CS 423
Operating System Design

Tianyin Xu

* Thanks for Prof. Adam Bates for the slides.

Learning Objectives

Before CS 423:

- Knowledge of C/C++
- Basic knowledge of Linux/POSIX APIs and functions

After CS 423:

- Mastery of Operating Systems concepts
- Comprehensive understanding of virtualization techniques
- Introduction to advanced OS topics: security, energy, redundant storage…
- Become a kernel hacker capable of establishing a kernel development environment and modifying operating system code

Today:

- Introduce the instruction team
- Go over the requirements and expectations for this course

The Team

Tianyin Xu (Instructor) tyxu@illinois.edu
Jack Chen (TA) jianyan2@illinois.edu
Andrew Yoo (TA) abyoo2@illinois.edu

Office Hours

- Monday 5-6pm, 207 SC (Jack)
- Tuesday 10-11pm, 207 SC (Andrew)
- Wednesday 10-11pm, 207 SC (Andrew)
- Thursday 5-6pm 207 SC (Tianyin)
- Friday 5-6pm 207 SC (Jack)

Ass. Prof. Tianyin Xu

I'm working on software and system reliability.

I worked at Facebook on dealing with datacenter level failures before joining UIUC.

- FAQ: academia or industry?
- I gained 20 LB eating free food.

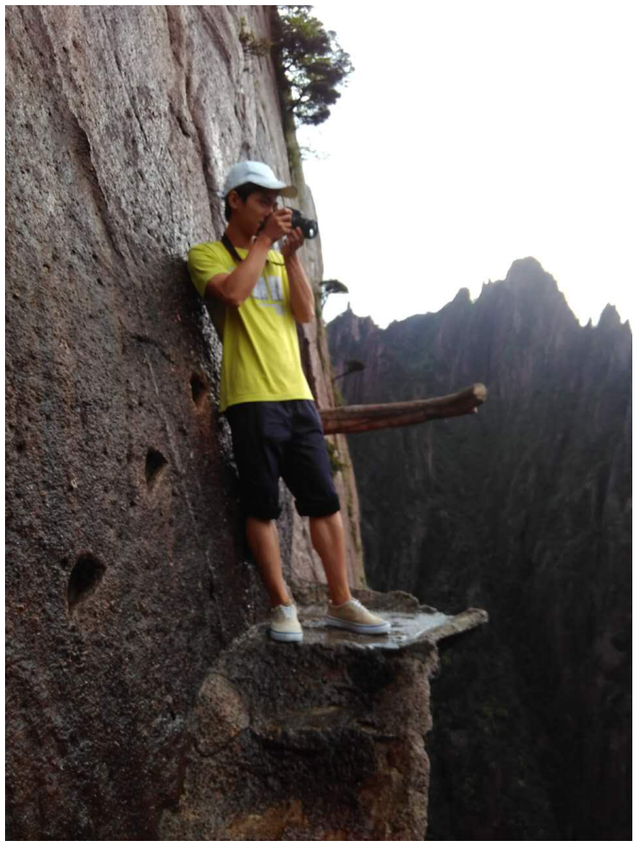
I graduated from UC San Diego in 2017.

- I worked on hardening cloud and datacenter systems against misconfigurations.
- Dream job: a tenured grad student

I applied twice for grad school.

- I failed the first time (as always).

Jianyan (Jack) Chen



I am working on software and system reliability... as well

I graduated from UW Madison in 2018.

I like lemon pound cake.

I didn’t go anywhere this winter break.

I feel cold right now...

Andrew Yoo

I'm an M.S. student working on distributed systems.

I received my B.S. from UC San Diego.

* I worked on using ML for 5G.

* I also worked on preventing misconfigurations.

I went back to my home next to LA over winter break.

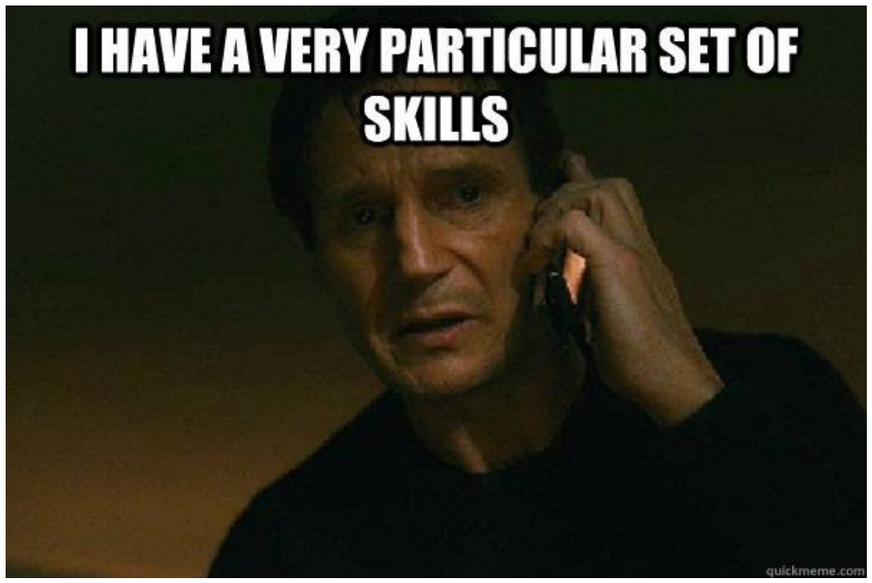
On my free time, I watch sports and play games!

* Not too much because of research!

What's in it for you?

- Understand the foundations of all system software
- Apply systems concepts and methodologies to higher layer software systems. Modern browsers, language virtual machines, and IoT devices all run their own forms of operating systems!
- Acquire a very particular (and lucrative) set of skills!

