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

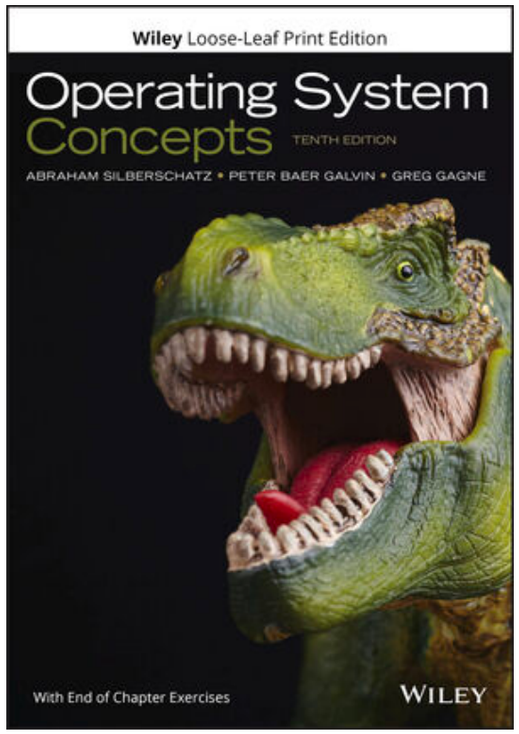
About David

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

- Background
  - PhD, Computer Science & Engineering, Ohio State, 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 (both required)
  - Dive into Systems (free e-book)
    - 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
  - 40% - Homework Assignments
  - 15% - Midterm Exam I
  - 18% - Midterm Exam II
  - 22% - Final Exam (Comprehensive)

  - 5% - Participation
    - 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 at first
    - But C is 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

Comprehensive Exams

- Two **1-hr midterms** and a **2-hr final** will be based on
  - Lectures, notes, homework
  - Comprehensive, but weighted heavily on new material

- 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

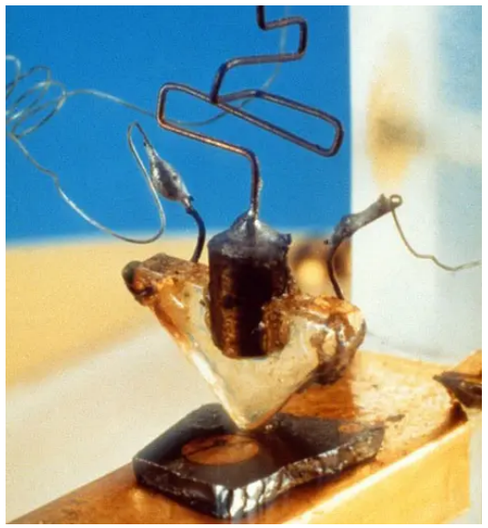
- Dedicate time to learning C
  - C is not intrisically *hard*, but it's nuanced and debugging is frustrating!
    - There's no "stack trace" when a program crashes.
  - Trivial tasks will take you longer to write in C (vs. other languages)
    - Imperative that you *master* strings and pointers early
      - (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
- Technological Trends
- What Is an OS, Anyway?
- A Brief History of Computer and Operating Systems
- Conclusion

Bell Labs and the Transistor

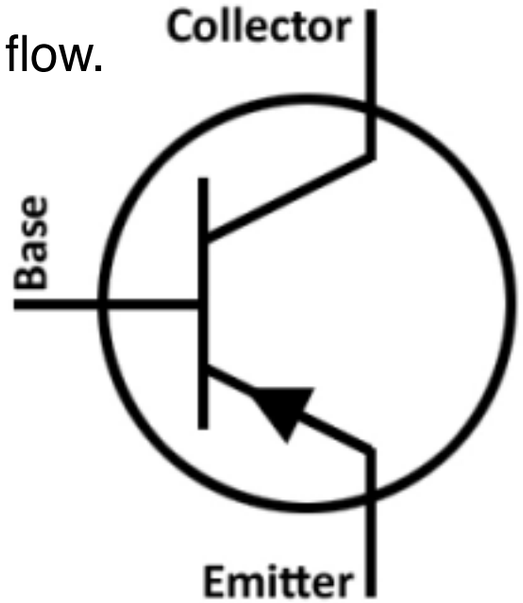
- The *"transistor"* is one the most important inventions in the 20th century.
  - Basic building blocks of logic gates, adders, flip-flops, and so on.
- Invented in Bell Labs in 1947
  - John Bardeen, Walter Brattain, and William Shockley
  - Earned the 1956 Nobel Prize in Physics



The original Bell Labs transistor

It acts like a faucet (or *switch*!) for electric flow.

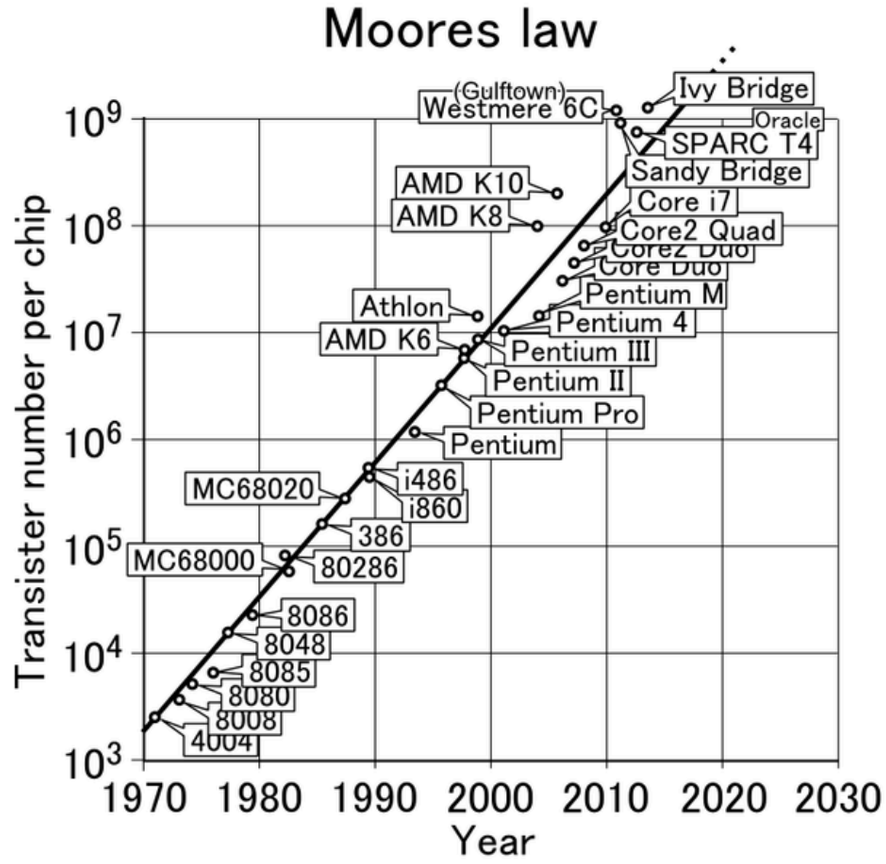
1. Emitter (electricity flows in)
2. Base (faucet handle)
3. Collector (electricity flows out)



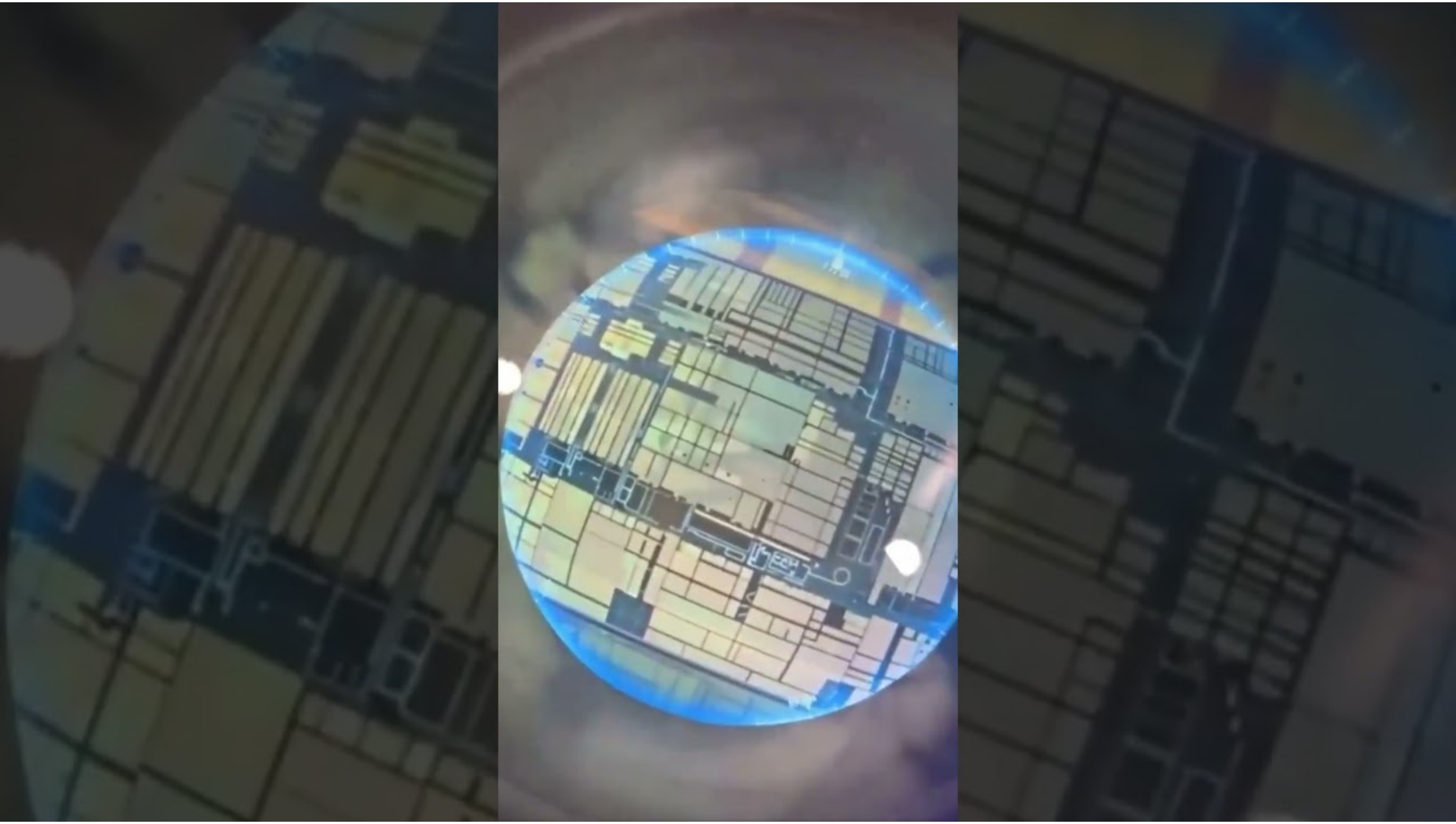
Transistors Enabled Microchips

- Gordon Moore (Co-Founder of Intel)
  - Former employee of Shockley Semiconductor
  - Part of the infamous *"Traitorous 8"*

- Known for his "law" or rather, prediction:
  - *"The number of transistors on a chip doubles every 18-24 months."*
  - Reaching physical limits: if transistor width is a few **nanometers**, quantum tunneling can occur, making them unreliable.

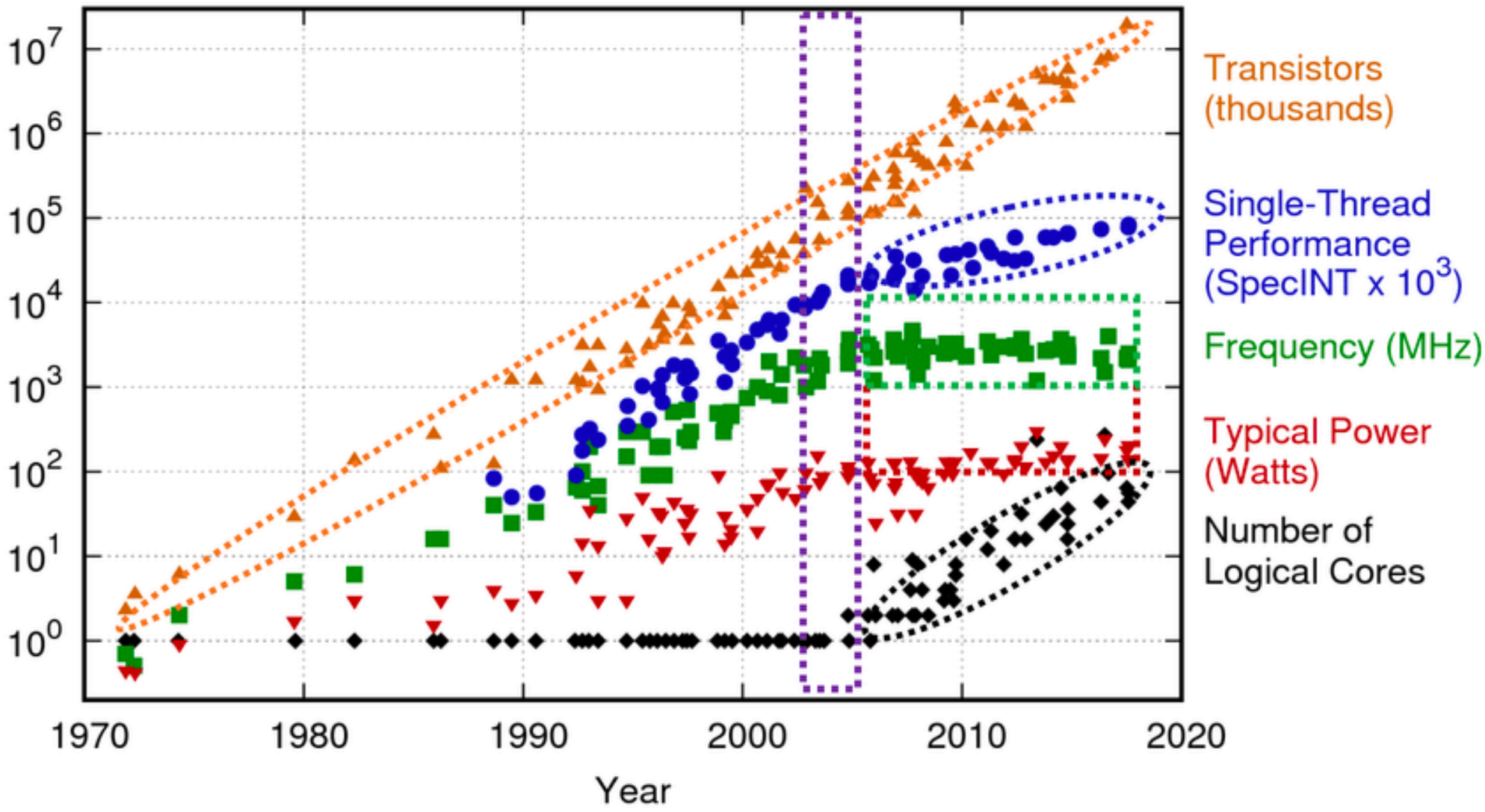


Scale of Chip Components



The Slowdown of Joy's Law

- *Bill Joy's Law:* CPU Performance doubles every ~18 months
  - But notice that the CPU Performance suddenly dropped off circa 2001.



