Basic Concepts & OS History

Nima Honarmand

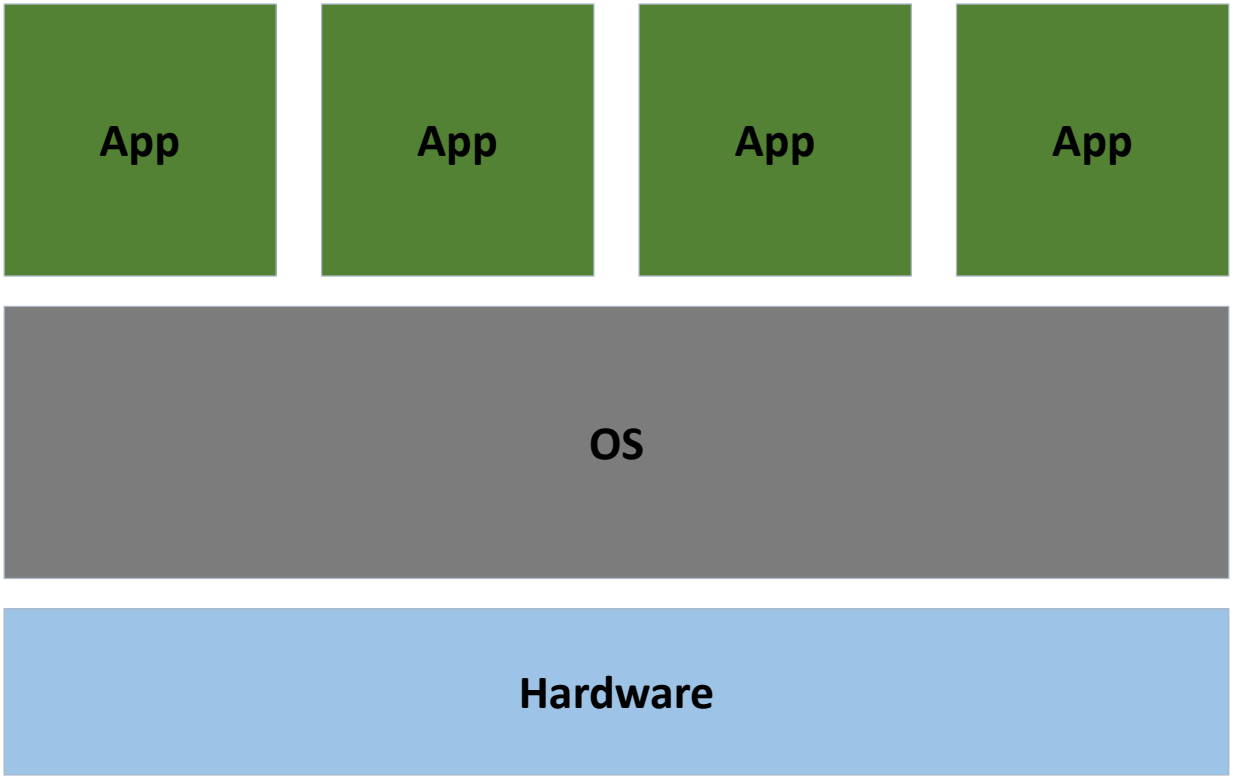
Administrivia

- TA: Babak Amin Azad
  - Office hours: Monday & Wednesday, 5:30-7:00 PM
  - Location: 2217 old CS building
- VMs ready; SSH Keys will be emailed today
- Lab1 released
  - Due date: 09/24 (11:59 PM)
  - Send me your group composition (if working as a pair) by the lab deadline

