CS 475
Operating Systems

Department of Mathematics
and Computer Science

Lecture 1
History and Trends in
Computer Systems

Today's Topics

- Course Overview
- Technological Trends
- What Is an OS, Anyway?
- A Brief History of Computer and Operating Systems
- Conclusion

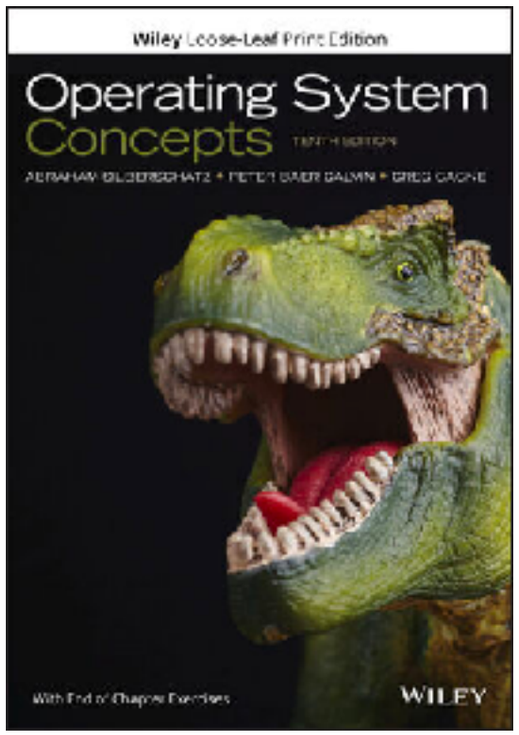
About David

- David Chiu (pronounced "chew")
  - "Professor Chiu", "Dr. Chiu", or just "David"

- Background
  - PhD, Computer Science & Engineering, Ohio State University, 2005-2010.
    - (Majored in systems, high performance computing, and databases)
- Assistant Professor at Washington State University, 2010-2014.
- Professor at Puget Sound since 2014.

Course Requirements

- Required textbooks
  - Dive into Systems (free online)
    - https://diveintosystems.org/book
  - Silberschatz, Galvin, and Gagne. Operating System Concepts. 8th ed. or higher.



How to Reach Me

- How to reach me
  - Email: dchiu@pugetsound.edu
  - Office: TH 303 (near south elevator)

- Drop in when door is open/cracked
  - Or, schedule an appointment here:

Course Webpages

- Two important webpages to bookmark.
  - Course Calendar - https://davidtchiu.github.io/teaching/cs475/
  - Canvas - https://canvas.pugetsound.edu
    - Assignment submission, code examples, study guides found here
  - [Let's take a quick tour]

- Scan below to take you to the course calendar

Course Policies

- Class Disruption
  - Put phone on silent
  - *No laptops out, unless permitted for note-taking by SAA or permitted on lab days*

- Cheating: Zero-Tolerance
  - I compare current and past assignments
  - OK to brainstorm, and ask for help, but *you must produce your own code*

ChatGPT and Other Generative AI

- **Do** use it to be your personalized tutor, and that is as far as it should go. Use it to explain C code you don't fully grasp.
  - Prompt: *"**Without giving me any code, explain …**"*

- **Do** use it to explain code to you. Is there a piece of code we went over in class that's hard to grasp? Paste it, and have it explain line-by-line as well as holistically.

- **Do** use it to explain errors to you. Paste your code and the errors you get when compiling or running it. Give hints on what might be the issue.

- **Do** use it to explain any OS concepts that you need to know.

ChatGPT and Other Generative AI (2)

- **Don’t turn in** anything that was generated by these tools. Copying-and-pasting AI output is considered plagiarism and will be treated as such.

- **Don’t underestimate** how easy it is for us to detect cases where students are turning in code written by generative AI tools.

- **Don’t forget** that you *still* need to demonstrate proficiency on all your exams to pass the course.

Evaluation

- Breakdown of Evaluation
  - 30% - Homework Assignments
  - 10% - Tech Report
  - 15% - Midterm Exam I
  - 18% - Midterm Exam II
  - 20% - Final Exam (Comprehensive)
  - 7% - Participation
    - Participation points starts at 10.
    - Unexcused absence (-2), late (-1), non-class related activities (-1)

Homework

- You can expect 7-8 homework assignments
  - Programs must compile and run on the remote server
  - $-3^d\,\%$ point late penalty, including weekends
  - All assignments must be submitted using **git/github**

- **Budget more time than usual for coding assignments**
  - Assignments *seem* easy as high level programmers
    - But C can be very finicky and low-level
    - The gcc compiler doesn't do a great job verbalizing runtime errors
  - Carefully doing and learning from the early "labs" is crucial

Exams

- Two **1-hr midterms** and a **2-hr final** will be based on
  - Lectures, notes, homework
  - Weighted heavily on new material
  - **Final exam is comprehensive**

- I don't test anything I did not specifically cover in class
  - But things covered in class, but isn't found in the readings, is fair game
  - So don't miss class!

- Calculator and 1-page notes (front/back) allowed

How to Succeed in This Course

- Dedicate time to learning C
  - C is not intrisically *hard*, but it's nuanced and debugging is frustrating!
    - There's no "stack trace" when a program crashes.
  - Trivial tasks will take you longer to write in C (vs. other languages)
    - Imperative that you *master* strings and pointers early
      - (They *will* drive you nuts)
- Meanwhile...
  - You must also develop a solid grounding with Linux and its command line shell

Today's Topics

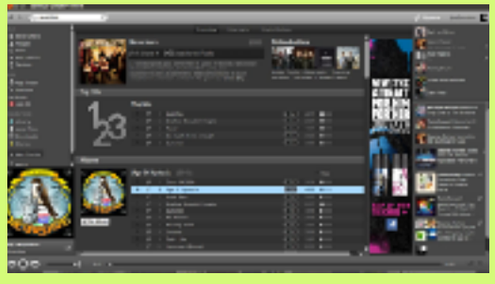
- Course Overview
- What Is an Operating System, Anyway?
- A Brief History of Computer and Operating Systems
- Conclusion

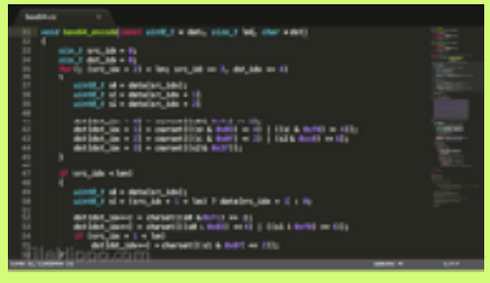
What's Life Like without OS?

- MITS Altair 8800 (1975)
  - World's first affordable PC
    - Intel 8080 (8-bits, 2MHz), 256B RAM, $397
  - No keyboard, no monitor display, no OS
    - Program by toggling binary code, hit run, read results off the LED display.
    - Sparked many startups (Microsoft was born!)
- For the curious. Here's early Bill Gates
  - https://youtu.be/suyiMfzmZKs

OS: Middle Layer between Users and Devices

User Applications





OS Services
(via System API)

Operating System

Kernel

Browser | Shell | GUI / windows | Other auxiliary apps

Scheduler | File System | ...

Memory Management | Process and Thread Management | Device Drivers

Hardware Interface
(called "device drivers")

Hardware

OS: The Illusionist

- Provides illusion (or, virtualization)
  - Programs (processes) think they get unrestricted access to all hardware resources.
- Example:
  - Each program believes it has total control and attention of the CPU.
  - Reality: OS must multiplex multiple programs over one CPU in real time.

Course Topics: "switching", "process/thead management", "memory management", "virtual memory"

Example: Virtualizing the CPU

- Each running program *thinks* it gets to run on its own CPU.

PowerPoint

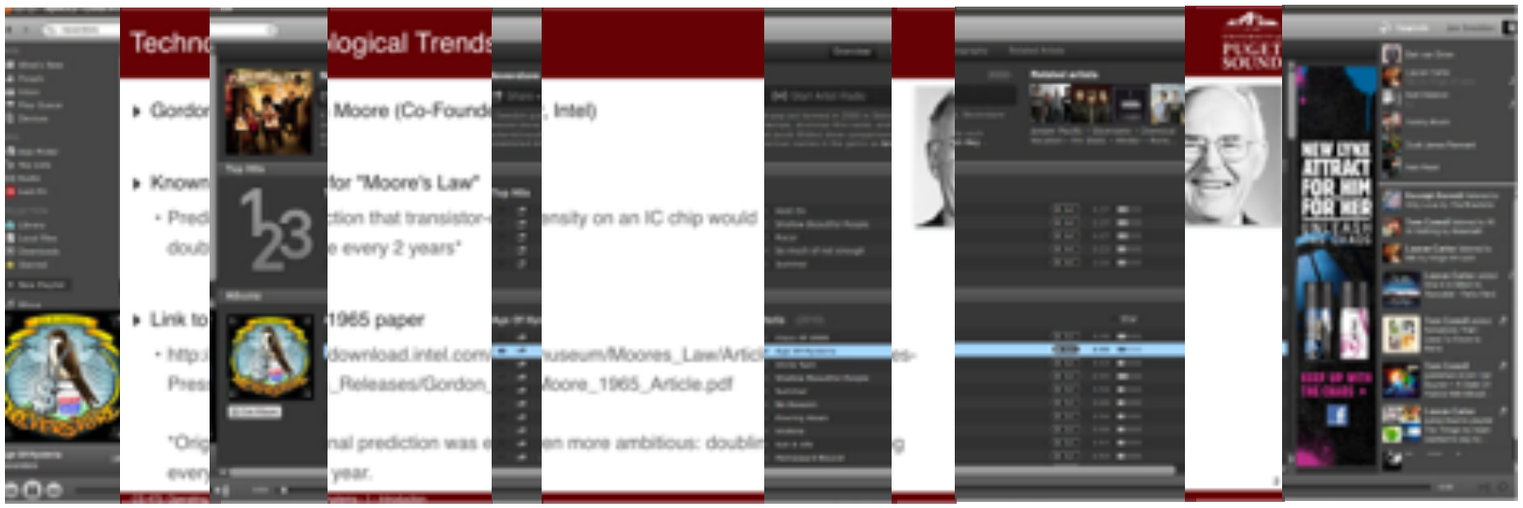
I think I have my own CPU. Don't have to wait for others to finish using it.