Dynamic Power Consumption of CPUs

- P approximates the power consumed by the CPU:

$$P = CV^2f$$

capacitance (chip area)

voltage

CPU frequency (i.e., clock rate)

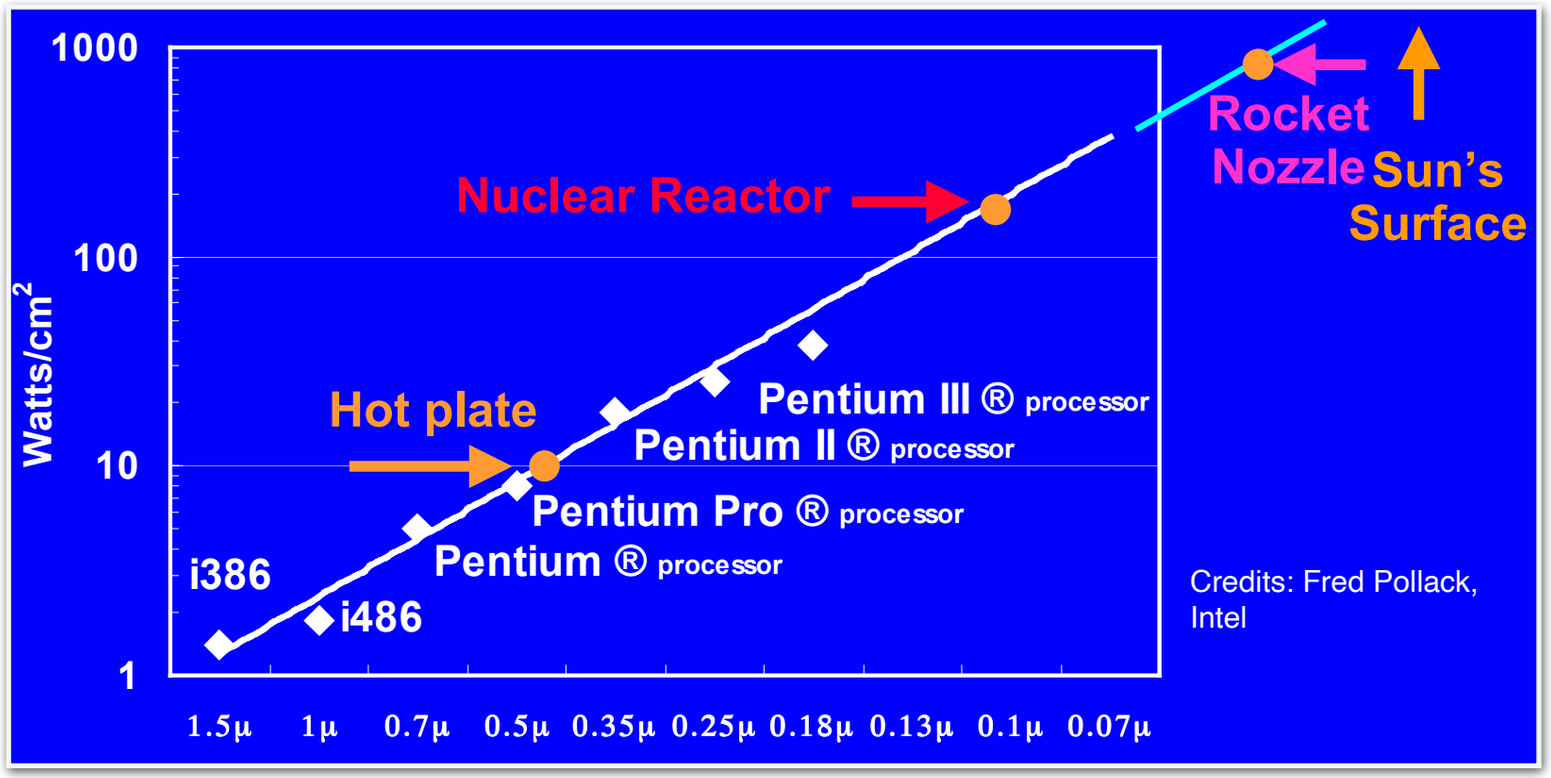
- Before 2001, chip manufacturers had always found ways to increase clock rate f to increase CPU performance.
  - (The CPU used to be just one massive, monolithic core!)

Problem: Power (Heat) Wall

- "Constraint [of chip making] is power, not manufacturability."

  - Dr. Fred Pollack, Intel Fellow

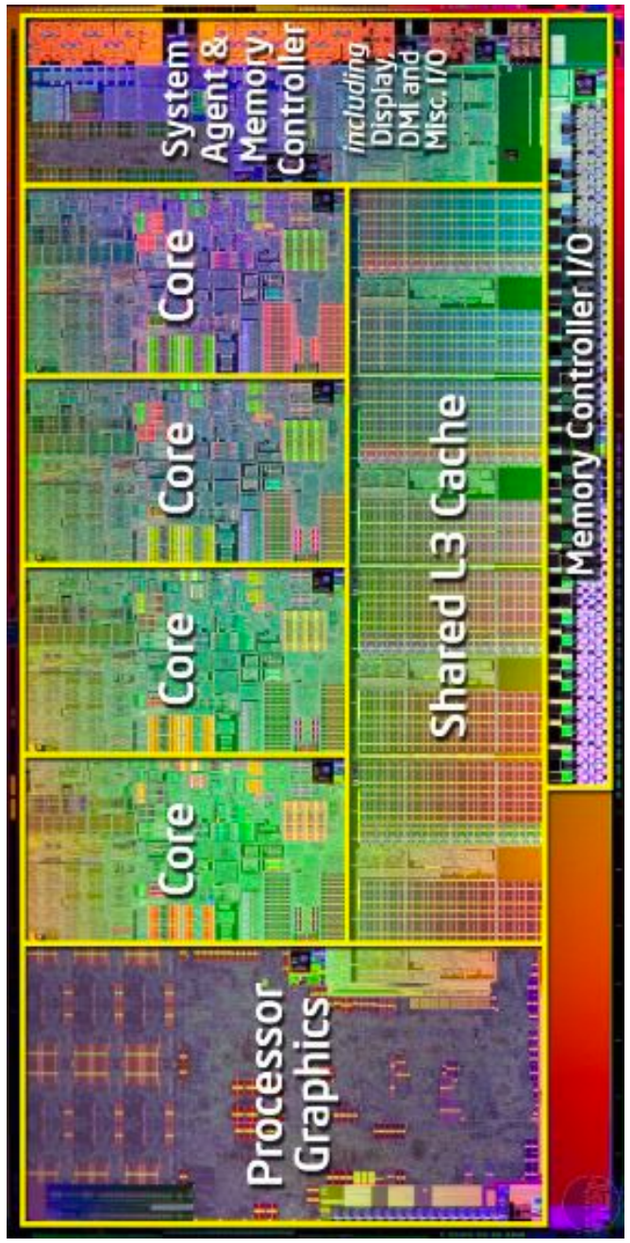
- Chip manufacturers were met with the *"power wall"* in the early 2000s.
  - Profound amount of heat generated in a very small area.



Solution: Multicore CPUs

- Reduce f (clock rate) on the single CPU to make it puny
  - But Moore's Law permitted more transistors per unit area.
  - **Ah ha!** Add more puny CPUs ("cores") to each chip

- Dynamic Power P can now be held constant, but what about *performance?*
  - Instead of one massive 4 GHz CPU, make a CPU with two puny cores running at 2 GHz a piece. 2+2 = 4 right?
  - Can we get close to 4 GHz performance when both cores are utilized simultaneously? *(The need for parallelism)*



Big Picture Today

- Life used to be easy. Slow code? No problem, wait for Moore's Law.
- Today
  - Multi-core CPUs are now commonplace
  - Need support from OS
  - Increased need for parallel programmers
- What's Next?
  - The decline of Moore's Law reveals new opportunities.
    - Co-Processors like Nvidia GPUs
    - 3D stacking of CPU planes
    - Advanced materials, like graphene, instead of silicon to overcome physical limits
    - Quantum computers

Today's Topics

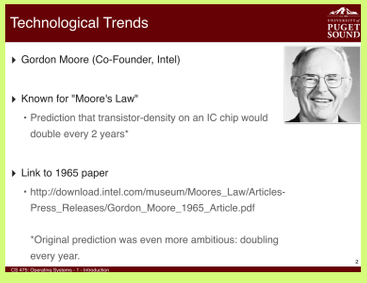
- Course Overview
- What Is an Operating System, Anyway?
- A Brief History of Computer and Operating Systems
- Conclusion

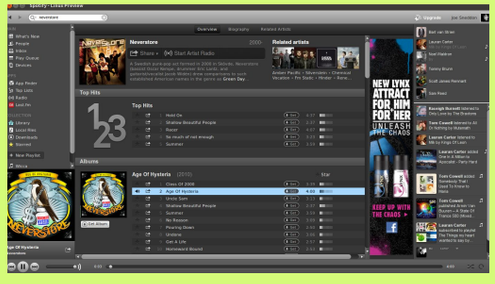
What's Life Like without OS?

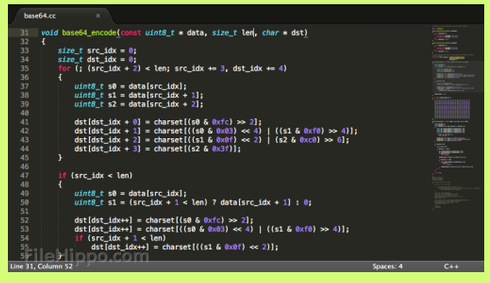
- MITS Altair 8800 (1975)
  - World's first affordable PC
    - Intel 8080 (8-bits, 2MHz), 256B RAM, $397
  - No keyboard, no monitor display, no OS
    - Program by toggling binary code, hit run, read results off the LED display.
    - Sparked many startups (Microsoft was born!)
- For the curious
 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

What Is an Operating System (Kernel)?

User Applications







OS Services
(via System Call 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

What Is an Operating System?

- The OS is an illusionist

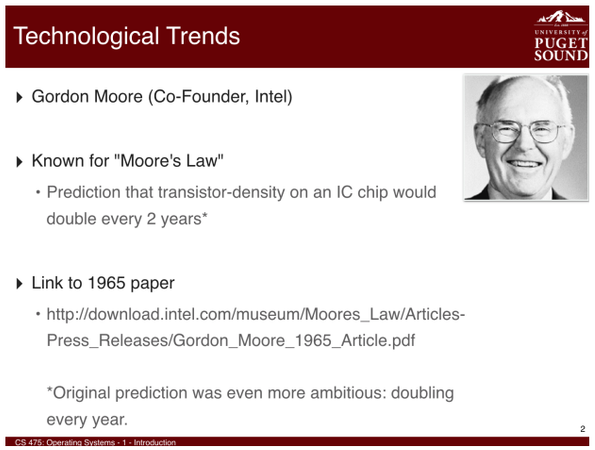
- Illusion:
  - Programs (processes) think they get unrestricted access to all hardware units.

- Reality: OS must establish controlled sharing among multiple processes.
  - This is called *"hardware virtualization"*

Illusionist: Virtualizing the CPU

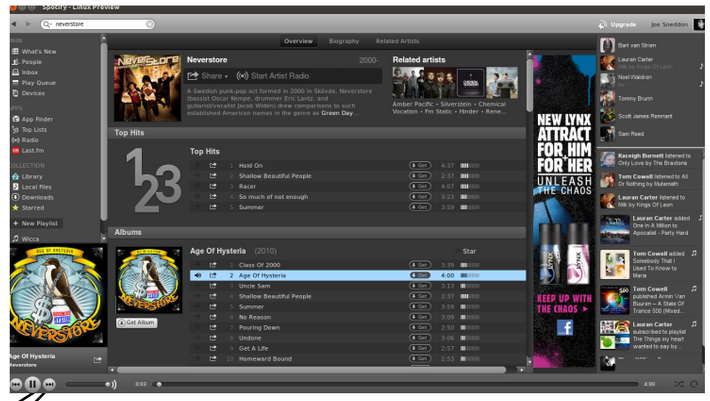
- Each running program *thinks* it gets to run on its own CPU.

