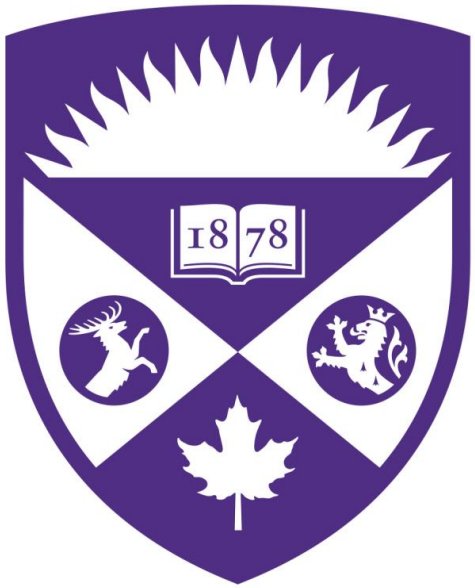


Western

UNIVERSITY · CANADA

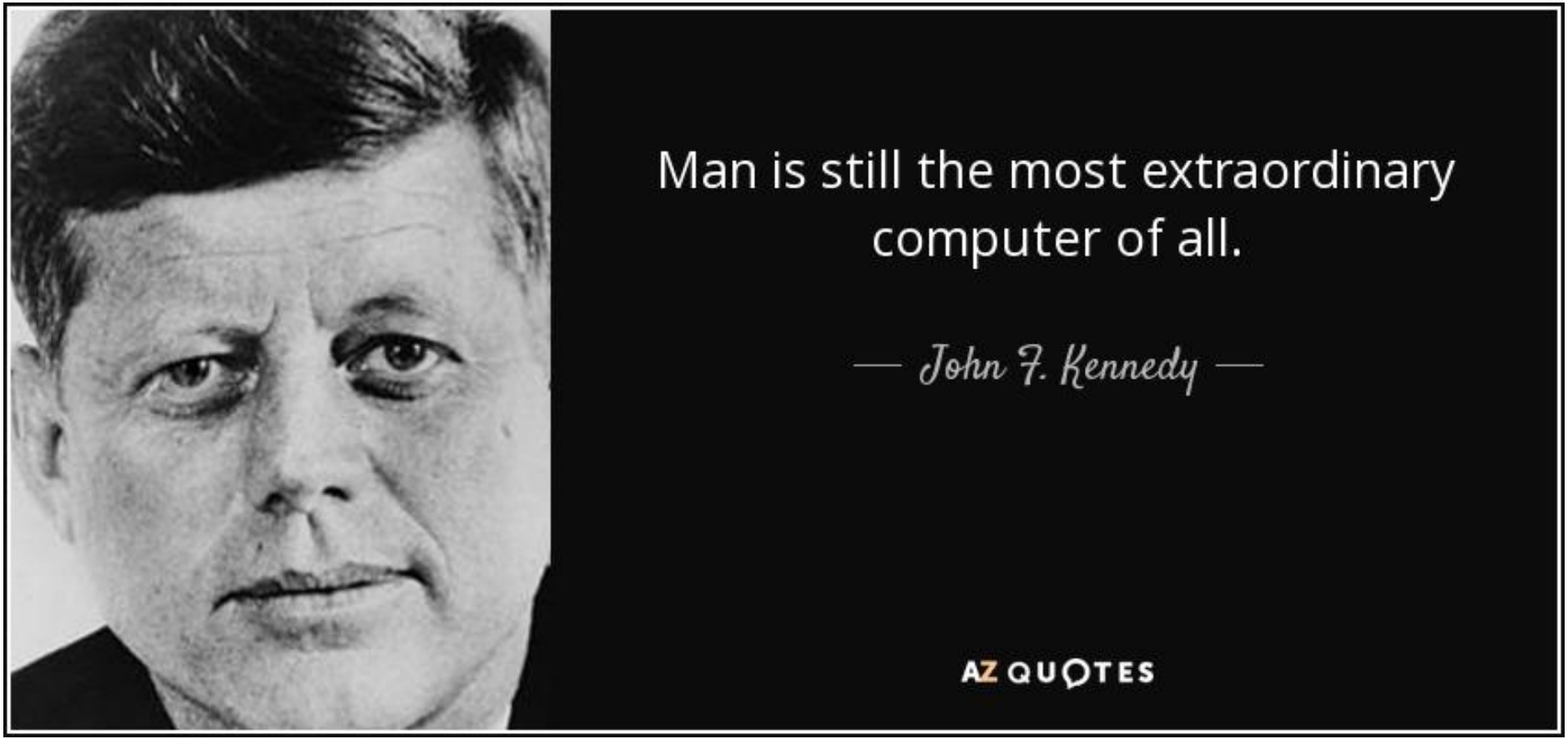
Systems Programming

Winter 2022

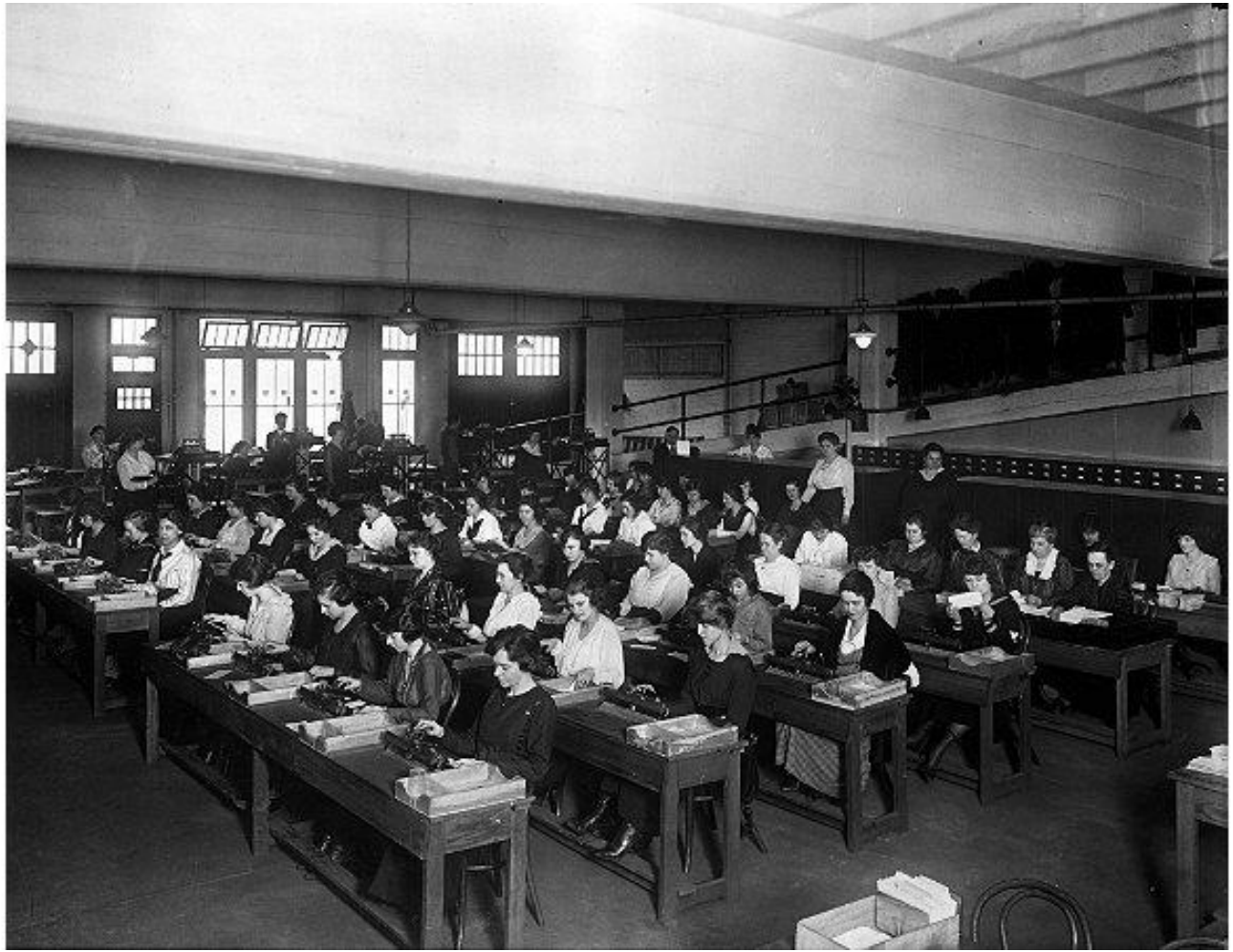
Introduction to Systems Programming

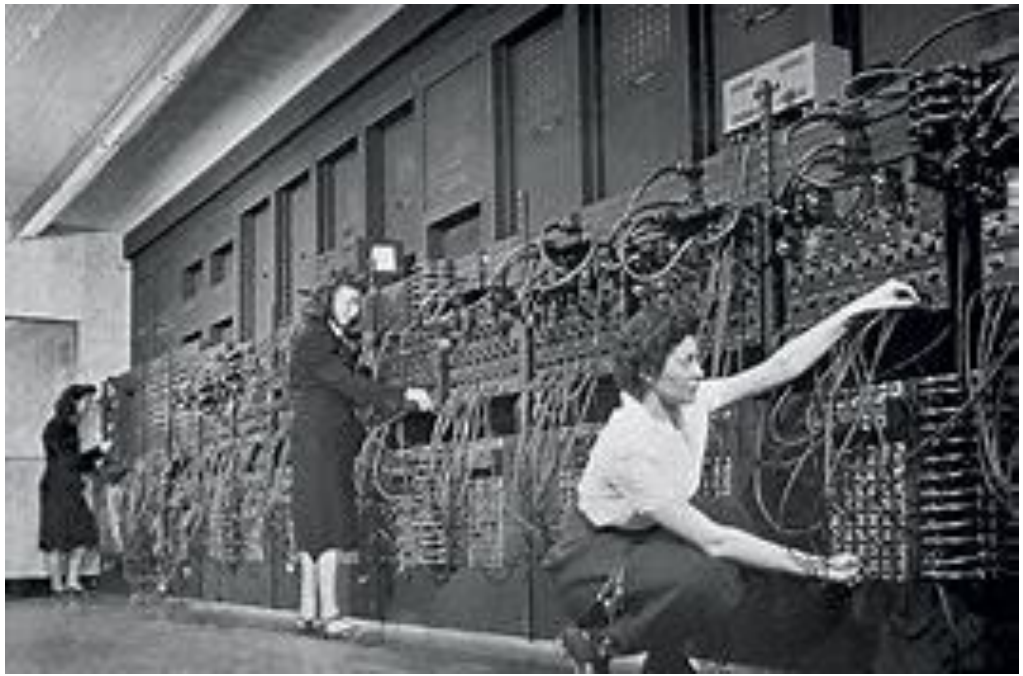
- "Systems"
  - We will use UNIX as our base
- "Programming"
  - We will use C as our base

Systems

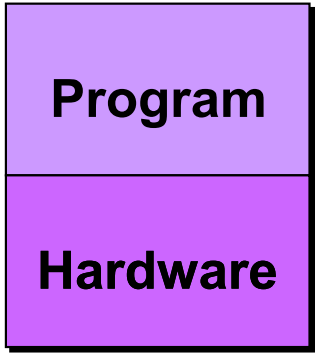

Man is still the most extraordinary computer of all.
— John F. Kennedy —
AZ QUOTES


COMPUTING
DIVISION
COMPUTING
SECTION





- Before the idea of an “operating system”
  - A single program runs on the hardware



- Before the idea of an “operating system”
  - You had the bare hardware
  - Someone had to write the software (program) to run on that hardware
  - This program did EVERYTHING
    - Managing the hardware components
    - Managing your task