Spotify

Me too!

Example: Virtualizing the CPU

- Reality: number of processes > number of CPUs.
  - OS needs to make it appear as if multiple processes are running simultaneously.



Time

Share a single CPU

OS: The Government

- The OS is your machine's government
  - It provides a well-regulated services that users need but are too risky to let users use without oversight. Examples:
    - Create, terminate, suspend a process
    - Open/read/write a file
    - Printing to the terminal or the printer
    - Requesting more memory during runtime

  - Users can request the OS to take action made through *"system calls"*
    - Each OS provides a standard library of system functions

Course Topics: "dual-mode operation", "protection", "virtual addressing"

OS: The Traffic Cop

- The OS is a traffic cop
  - Coordinates many running programs (processes)
  - Ensures who gets to run? When? For how long?
  - Ensures that processes can't monopolize resources or starve for them

Course Topics: "CPU scheduling", "synchronization", "concurrency control", "deadlock detection"

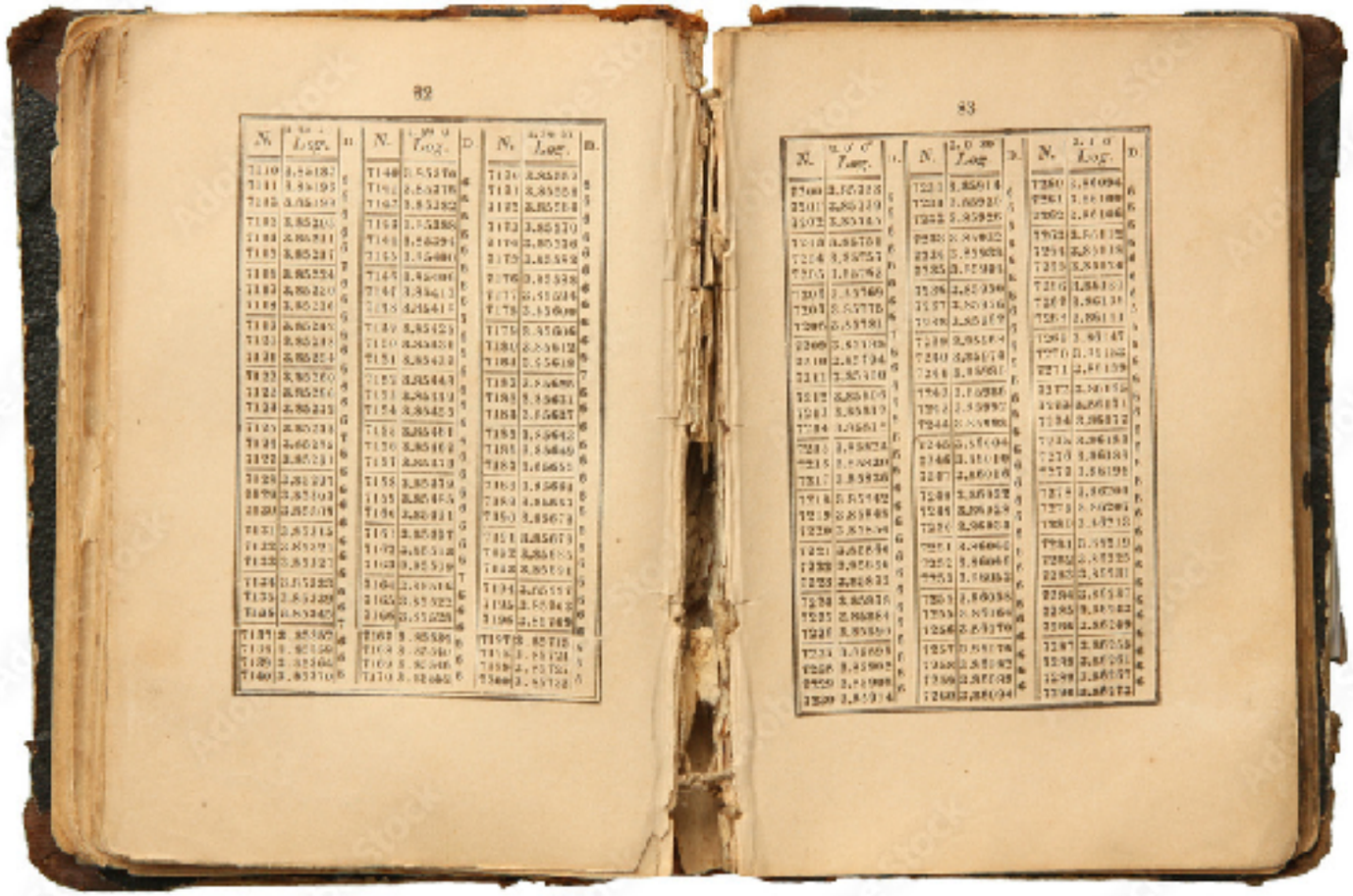
Today's Topics

- Course Overview
- What Is an OS, Anyway?
- A Brief History of Computer and Operating Systems
- Conclusion

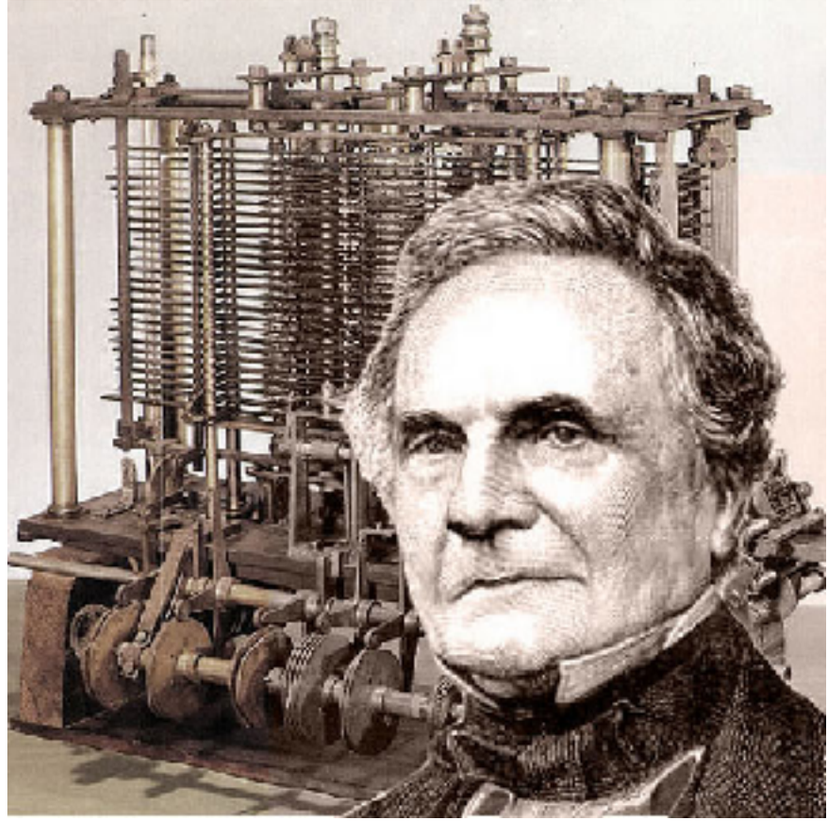
1700s - 1800s: Origins of Computing

- The prevalence of mathematical and data tables
  - Used for engineering, studying astronomy, ...
  - Calculations done by hand. Lacked precision and accuracy

Example: Table of Logarithms

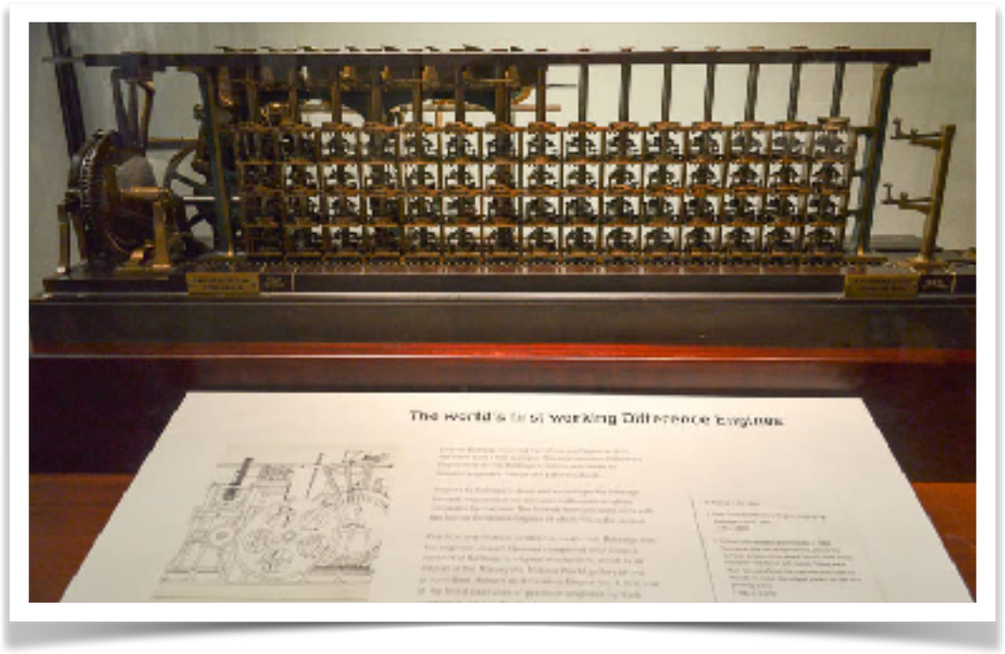


Charles Babbage



Origins: Analytical Engine (1800s)