Keynote



I think I have my own CPU

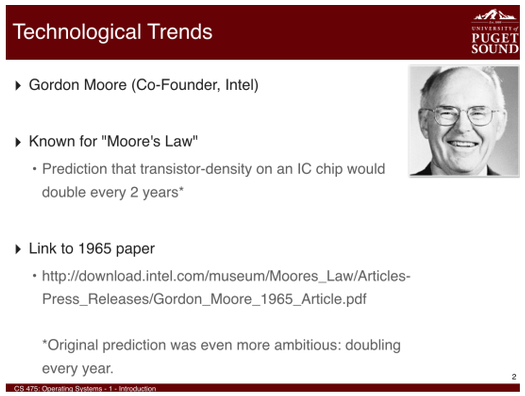
Spotify

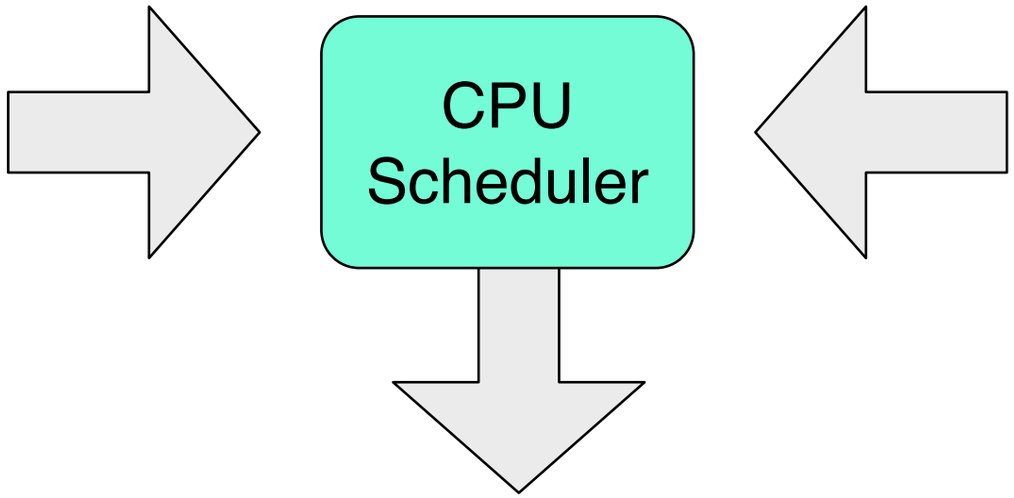


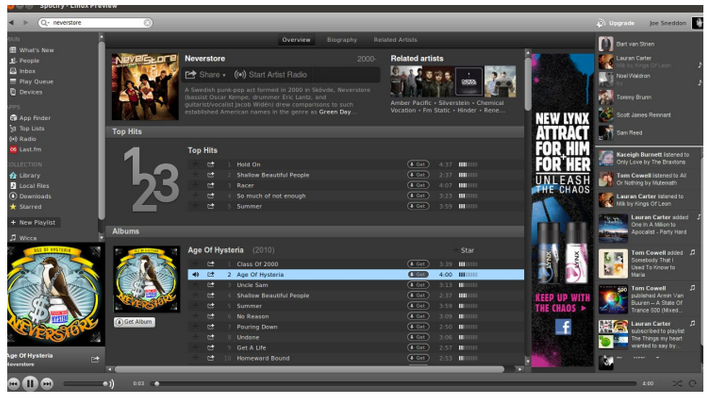
So do I!

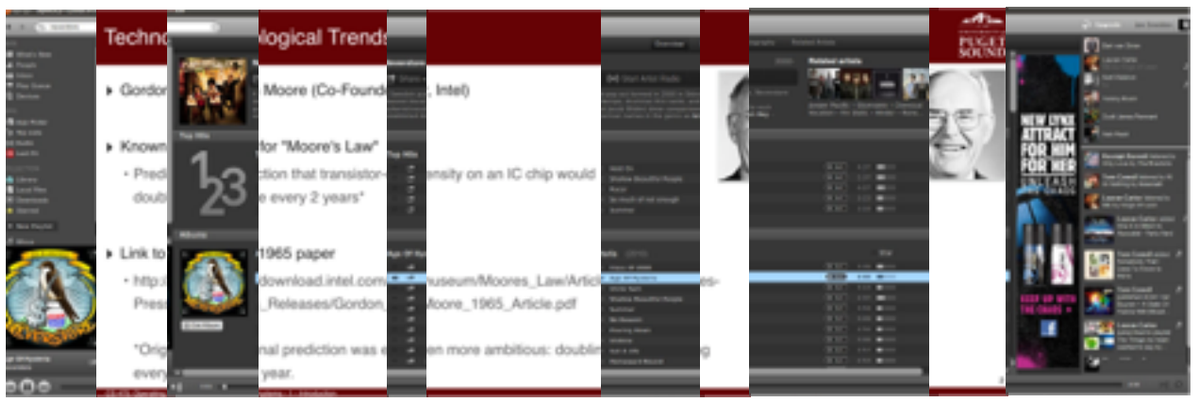
Illusionist: Virtualizing the CPU (Cont.)

- **Reality:** number of processes **>** number of CPUs
  - *CPU scheduler* establishes time-sharing via "context switching."





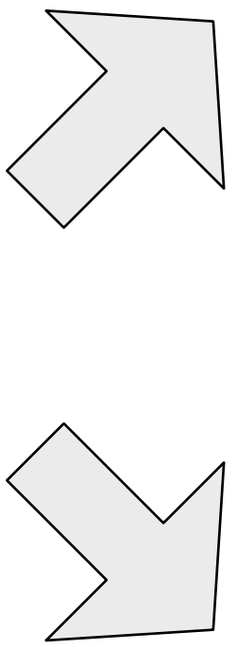


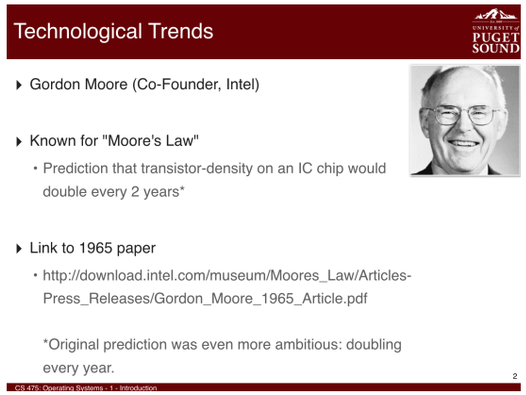


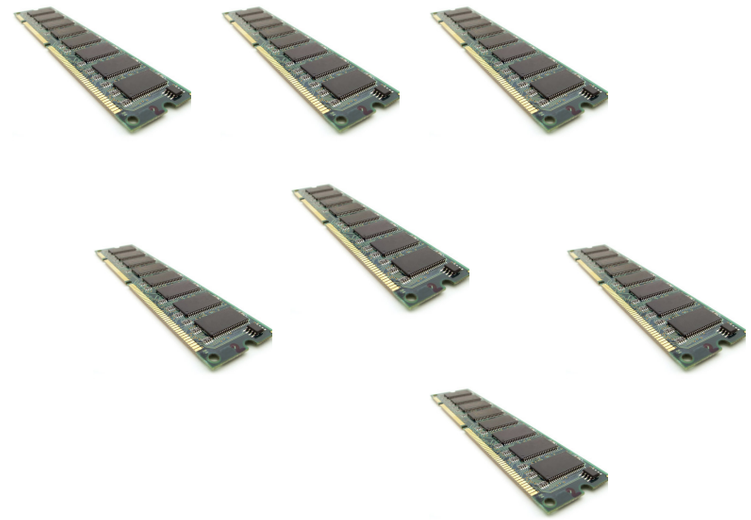
Timeshare a single CPU core

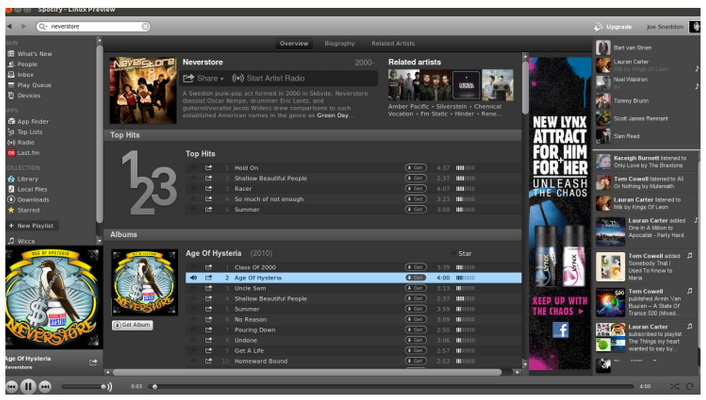
Illusionist: Virtualizing Memory

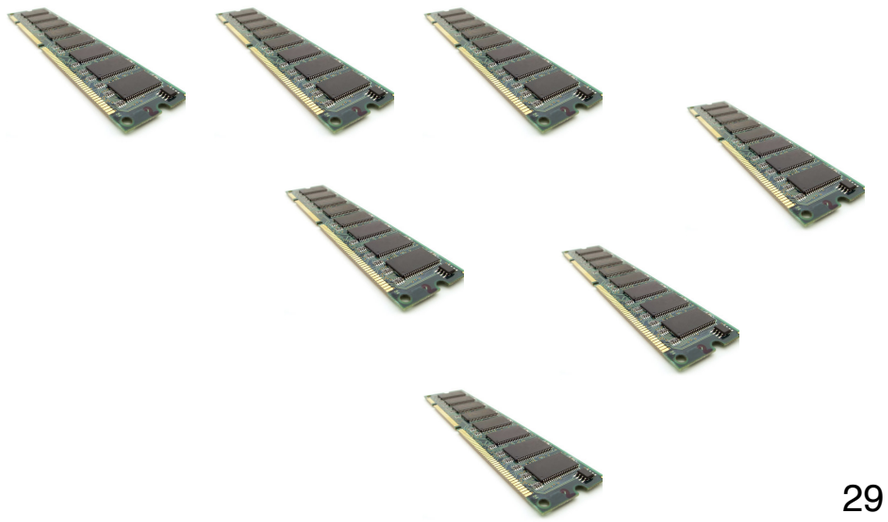
- Provide an illusion of having infinite memory dedicated to each program





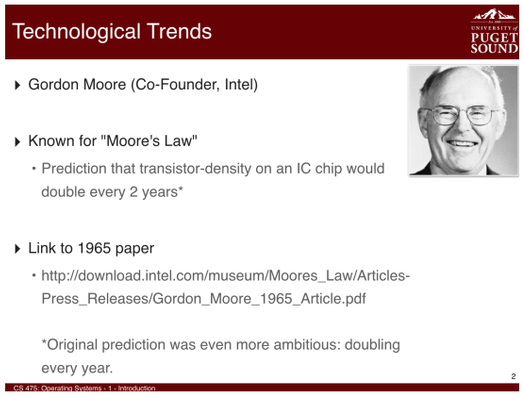


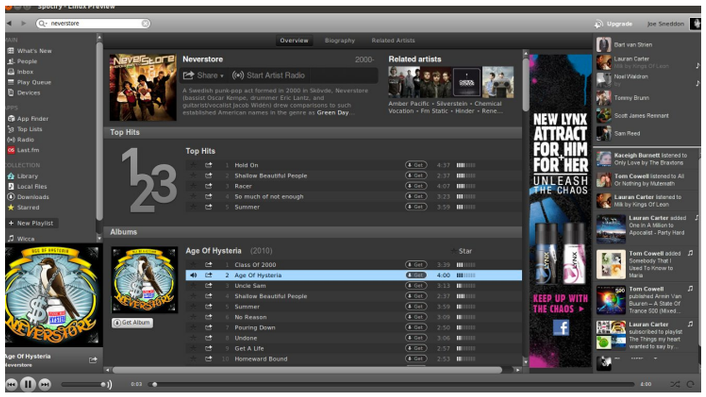


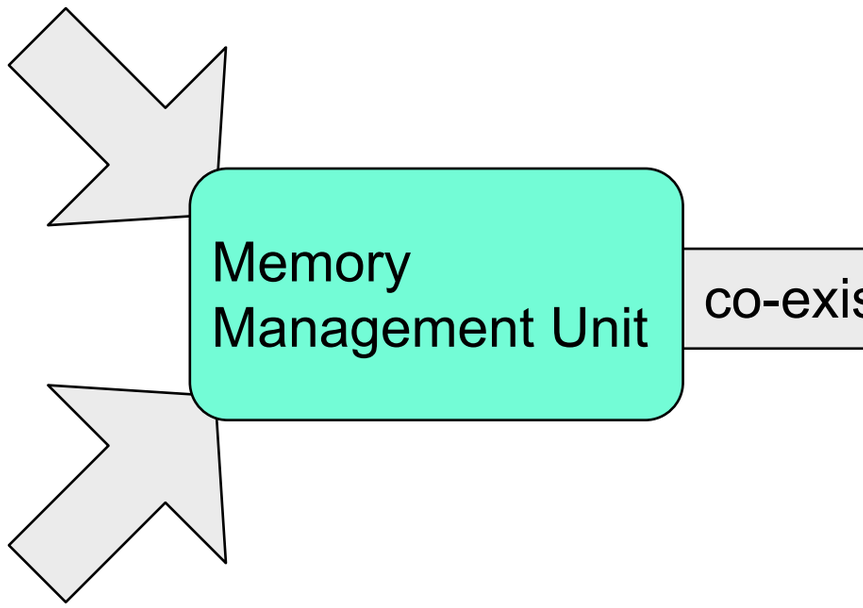


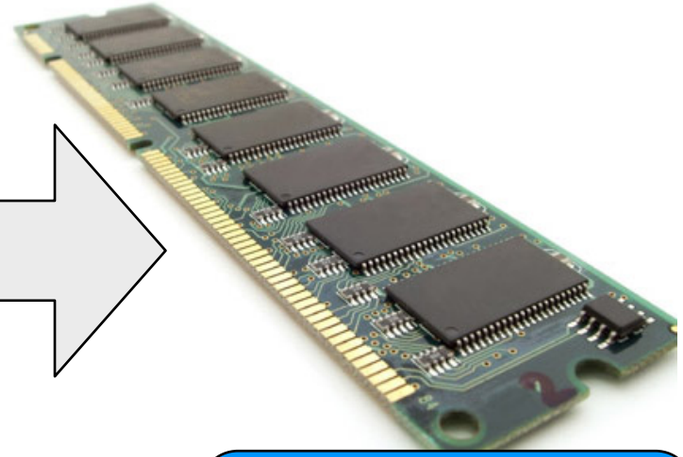
Illusionist: Virtualizing Memory

- **Reality**: Limited amount of memory. OS's memory management unit allows for sharing (and protection).