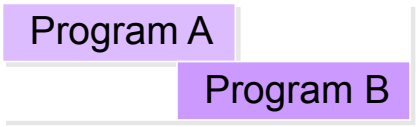
- Before the idea of an “operating system”
  - What if you wanted to do something else with your hardware?
    - You had to write EVERYTHING
  - What if you bought new and improved hardware?
    - You had to re-write EVERYTHING

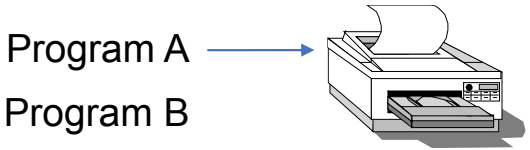
- With an “operating system”
  - Manages the hardware components for you
  - Your program can focus on the task

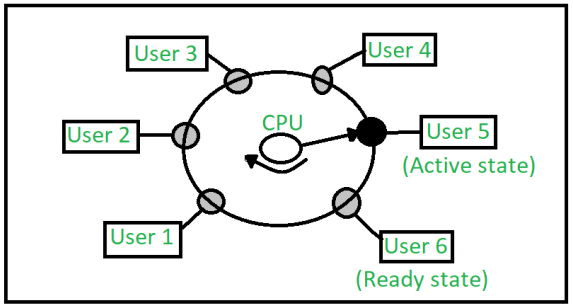
- With an “operating system”
  - What if you wanted to do something else with your hardware?
    - Just write a new program for your operating system
  - What if you bought new and improved hardware?
    - *Assuming your operating system runs on the new hardware*, there’s not much to change

- Other advantages of operating systems
  - Resource sharing
    - Several users can run several programs at once
    - Save time and money

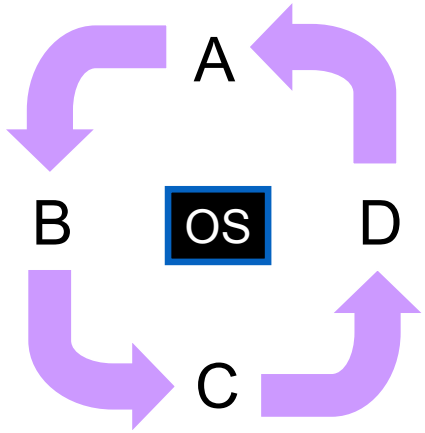
- How hardware resources are shared
  - Memory and disk space
    - Use different regions
  - Printers and other I/O devices
    - Line up and wait (queue)
  - CPU
    - “time sharing”



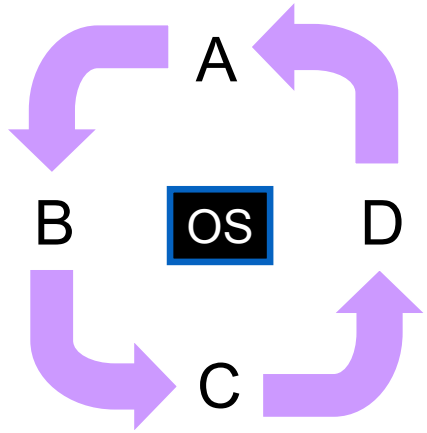




- CPU time sharing
  - Different processes are run for small amounts of time in turns
  - Each task “thinks” it has the whole machine to itself



- CPU time sharing
  - Slightly slower for the individual process but faster in aggregate
    - Processes usually have A LOT of “idle” time
      - Waiting for user input
      - Network or Disk I/O

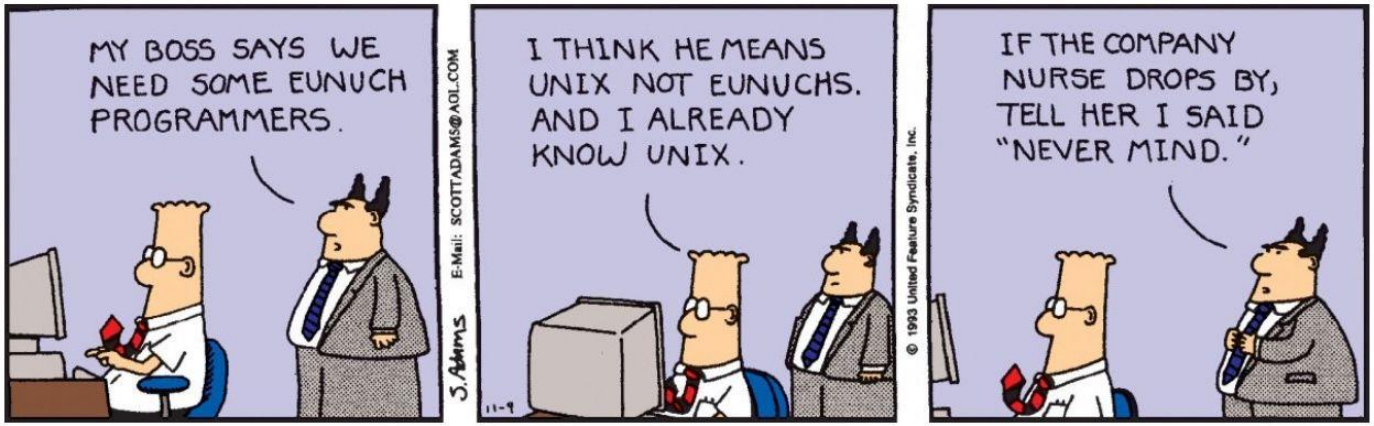


- A generic and incomplete computer family

Timeframe	Original types	Common and modern names	Purpose
1940s and onward	Mainframe computers	Mainframe, “Big Iron”, engineered systems	High level performance for large enterprises
1960s and onward	Supercomputers		Ultra high level performance for research
1960s and onward	Minicomputers	Servers	Medium level performance for small to medium sized enterprises
1980s and onward	Microcomputers	Personal Computer, Notebooks and laptops	Varying level performance for personal use
1980s and onward	Others	“Mobile” (Tablets, phones), MicroPCs (Nettops), Gaming systems	Varying level performance for specific personal use