- Charles Babbage (1791-1871) and Ada Lovelace (1815-1852)
  - Babbage invented the Difference Engine, a mechanical calculator
  - Drew up plans for the Analytical Engine, but was never built
  - Watch: Calculating Ada documentary



Part of the Difference Engine, Science Museum in London

Ada Lovelace:
Published the first program (for the analytical engine)

Babbage's Difference Engine

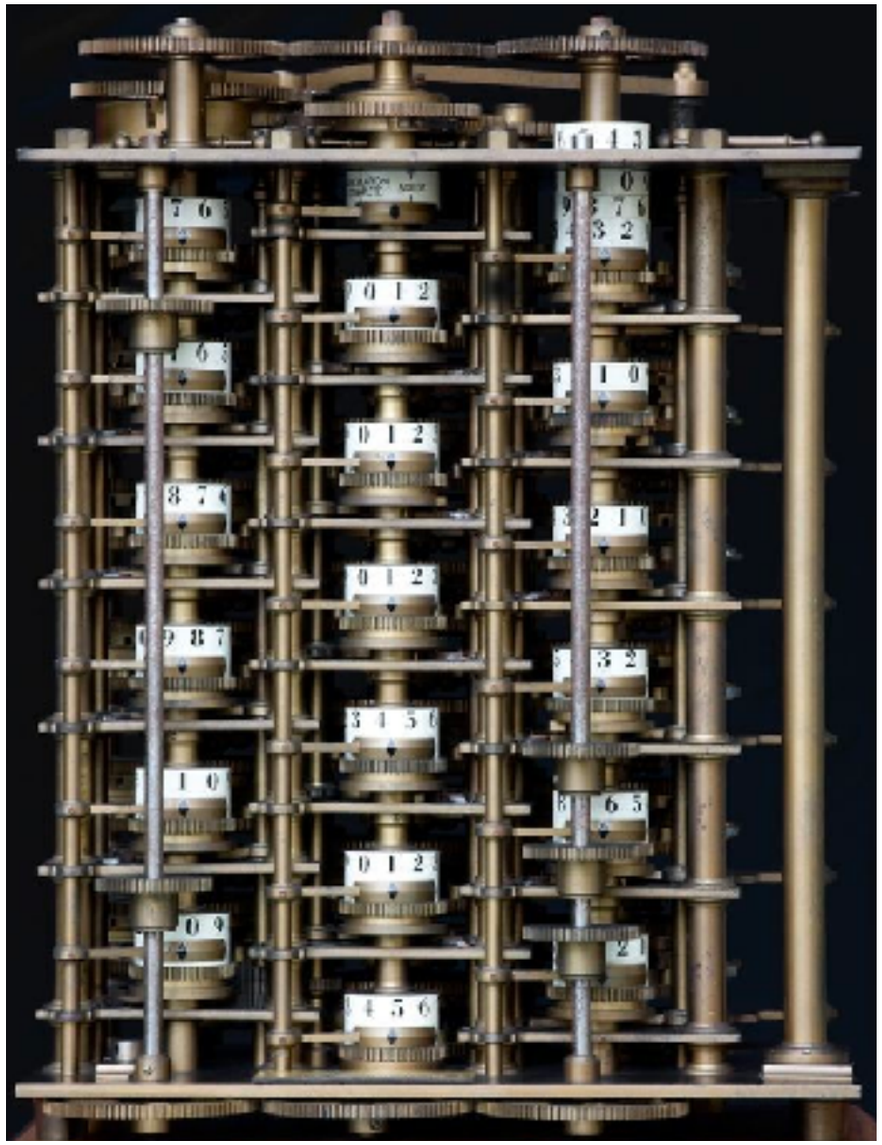
- Print results of polynomials in a table format
  - Problem: labor-intensive, prone to human error
- Key insight:
  - Automate the *method of finite differences*
    - *Decompose task using simple additions*
  - For an degree-n polynomial, input all n differences for first few values of x.
  - *Then* $f(x) = f(x-1) + \sum_{i=1}^{n} \Delta_i$
  - https://bit.ly/3kF8srm
- ***"Registers"*** *held the current iteration's values*

$$f(x) = 2x^2 + 4$$

x	f(x)	Δ_1	Δ_2
1	6		
2	12	6	
3	22	10	4
4	36	14	4
5	54	18	4
6	?	...	...

Babbage's Difference Engine

- A reproduction built for the Computer History Museum (CHM)

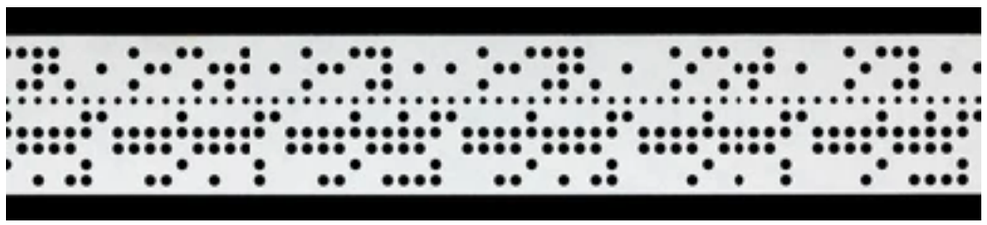


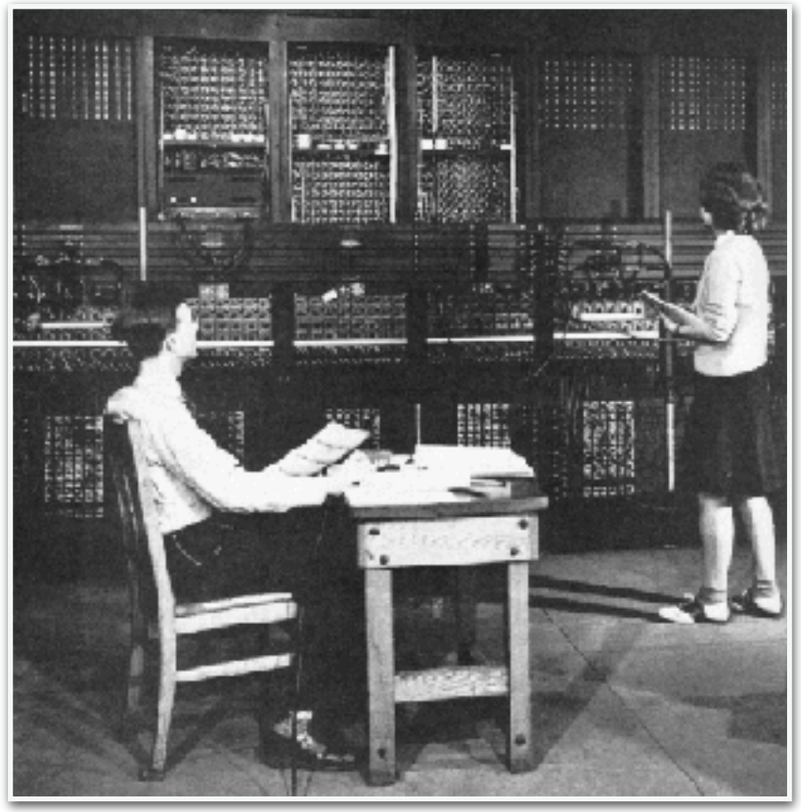
"Registers"

A Replica at the Computer History Museum

1940s: Dawn of Time

- Hardware is *very expensive* and *experimental forms of art*
  - First manufactured by IBM as general-purpose "calculators"
- Computers have lots of **moving parts (slow; unreliable)**
- No stored programs
- Input/output
  - Programs on punched tape
  - Printer
- No OS