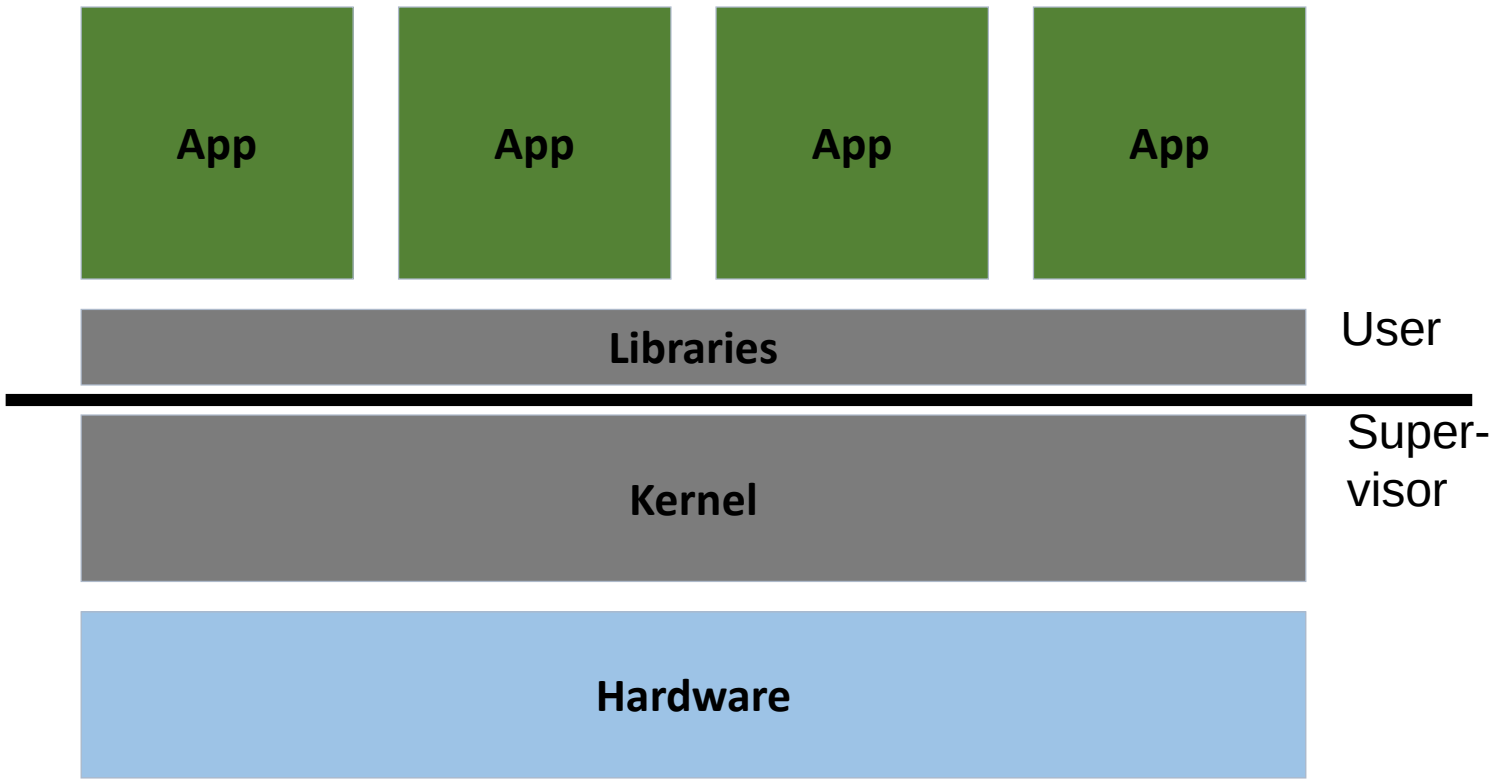
Background

- CPUs have 2 modes: user and supervisor
  - Sometimes more (4 in Intel) but 2 is all we need
- Supervisor mode:
  - Issue commands to hardware devices
  - Power off, Reboot, Suspend
  - Launch missiles, Do awesome stuff
- User mode:
  - Run other code, hardware tattles if you try anything reserved for the supervisor

