Operating Systems Introduction

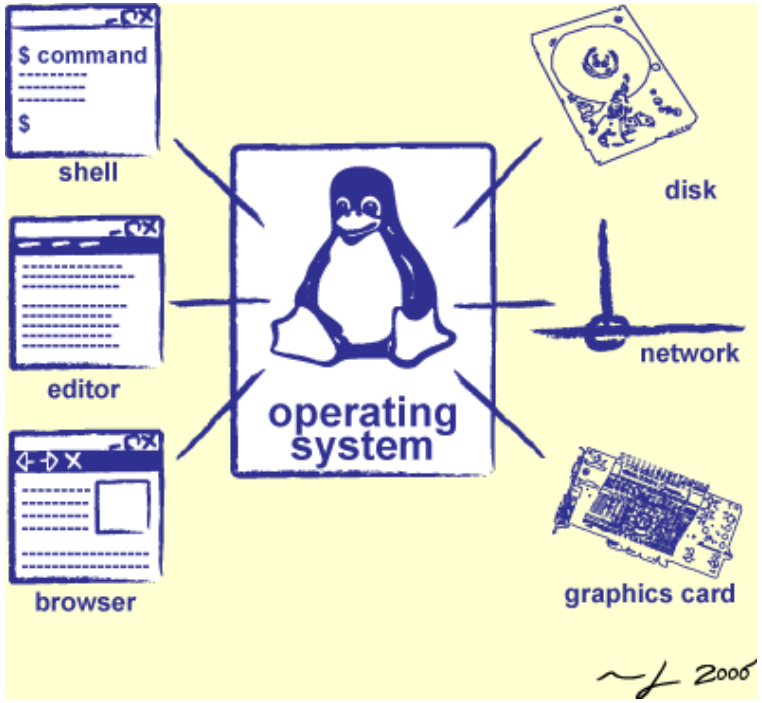
Lecture 1

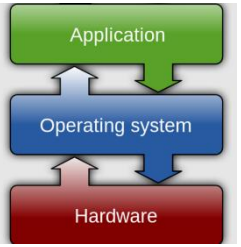
Hartmut Kaiser

https://teaching.hkaiser.org/spring2026/csc4103/

Goals for Today

- Why should you care?
- Why is it hard?
- What is an Operating System?
- Administratrivia & Course Policy

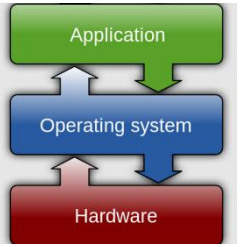




Why should you care?

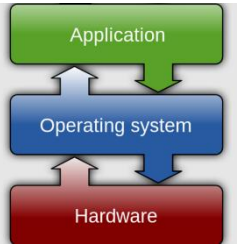
The OS is everywhere

- Every device, from your smartwatch, your smart light bulb, to your mobile phone and laptop runs an operating system
- Every program you will ever write will run on an operating system
- Its performance and execution behavior will depend on the operating system

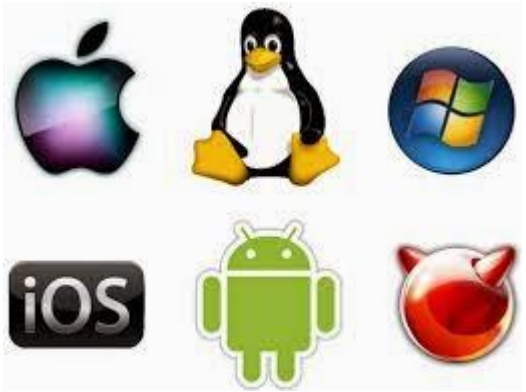


Why is designing an OS hard?

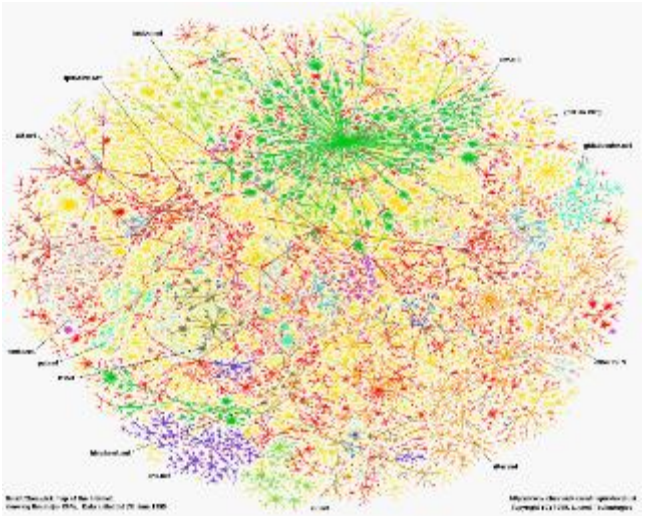
What do these have in common?