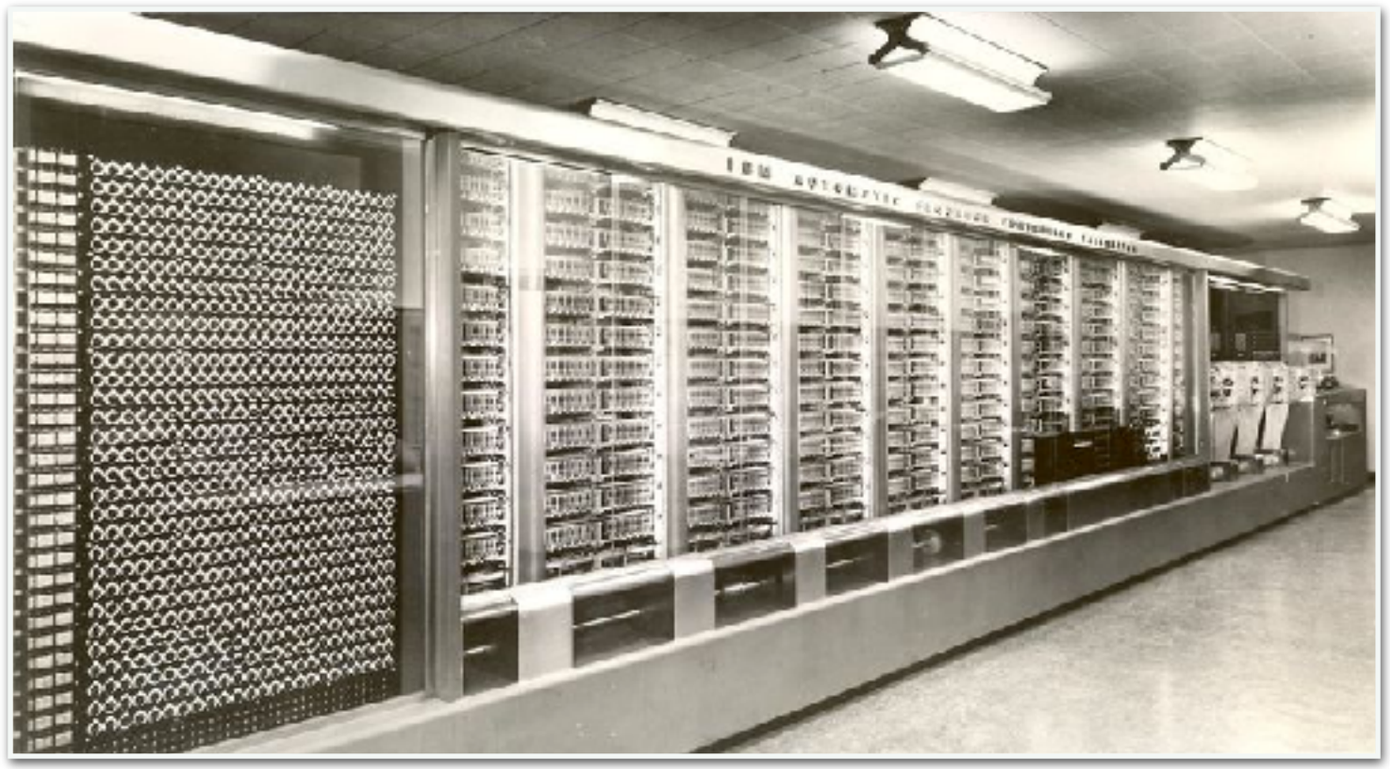




1940s: Harvard IBM Mark I (1944)

- First fully-automatic, *electro-mechanical* general-purpose computer
  - 51ft x 8ft x 8ft, weighed 5 tons
  - 3 adds/subtracts per sec; 1 multiplication in 6 secs
  - Fully programmable!
    - Stopped and waited for "go" signal between operations; made for easy debugging
  - No OS, programs on tape fed by humans

- *Fun fact: "Sounded like a room full of people knitting."*



1940s: Harvard IBM Mark I (1944)

- Mark I programming team led by **Dr. Grace Hopper**
  - Professor of Mathematics @ Vassar College
  - Then, as a US Naval Officer, became the first principle programmer of the Harvard Mark I

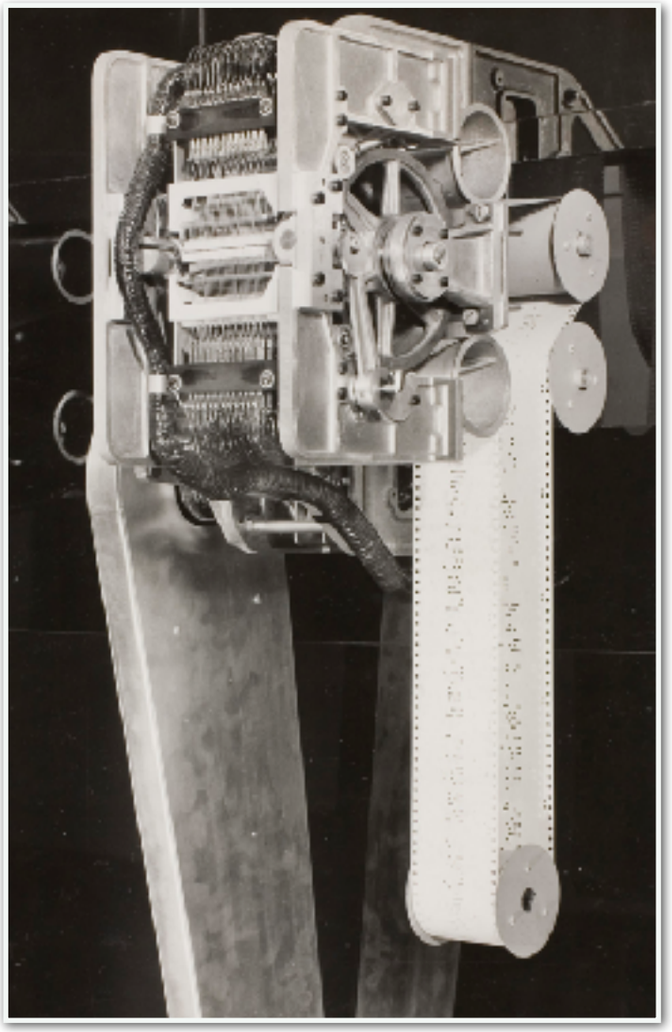
On the Harvard Mark I:

"[it was] the first machine that was built that was supposed to assist the power of man's brain instead of the strength of his arms."

