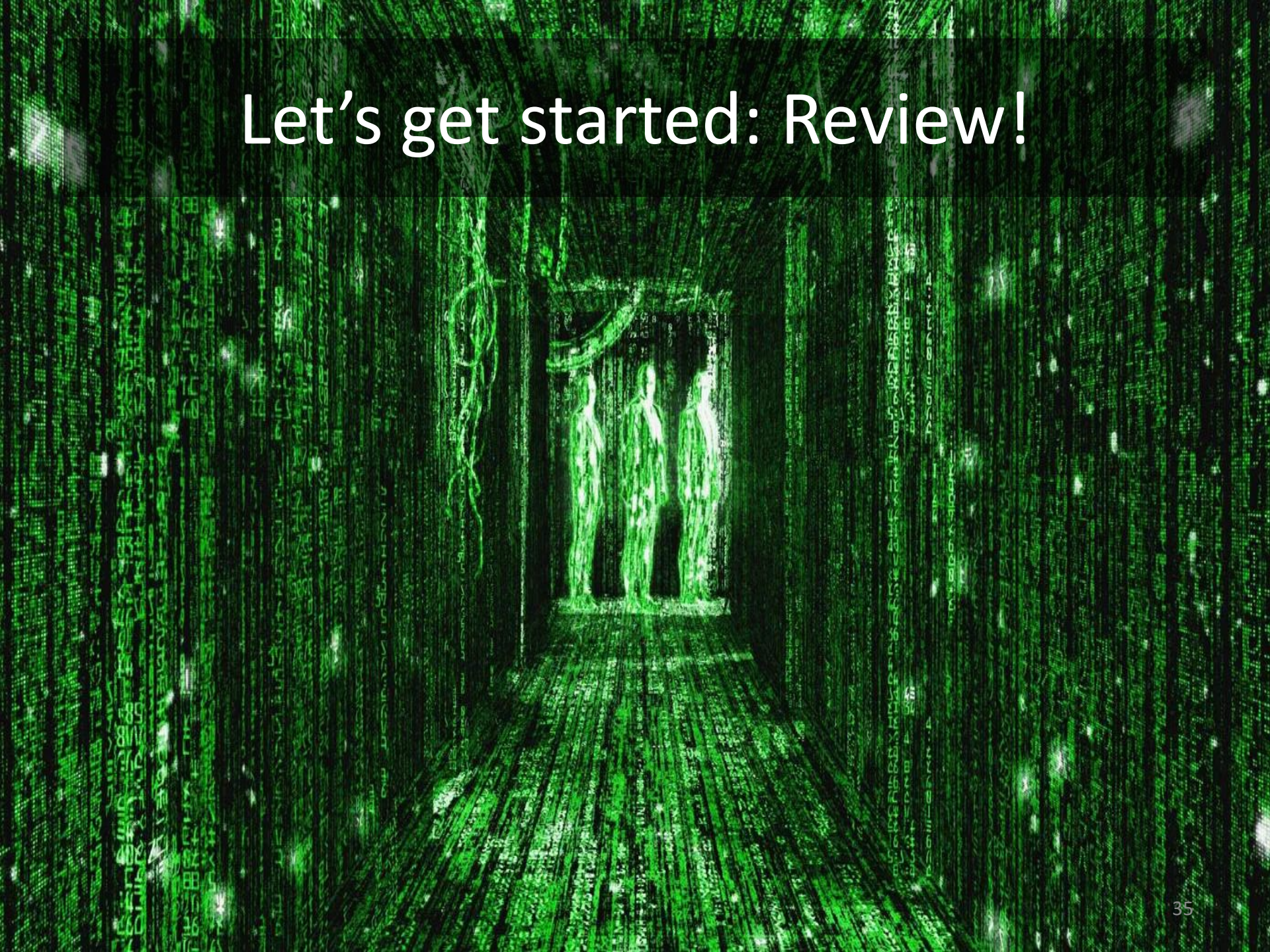
→Andrew Tannenbaum created MINIX as a small OS for his students to learn from

→Linus Torvalds creates Linux under the GPL

> "Everybody puts in effort into making Linux better, and everybody gets everybody else's effort back. "

Linus Torvalds

Let’s get started: Review!



Development Environment

- Operating System: Linux
  - Command Line Interfaces (CLI)
  - Bash shell
  - System libraries
- Programming: C/C++
- Editor: nano, vim, emacs, VS Code

Review: Bash

- What is bash?

- What are some bash commands?

Review: Files and Directories

- What is a file? A directory?
- What are some common file types?
- What is the difference between text and binary files?
- How do we distinguish between file types?

Review: Files and directories

- In a file system, what is a path?
- What is the current working directory?
- What is the difference between an absolute and relative path?
- What do the following symbols mean when they are in a path?
  - /
  - .
  - ..
  - ~

Example

```
root
-- A
---- hello.txt
-- B
```

What is the absolute path of hello.txt?

What is the relative path of `hello.txt` from

- the root directory?
- the A directory?
- the B directory?

Draw the directory hierarchy after the following commands

```
$ pwd
/home/alinen
$ mkdir A
$ cd A
$ mkdir Z
$ touch talk.c
$ cd ..
$ touch listen.c
$ cd
$ touch sing.c
```

Wildcards

Paths and filenames support wildcards with *

```
alinen@sutekh:~/cs355/os-devel/lectures/week01$ ls
Makefile  error.txt  input.txt  log.txt  stdio0.c  stdio1.c  stdio2.cpp
alinen@sutekh:~/cs355/os-devel/lectures/week01$ ls *.c
stdio0.c  stdio1.c
```

```
alinen@sutekh:~/cs355/cpp-lecture$ ls
CMakeLists.txt  LICENSE  README.md  basic  boxes  examples  files
alinen@sutekh:~/cs355/cpp-lecture$ ls */Makefile
basic/Makefile  boxes/Makefile
```

Exercise: Wildcards

```
alinen@sutekh:~/cs355/os-devel/lectures/week01$ ls
Makefile  error.txt  input.txt  log.txt  stdio0.c  stdio1.c  stdio2.cpp
```

- How to print all C++ source files?
- How to print all txt files?
- How to print all files that start with s?
- How to print all c files that start with s?