Across many devices

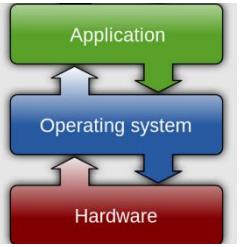


Have an operating system

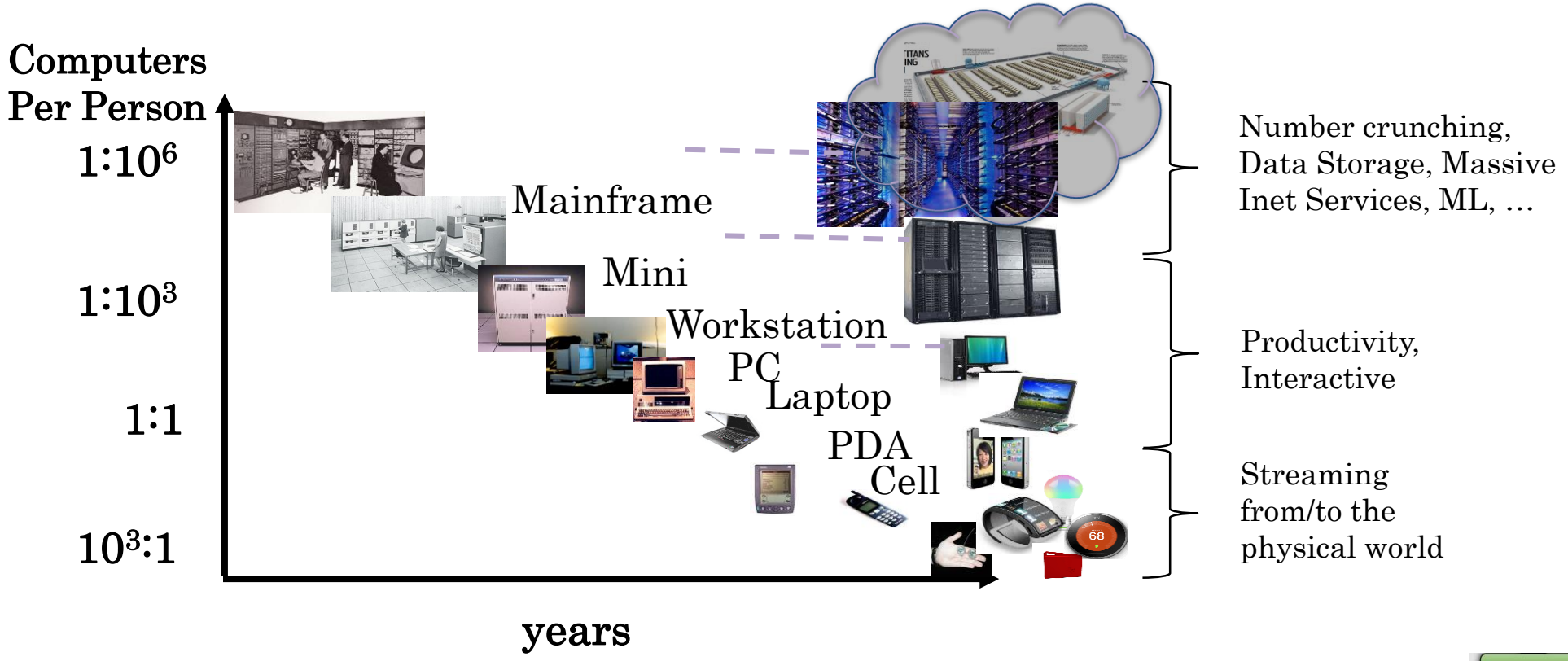


Communicate over the Internet

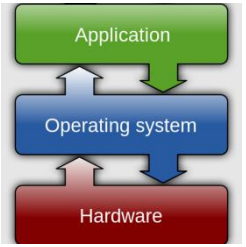
Interface across huge diversity of devices



Bell's Law



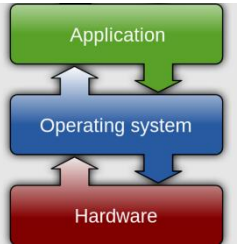
One new device class every 10 years



Across many timescales

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Jeff Dean's Numbers Everyone Should Know



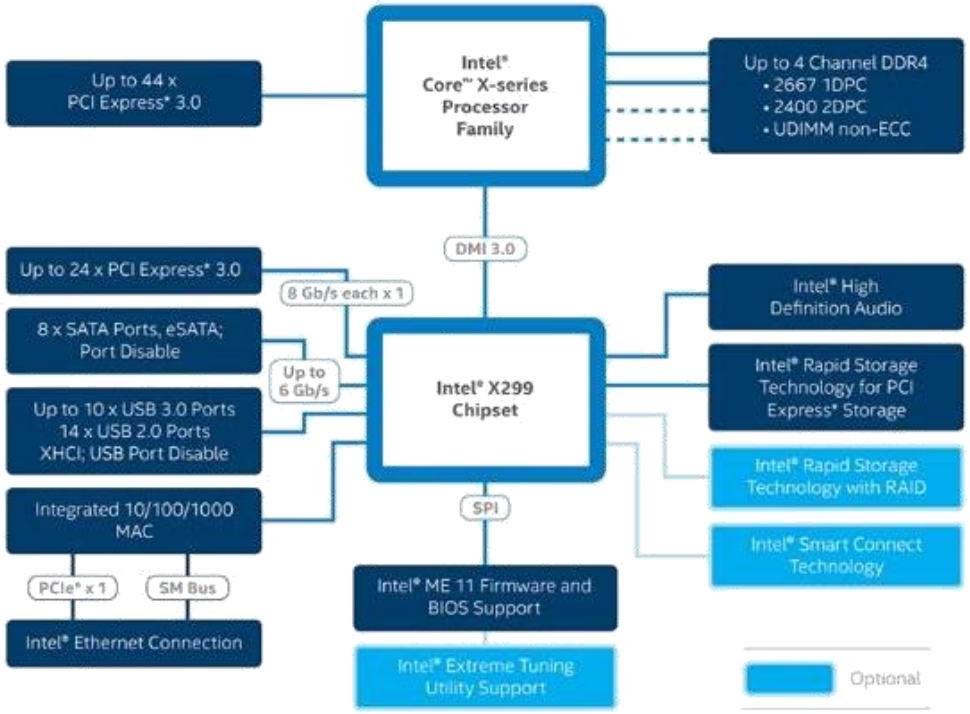
Why so much Complexity?

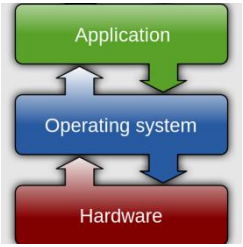
Hardware is becoming smarter!

Better reliability and security

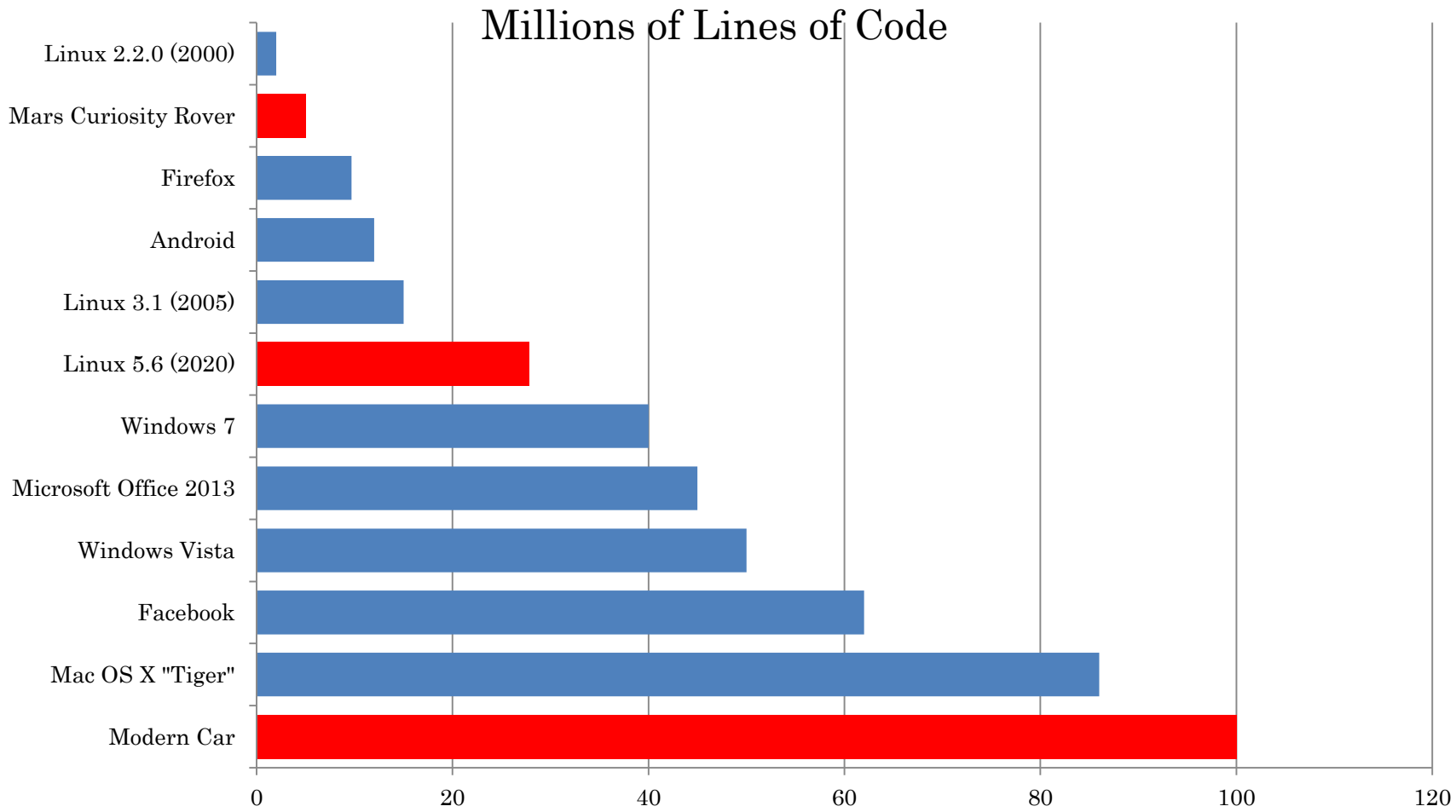
Better performance (more efficient code, more parallel code)