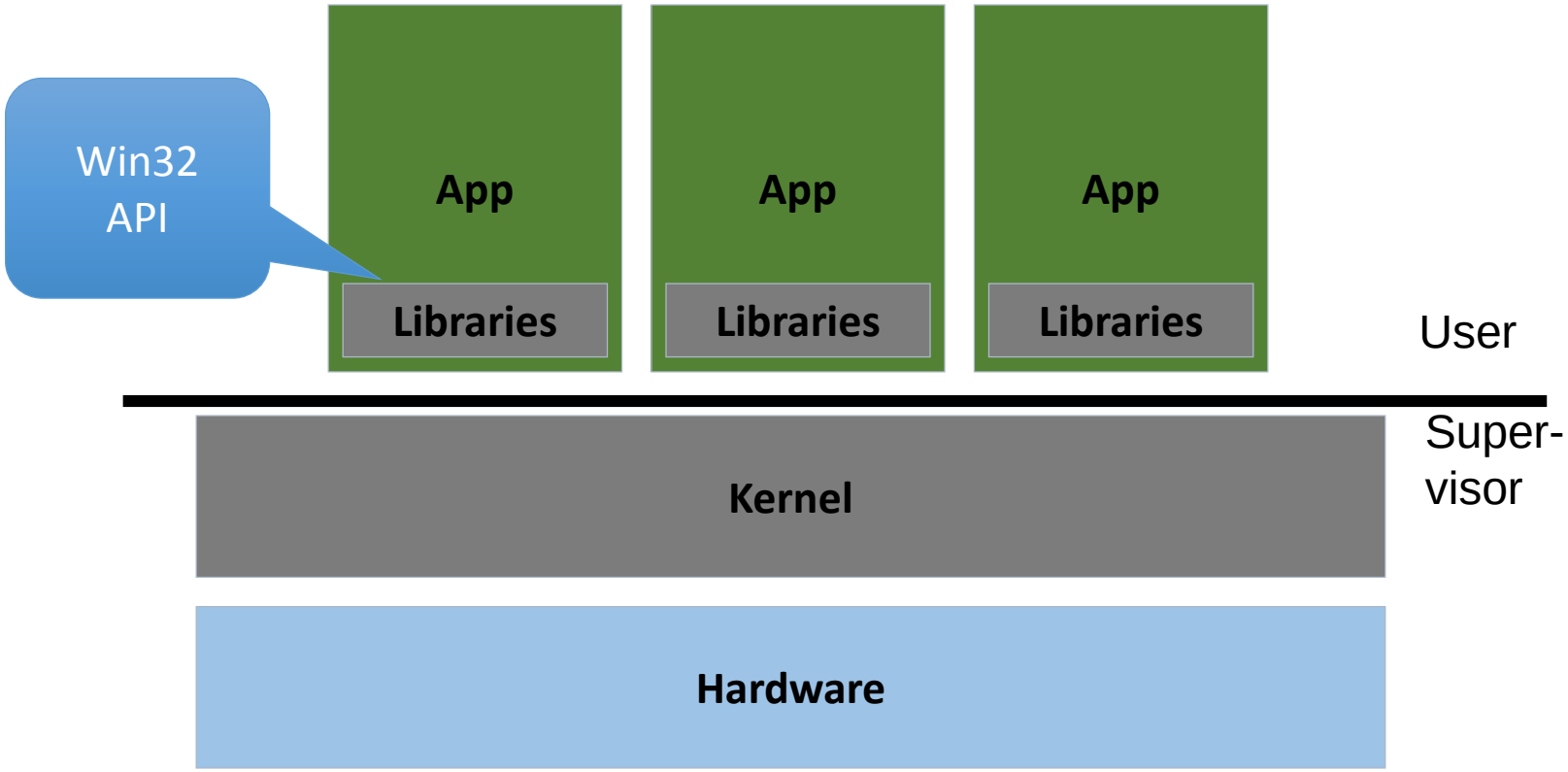
A Simple View of an OS



OS Architecture



OS Architecture



Famous libraries, anyone?