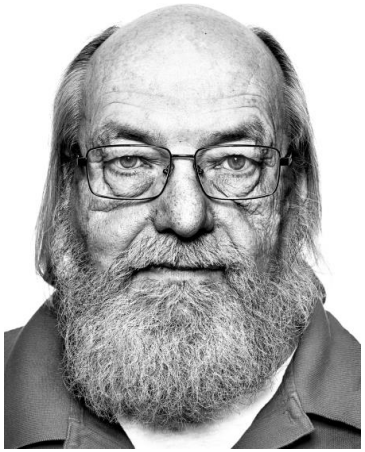
Introduction to Unix

1) Nov. 9, 1993: "Unix programmers"



Courtesy of Scott Adams

- Introduction to Unix
  - 1969 - First developed by Ken Thompson at AT&T Bell Labs
    - First written in assembly (not C)
    - First run on DEC PDP-7 (minicomputer)
      - “Digital Equipment Corporation”
      - “Programmed Data Processor” version 7

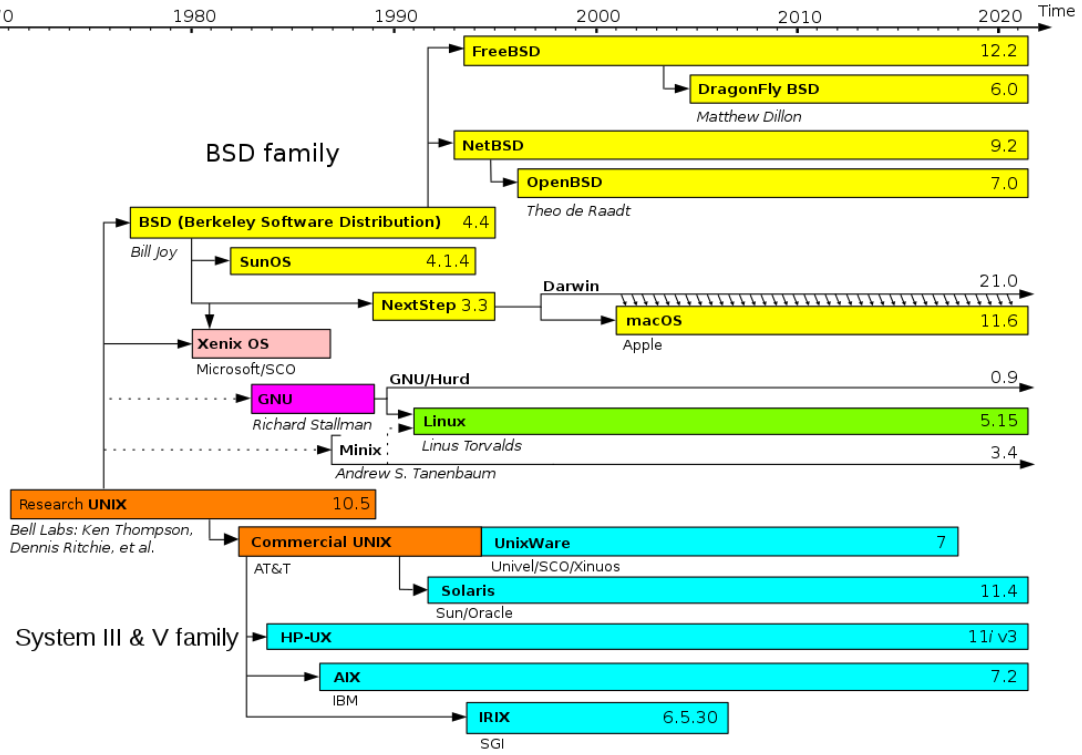


- Introduction to Unix
  - DEC provided an operating system but it was rigid and restrictive
  - Unix was flexible and more “cutting-edge”
  - “An operating system by computer scientists, for computer scientists”
  - When the PDP-11 came out, Unix was rewritten again

- Introduction to Unix
  - 1973 – To prepare for next generation computers, Unix was rewritten but this time in C to improve portability
  - Only the C compiler had to be ported to the new hardware. Unix didn't have to be re-written
  - More on C later

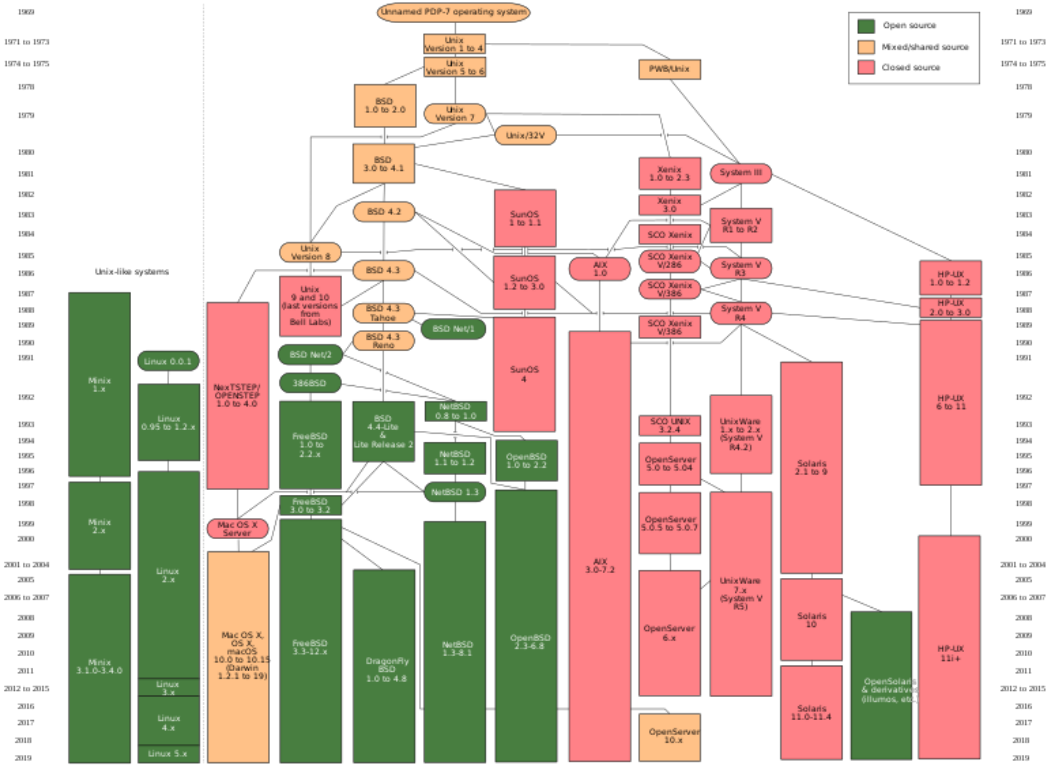
- Introduction to Unix
  - Why is a telephone company writing an operating system?
  - AT&T was a regulated monopoly with a “guaranteed” income so Bell Labs was their “creative” branch
  - As a research project from a public monopoly, profiting off Unix would be problematic, so it was initially freely licensed to academic institutions