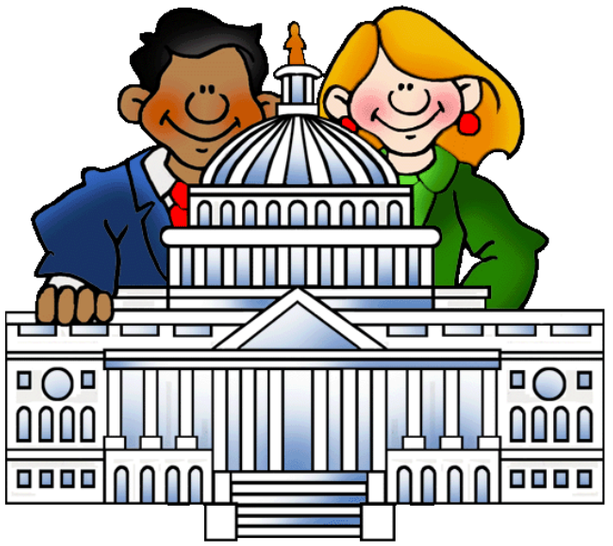






What Else Does an OS Do for Us?

- The OS is also like a government
  - It provides a set of regulated services that everyone needs but are too risky to let users use without oversight. Examples:
    - Create, terminate, suspend a process
    - Open/read/write a file
    - Printing to the screen
    - Requesting more memory during runtime



- Users can request the OS to take action made through a *"system call"*
  - Each OS provides a standard library of system functions
  - (In Linux, calls made through a C library called **unistd.h**)

Sample System Call APIs

- *System calls:* These actions are privileged, so you must request the OS to do them on your behalf. (Defined in **unistd.h**)

Category	Linux
Process Control	`fork()`
	`exit()`
	`wait()`
File Manipulation	`open()`
	`read()`
	`write()`
	`close()`
File Protection	`chmod()`
	`chown()`
...	**...**

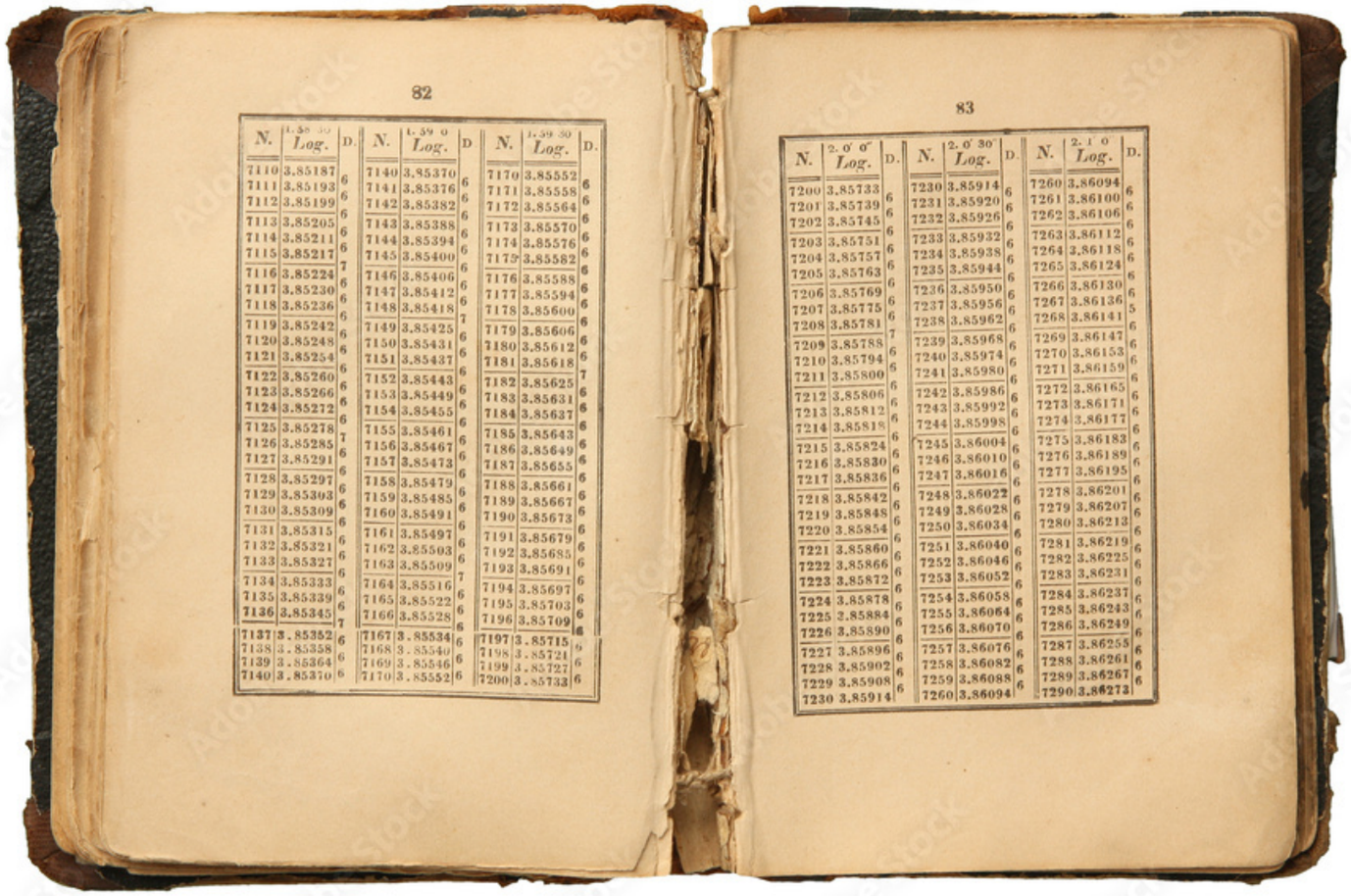
Today's Topics

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

1700s - 1800s: Origins of Computing

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

Example: Table of Logarithms



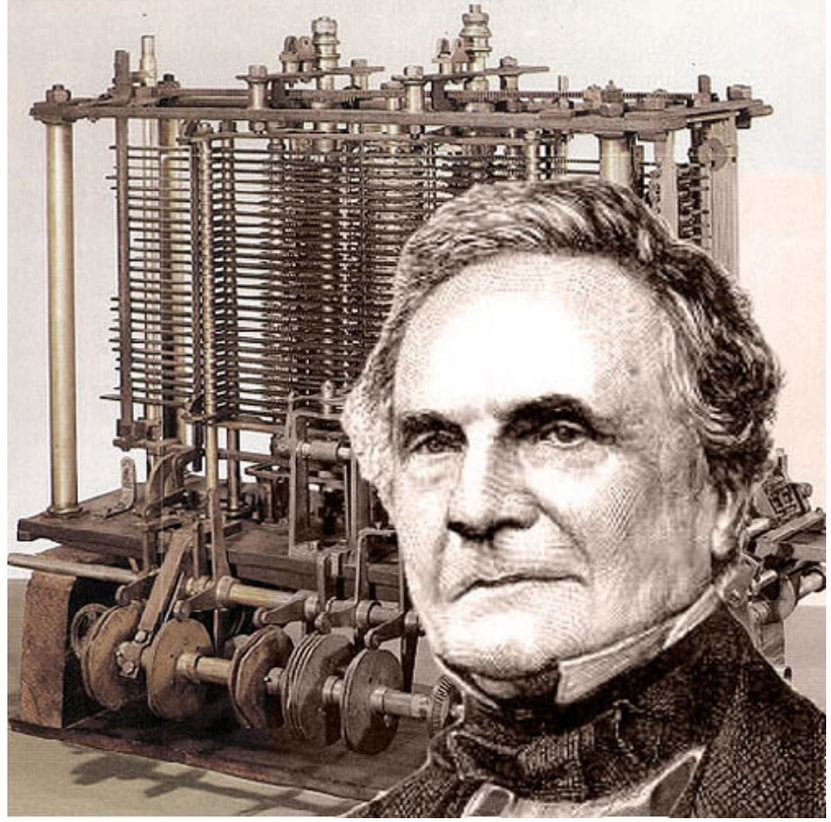
82

N.	Log.	D.	N.	Log.	D	N.	Log.	D.
7110	3.85187	6	7140	3.85370	6	7170	3.85552	6
7111	3.85193	6	7141	3.85376	6	7171	3.85558	6
7112	3.85199	6	7142	3.85382	6	7172	3.85564	6
7113	3.85205	6	7143	3.85388	6	7173	3.85570	6
7114	3.85211	6	7144	3.85394	6	7174	3.85576	6
7115	3.85217	7	7145	3.85400	6	7175	3.85582	6
7116	3.85224	6	7146	3.85406	6	7176	3.85588	6
7117	3.85230	6	7147	3.85412	6	7177	3.85594	6
7118	3.85236	6	7148	3.85418	7	7178	3.85600	6
7119	3.85242	6	7149	3.85425	6	7179	3.85606	6
7120	3.85248	6	7150	3.85431	6	7180	3.85612	6
7121	3.85254	6	7151	3.85437	6	7181	3.85618	7
7122	3.85260	6	7152	3.85443	6	7182	3.85625	6
7123	3.85266	6	7153	3.85449	6	7183	3.85631	6
7124	3.85272	7	7154	3.85455	6	7184	3.85637	6
7125	3.85278	6	7155	3.85461	6	7185	3.85643	6
7126	3.85285	6	7156	3.85467	6	7186	3.85649	6
7127	3.85291	6	7157	3.85473	6	7187	3.85655	6
7128	3.85297	6	7158	3.85479	6	7188	3.85661	6
7129	3.85303	6	7159	3.85485	6	7189	3.85667	6
7130	3.85309	6	7160	3.85491	6	7190	3.85673	6
7131	3.85315	6	7161	3.85497	6	7191	3.85679	6
7132	3.85321	6	7162	3.85503	6	7192	3.85685	6
7133	3.85327	6	7163	3.85509	7	7193	3.85691	6
7134	3.85333	6	7164	3.85516	6	7194	3.85697	6
7135	3.85339	6	7165	3.85522	6	7195	3.85703	6
7136	3.85345	7	7166	3.85528	6	7196	3.85709	6
7137	3.85352	6	7167	3.85534	6	7197	3.85715	6
7138	3.85358	6	7168	3.85540	6	7198	3.85721	6
7139	3.85364	6	7169	3.85546	6	7199	3.85727	6
7140	3.85370	6	7170	3.85552	6	7200	3.85733	6

83

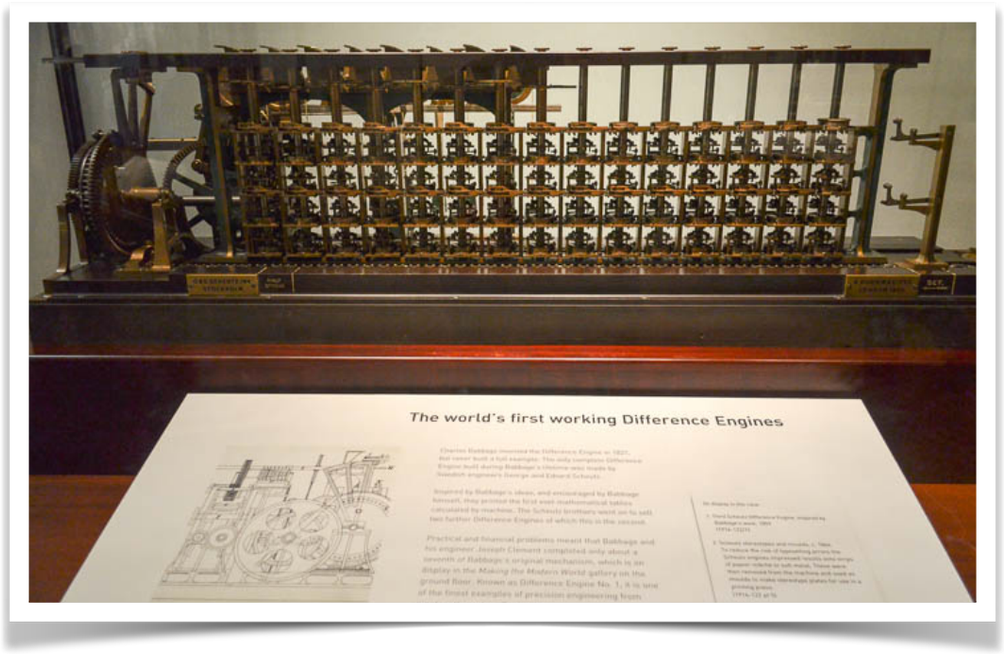
N.	2. 0 0 Log.	D.	N.	2. 0 30 Log.	D.	N.	2. 1 0 Log.	D.
7200	3.85733	6	7230	3.85914	6	7260	3.86094	6
7201	3.85739	6	7231	3.85920	6	7261	3.86100	6
7202	3.85745	6	7232	3.85926	6	7262	3.86106	6
7203	3.85751	6	7233	3.85932	6	7263	3.86112	6
7204	3.85757	6	7234	3.85938	6	7264	3.86118	6
7205	3.85763	6	7235	3.85944	6	7265	3.86124	6
7206	3.85769	6	7236	3.85950	6	7266	3.86130	5
7207	3.85775	6	7237	3.85956	6	7267	3.86136	6
7208	3.85781	7	7238	3.85962	6	7268	3.86141	6
7209	3.85788	6	7239	3.85968	6	7269	3.86147	6
7210	3.85794	6	7240	3.85974	6	7270	3.86153	6
7211	3.85800	6	7241	3.85980	6	7271	3.86159	6
7212	3.85806	6	7242	3.85986	6	7272	3.86165	6
7213	3.85812	6	7243	3.85992	6	7273	3.86171	6
7214	3.85818	6	7244	3.85998	6	7274	3.86177	6
7215	3.85824	6	7245	3.86004	6	7275	3.86183	6
7216	3.85830	6	7246	3.86010	6	7276	3.86189	6
7217	3.85836	6	7247	3.86016	6	7277	3.86195	6
7218	3.85842	6	7248	3.86022	6	7278	3.86201	6
7219	3.85848	6	7249	3.86028	6	7279	3.86207	6
7220	3.85854	6	7250	3.86034	6	7280	3.86213	6
7221	3.85860	6	7251	3.86040	6	7281	3.86219	6
7222	3.85866	6	7252	3.86046	6	7282	3.86225	6
7223	3.85872	6	7253	3.86052	6	7283	3.86231	6
7224	3.85878	6	7254	3.86058	6	7284	3.86237	6
7225	3.85884	6	7255	3.86064	6	7285	3.86243	6
7226	3.85890	6	7256	3.86070	6	7286	3.86249	6
7227	3.85896	6	7257	3.86076	6	7287	3.86255	6
7228	3.85902	6	7258	3.86082	6	7288	3.86261	6
7229	3.85908	6	7259	3.86088	6	7289	3.86267	6
7230	3.85914	6	7260	3.86094	6	7290	3.86273	6