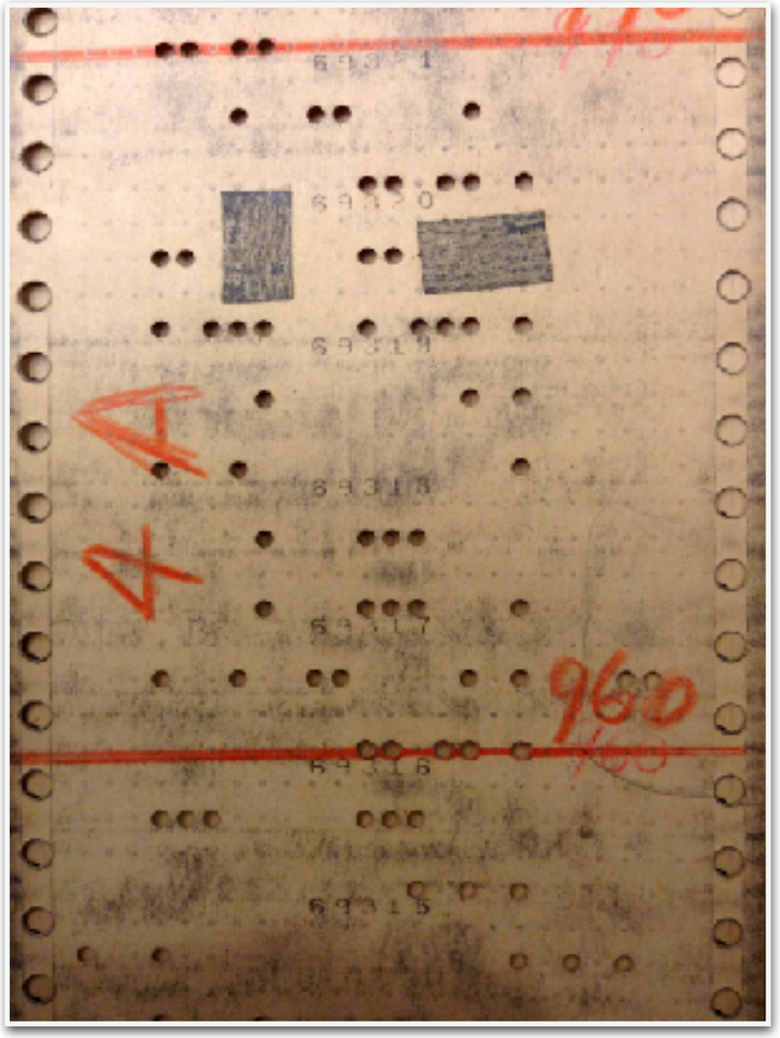
Grace Murray Hopper

Aside: Etymology of Common Terms

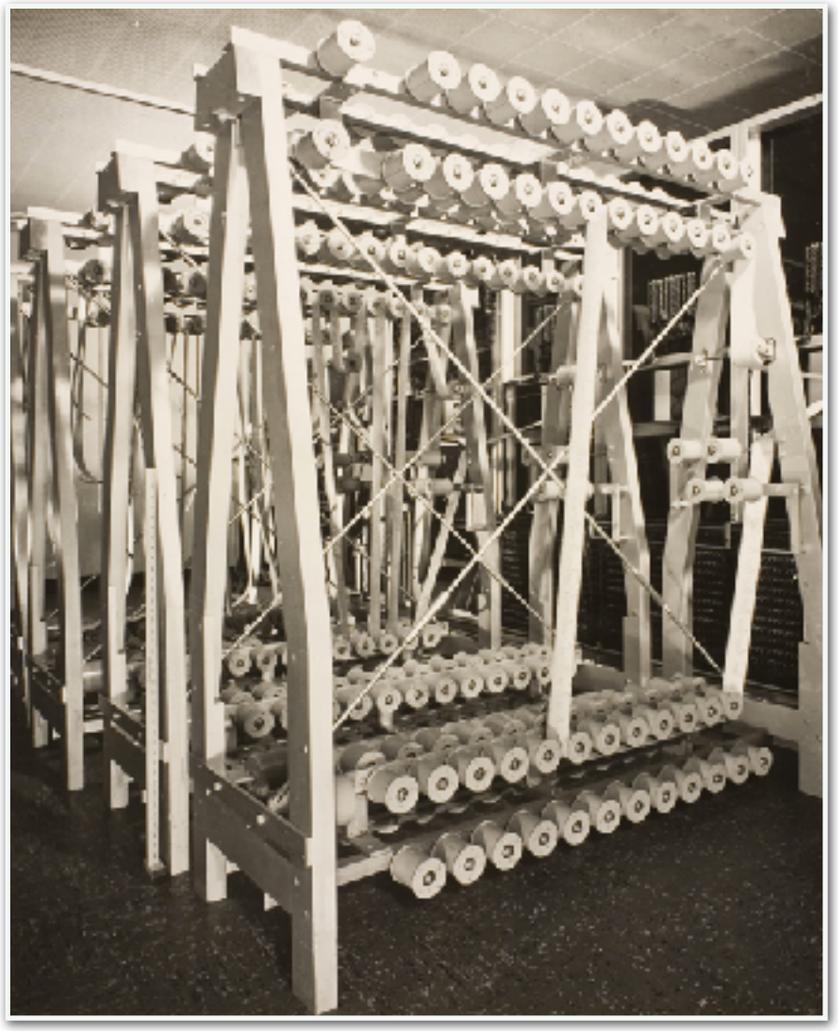
- A lot of common CS terms were coined by Mark I programmers
  - Documentary here: https://chsi.harvard.edu/harvard-ibm-mark-1-video



"Loop"

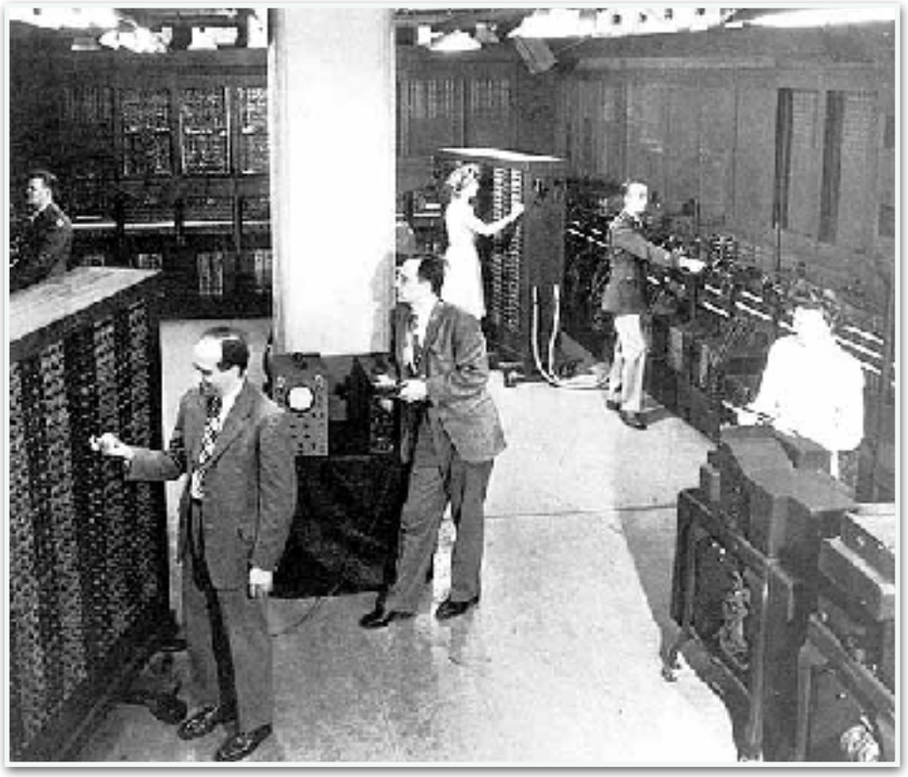


"Patch"



"Library"

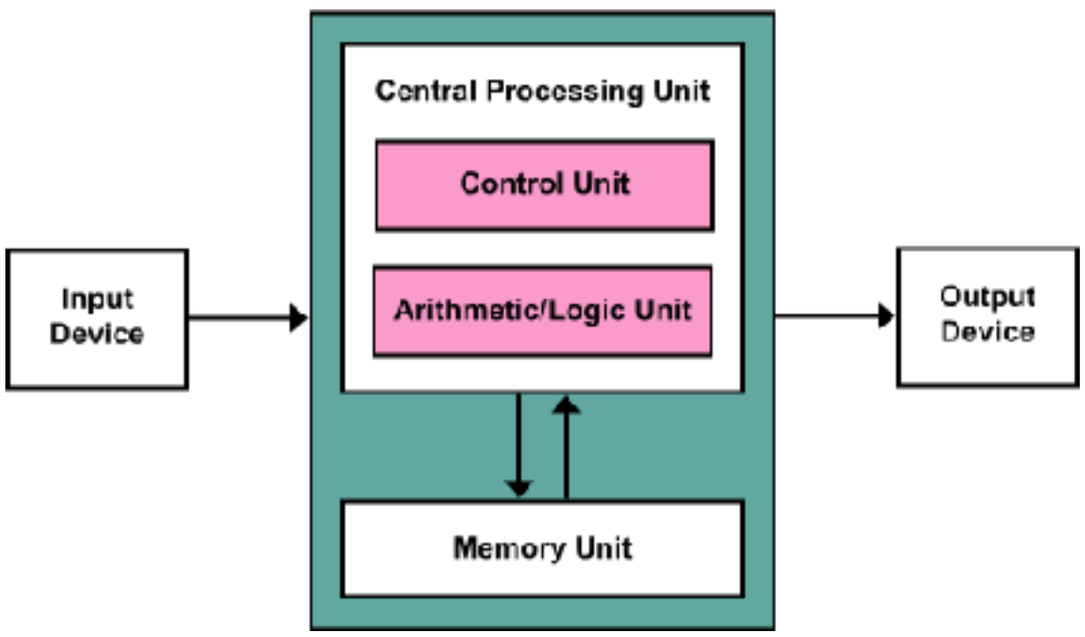
1940s: ENIAC (1946)



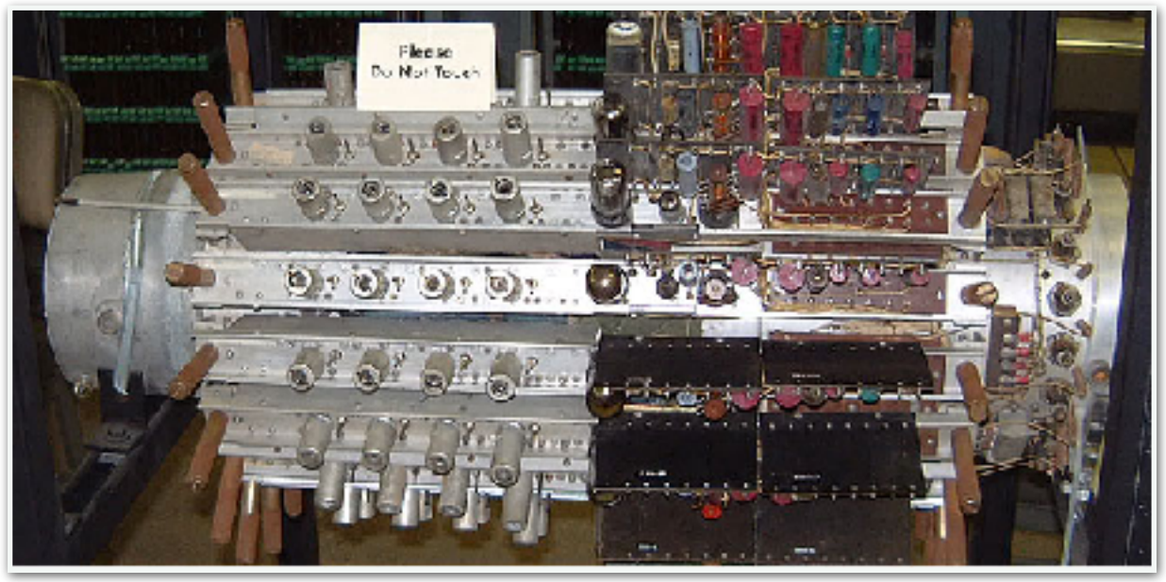
- ENIAC: Electronic Numerical Integrator and Computer
  - First general-purpose *fully electronic* computer
    - Moore School of Engineering @ UPenn
    - J. Presper Eckert and John Mauchly
  - No OS
    - More limiting than Mark I: Can't program ahead of time with tape/cards
    - Used switches and connections of cables (laborious) -- programmed "live" like Altair 8800

Mid-late '40s: EDVAC Changes Everything

- Mathematician John von Neumann teams up with Eckert and Mauchly
  - *Published: "First Draft of a Report on the EDVAC"*
    - The *Von Neumann model* of computing is born!
    - The *Stored-program model* *(Hmm.. how to "store" code/data in a machine?)*



The von Neumann model
(Still in use today)