- Introduction to Unix
  - The 1970s/1980s and the “Unix wars”



- Introduction to Unix
  - The 1970s/1980s and the “Unix wars”
    - 1988 - The POSIX (Portable Operating System Interface) standard created by the IEEE standardized APIs, shells, utilities, etc.
    - Most Unix derivatives are either completely or mostly POSIX compliant

Introduction to Unix



- Introduction to Unix
  - Linux
    - 1991 – Developed by Linux Torvalds as an Undergraduate student at the University of Helsinki
    - He wanted a version of Unix that ran on his old computer
    - Used on some desktops and extremely popular as a server operating system

- Introduction to Unix
  - Linux
    - Extended to numerous architectures (CPU types). E.g. x86, PowerPC, SPARC, etc.
    - Extensive software tools
    - Free (as in “freedom”) under the GNU General Public License

- Introduction to Unix
  - What about other operating systems like DOS, Windows, macOS, iOS, ChromeOS, Android,?
    - IBM and Microsoft implemented DOS for the new Microcomputer (PC) market. Microsoft continued with Windows
    - Apple adopted and modified BSD for their PCs

- Introduction to Unix
  - What about other operating systems like DOS, Windows, macOS, iOS, ChromeOS, Android,?
    - iOS is also a BSD derivative designed for iPhones and iPads
    - ChromeOS and Android are Linux based systems designed for Chromebooks and smart phones

- Introduction to Unix
  - What about other operating systems like DOS, Windows, macOS, iOS, ChromeOS, Android,?
    - There are now versions of Windows tailored for servers
    - There are now versions of Unix tailored for PCs

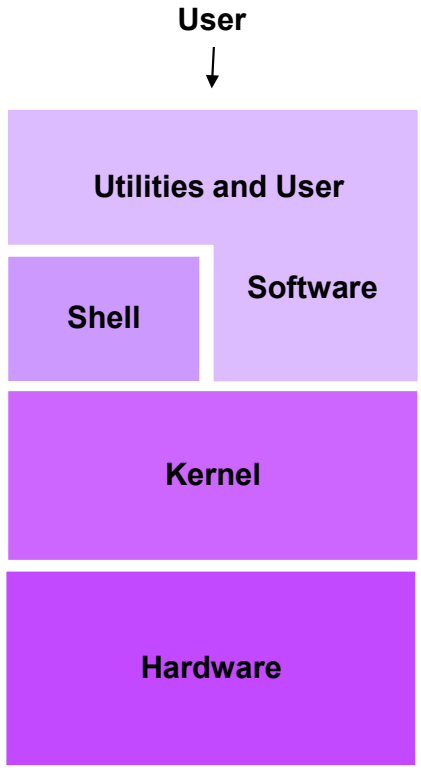
- Introduction to Unix
  - The Unix philosophy
    - "User-helpful" rather than "User-friendly"
    - Protects users from other users, but not necessarily from themselves
    - Gives you the tools or basic building blocks – the rest is up to you

multi-user system.

rm -rf

- Introduction to Unix
  - The Unix philosophy
    - Keep each tool simple and do each task well (atomic)
    - Complex tasks can be accomplished by “gluing” multiple tools together
    - Input and output to the terminal used to be very cumbersome so cutting out the middle steps saved time

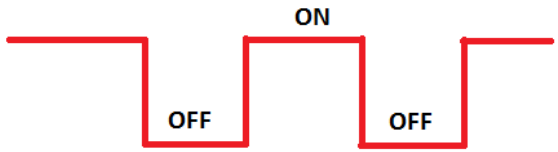
- Introduction to Unix
  - Parts of a Unix operating system
    - Utilities – Standard tools and applications
    - Shell – An interface between users and the kernel. Provides a command-line interpreter
    - Kernel – Manages the processes, resources, hardware



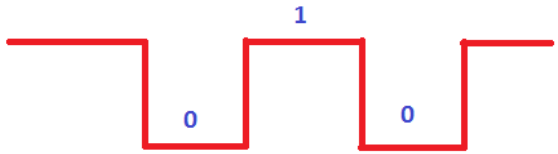
Programming

- When we use the term “computer” in this course
  - Electronic device
  - Aware of only the presence or absence of an electrical charge

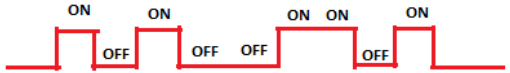
- When we use the term “computer” in this course
  - Electronic device
  - Aware of only the presence or absence of an electrical charge