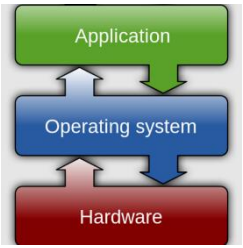
Better energy efficiency





Increasing Software Complexity





What is an Operating System anyways?

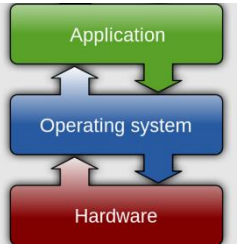
Operating System

Operating

- Manages multiple tasks and users

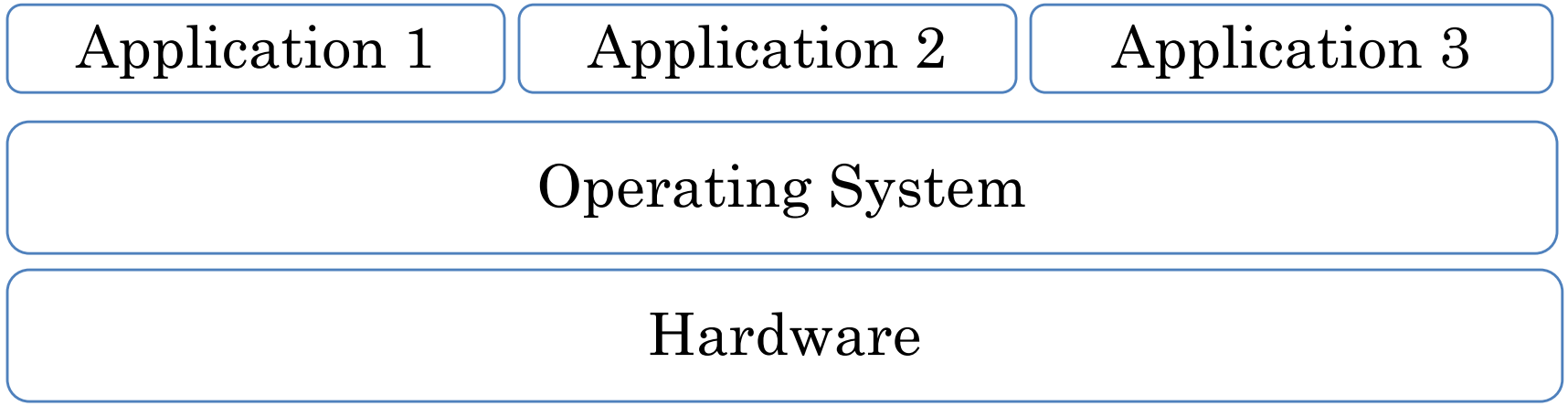
System

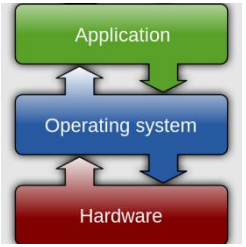
- A set of interconnected components with an expected behavior observed at the interface with its environment



Operating System (v1)

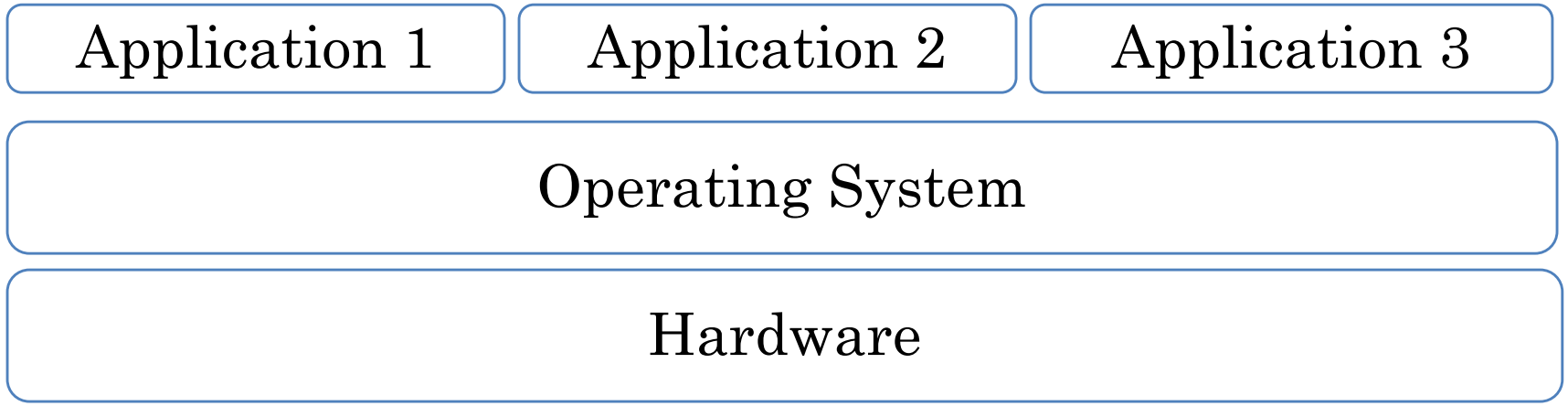
- An operating system is the layer of software that interfaces between (diverse) hardware resources and the (many) applications running on the machine

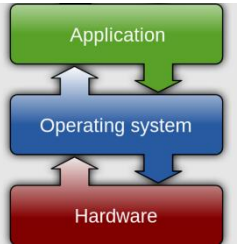




Operating System (v2)

- An operating system implements a virtual machine for the application whose interface is more convenient than the raw hardware interface (convenient = security, reliability, portability)