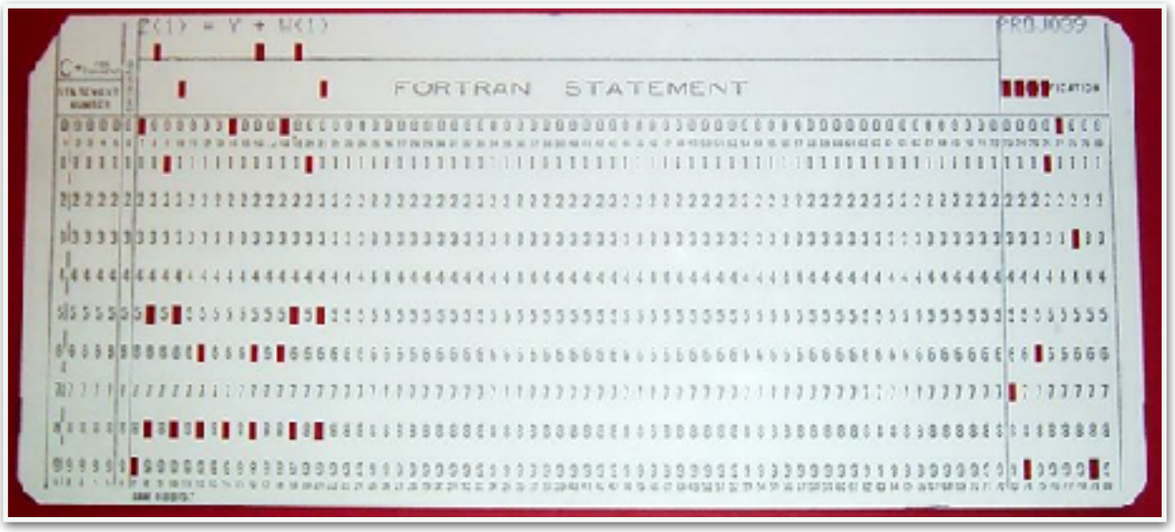


Mercury Delay-Line Storage
(https://www.youtube.com/watch?v=kignGE77I_I)

History Era 1: 1950-70's

- Hardware is expensive and humans are cheap!

- The OS begins to take form
  - Stored programs now commonplace
  - Programs on punch cards or tape
    - Load program, run, print results, dump, repeat.
    - Removed humans from the loop, mostly



A-0 Simplifies Programming

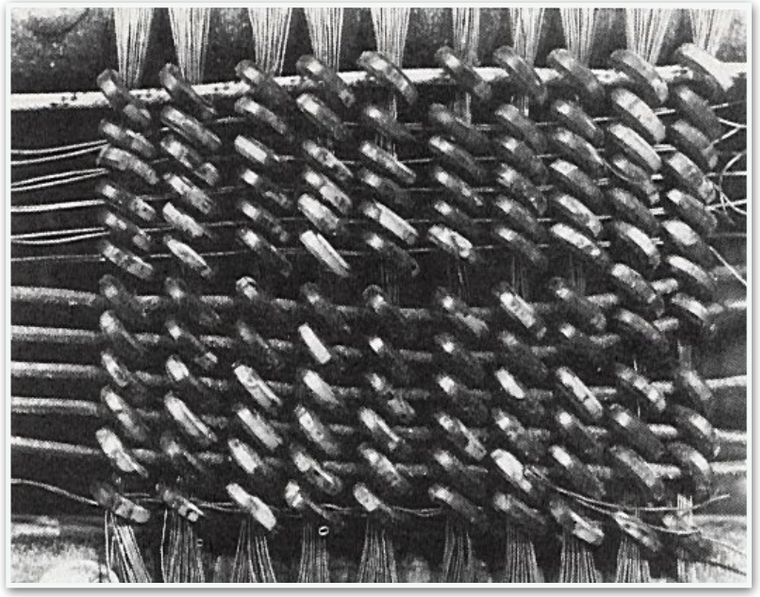
- 1951-1952:
  - Eckert-Mauchly's Univac becomes the first commercially-viable computer!

- In 1952 Hopper writes the first-ever compiler (A-0) for the UNIVAC.
  - *Fun fact: People belittled and resisted using the compiler for years!*

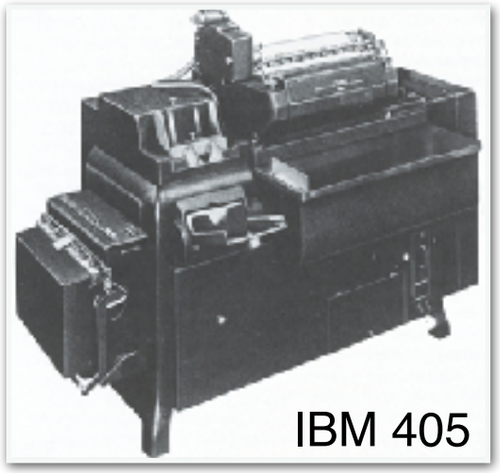
> "I had a running compiler," said Hopper, "And nobody would touch it because, *they* carefully told me, computers could only do arithmetic; they could not do programs. It was a selling job to get people to try it."
>
> "*Grace Hopper: Navy Admiral and Computer Pioneer*" by Charlene W. Billings. 1989.

1950's: Solid-State Core Memory

- Race for solid-state storage
  - Also in 1952: The first magnetic *core memory* tested on the IBM 405 Accounting Machine



- Bits stored as magnetization in iron rings.
  - *Non-volatile!* Magnetic direction is persistent.



- A "core dump" still refers to the contents in memory (for investigation) after a process fails.

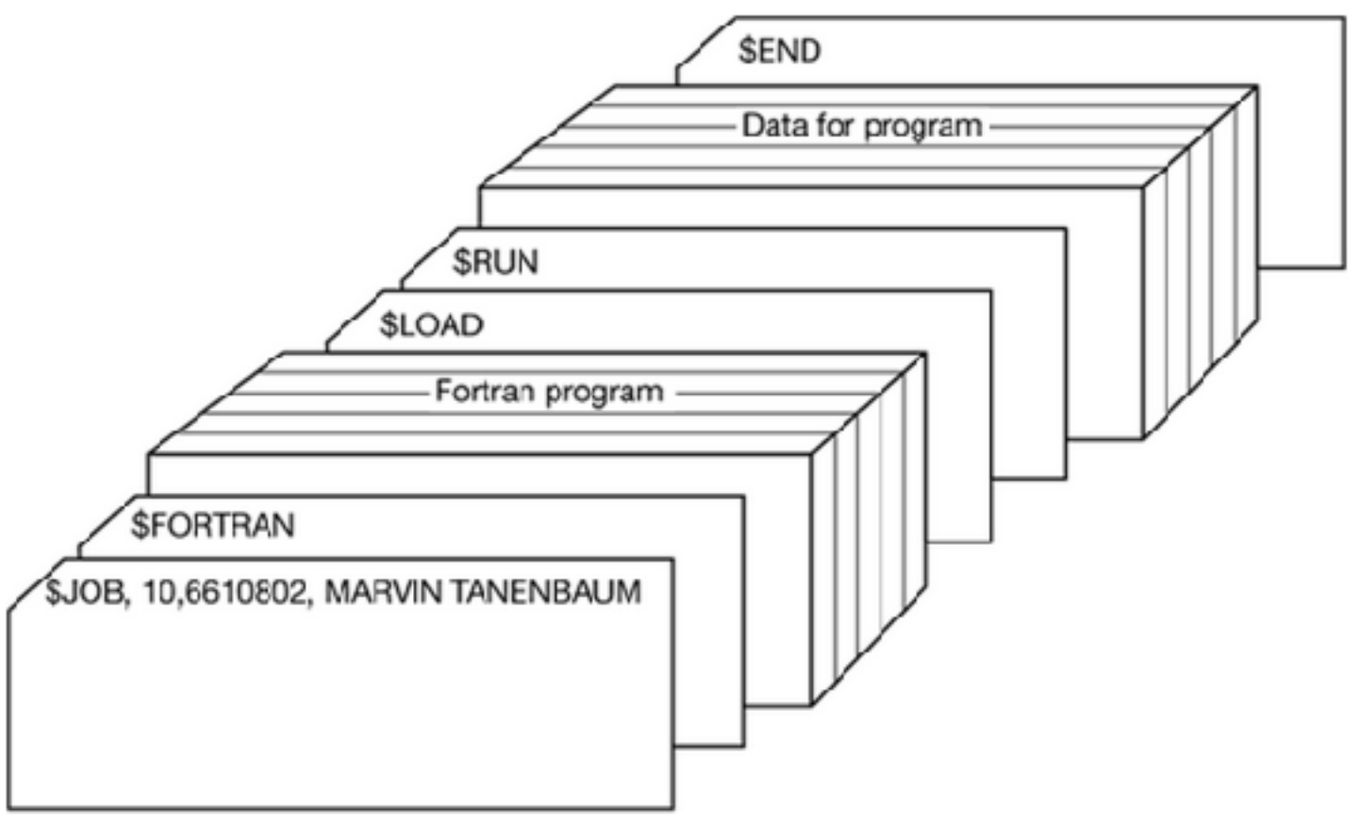
1950's (Batch Processing Model)

- 1952: IBM 701 Mainframe
  - This machine transitions IBM from a tabulation company to the computer business.

- The **First OS:** *"Resident Monitor"*
  - Established the *Batch Processing Model:* Run a job until it completes.
  - Jobs are loaded using card readers.