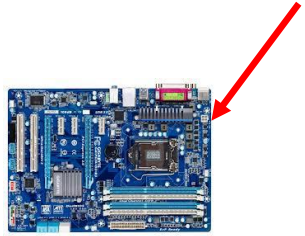


- When we use the term “computer” in this course
  - Electronic device
  - Aware of only the presence or absence of an electrical charge

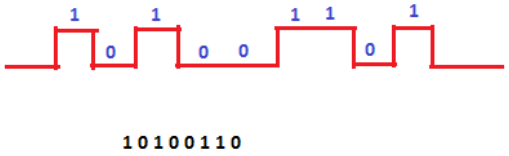


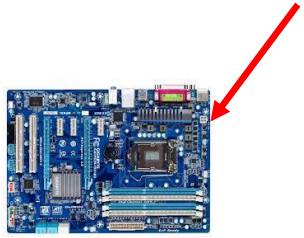
- When we use the term “computer” in this course
  - Electronic device
  - Aware of only the presence or absence of an electrical charge





- When we use the term “computer” in this course
  - Electronic device
  - Aware of only the presence or absence of an electrical charge
  - “Machine Language”





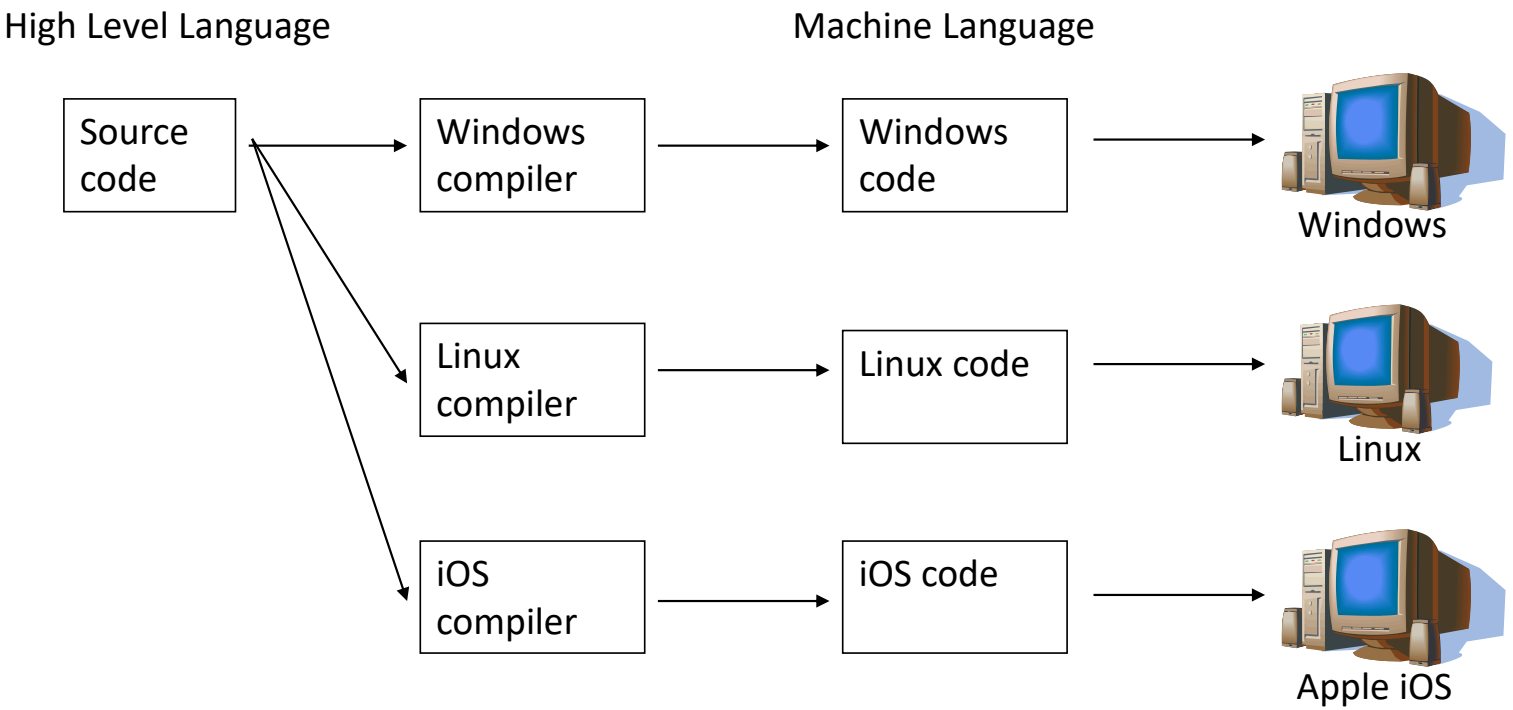
- 3 levels of programming languages
  - 1st level – direct machine code, binary
  - 2nd level – Assembly code
  - 3rd level – Higher level code (e.g. C)