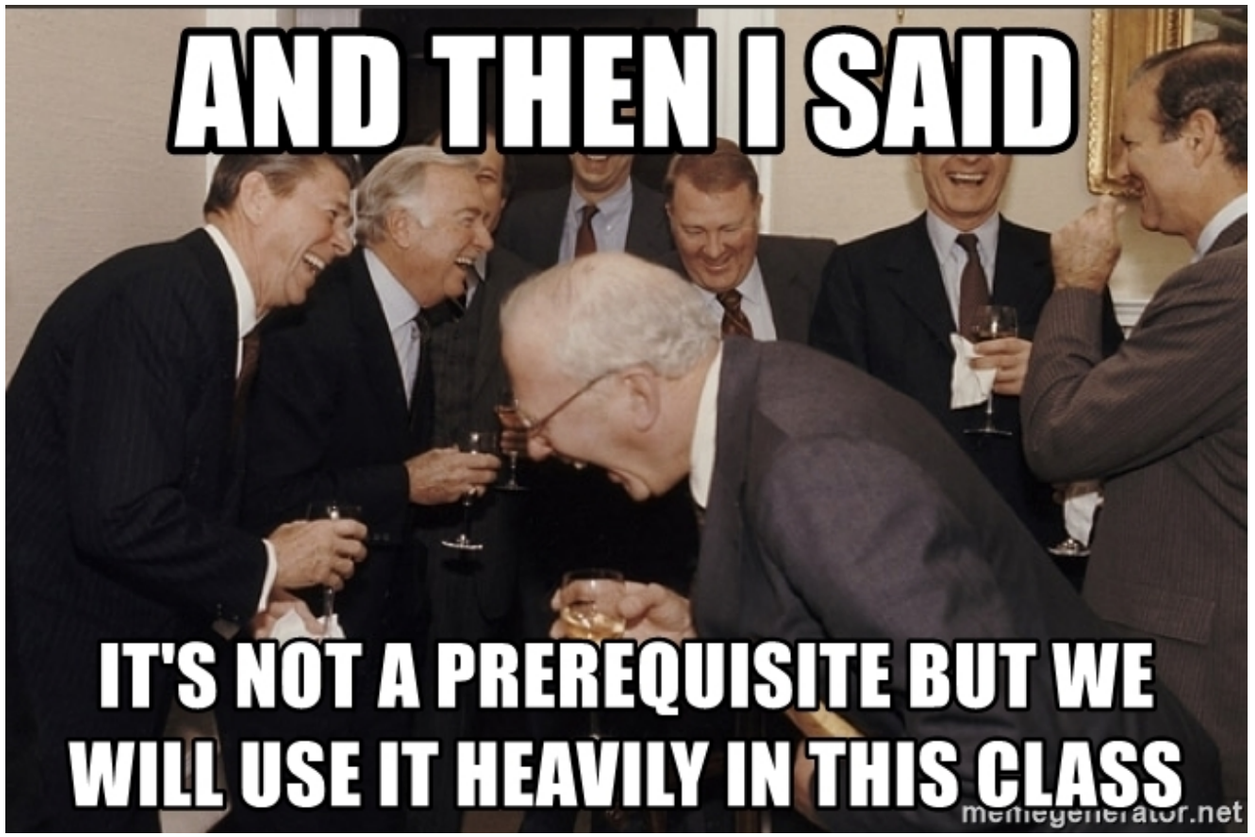
"I attended a Microsoft-organized meeting where the Director of Engineering of (Microsoft in Redmond) talked to me about a great need for engineers who know operating systems/device drivers, and know linux kernel/programming at such lower levels. He bitterly complained that many CS departments are dismanteling their OS programs. I told him that we have actually multiple OS undergraduate classes at UIUC the current instructor to advertise among the students who take these courses that there are many jobs at Microsoft in OS area (more than ever!)."



My Own Version

- Prepare you for the real world!
  - Real-world software is imperfect and buggy
    - Unfortunately, you have to built on top of them;
  - Real-world infrastructure is fragile and vulnerable.
    - Unfortunately, you have to bear with that.
  - Document is obsolete and even misleading
    - You won't have teachers or TAs
- No matter what, have **FUN**!

Prerequisites



- Did you take CS241?
- Did you take ECE391?
- Do you have systems programming experiences from another university?
- If not, you might have a bad time in this course…

Online Discussion

piazza

You are already added on the Piazza.
(if not, find the link on the course website)

Go here for announcements and to ask questions.

Instruction team will be checking forums regularly!

Textbook

- "Operating Systems: Three Easy Pieces" OSTEP Remzi and Andrea Arpaci-Dusseau
  - It is **FREE**.
  - Why Textbooks Should Be Free
  - The chapters are linked on the website.

Additional Texts

- Alternative Textbooks (Not Free):
  - Operating Systems: Principles & Practice
    - Anderson and Dahlin, 2018
  - Modern Operating Systems
    - Tanenbaum and Bos, 2014
  - Operating System Concepts
    - Silberschatz, Galvin and Gagne, 2012

- Other Recommended Reading:
  - Virtual Machines
    - Smith and Nair, 2005
  - Linux Kernel Development**
    - Love, 2010

** Helpful for MPs

CS 423 Requirements

- Attendance/Participation
  - Come to class, Tue/Thu, 2:00-3:15am
  - Participate actively in class and on piazza
- Machine Problems (MPs): 4.5 major programming assignments + one warm-up
- Periodic Homeworks: includes "prereqs" and "practice final", may assign more
- Midterm & Final Exams: Dates TBD
- 4 Credit Class: Read additional assigned literature and submit summaries weekly.

ALL WORK IS TO BE INDEPENDENTLY COMPLETED!

Participation

- Contribute in class — ask questions, respond to questions, share relevant outside knowledge.
- Contribute *good* questions and answers on Piazza!
- *"The kind of answers you get to your technical questions depends as much on the way you ask the questions as on the difficulty of developing the answer."*
- **How To Ask Questions The Smart Way:** http://www.catb.org/esr/faqs/smart-questions.html
- Other questions (e.g., administrative) on Piazza are also welcome, but won't give you participation credit.

Four Credit Section

- Intended audience: graduate students, ambitious undergraduate students interested in research.
- Earn your 4th credit by reading and summarizing weekly literature assignments
- **Summaries due on the Frida, 11:59pm y of each week. The first summaries are due January 24th.**
- Upload summaries as PDFs on compass.
- PDFs should be typeset in LaTeX.
- Assigned readings are marked as **C4** in the Assignments section of the class schedule. Other students are not required to read these papers.
- Grading: Summaries will contribute to C4 student's homework and participation credit.