Charles Babbage

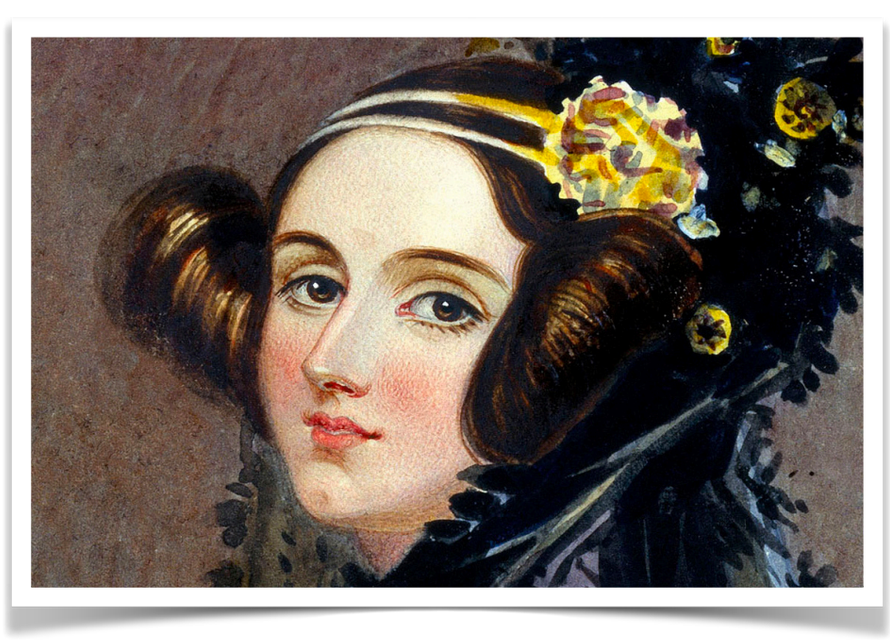


Origins: Analytical Engine (1800s)

- 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

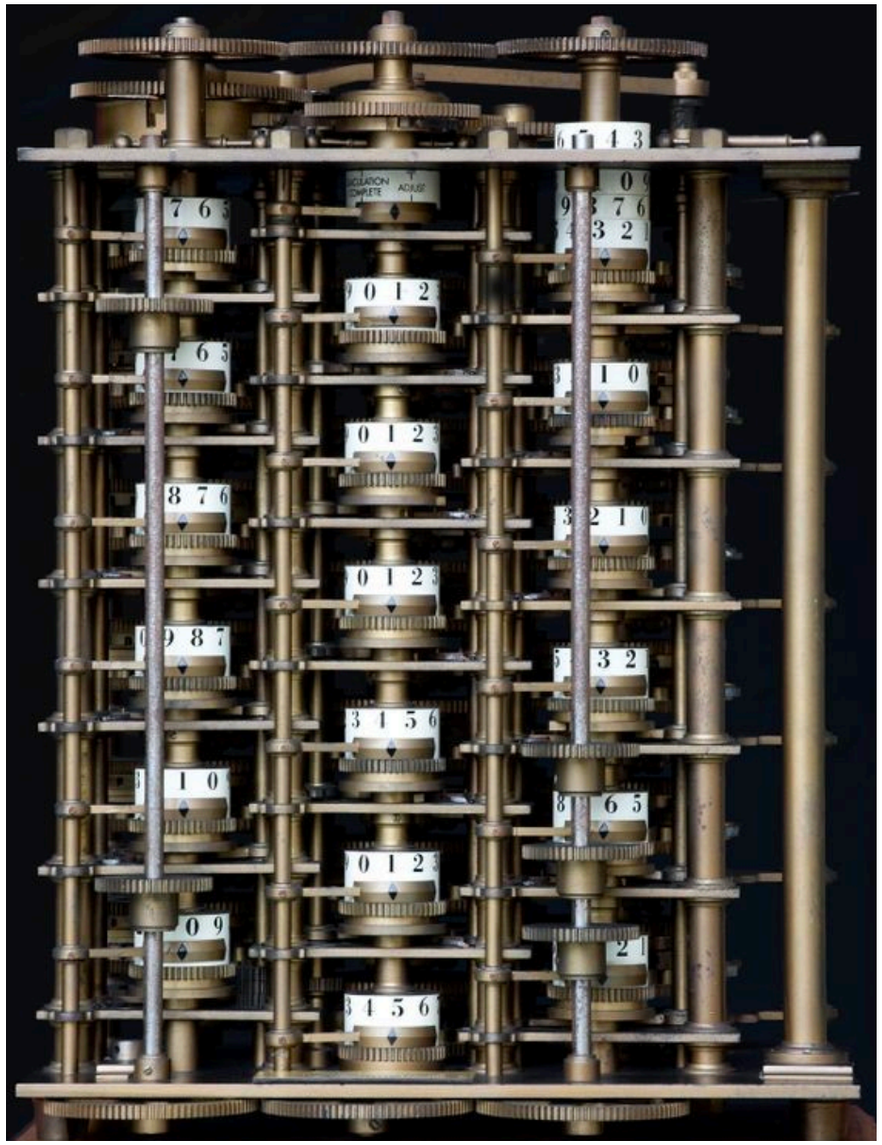
- Print results of polynomials in a table format
  - Problem: labor-intensive, prone to human error
- Key insight:
  - Automate the *method of finite differences*
    - *Decompose task using simple additions*
  - For an degree-n polynomial, input all n differences for first few values of x.
  - *Then*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

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

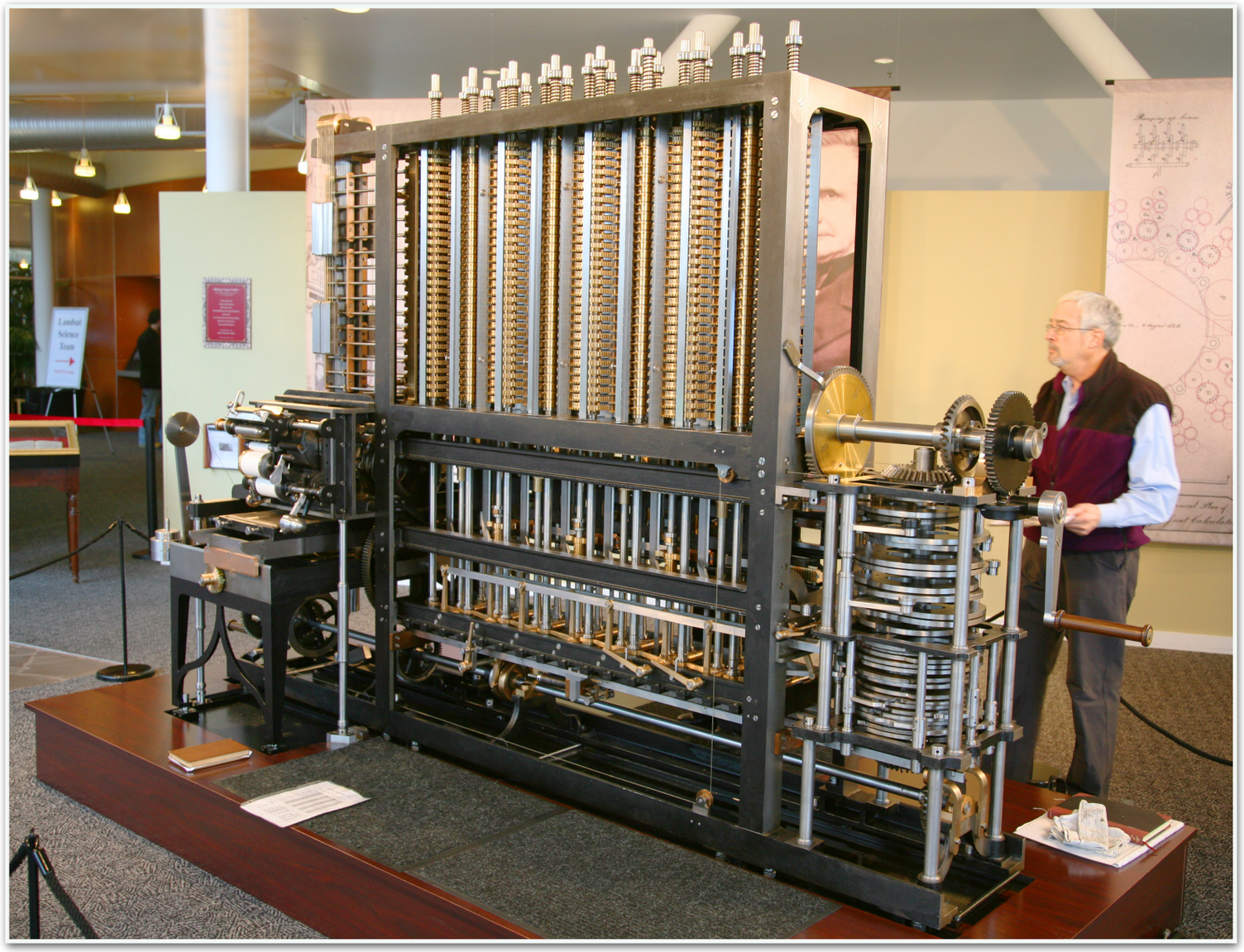
x	f(x)	Δ_1	Δ_2
1	6		
2	12	6	
3	22	10	4
4	36	14	4
5	54	18	4
6	?	...	...

Babbage's Difference Engine

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



"Registers"

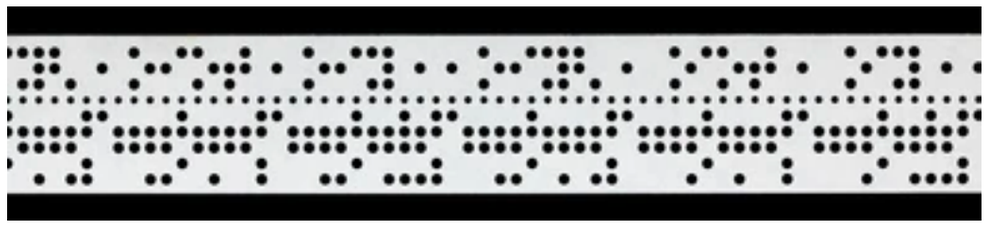


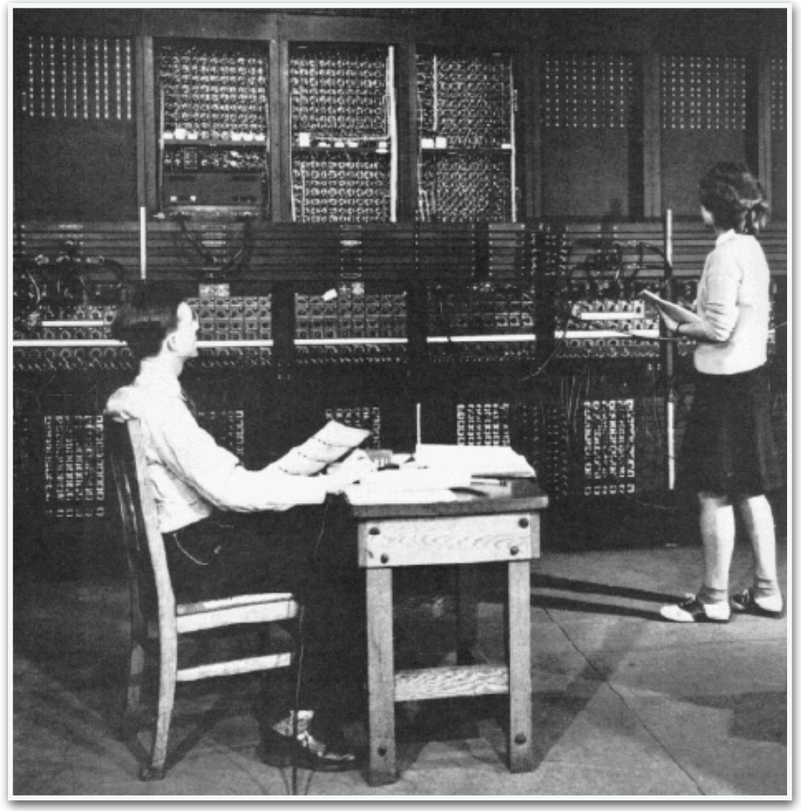
A Replica at the Computer History Museum

1940s: Dawn of Time

- Hardware is *very expensive* and *experimental forms of art*
  - Only universities and Federal Labs had "automatic computers"
  - 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