- 3 levels of programming languages

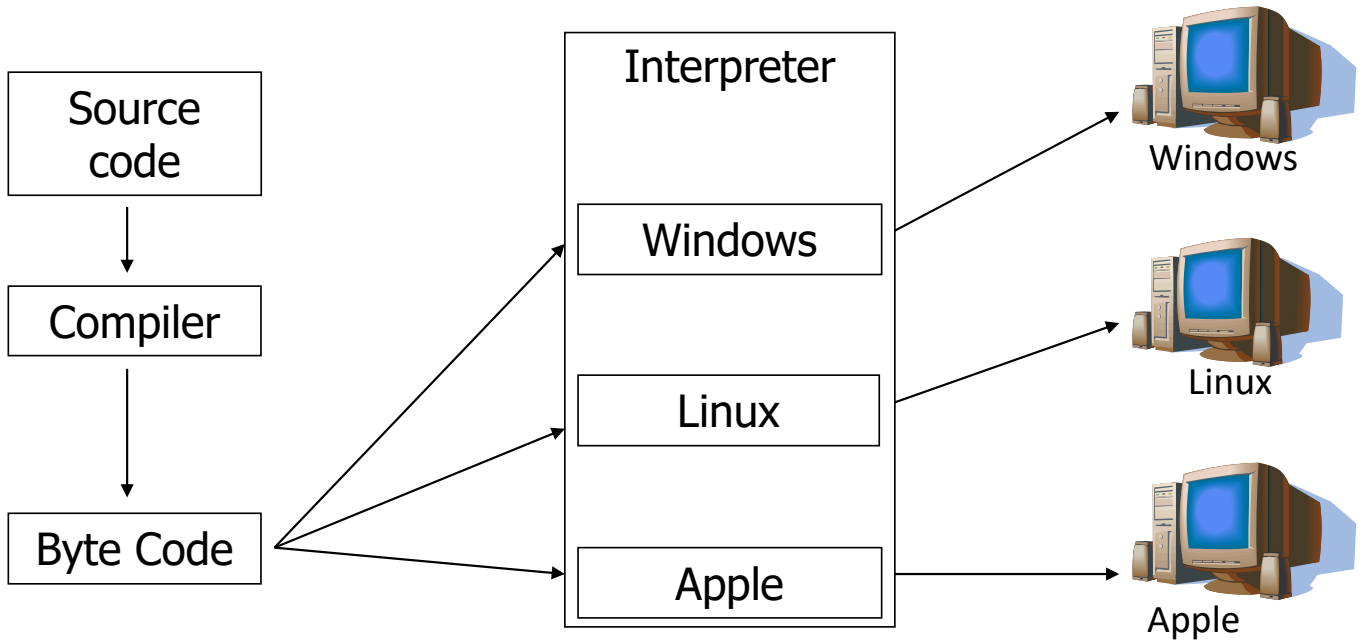
```
int f( int x )          (C Code)
  {
    return ( x*x );
  }
-------------------------------------------------------------------------------------------
Compiled code:

0000000000000000 <f>:
  0: 0011010101101011     push   %rbp
  1: 1101010101000110     mov    %rsp,%rbp
  4: 1001011010100011     mov    %edi,-0x4(%rbp)
  7: 1011110101001000     mov    -0x4(%rbp),%eax     // get x in reg. eax
  a: 0011010110110011     imul   -0x4(%rbp),%eax     // (x*x)
  e: 0101010110001101     leaveq
  f: 11101110110101100    retq
```

- Translating from 3rd level to 2nd level
  - Compiler – A program that transforms code from one format to another
  - Interpreter – A program that translates and executes code

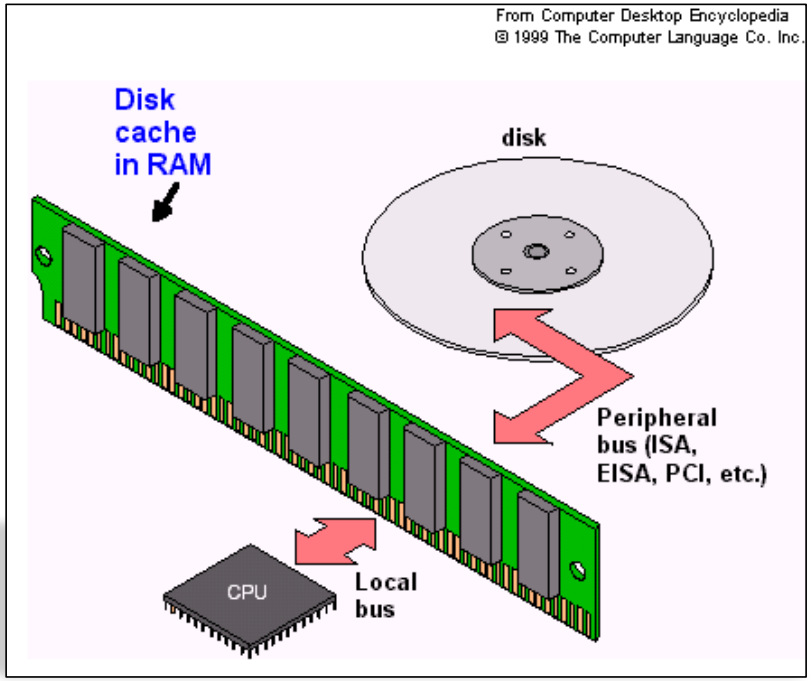
- # Compiling (e.g. C)



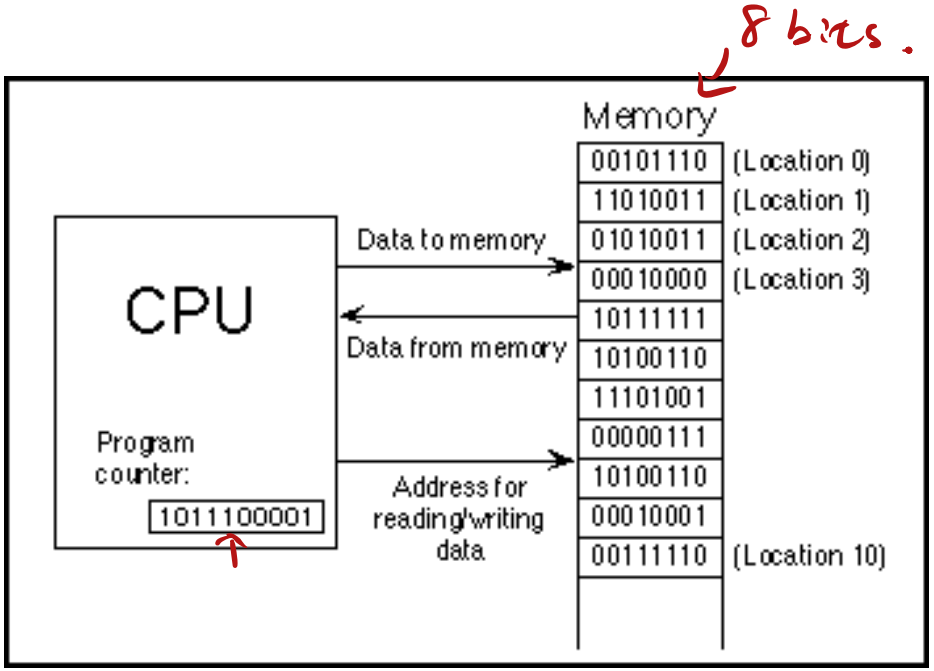
- Interpreting (e.g. Java, Python)