C4 Paper Summaries

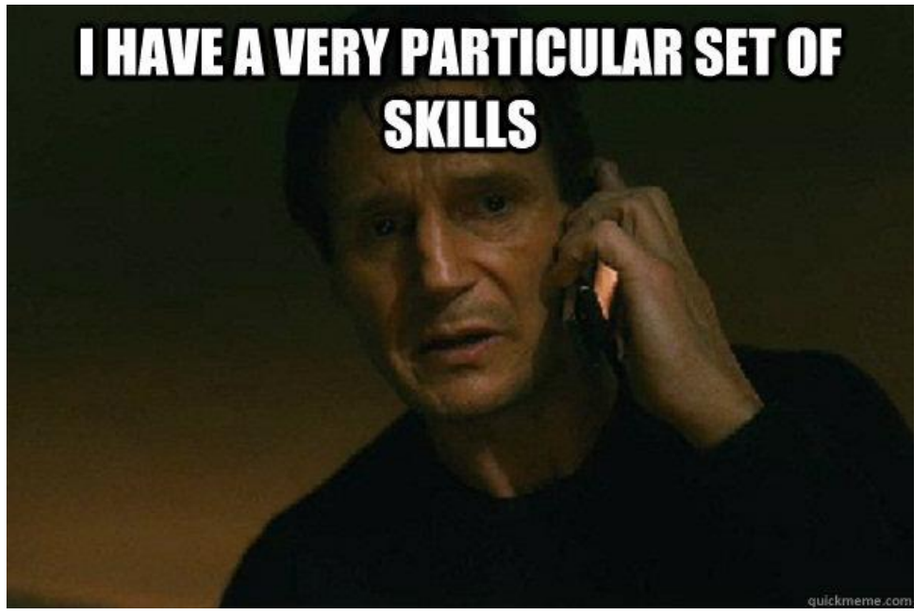
- Each summary should be about a page in length.
- Structure your summary to cover:

1. Area
2. Problem
3. Solution
4. Methodology
5. Results
6. Takeaway

Machine Problems

- Implement and evaluate concepts from class in a commodity operating system
- Kernel Environment: Linux. Not a toy OS, but a real 25 million LoC behemoth.
- Why? Building out a small OS is good experience, but navigating an existing code base is a more practical skill.
- Recall from earlier:

  "I attended a Microsoft-organized meeting where the Director of Engineering of (Microsoft in Redmond) talked to me about a great need for engineers who know operating systems/device drivers, and know linux kernel/programming at such lower levels. He bitterly complained that many CS departments are dismantling their OS programs. I told him that we have actually multiple OS undergraduate classes at UIUC the current instructor to advertise among the students who take these courses that there are many jobs at Microsoft in OS area (more than ever!)."



MP Development Environment

- Engr-IT managed VMs will be provided for you
- If you brick your machine (happens often), we'll need to open a ticket with Engr-IT (>= 24 hour delay)
- Brick your machine on a weekend? Too bad for you.
- Occasionally, the VM cloud just goes down! That's fun.

MP Development Environment

- Coping Strategies:
  - Develop in your own local VMs (e.g., VirtualBox).
    - Distro information, etc. will be provided
  - Use version control (private repositories only)
    - You will still need to test and submit your MPs in the Engr-IT VM, which is where we will grade.
- *Extensions due to VM failures will only be granted for cloud-wide availability disruptions and other extraordinary circumstances, not for self-inflicted issues!*

Submission

- Code repository
  - You will need to submit your source code
  - We will create a private GitHub repo for you.
  - Everything will be based on GitHub.

Grading

Final Exam: **20%**
Mid-term Exam: **20%**
Homework: **4%**
Machine Problems (5.5 total): **46%**
- **2%, 4%, 10%, 10%, 10%, 10%**

Participation: **10%**
- Class / Forum involvement

Policies

- **No late homework/MP submissions**
- 1 week window for re-grades from return date
- **Cheating policy: Zero tolerance**
  - 1st offense: get zero
  - 2nd offense: fail class
  - Example: You submitted two MPs in which solutions were not your own. Both were discovered at the same time. You fail class.

Feedback welcome!

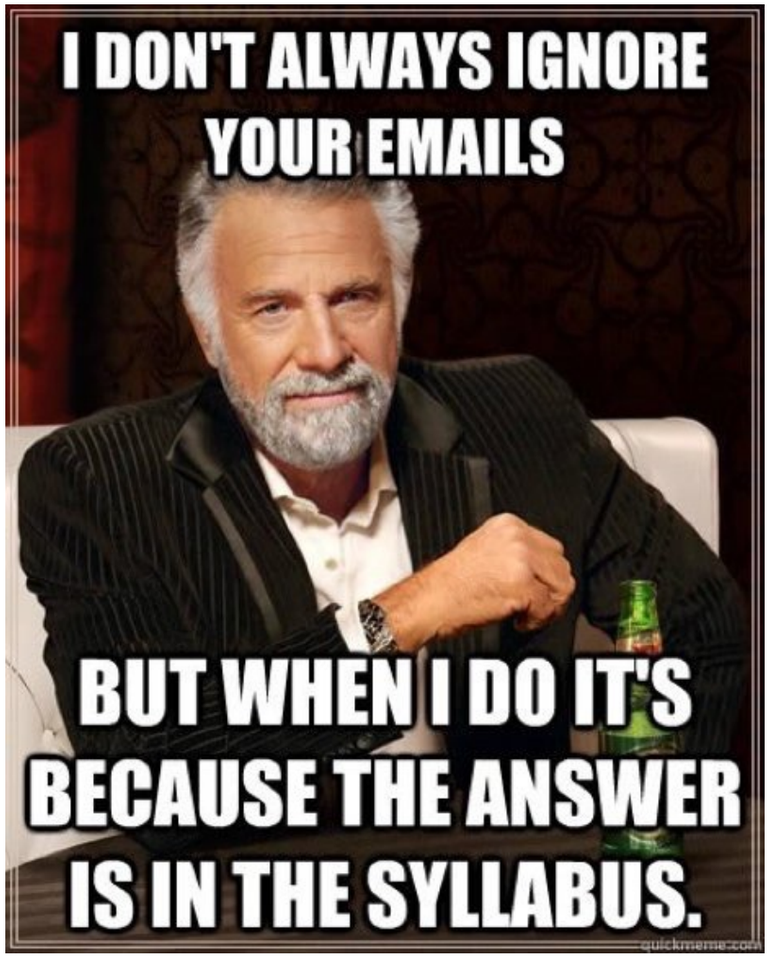
- This is my first time teaching 423.
- I will likely be screwing up things.
- Feedback is ALWAYS welcome (including criticisms of my bad jokes).