- Windows: ntdll.dll, kernel32.dll, user32.dll, gdi32.dll
- Linux/Unix: libc.so, ld.so, libpthread.so, libm.so

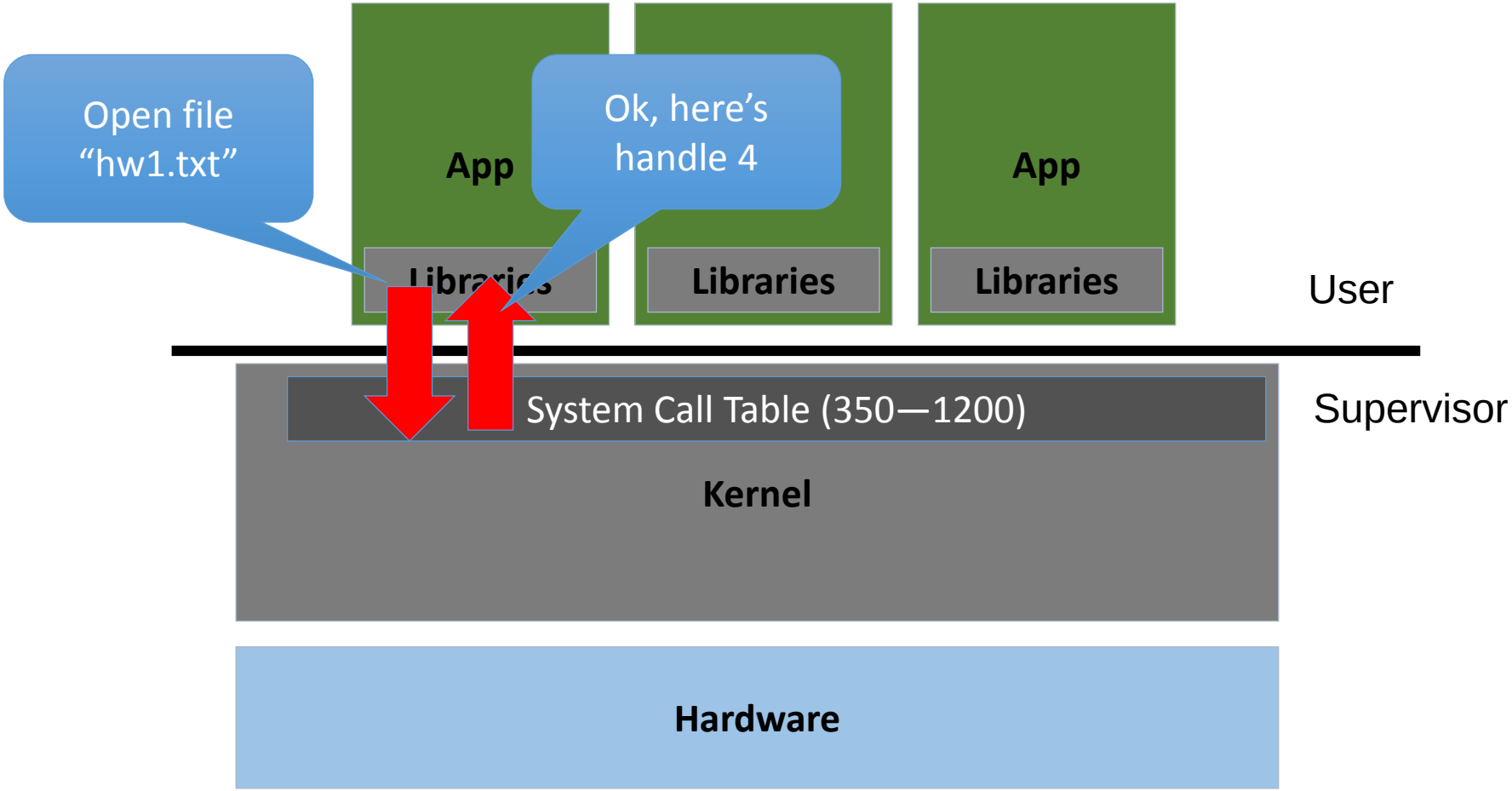
Caveat 1

- Libraries include a lot of code for common functions
  - Why bother re-implementing `sqrt`?
- They also give high-level abstractions of hardware
  - Files, printer, etc.
- How does this work?

System Call

- Special instruction to switch from user to supervisor mode
- Transfers CPU control to the kernel
  - One of a small-ish number of well-defined functions
- How many system calls does Windows or Linux have?
  - Windows ~1200
  - Linux ~350