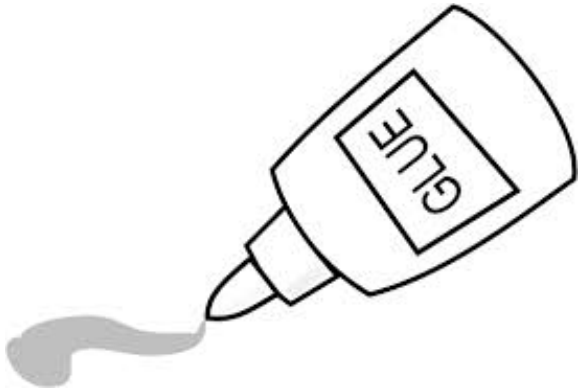




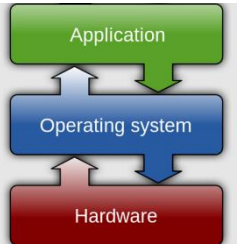
Three main Hats

Referee
Manage protection, isolation, and sharing of resources

Illusionist
Provide clean, easy-to-use abstractions of physical resources

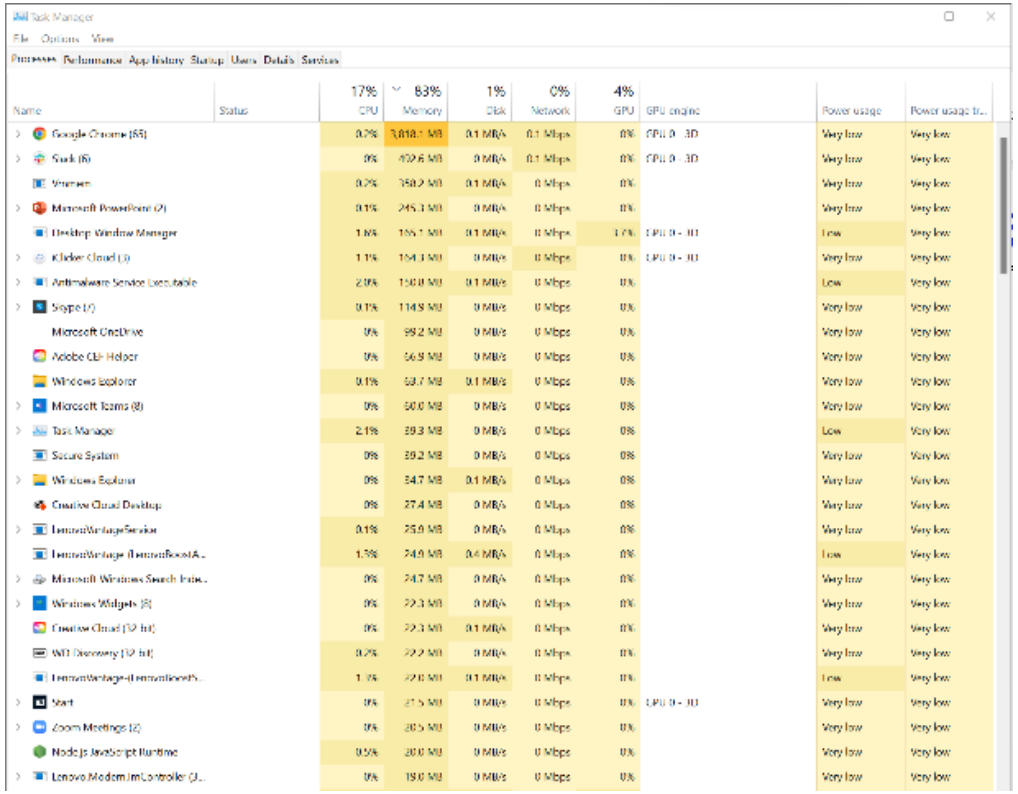


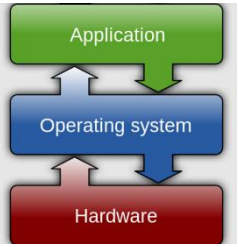
Glue
Provides a set of common services



OS as a Referee

Allow multiple (untrusted) applications to run concurrently





OS as a Referee

Fault Isolation

Isolate programs from each other

Isolate OS from other programs

Process

Dual Mode Execution

Resource Sharing

How to choose which task to run next?

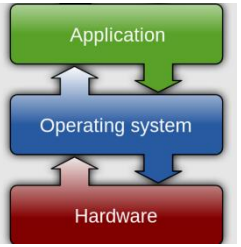
How to split physical resources?

Scheduling

Communication

How can OS support communication to share results?

Pipes/Sockets



What does this Program do?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  char *str = argv[1];
  while (1) {
     printf("%s\n", str);
  }
  return 0;
}
```

```
laptop> gcc -o cpu cpu.c -Wall

laptop> ./cpu A
A
A
A
A
…
```

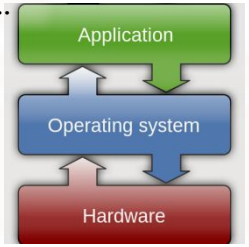
```
laptop> ./cpu A & ./cpu B & ./cpu C
```

a)
```
A
A
A
A
…
```

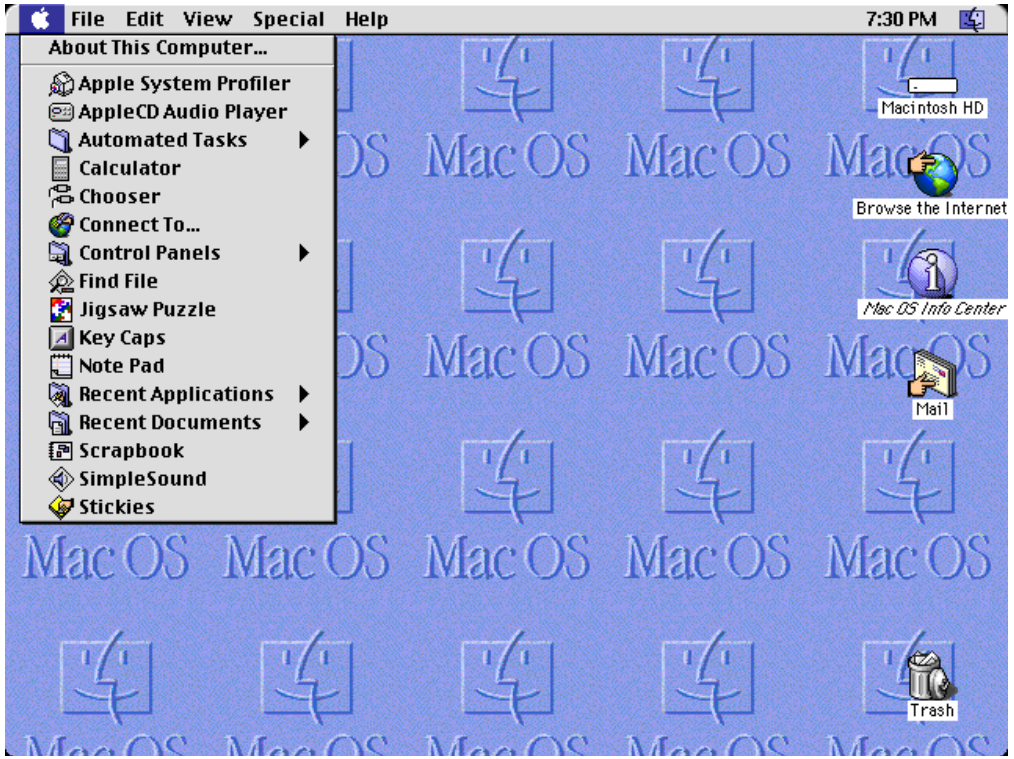
b)
```
A
B
C
A
B
C
…
```

c)
```
A
B
B
A
C
B
C
…
```

```
laptop> ./cpu ; ./cpu B
Segmentation Fault
B
…
```



Refereeing is hard!

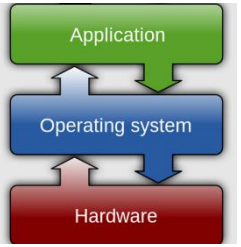


Mac V8 (1997)

OS cannot force program to give up control!