Course Website

https://cs423-uiuc.github.io/spring20/

Go here for…

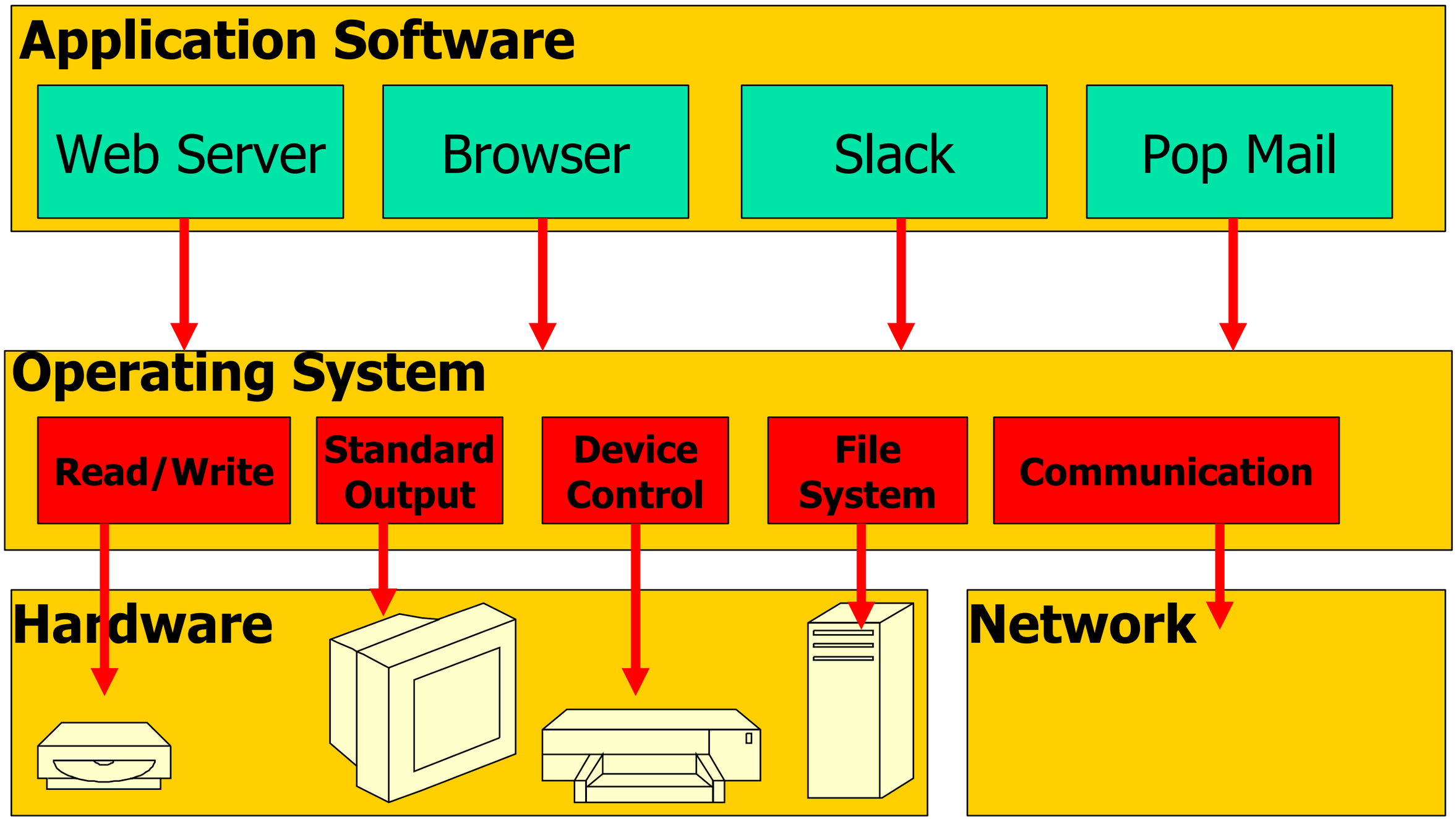
- Syllabus
- Course Schedule
- Lecture Slides/Recordings
- Links to other resources



What is an operating system?