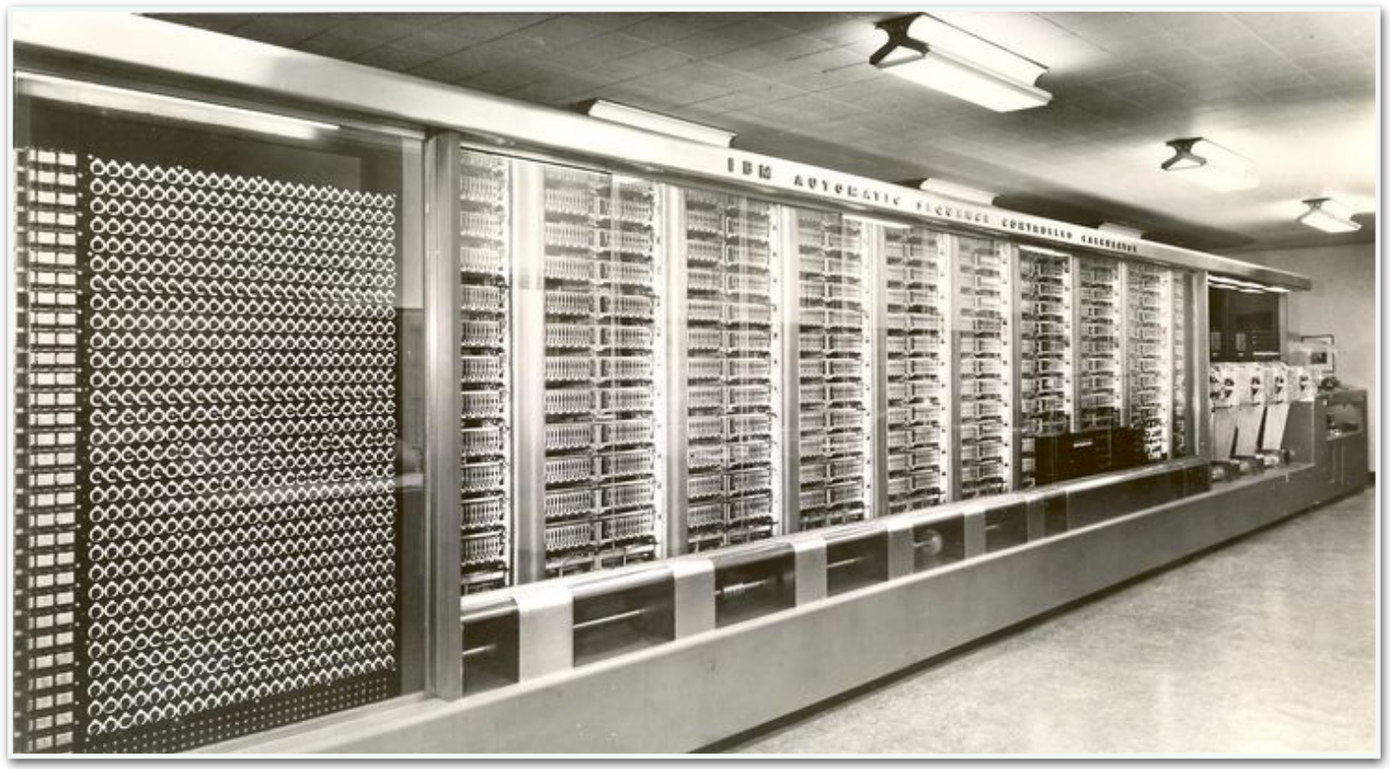




1940s: Harvard IBM Mark I (1944)

- 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**
  - Initially, a Professor of Mathematics @ Vassar College
  - Then, as a US Naval Officer, became the first principle programmer of the Mark I

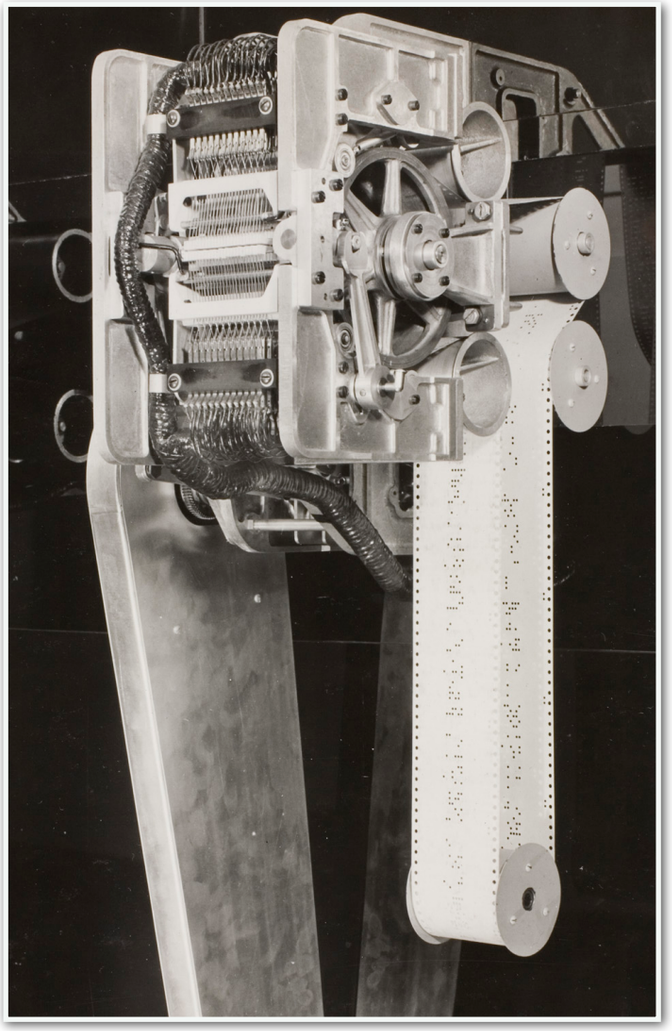
On the Harvard Mark I:

"[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."

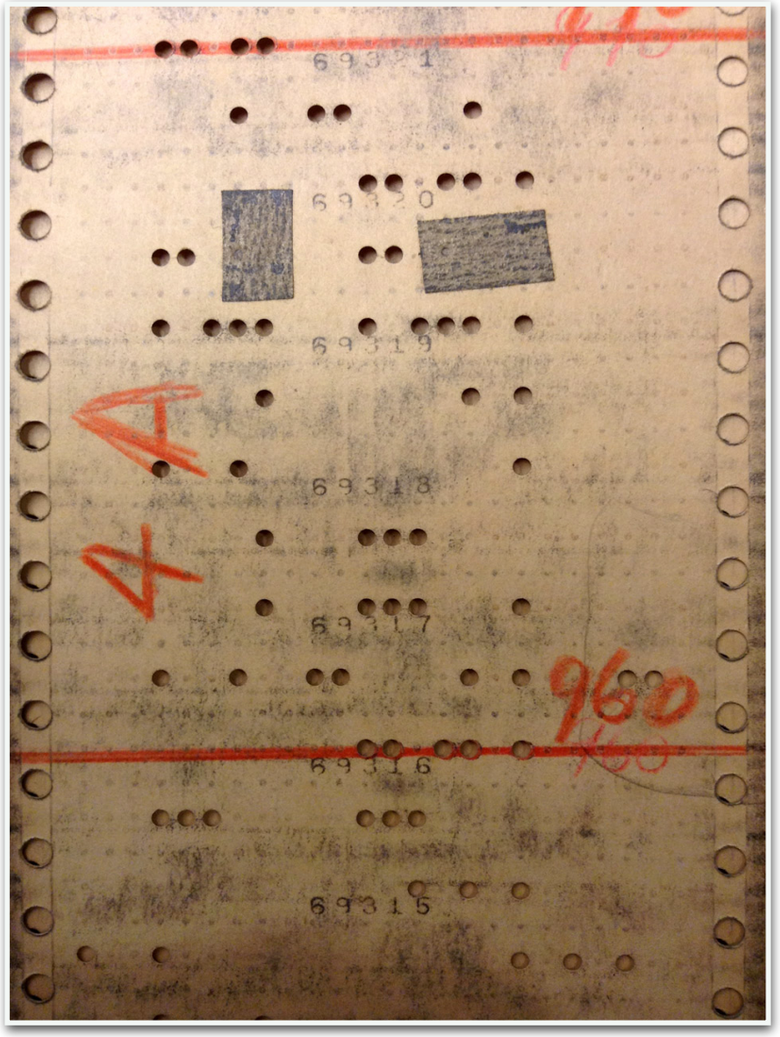
Grace Murray Hopper

Aside: Etymology of Common Terms

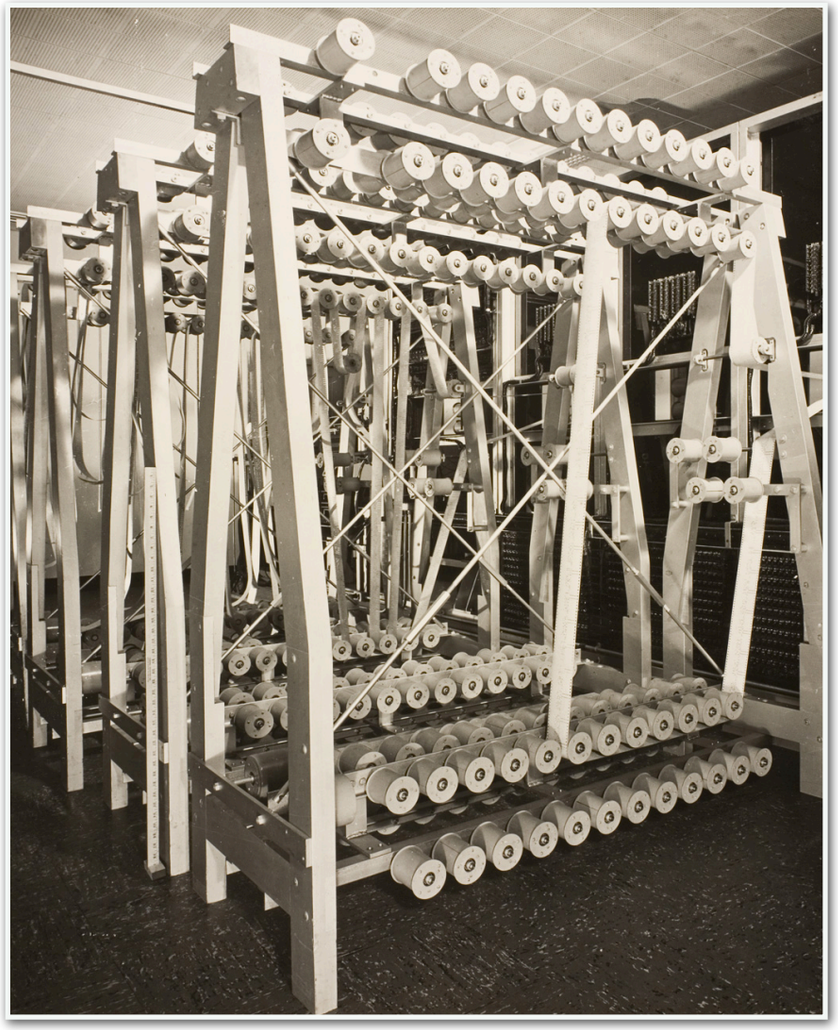
- A lot of common CS terms were coined by Mark I programmers
  - Documentary here: https://chsi.harvard.edu/harvard-ibm-mark-1-video



"Loop"



"Patch"

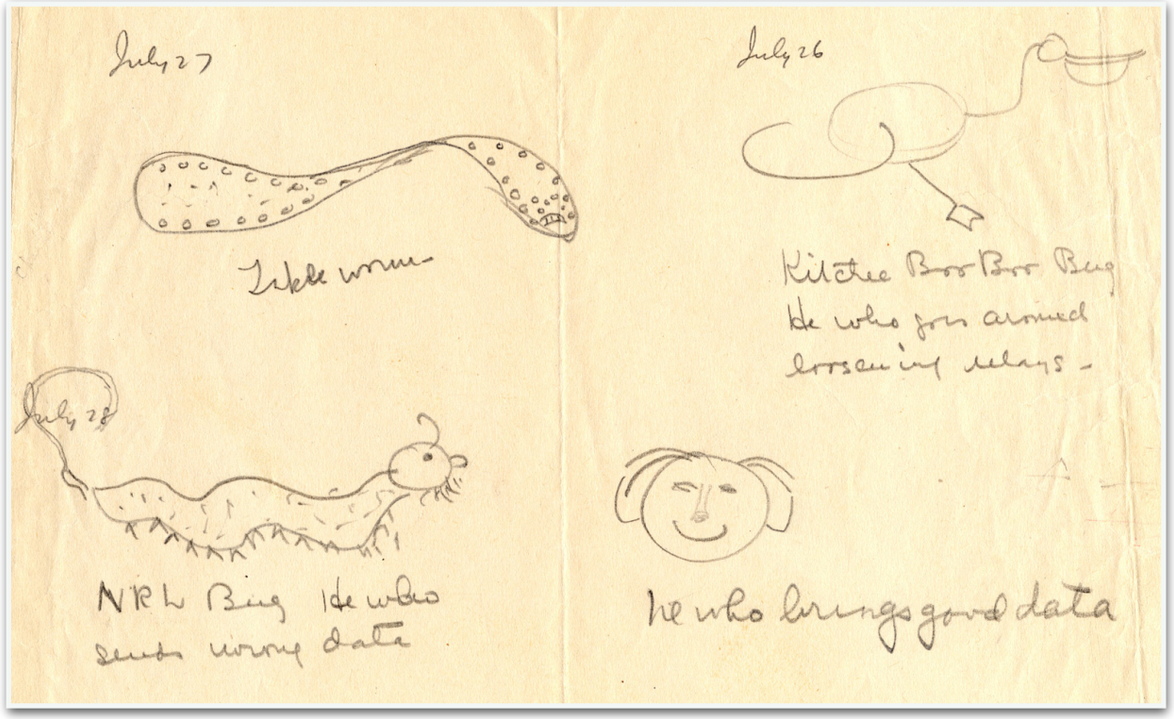


"Library"

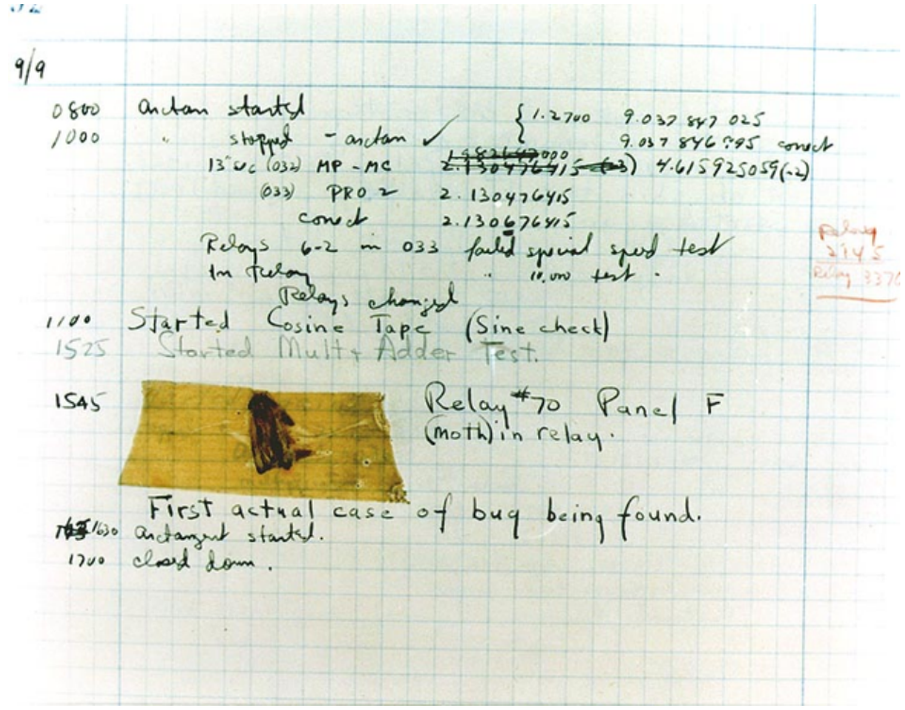
Aside: Etymology of Common Terms

- The term "bug" had been used in the 1800s to describe flaws
  - Moth discovered in Harvard Mark II
    - "First actual case of ***'bug'*** being found." (1947)
    - Mark II was still electromechanical, but had special units for: sqrt, 1/x, ...