1950s: Communicating with the First OS

- *Control cards* marked the beginning and end of *segments*:
  - $JOB - First card of a job
  - $END - Final card of a job
  - $FORTRAN - run the fortran compiler
  - $LOAD - run the loader
  - ...



IBM 711 Punchcard Reader

History Era 2: 1960's

- Batch Processing was too wasteful
  - 

  - Spent too much time waiting around for slow devices
  - Time is precious, computer is expensive

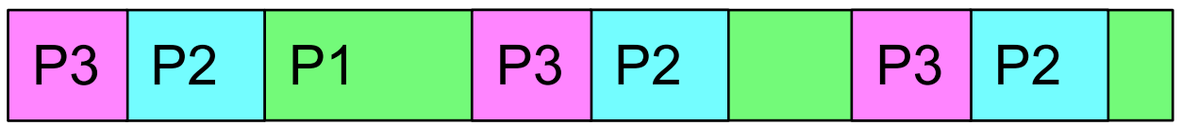
- Computers were becoming more capable by adding lots of slow input/output (IO) devices
  - But IO is slow, and the job must block until the IO operations finish

- We wanted better CPU utilization over batch processing
  - Run another job while a job blocks on IO!

1960's: Multiprogramming Is Born

- *"Multiprogramming Model"*
  - Run a job until it:
    - Completes, or
    - yields to another program, or
    - uses an IO device

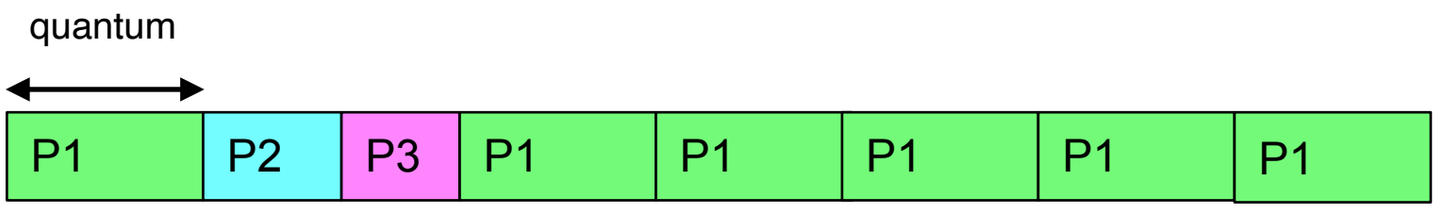
- On yield or IO, put the current job back into the ready queue, and choose a new job to run.



1960's: The Need for Timesharing

- Multiprogramming gets complicated
  - CPU utilization increased over batch processing, but...
  - Jobs that don't require IO can *still* monopolize the computer

- *Timesharing (Multitasking) Model:* Multiprogramming + Timeout
  - Each job is given a fixed amount of CPU time to run.
    - Called a "time slice" or a "quantum"
  - If job doesn't finish or do I/O within the quantum, then it gets swapped out.



Timesharing is Hard

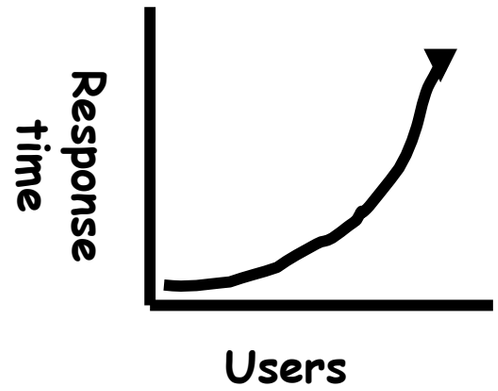
- Complexity gets out of hand!
  - Multiple jobs can coexist. Security matters now.
    - Job A should not be able to access Job B's memory

- **Multics:** Multiplexed Information and Computing Service (MIT, 1969)
  - Goal was to implement a fully *timesharing OS*
    - Conceptualized ideas for: a process, address spaces, virtual memory, shared systems libraries
  - Project failed: *"Too big and slow ... an overdesigned behemoth"*
    - But those concepts developed were highly influential and enduring.

History Era 3: 1970's

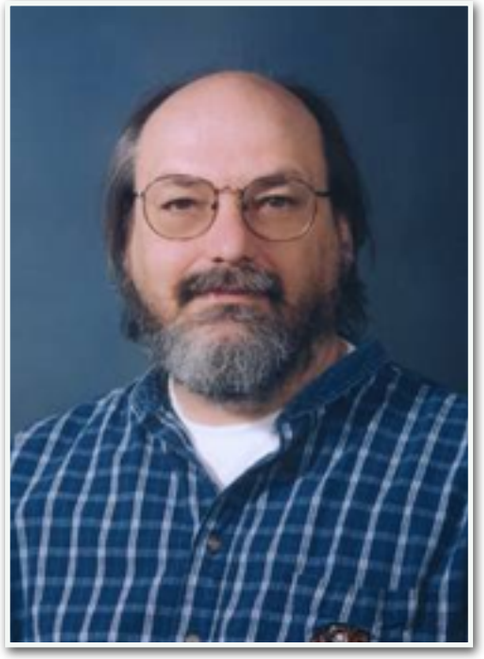
- Hardware becoming less expensive; humans getting more expensive
  - Computers now $100,000s rather than $1,000,000s
  - OS technology stabilizing, maturing, becoming *portable*
    - No longer designed for a specific machine hardware
  - *Want many users to **share** a single system simultaneously using cheap terminals*

- New Idea: *Interactive Timesharing Model*
  - Prioritize the jobs that do lots of I/O!
  - New problem: Thrashing
    - Too many context switches
    - Response time increase as users increase



1970: The Success of UNIX

- UNIX operating system is developed in Bell Labs
  - ("Unics" is a "cut-down version of Multics")
  - Developed by Ken Thompson and Dennis Ritchie (a former Multics programmer)



Ken Thompson

- UNIX rewritten in Dennis Ritchie's new language C
  - *Established C to be the de-facto systems language*
  - Lasting contribution: UNIX is portable!
    - Supported a wide-array of underlying hardware

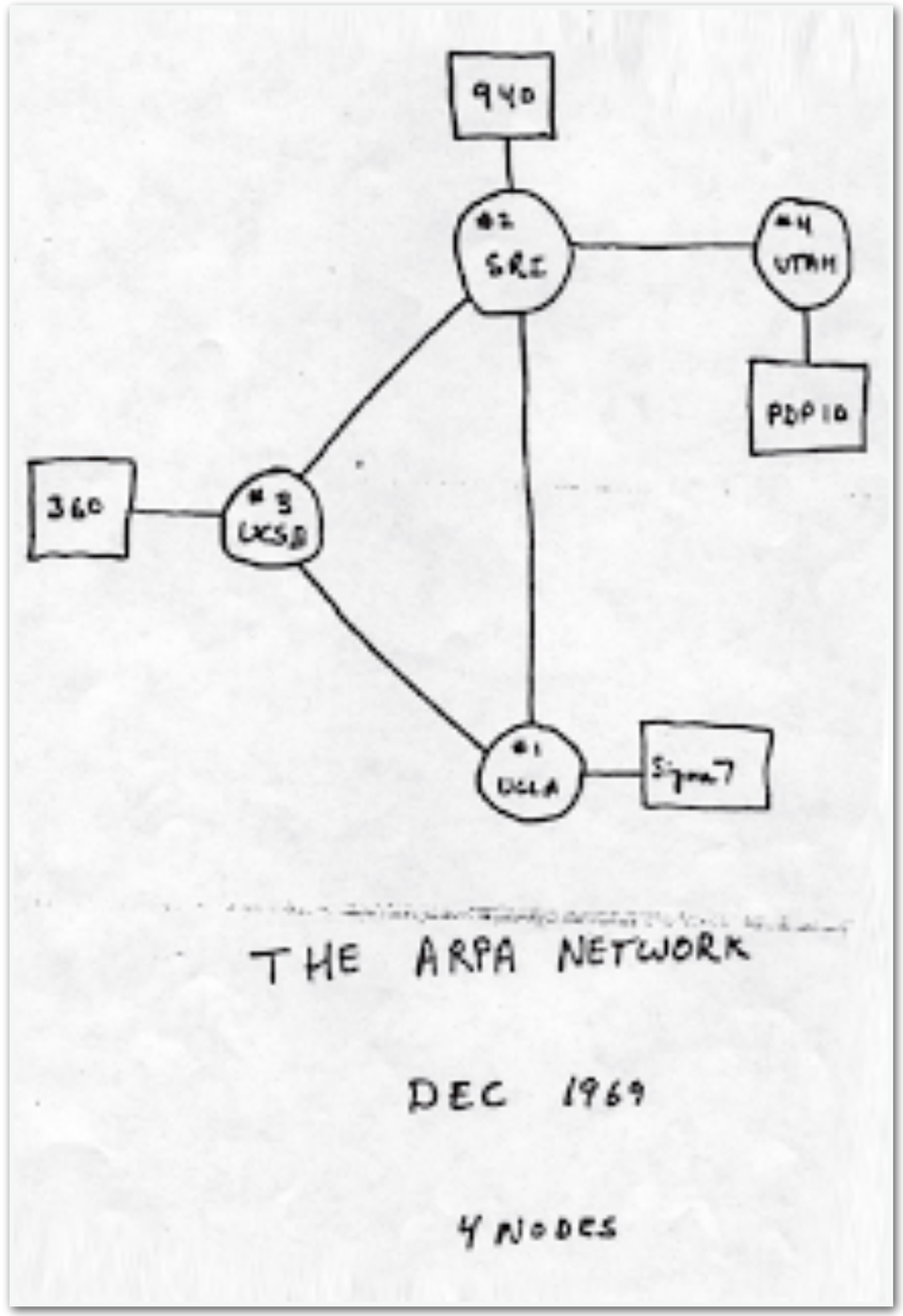
Dennis Ritchie

- Co-Recipients of the ACM Turing Award in 1983

Also around This Time... 1968-1970

- The emergence of ARPA Net
  - Resource-sharing computer network

- September 1969: UCLA becomes first node of ARPA Net
  - First message sent from UCLA to SRI: **"Lo"**
  - Four nodes connected by 56 Kbps phone lines:
    - UCLA, SRI, UCSB, University of Utah