```
very-old-laptop> ./cpu A & ./cpu B & ./cpu C
```

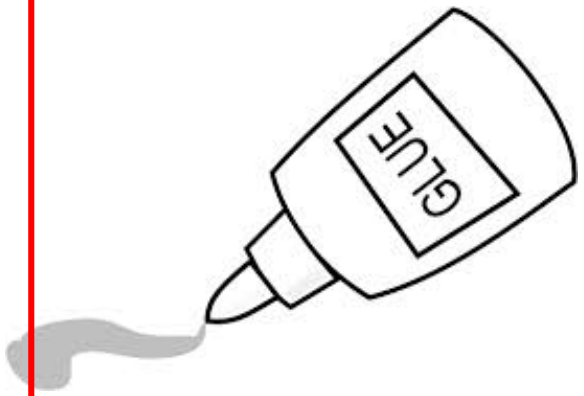
```
A
A
A
A
A
A
A
...
```



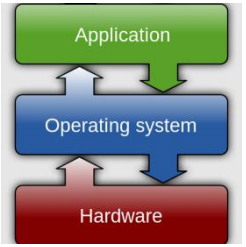
Three main Hats

Referee
Manage protection, isolation, and sharing of resources

Illusionist
Provide clean, easy-to-use abstractions of physical resources



Glue
Provides a set of common services



OS as Illusionist

- Mask the restrictions inherent in computer hardware through virtualization

All alone

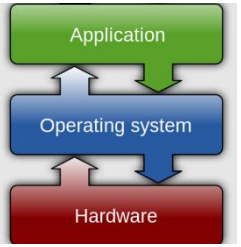
Provide abstraction that application has exclusive use of resources

All powerful

Provide abstraction that hardware resources are infinite

All expressive

Provide abstraction of hardware capabilities that are not physically present



What does this Program do?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
  int *p = malloc(sizeof(int));
  printf("(%d) p: %p\n", getpid(), p);
  *p = 0;
  while (1) {
    *p = *p + 1;
    printf("(%d) p: %d\n", getpid(), *p);
  }
  return 0;
}
```

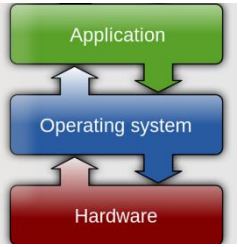
```
laptop> gcc -o memory memory.c -Wall
laptop> ./memory
(120) p: 0x200000
(120) p: 1
(120) p: 2
(120) p: 3
(120) p: 4

laptop> ./memory & ./memory
(120) p: 0x200000
(254) p: 0x200000
```

a)

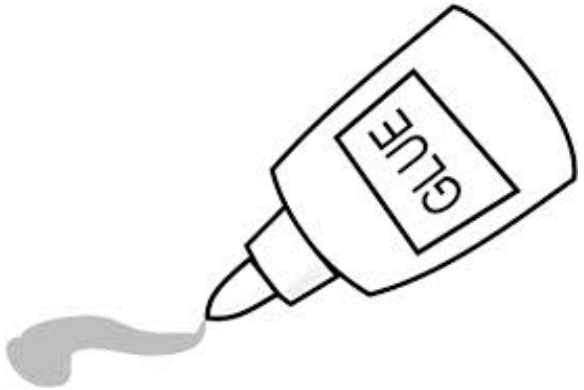
```
(120) p: 1
(120) p: 2
(254) p: 1
(254) p: 2
(254) p: 3
(120) p: 3
...
```

b)

```
(120) p: 1
(254) p: 1
(120) p: 2
(254) p: 2
(120) p: 3
(254) p: 3
...
```



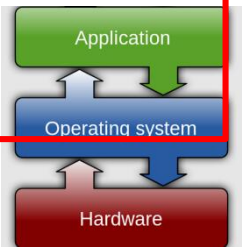
Three main Hats



Referee
Manage protection, isolation, and sharing of resources

Illusionist
Provide clean, easy-to-use abstractions of physical resources

Glue
Provides a set of common services



OS as Glue

- Provide set of common, standard services to applications to simplify and regularize their design

Make sharing easier

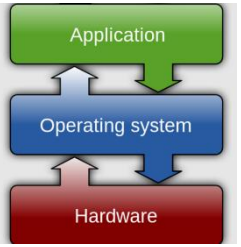
Simpler if all assume same basic primitives

Maximize reuse

Avoid re-implementing functionality from scratch.