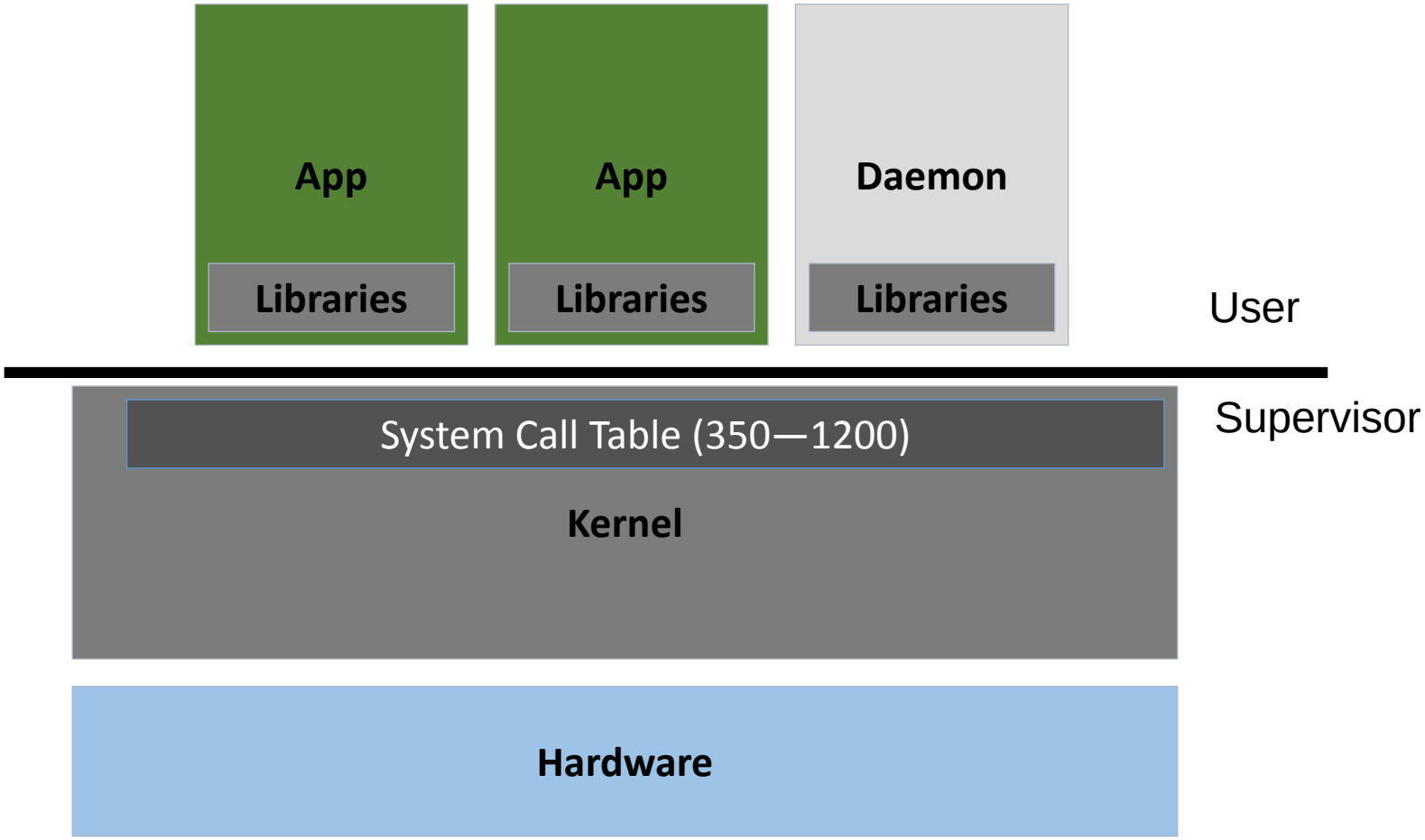
OS Architecture



Caveat 2

- Some libraries also call special apps provided by the OS, called a ***daemon (or service)***
  - Communicate through kernel-provided API
- Example: Print spooler
  - App sends pdf to spooler
  - Spooler checks quotas, etc.
  - Turns pdf into printer-specific format
  - Sends reformatted document to device via OS kernel