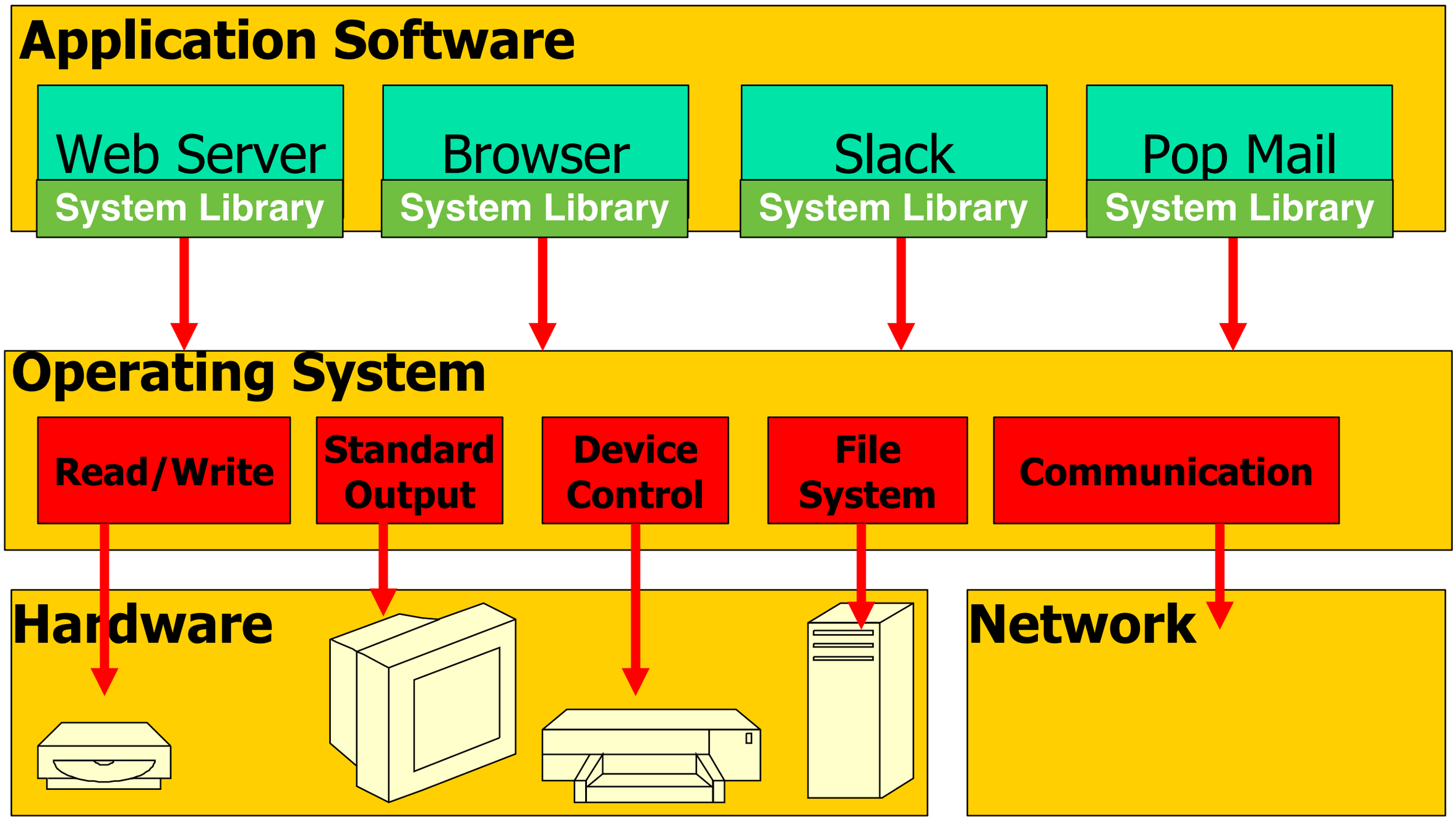
Why Operating Systems?

Software to manage a computer's resources for its user



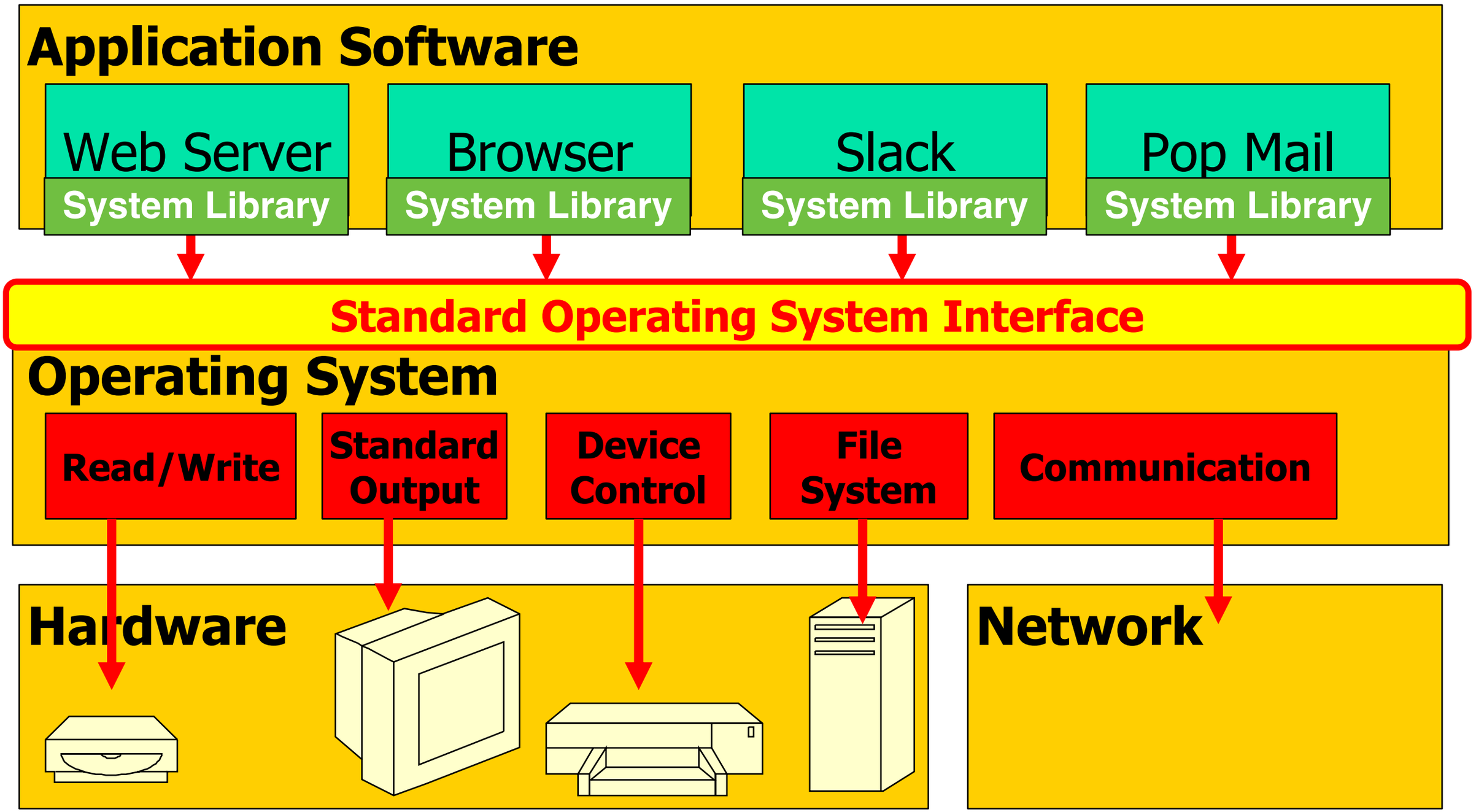
Why Operating Systems?