Drawings of bugs by Grace Hopper

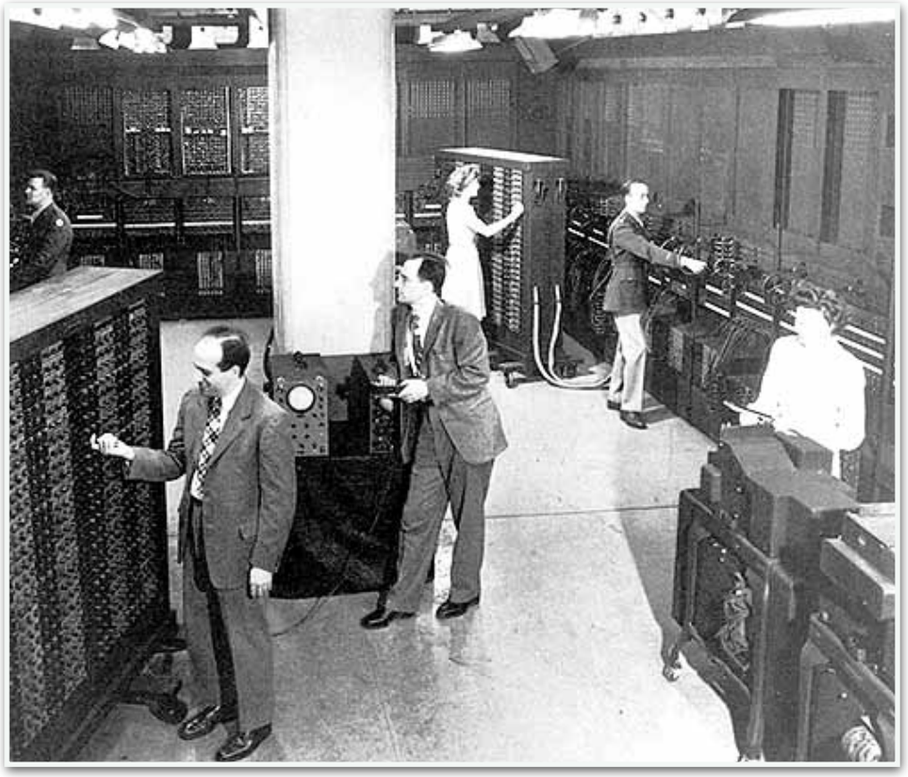


"NRW Bug He who sends wrong data" *"He who brings good data"*



"First actual case of bug being found"

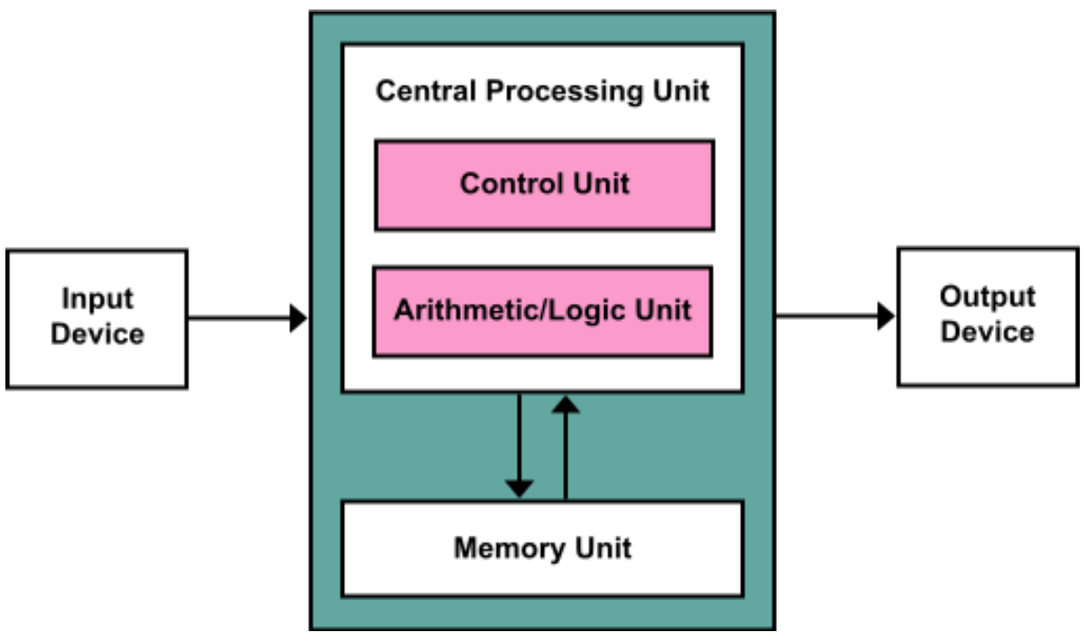
1940s: ENIAC (1946)



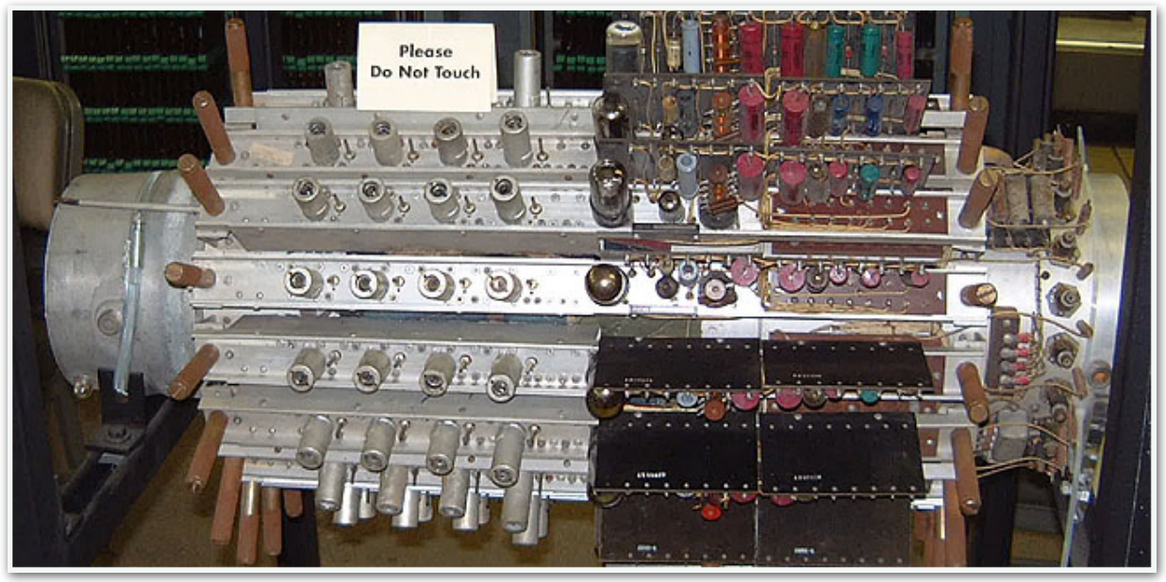
- ENIAC: Electronic Numerical Integrator and Computer
  - First general-purpose *fully electronic* computer
    - Moore School of Engineering @ UPenn
    - J. Presper Eckert and John Mauchly
  - Still no OS
    - More limiting than Mark I: Can't program ahead of time with tape/cards
    - Used switches 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)

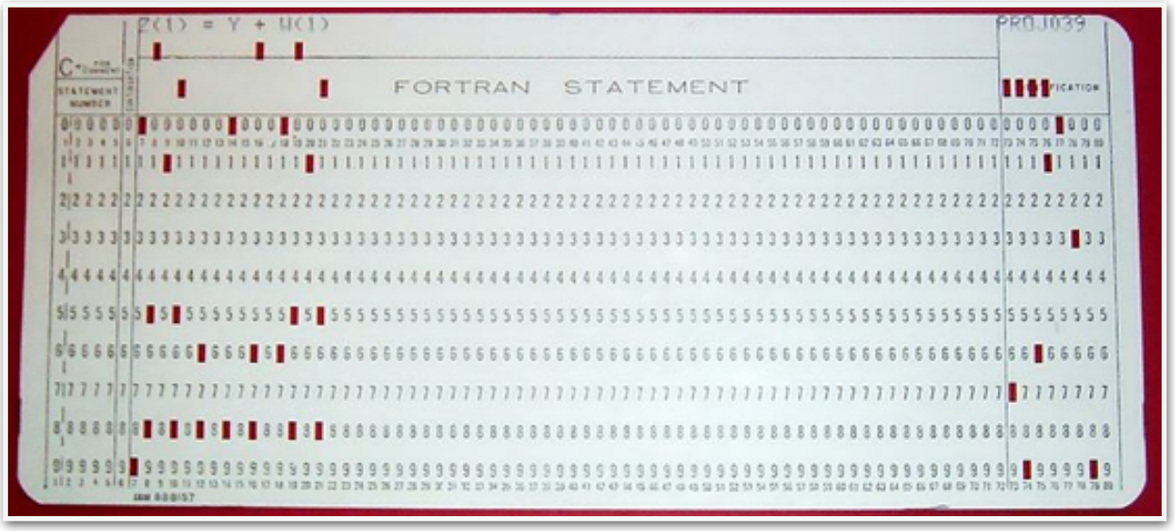


Mercury Delay-Line Storage
(https://www.youtube.com/watch?v=kignGE77I_I)

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

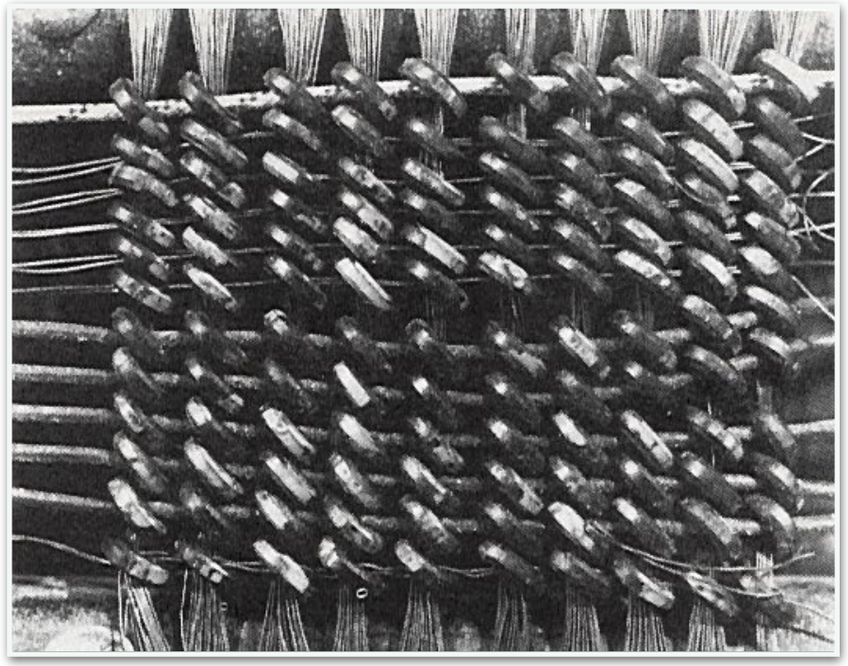
- 1951-1952:
  - Eckert-Mauchly's UNIVAC becomes the first commercially-viable computer!
    - Go see a working UNIVAC @ Living Computer Museum in Seattle! (If it opens again)
- In 1952 Hopper writes the first-ever compiler (A-0) for the UNIVAC.

- 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. I think with any new idea, because people are allergic to change, you have to get out and sell the idea."
>
> *Grace Hopper: Navy Admiral and Computer Pioneer* by Charlene W. Billings. 1989.

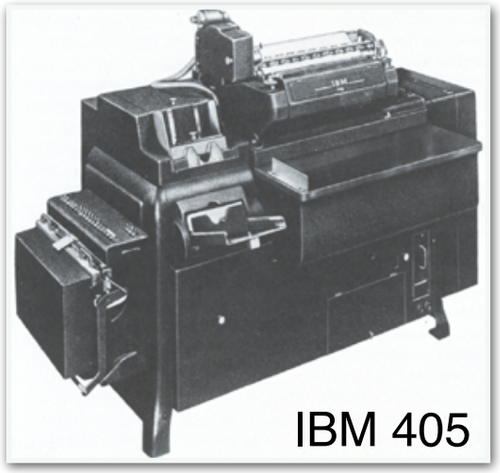
1950's: Solid-State Core Memory

- 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.