OS Architecture

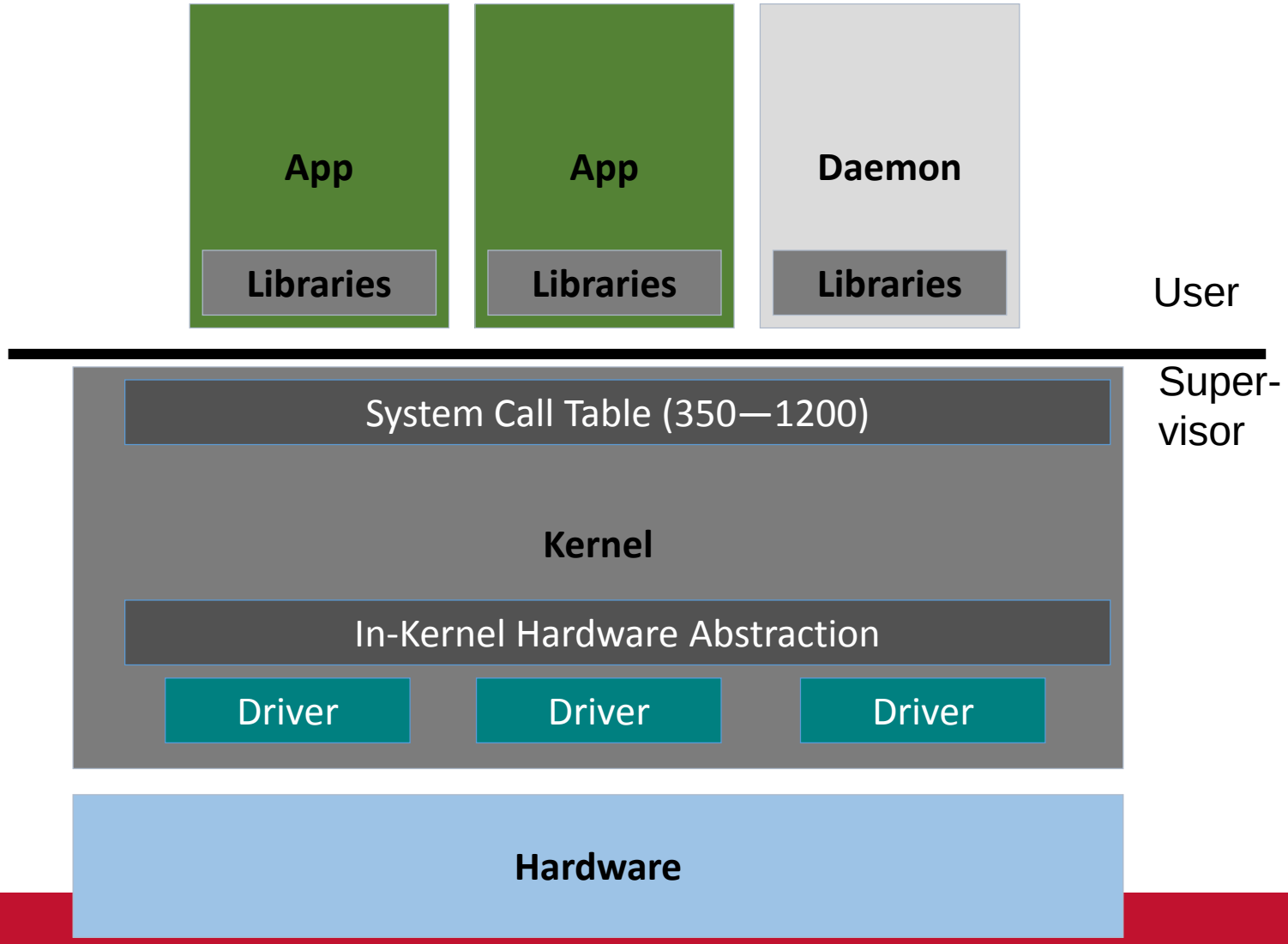


OS = Kernel + System Libraries + System Daemons

In-Kernel Hardware Abstractions

- Kernels are programmed at a higher level of abstraction
  - Block devices (in-kernel abstraction) vs. specific types of disks (real hardware)
- For most types of hardware, the kernel has a "lowest common denominator" interface
  - E.g., Disks, video cards, network cards, keyboard
  - Think Java abstract class
- Each specific device (Nvidia GeForce 600) needs to implement the abstract class
  - Each implementation is called a **device driver**

OS Architecture



So what is Linux?

- Really just an OS kernel
  - Including lots of device drivers
- Conflated with environment consisting of:
  - Linux kernel
  - Gnu libc
  - X window manager daemon