Software to manage a computer's resources for its user



Why Operating Systems?

The OS exports a user interface. Why?



Why Operating Systems?

Standard interface increases portability and reduces the need for machine-specific code.

Application Software

Web Server | Browser | Slack | Pop Mail

System Library | **System Library** | **System Library** | **System Library**

Standard Operating System Interface

Operating System (machine independent part)

Read/Write | **Standard Output** | **Device Control** | **File System** | **Communication**

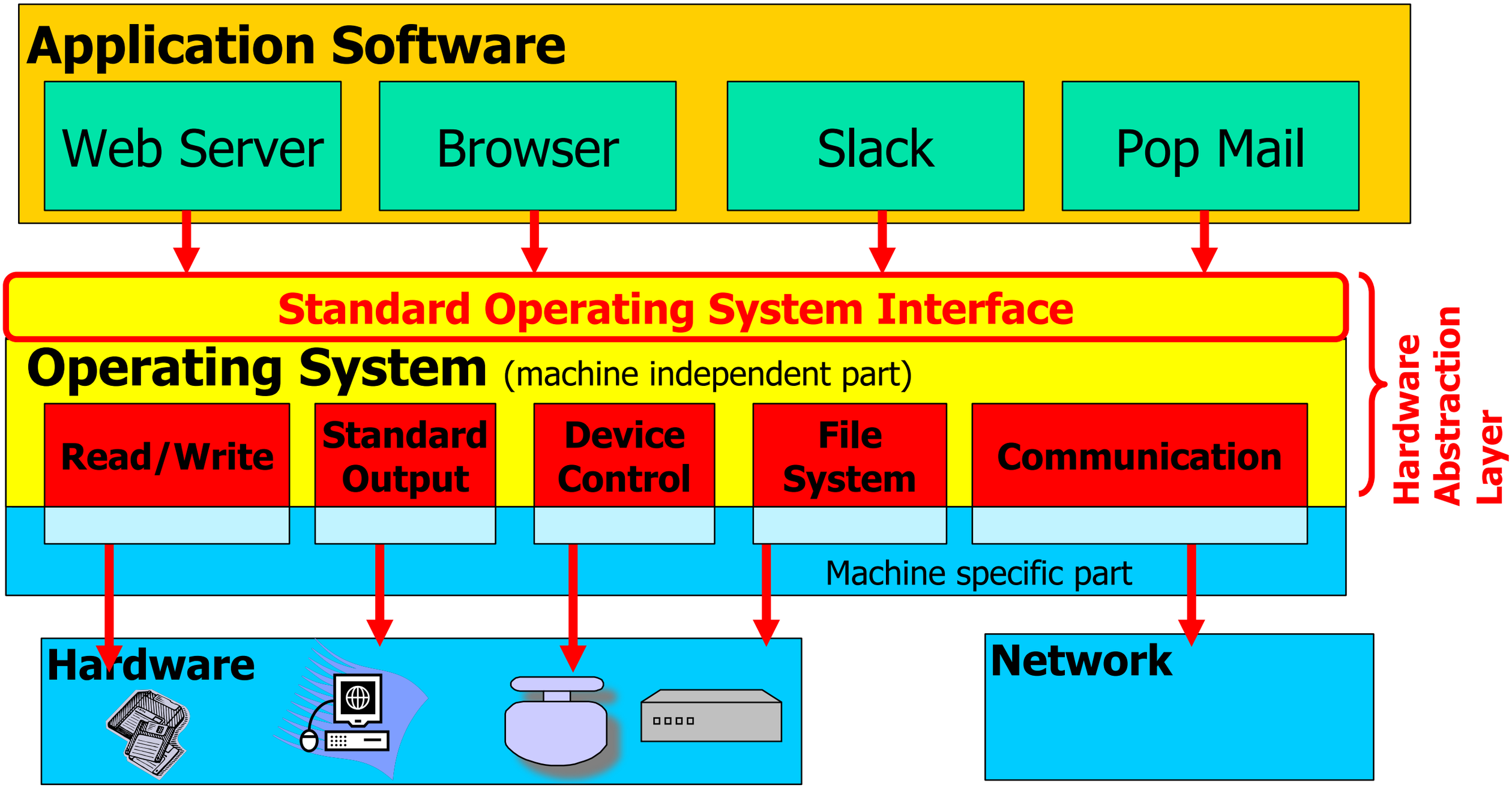
Machine specific part

Hardware

Network

Why Operating Systems?

OS Runs on Multiple Platforms while presenting the same Interface:



What are the responsibilities of an operating system?

Operating System Roles

Role #1: Referee

- Manage resource allocation between users and applications
- Isolate different users and applications from one another
- Facilitate and mediate communication between different users and applications

Operating System Roles

Role #2: Illusionist

- Allow each application to believe it has the entire machine to itself
- Create the appearance of an Infinite number of processors, (near) infinite memory
- Abstract away complexity of reliability, storage, network communication…

Operating System Roles

Role #3: Glue

- Manage hardware so applications can be machine-agnostic
- Provide a set of common services that facilitate sharing among applications
- **Examples of "Glue" OS Services?**

Operating System Roles

Role #3: Glue

- Manage hardware so applications can be machine-agnostic
- Provide a set of common services that facilitate sharing among applications
- **Examples of "Glue" OS Services?**
  - Cut-and-paste, File I/O, User Interfaces…

Consider file systems and storage devices…

How is the OS a referee?

An illusionist?

Glue?