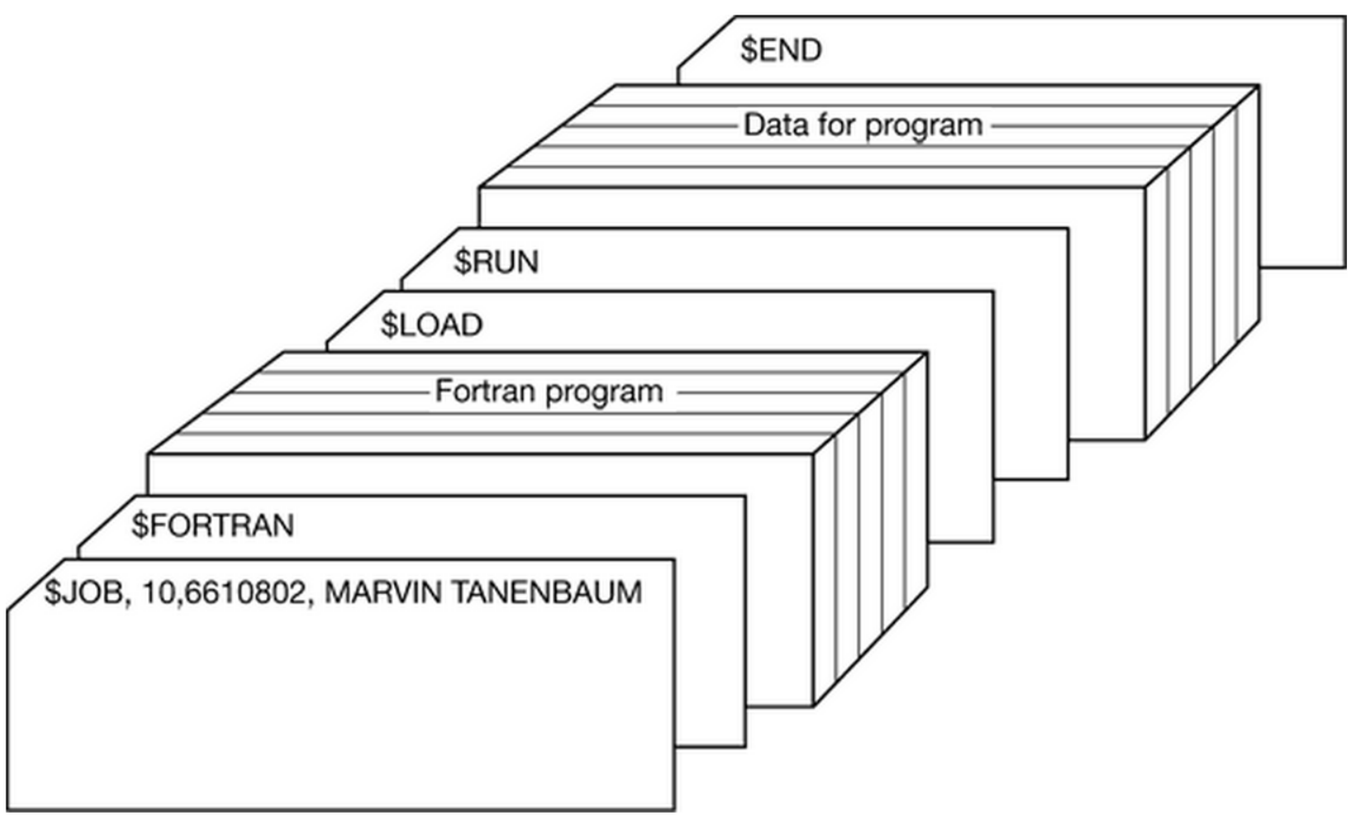
1950's (Batch Processing Model)

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

- Finally, the **First OS** called the *"resident monitor"*
  - Established the *Batch Processing Model*
    - Load many programs to run.
    - Program cards separated by *control cards.*
    - Run one program (job) at a time.

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

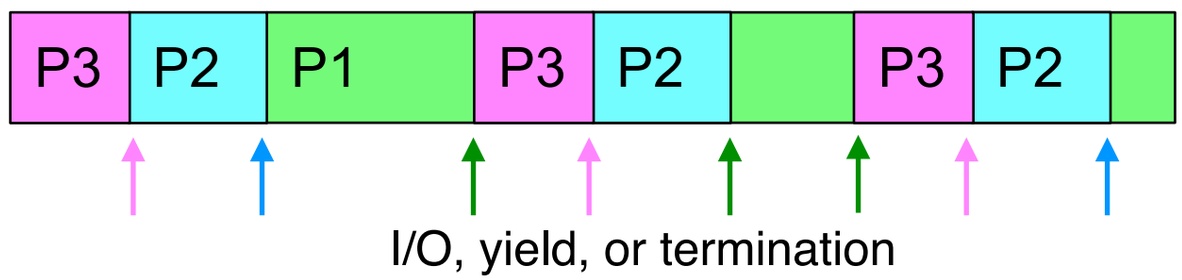
- Computers were adding many **slow** input/output (I/O) devices
  - Time is precious, CPU is expensive, I/O is slow
  - **Breakthrough:** context switching

- *Multiprogramming Model*
  - We want better CPU utilization over batch processing.
  - Make use of stored-programs: Several jobs loaded into memory at once
  - Run a program until it..
    - (1) Completes, (2) yields CPU to another program, or (3) uses I/O device
    - Put the current job back in memory, select a new job to run (*i.e.*, context switch)

1960's: Multiprogramming Is Born

- *Multiprogramming Model*
  - Run a job until it either:
    - Completes, *or* yields the CPU to another program (play nice), or uses device I/O
    - Put the current job back into ready queue, and select a new job to run
      - (This mechanism is called as a *"context switch"*)

When multiprogramming is good

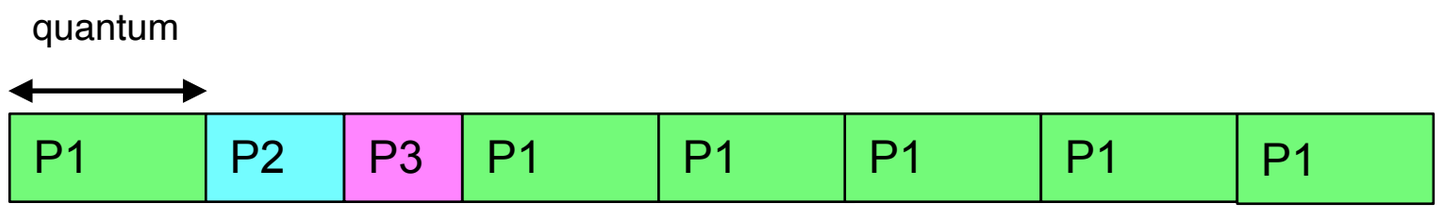


When multiprogramming doesn't help. (same as batch processing)

1960's: The Need for Timesharing

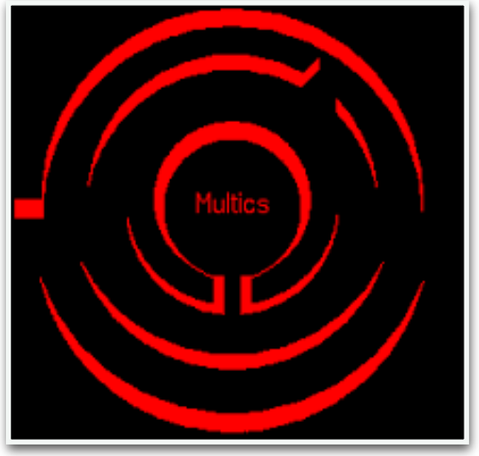
- Multiprogramming gets complicated
  - CPU utilization increased over batch processing, but...
  - Really long jobs (that don't require I/O) can *still* monopolize computer

- *Timesharing Model:* The OS repeatedly cycles through all jobs
  - Timesharing is enhanced multiprogramming.
  - It's multiprogramming plus the following:
    - Each job is given a fixed amount of CPU time (called a time slice, or a quantum)
    - If job doesn't finish within the quantum, then it gets switched out



1960's: Timesharing is Hard

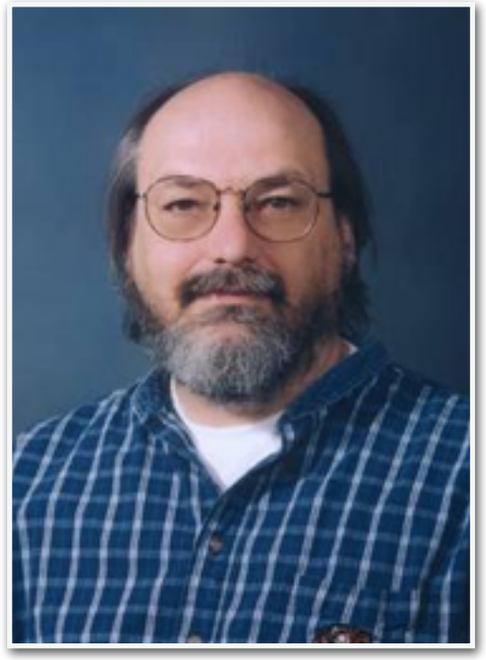
- Complexity gets out of hand!
  - How to ensure *security* between running-jobs?
    - Program A should not be able to access Program B's memory
    - User programs should not be able to access OS's memory



- **Multics:** Multiplexed Information and Computing Service (MIT, 1969)
  - Implementing *timesharing* was the main objective
    - Formalization of a process, address space, virtual memory, shared libraries
  - Multics ultimately failed: *"Too big and slow ... an overdesigned behemoth"*
    - But concepts developed were highly influential and stood the test of time.

1970: The Success of UNIX

- UNIX operating system is developed in 1970
  - ("Unics" is a play on "Multics")
  - Developed in Bell Labs by Ken Thompson (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

History Era 3 (1970's - 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!

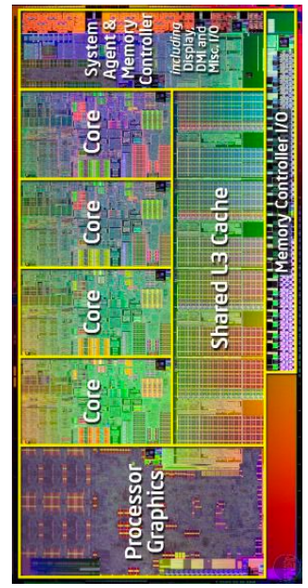
- Advocacy of Personal Computing (PC)
  - OS needed better user-friendliness and GUIs
  - A return to simplicity (due to 1-to-1 usage model)
    - Back to one program running at-a-time: MS-DOS
  - Followed by resurgence of complexity
    - People can multi-task: 1 user, many programs running

Apple Macintosh (1984)

History Era 4 (Now)

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

- Multicore CPUs (~ 2001)
  - Monolithic CPU approaching *"power wall"*
  - Slow them down and add more cores per CPU!
    - New problem: How to exploit parallelism?

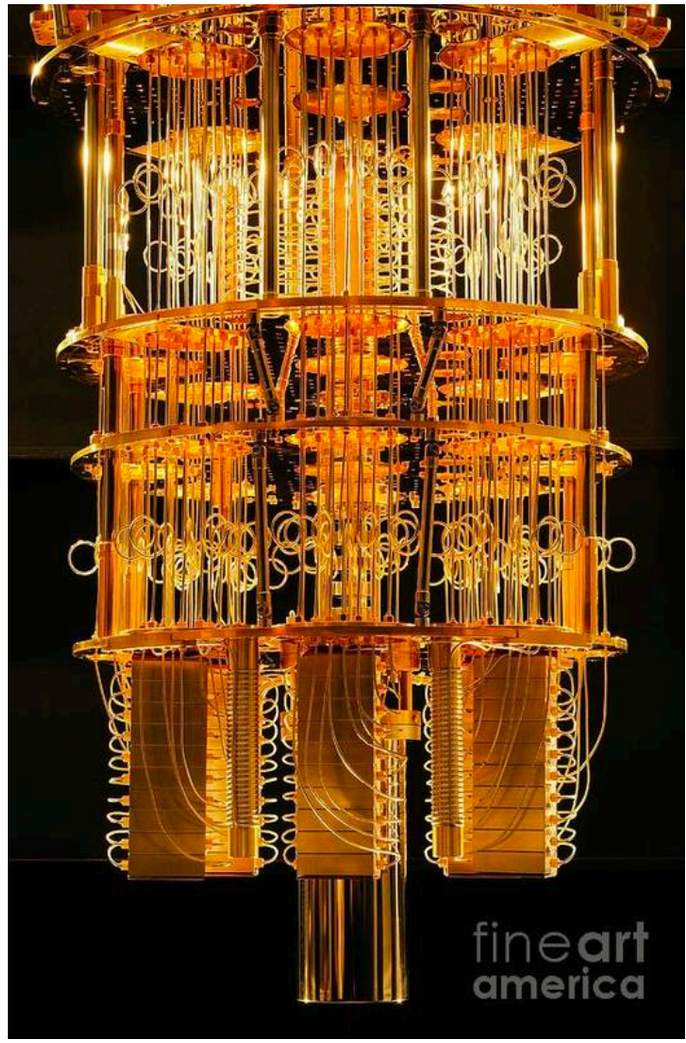


- Mobile computing (~ 2010)
  - Power efficiency now the chief concern
  - Lack of storage
    - OS must have a small "footprint". Back to batch processing again

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
  - 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 1/17

- Welcome to the first day of OS!
  - Reading
    - Chap 1 of Dinosaur Book
    - Chap 1-3 of Dive into Systems (on C)
  - Hwk 1 due 11:59p Friday!
    - Your username is your puget sound login
    - Your password is **qwe123**