  - CUPS printer manager
  - Etc.

So what is Ubuntu? Centos?

- A **distribution:** bundles all of that stuff together
  - Pick versions that are tested to work together
  - Usually also includes a software update system

OSX vs iOS?

- Same basic kernel (a few different compile options)
- Different window manager and libraries

What is Unix?

- A very old OS (1970s), innovative, still in use
- Innovations:
  - Kernel written in C (first one not in assembly)
    - Co-designed C language with Unix
  - Several nice API abstractions
    - Fork, pipes, everything a file
- Several implementations: *BSDs, Solaris, etc.
  - Linux is a Unix-like kernel

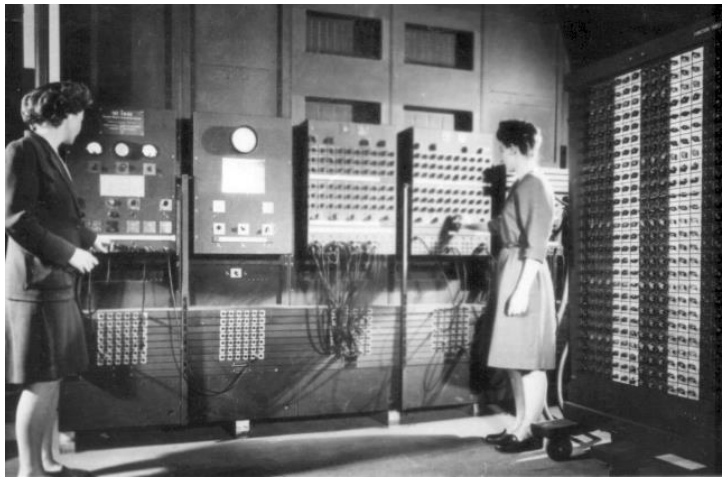
What is POSIX?

- A standard for Unix compatibility
- Even Windows is POSIX compliant!