Ex: File System Support

Referee

- Prevent users from accessing each other's files without permission
- Even after a file is deleting and its space re-used

Illusionist

- Files can grow (nearly) arbitrarily large
- Files persist even when the machine crashes in the middle of a save

Glue

- Named directories, printf, other system calls for File I/O

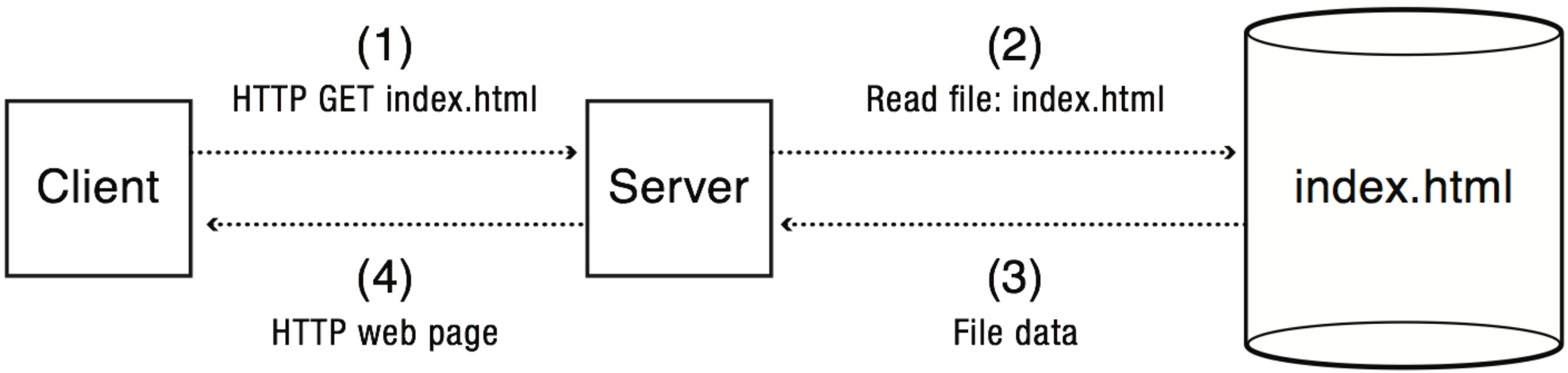
A Question

What does an OS need to do in order safely run an untrustworthy application?

Another Question

How should an operating system allocate processing time between competing uses?

Example: Web Service



- How does the server manage many simultaneous client requests?
- How do we keep the client safe from spyware embedded in scripts on a web site?
- How do handles updates to the web site such that clients always see a consistent view?

OS Challenges

Reliability

- Does the system do what it was designed to do?

Availability

- What portion of the time is the system working?
- Mean Time To Failure, Mean Time to Repair

Security

- Can the system be compromised by an attacker?

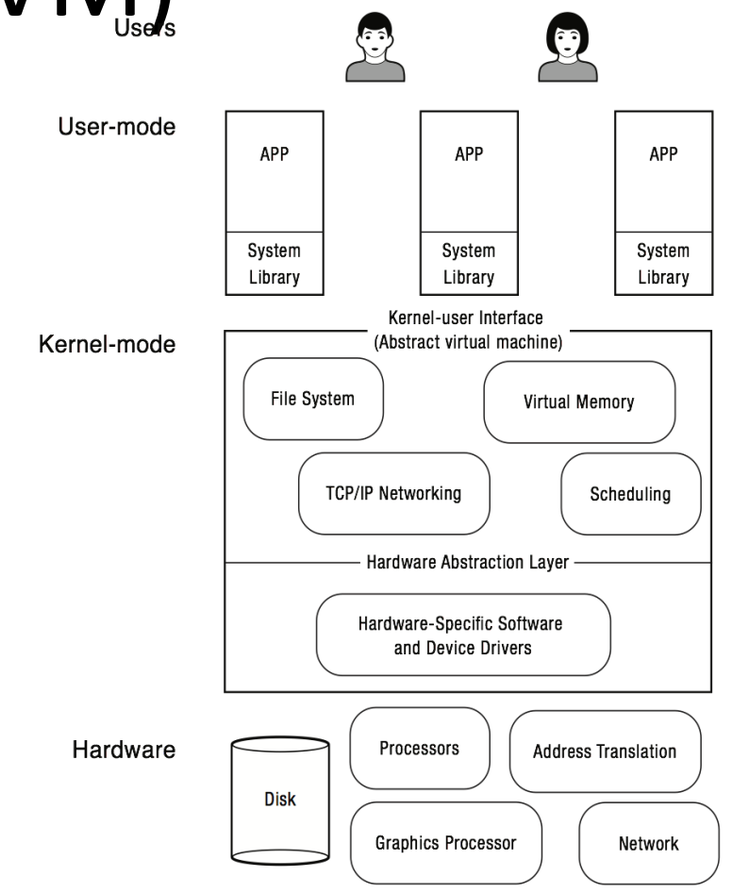
Privacy

- Data is accessible only to authorized users

OS Challenges

Portability

- For programs:
  Application programming interface (API)
  Abstract virtual machine (AVM)
- For hardware
  Hardware abstraction layer