- Storing and using machine code

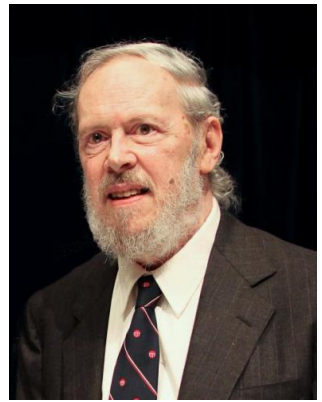


- Storing and using machine code



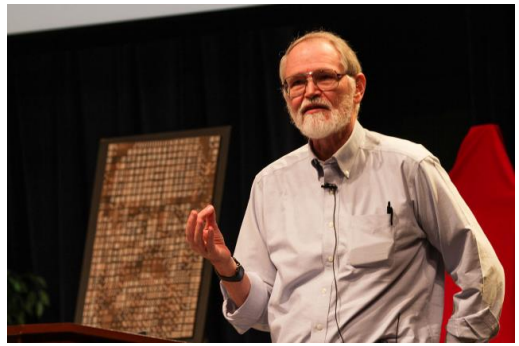
- The C programming language
  - BCPL (Basic Combined Programming Language) was an early high level language with a highly portable compiler
    - Introduced common programming syntax such as { } and //
    - Had only one data type “word”

- The C programming language
  - Developed by Ken Thompson, Dennis Ritchie, and others in the late 1960s
  - Removed some pieces of BCPL and called it B
    - B introduced other programming syntax such as `=` vs `==`, `+=`, `++`, `--` but retained the single “word” type



- The C programming language
  - Thompson further improved B and called it NB (“New B”).
  - The name was later changed to “C”
  - C kept the features of B and included (among other features) multiple types (`int`, `char`, `float`, and later “multi-types” like `struct`)
  - By 1973, C was mature enough to rewrite Unix in C

- Standardization
  - 1972 – Birth. No standardization
  - 1978 – Brian Kernighan and Dennis Ritchie wrote the first manual so the K&R "standard" began (aka C78)
  - 1989/1990 – ANSI C (aka. Standard C or C89)
  - Later standards: C99, C11, C17, C2x (probably 2023)



C99, C11 (Bonus)

- Why C?
  - Intended as a language by programmers for programmers
  - Intended to still be as powerful and efficient as assembler
  - Structured for easy to read and write code
  - Standardized for easy portability

- Why not C?
  - It’s easy to be error-prone. C is very permissive and it can’t protect you from yourself!
  - Compared to newer languages, it can still be difficult to understand
  - If you don’t account for potential modifications, C can be difficult to modify

- Is C still useful in 2022?
  - Yes! TIOBE programming community index maintains the de facto list of the most popular programming languages
  - https://www.tiobe.com/tiobe-index/c/

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

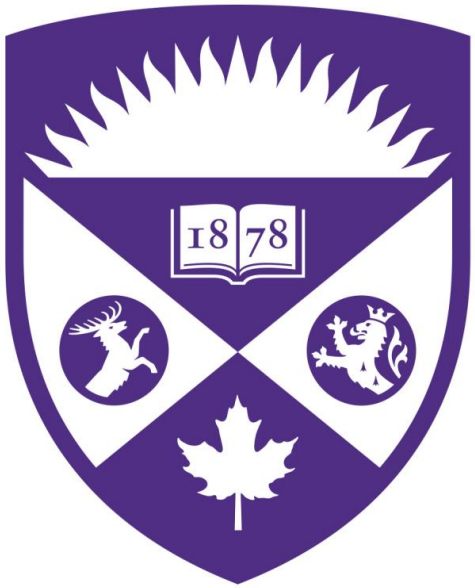
Programming Language	2021	2016	2011	2006	2001	1996	1991	1986
C	1	2	2	2	1	1	1	1
Python	2	5	7	8	20	27	-	-
Java	3	1	1	1	2	15	-	-
C++	4	3	3	3	3	2	2	5
C#	5	4	4	7	11	-	-	-

- Is C still useful in 2022?
  - C saw a sharp uptick thanks to the pandemic
  - https://www.infoworld.com/article/3542028/c-programming-language-rises-with-covid-19.html

- Is C still useful in 2022?
  - Many popular languages are in the “C family”
    - Java, C++, C#, Objective C, PHP, Go, Rust, R, Swift, JavaScript, Perl
  - C’s efficiency makes it popular in data analytics
  - Any non web-based or mobile-based software is probably written in C (e.g. OS, Databases, Networking, Robotics)

- Youtube links
  - AT&T Archives: The UNIX Operating System (https://youtu.be/tc4ROCJYbm0)
  - Computerphile
    - Mainframes and the Unix revolution (https://youtu.be/-rPPqm44xLs)
    - When Unix landed (https://youtu.be/fCDsn7OTNMg)
    - A Unix Special (https://youtu.be/vT_J6xc-Az0)
    - The C Programming language and the K&R standard (https://youtu.be/de2Hsvxaf8M)

- Youtube links
  - Computerphile
    - Why is C so influential (https://youtu.be/ci1PJexnfNE)
  - Fireship
    - C in 100 seconds (https://youtu.be/U3aXWizDbQ4)
    - Bash in 100 seconds (https://youtu.be/I4EWvMFj37g)
    - Linux directories in 100 seconds (https://youtu.be/42iQKuQodW4)
    - The weird history of Linux (https://youtu.be/ShcR4Zfc6Dw)



Western

UNIVERSITY · CANADA