History Era 3 (1980's)

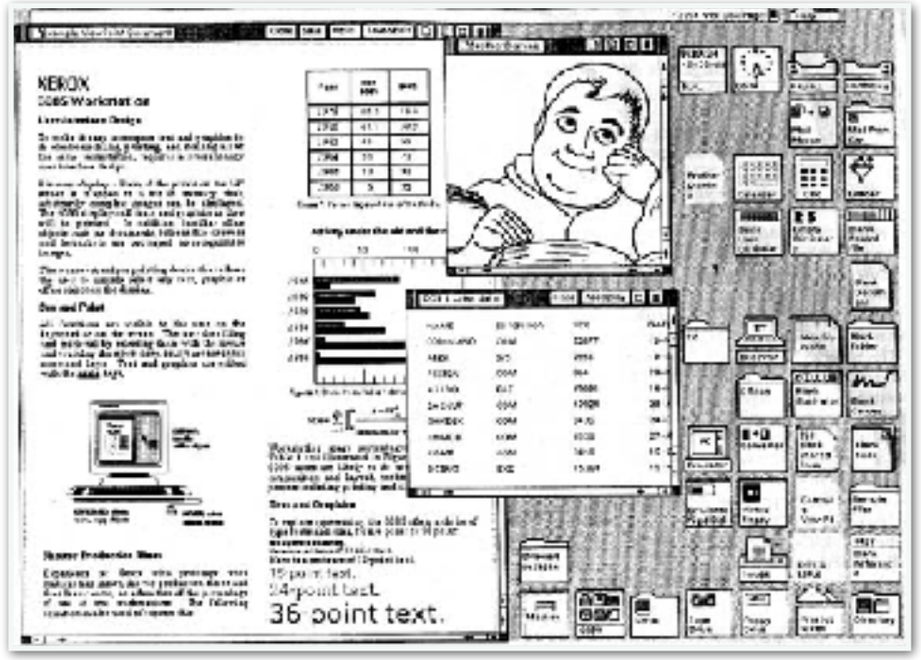
- Hardware very cheap, humans very expensive
  - Computers cost $10,000s, but programmers now cost $100K/yr
  - Cultural shift: Give people computers to make them more productive!

- Advocacy of Personal Computing (PC)
  - OS needed better user-friendliness and GUIs
  - A return to simplicity (due to 1-to-1 usage model)
    - Back to one program running at-a-time: MS-DOS
  - Followed by resurgence of complexity
    - People can multi-task: 1 user, many programs running

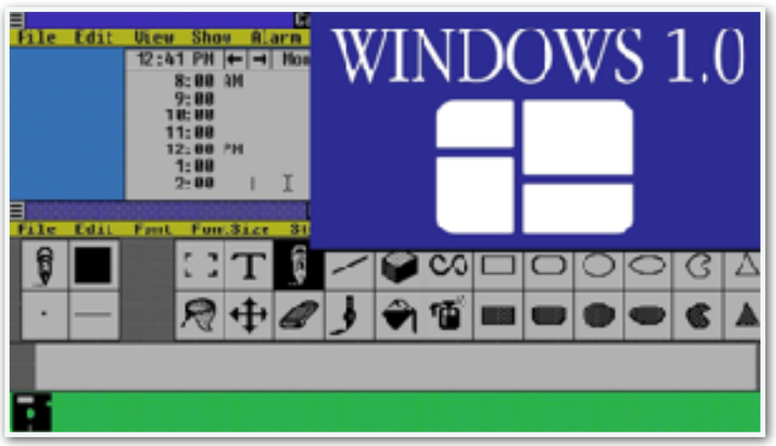
Apple Macintosh (1984)

Graphical User Interfaces (1980s)

- User Friendliness: GUIs added to OS
  - Xerox Star (1981) - $75,000
    - First use of mouse and GUI
  - Apple Lisa (1983) - $10,000
  - Apple Macintosh (1984) - $2500
  - X Window/X11 for Unix (1987)
  - Microsoft Windows (1985 - now)



Xerox Star

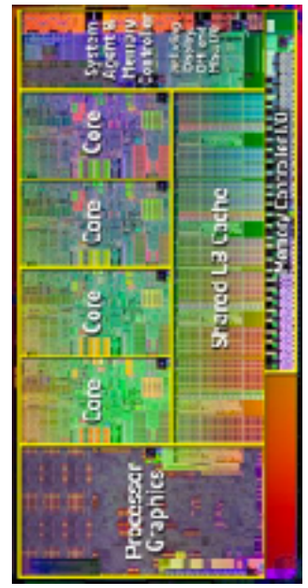


First Mouse: Douglas Englebert

History Era 4 (Today)

- Hardware is very cheap and ubiquitous (many devices per person)

- Multicore CPUs (~ 2001)
  - Monolithic CPU approaching physical limits
  - Slow them down and add more cores per CPU!
    - New problem: How to exploit parallel hardware?

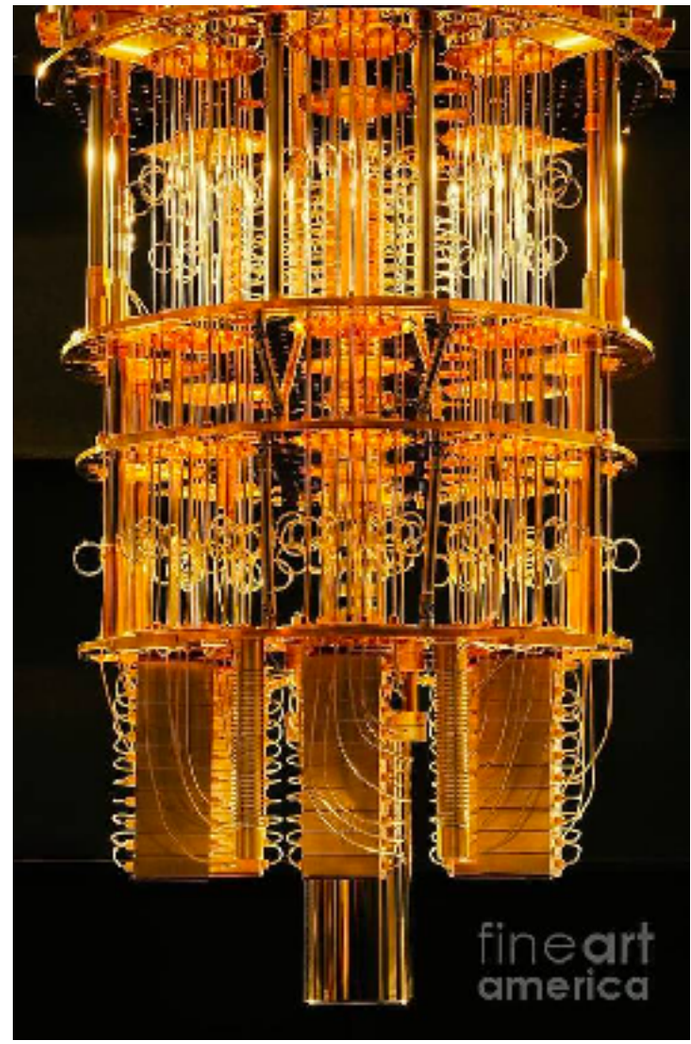


- Mobile computing (~ 2010)
  - Power efficiency now the chief concern
  - Lack of storage
    - OS must have a small "footprint". Back to batch processing again

What's Next? Quantum Computing (Probably)

- We're entering a new era of computing!
  - The dawn of something revolutionary (Again!)

- *Quantum Computers*
  - Experimental and very expensive ... like the ENIAC
  - D-WAVE quantum computer costs $10M - $15M



- *Qubits* harness quantum superpositioning
  - A *qubit* can represent a 0 and 1 simultaneously
  - n qubits means 2^n possibilities may be explored simultaneously, but verification of results is tricky.

Administrivia 1/24

- Bingo Night at 4pm following class in CS Lounge!

- Reading
  - Chap 1 of Dinosaur Book
  - Chap 1-3 of Dive into Systems (on C)

- Hwk 1 due 11:59p tonight!
  - Your username is your puget sound login
  - Your password is **qwe123**

Administrivia 2/3

- Reminders:
  - Hwk 2 due 11:59p tonight!
  - Hwk 3 posted (due Friday, Feb 14)

- Last time:
  - What is the OS?
- Today:
  - A brief history of computing machinery
  - Batch processing, multiprogramming, timesharing

`

- Reading: Finish Chapter 1; Start Chapter 2
- Last Time...
  - Computers driven by economic forces
    - Military supremacy (ballistics tables), US census (data collection)
  - Realization of Von Neumann's stored program model changes everything
- Today:
  - Batch processing
  - Multiprogramming
  - Timesharing