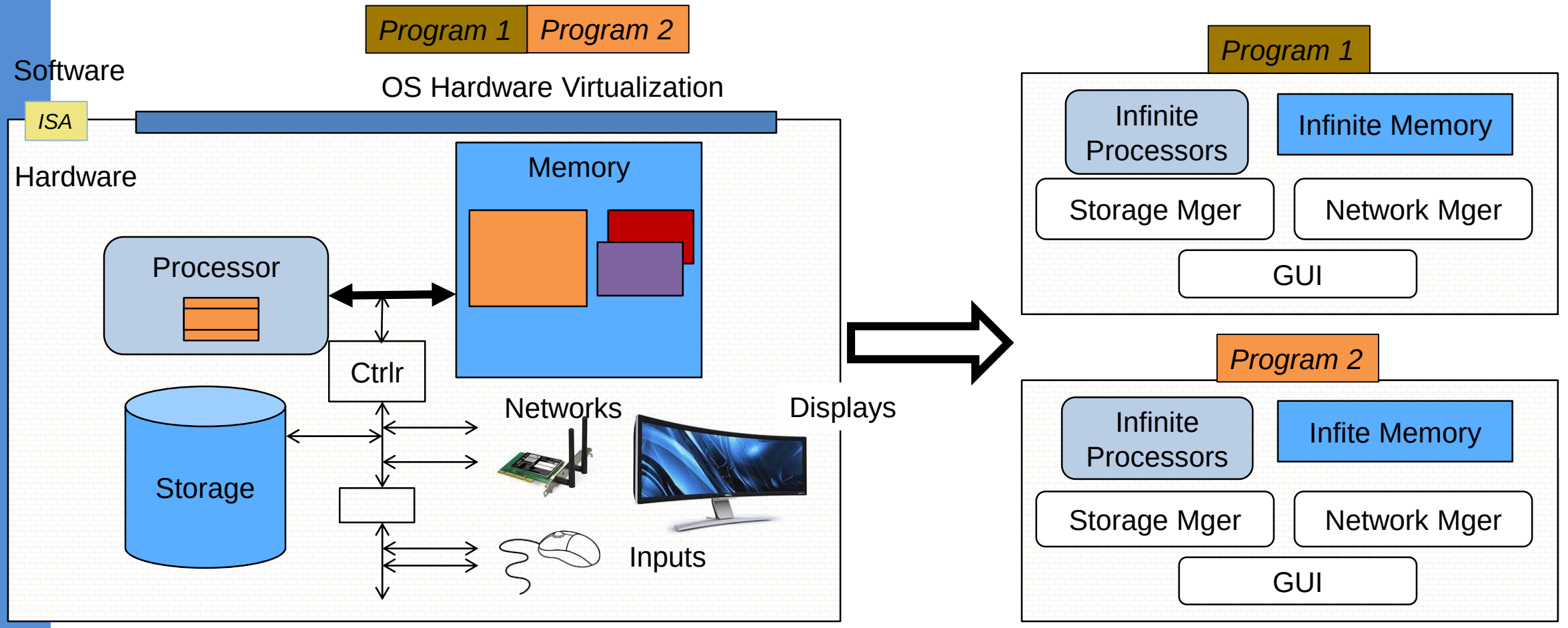
Evolve components independently

File System, User Interface, Network, etc.



Putting it all together

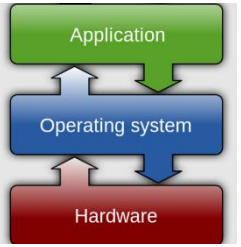
- Referee + Illusionist + Glue => Easy to use virtual machine



Evaluation Criteria: Performance

OS must implement the abstraction efficiently, with low overhead, and equitably

Overhead: added resource cost of implementing an abstraction
Fairness: How "well" are resources distributed across applications
Response time: how long does it take for a task to complete
Throughput: Rate at which group of tasks can be completed
Predictability: Are performance metrics constant over time?

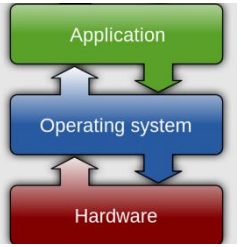


Evaluation Criteria: Reliability

System does what it is supposed to do

OS failures catastrophic!

Availability: mean time to failure + mean time to repair



Evaluation Criteria: Security

Minimize vulnerability to attack

Integrity: Computer's operation cannot be compromised by a malicious attacker

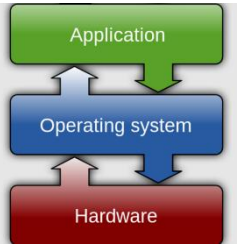
Privacy: data stored on computer accessible to authorized users

Enforcement Policy

How the OS ensures only permitted actions are allowed

Security Policy

What is permitted



Evaluation Criteria: Portability

A portable abstraction does not change as the hardware changes

Can't rewrite application (or OS!) every time

Must plan for hardware that does not exist yet!

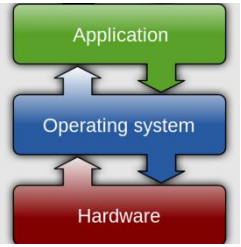
Application

Abstract Machine Interface

Operating System

Hardware Abstraction Layer

Hardware



Administrativia

Three "Prongs" for the Class

Understanding OS principles

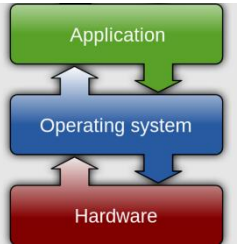
System Programming

Lectures

Assignments

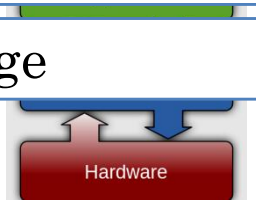
Map Concepts to Real Code

Group Projects



Topic Breakdown

- Virtualizing the CPU
  - Process Abstraction and API
  - Threads and Concurrency
  - Scheduling
- Virtualizing Memory
  - Virtual Memory
  - Paging
- Persistence
  - IO devices
  - File Systems
- Distributed Systems
  - Challenges with distribution
  - Data Processing & Storage

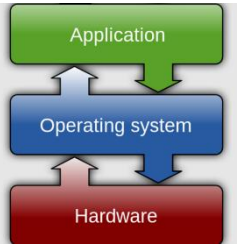


Enrollment

- Class has limit of 100 students
  - This semester we will have only 26

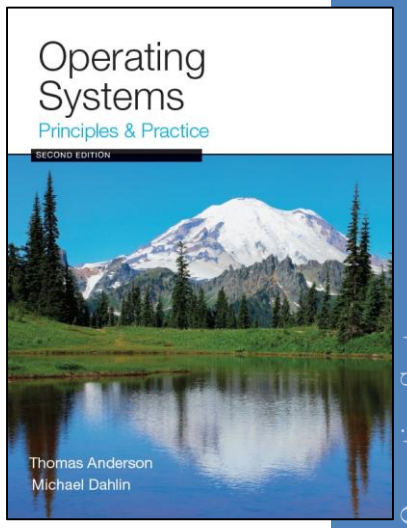
- The Early Drop Deadline is January 22, 2025
  - If you are not serious about taking the class, please drop early
  - Really hard to drop afterwards!
  - The class involves a group project, it makes life much more difficult for the others if you drop out in the middle

- Note: this is a hard class!
  - While it requires a lot of work, the class is highly rewarding

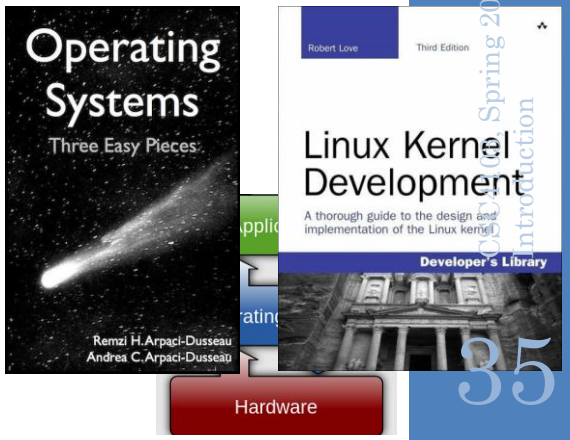


Infrastructure, Textbook & Readings

- Infrastructure
  - Website: https://teaching.hkaiser.org
  - Discord: https://discord.gg/NdKrbJTaUg
- Textbook: Operating Systems: Principles and Practice (2nd Edition) Anderson and Dahlin
  - Not required, get a copy if you feel that lectures are not sufficient
  - Suggested readings posted along with lectures
  - Try to keep up with material in book as well as lectures



- Supplementary Material
  - Operating Systems: Three Easy Pieces, by Remzi and Andrea Arpaci-Dusseau, available for free online
  - Linux Kernel Development, 3rd edition, by Robert Love

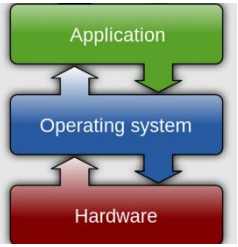


Class Expectations

- Lectures
  - Come! Cannot guarantee content will be identical to previous years
  - Electronic devices used only for note taking
  - Will explain many things related to assignments not otherwise explained
  - Attendance not mandatory but highly advised
  - If you decide to come, please be on time
  - Meet your fellow students, they are your future colleagues!