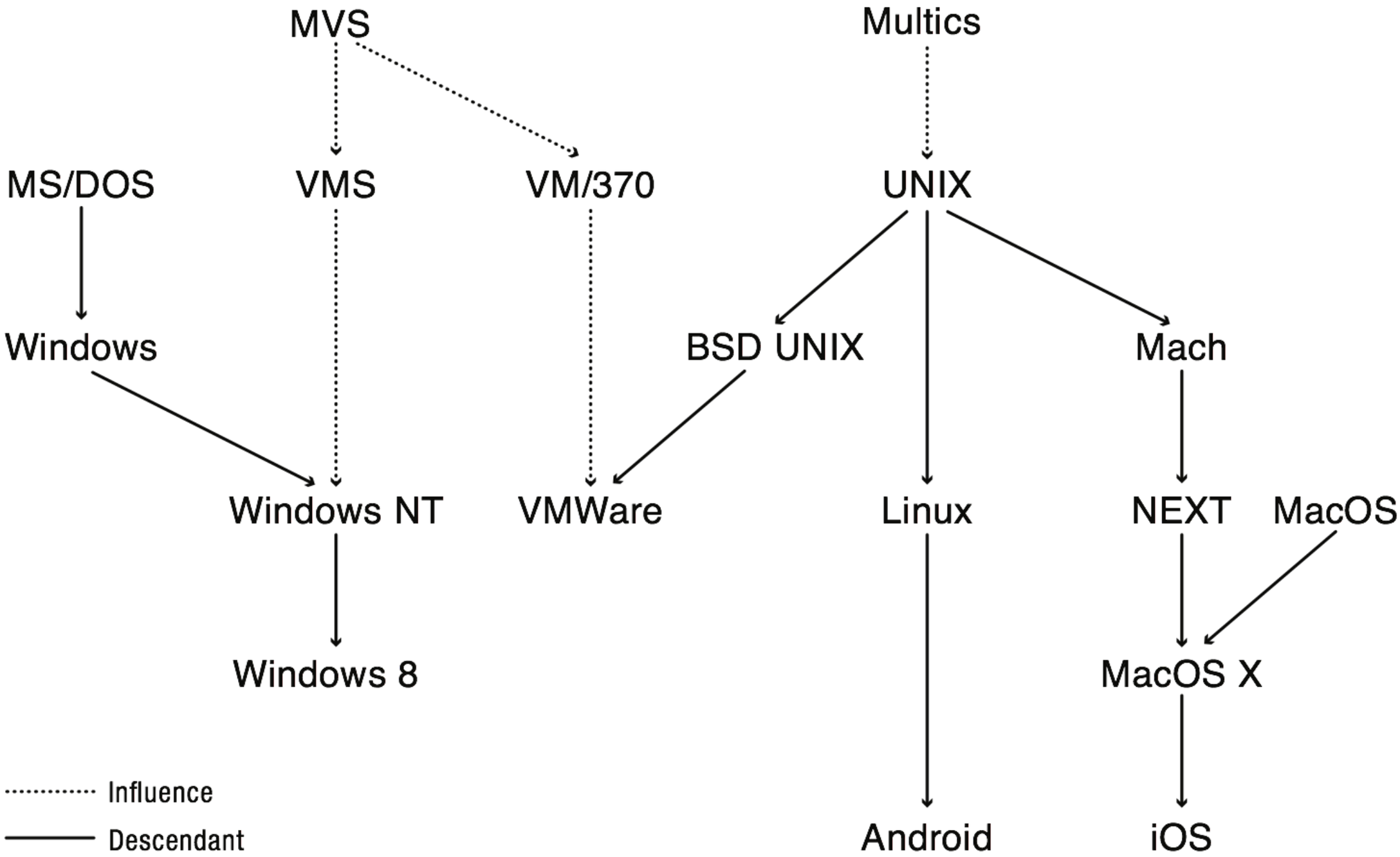


OS Challenges

Performance

- Latency/response time
  - How long does an operation take to complete?
- Throughput
  - How many operations can be done per unit of time?
- Overhead
  - How much extra work is done by the OS?
- Fairness
  - How equal is the performance received by different users?
- Predictability
  - How consistent is the performance over time?

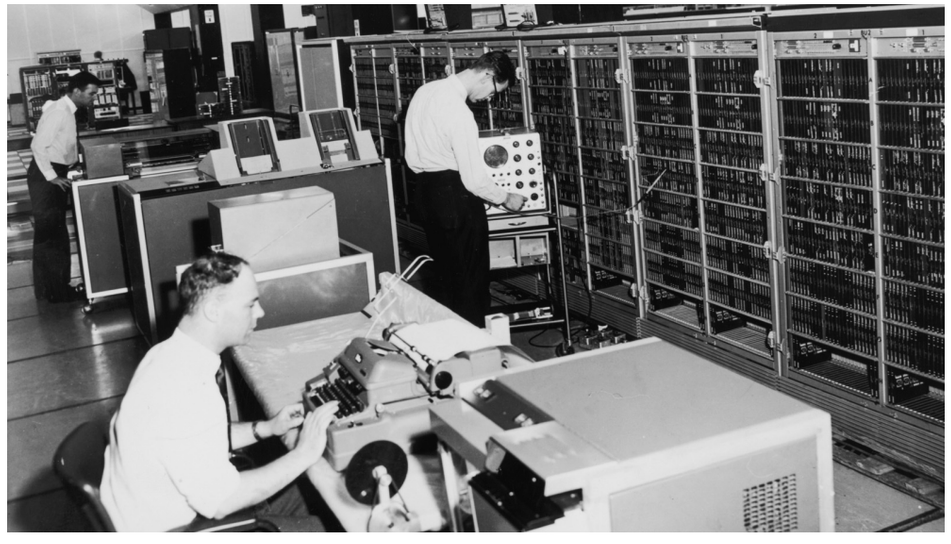
OS Family Tree



Performance / Time

	1981	1997	2014	Factor (2014/1981)
Uniprocessor speed (MIPS)	1	200	2500	2.5K
CPUs per computer	1	1	10+	10+
Processor MIPS/$	$100K	$25	$0.20	500K
DRAM Capacity (MiB)/$	0.002	2	1K	500K
Disk Capacity (GiB)/$	0.003	7	25K	10M
Home Internet	300 bps	256 Kbps	20 Mbps	100K
Machine room network	10 Mbps (shared)	100 Mbps (switched)	10 Gbps (switched)	1000
Ratio of users to computers	100:1	1:1	1:several	100+

Early Operating Systems



One application at a time

- Had complete control of hardware
- OS was runtime library
- Users would stand in line to use the computer

Batch systems

- Keep CPU busy by having a queue of jobs
- OS would load next job while current one runs
- Users would submit jobs, and wait, and wait, and

Time-Sharing OSs

Multiple users on computer at same time

- Multiprogramming: run multiple programs at same time
- Interactive performance: try to complete everyone's tasks quickly
- As computers became cheaper, more important to optimize for user time, not computer time

Today's OSs

- Smartphones
- Embedded systems
- Laptops
- Tablets
- Virtual machines
- Data center servers

Tomorrow's OSs

- Giant-scale data centers
- Increasing numbers of processors per computer
- Increasing numbers of computers per user
- Very large scale storage

Your To-Do List

Today:

- Visit the class webpage and check out all the info
  - https://cs423-uiuc.github.io/spring20/
  - Refresh your system programming skills (e.g., review CS 241 and see C language tutorial below)
    - http://www.lysator.liu.se/c/bwk-tutor.html
- Familiarize yourself with Piazza

Soon:

- Access CS 423 development VM, begin MP0
- Complete HW0