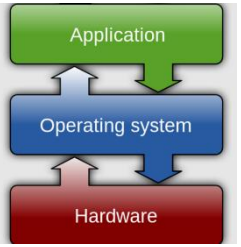
- Office Hours
  - Come and ask for help early. There are no stupid questions!
  - We like teaching and want to meet you!

- Communicate with course staff through Discord and Email.

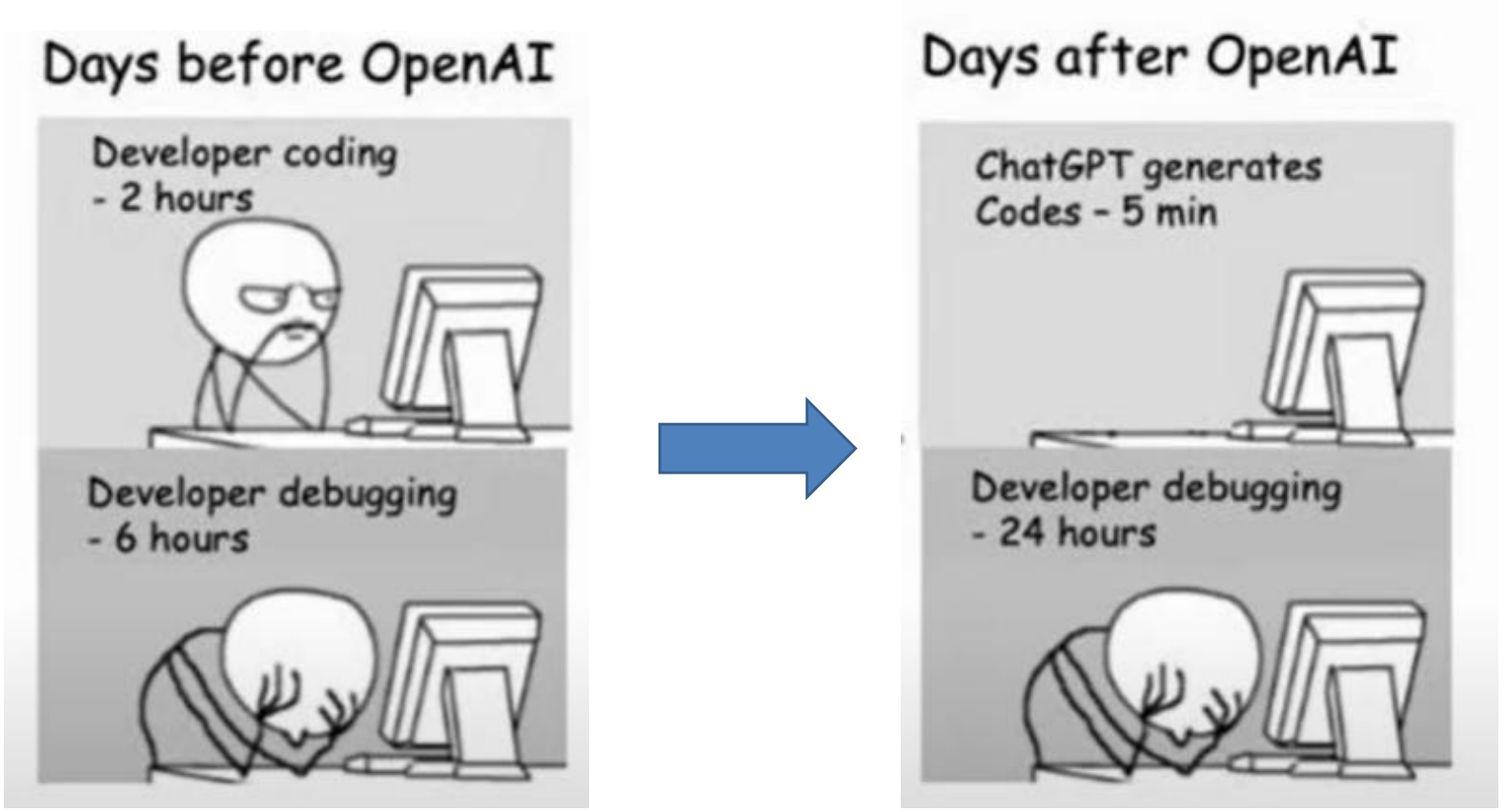


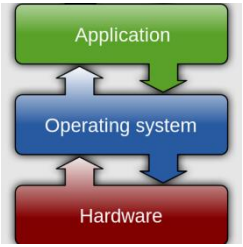
Honesty

- The LSU *Code of Student Conduct* defines plagiarism in Section 5.1.16:
  - "Plagiarism is defined as the unacknowledged inclusion of someone else's words, structure, ideas, or data. When a student submits work as his/her own that includes the words, structure, ideas, or data of others, the source of this information must be acknowledged through complete, accurate, and specific references, and, if verbatim statements are included, through quotation marks as well. Failure to identify any source (including interviews, surveys, etc.), published in any medium (including on the internet) or unpublished, from which words, structure, ideas, or data have been taken, constitutes plagiarism;"
- Plagiarism will not be tolerated and will be dealt with in accordance with and as outlined by the LSU Code of Student Conduct : https://www.lsu.edu/saa/students/codeofconduct.php



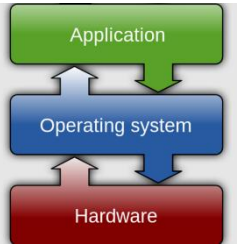
ChatGPT?


Days before OpenAI
Developer coding
- 2 hours
Developer debugging
- 6 hours
Days after OpenAI
ChatGPT generates
Codes - 5 min
Developer debugging
- 24 hours


Application
Operating system
Hardware

ChatGPT

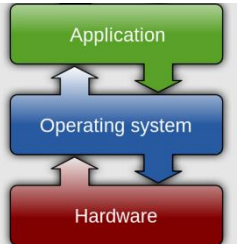
- The goal for this course is to train our brains, not ChatGPT
  - So, do yourself a favor and don't use it
  - There are too many software 'developers' out there who copy & paste their way to the next paycheck
- However, if you do use it:
  - Never use anything without carefully reviewing it
    - Assume what you got is wrong! Prove to yourself it is correct!
  - The skill of reading (and understanding) code becomes more important than ever
  - Add a note which part of your code was generated
- Whatever you do, remember:
  - It's plagiarism if you submit the same code as your neighbor
  - No matter where you got it from, be it Google, ChatGPT, or your neighbors computer



Learning by Doing

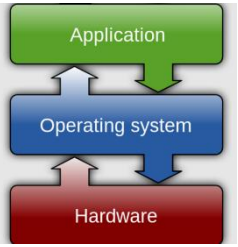
- Individual Assignments (3 weeks)
  - 0. Tools & Environment, Autograding, recall C, executable
  - 1. Lists in C
  - 2. BYOS – build your own shell
  - 3. Sockets & Threads in HTTP server
  - 4. possibly more…

- Two (and ½) Group Projects (4-6 weeks)
  - 0. Getting Started (Individual, before you have a group)
  - 1. User-programs (exec & syscall)
  - 2. Threads & Scheduling



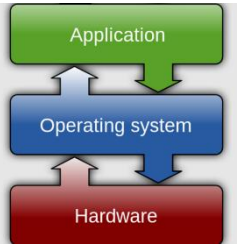
Group Projects

- Project teams have 4 members!
  - Never 5, 3 requires serious justification
  - Must work in groups in "the real world"
  - Suggest teammates via email, we'll try to accommodate
  - Everyone will be randomly assigned to a group by January 24th
- Everyone should do work and have clear responsibilities
  - You will evaluate your teammates at the end of each project milestone
  - Dividing up by Task is the worst approach. Work as a team.
- Communicate with supervisor (TAs)
  - What is the team's plan?
  - What is each member's responsibility?
  - Short progress reports are required
  - Design Documents: High-level description for a manager!



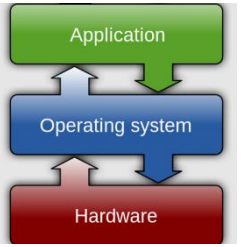
Group Projects

- Detailed exploration of operating system implementation
- This is a project class:
  - Every assignment requires substantial programming effort
  - Most programming is C; a small amount is IA32 assembly
  - Use Git version control for managing your code, making checkins, etc.
  - Also, Make/Gdb/etc. Linux tools.
- Course uses the PintOS instructional operating system
  - A small UNIX-like operating system with very limited capabilities
  - Implemented in 2005 for use with Stanford's CS140 OS class
  - Intended to be run on IA32/x86 processor emulator (Bochs, QEMU)
    - Can also run on actual IA32 hardware if properly coaxed…
    - We have set up a Docker image for this



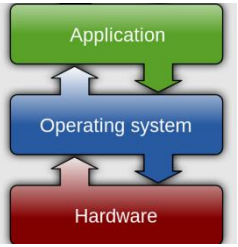
Assignments and Collaboration

- The assignments are hard!
- I repeat: the assignments are hard!
- Lots of code to understand, significant implementation effort, and lots of debugging to do
- You are required to work in groups
  - Biggest reason: you will have other people to talk with, when designing and debugging systems
- Students can drop the class, but this will affect others…
  - Please only take this course if you really intend to finish it!
- If students drop later in the term, we can adjust the teams
  - e.g. move a student into another team (student will have to learn the new team's code)



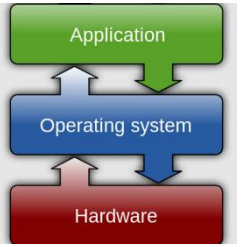
Assignments and Collaboration

- Each team's submission must be created entirely by that team alone. Teams cannot share implementation code.
- Cross-team sharing is encouraged in these areas:
  - Design and implementation ideas (but not code or pseudocode!)
  - Pitfalls you encountered, and how to solve them
  - Help with setup and debugging
- Also, PintOS has been around since 2005…
  - Do not look for solutions to projects online!
- You are encouraged to look at other resources, e.g. Linux sources, other textbooks, OS dev. websites, etc.
  - Don't copy code! (see first point above) Focus on understanding it.
  - Cite any external sources in your submission, so I can share them with the class this year and next year.



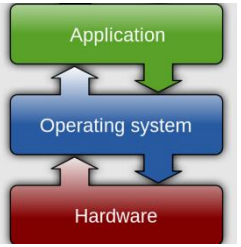
Assignments and Due-Dates

- Each assignment specifies a due-date (Monday's 11.59 pm)
- Assignments and Projects are on a tight schedule
- Late submissions are penalized
  - Usually one day pre-approved extension
  - On-time submissions receive 10% bonus
- Students/teams can also request extensions due to health or other reasons
  - Most important thing is to try to do this beforehand, if possible



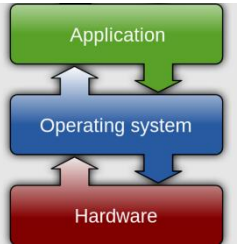
Development and Testing Platform

- PintOS is designed to be built and tested on Linux
- Submissions will be tested on 32-bit Ubuntu Linux
  - You are encouraged to at least test on this platform
  - A very good idea to develop on a similar 32-bit Linux platform
- We provide a virtual machine (Docker image) for you to use
  - Will have all necessary development tools (Git, gcc, emulators, …)
- ARM-based Mac users may have some challenges…
  - VirtualBox just released a slow, ARM-based developer preview
  - The TAs have put together a Docker image with cross-compiler tools etc.
  - I'm afraid your life will be more complicated ®
  - You can use Codespaces (i.e. fully online VSCode on github)



Getting started

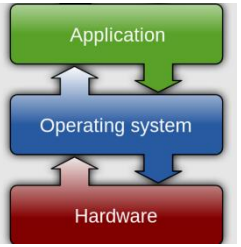
- Assignment 0 has been posted
  - Due date: Monday, January 27 2025
  - Individual assignment
  - Set up development environment, familiarize yourself with tools
- Project 0 has been posted as well
  - Due date: Monday, February 10 2025
  - Individual assignment as well, later projects will be group based
  - Familiarize yourself with PintOS, how to debug things
- Don’t wait for the last day!
  - Assignments and projects require a lot of work



Getting started

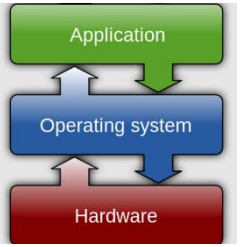
- Start Assignment 0 and Project 0 right away
  - Github account, assignments managed through Github classroom
  - Docker image - environment for the course
    - Consistent, managed environment on your machine
  - Get familiar with all the CSC4103 tools
  - Submit to autograder via git, feedback through Github

- Study Guides on website
  - We suggest to use those as a constant reference
  - Answer questions, requires looking around for solutions
  - Perfect guidelines for examinations



Preparing Yourself for this Class

- Projects will require you to be very comfortable with programming and debugging C
  - Pointers (including function pointers, void*)
  - Memory Management (malloc, free, stack vs. heap)
  - Debugging with GDB
- You will be working on a larger, more sophisticated code base than anything you've likely seen before!
- "Resources" page on course website
  - Ebooks on “git” and “C”
- C programming
  - Happy to answer questions during lectures



LSU CENTER FOR COMPUTATION & TECHNOLOGY

STE||AR GROUP