OS History

1940's – First Computers

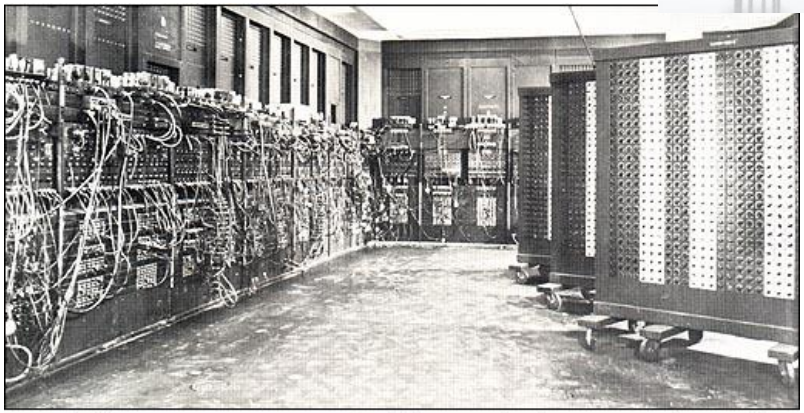
- One user/programmer at a time
- Program loaded manually using switches
  - Debug using the console lights
- ENIAC
  - 1st gen purpose machine
  - Calculations for Army
  - Each panel had specific function



ENIAC (Electronic Number Integrator and Computer)

1940's – First Computers

- Vacuum Tubes and Plugboards
- Single group of people designed, built, programmed, operated and maintained each machine
- No Programming language, only absolute machine language (101010)
- O/S? What is an O/S?
- All programs basically did numerical calculations



Among the first assignments given to Eniac, first all-electronics digital computer, was a knotty problem in nuclear physics. It produced the answer in two hours. One hundred engineers using conventional methods would have needed a year to solve the problem

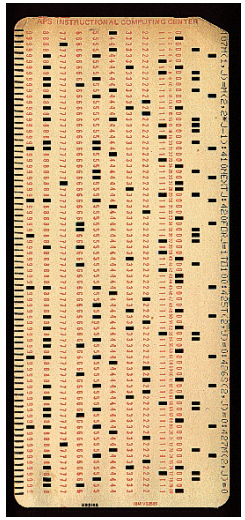
Pros:

- Interactive – immediate response on lights
- Programmers were women ☺

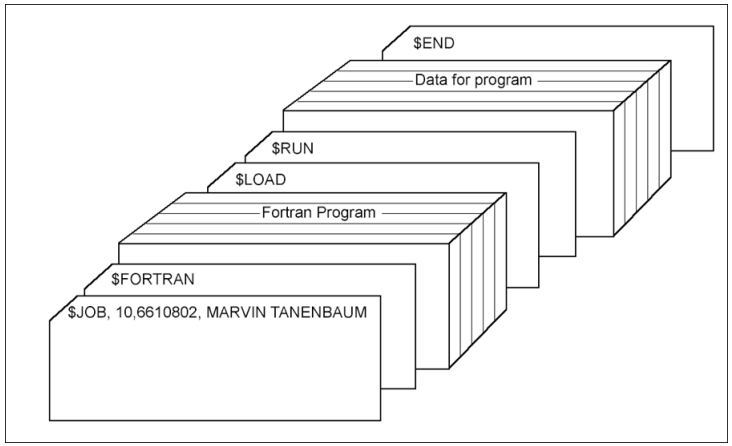
Cons:

- Lots of Idle time
  - Expensive computation
- Error-prone/tedious
- Each program needs all driver code

1950's – Batch Processing



- Deck of cards to describe *job*
- Jobs submitted by multiple users are sequenced automatically by a *resident monitor*
- *Resident monitor* was a basic O/S
  - S/W controls sequence of events
  - Command processor
  - Protection from bugs (eventually)
  - Device drivers



Monitor's Perspective

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

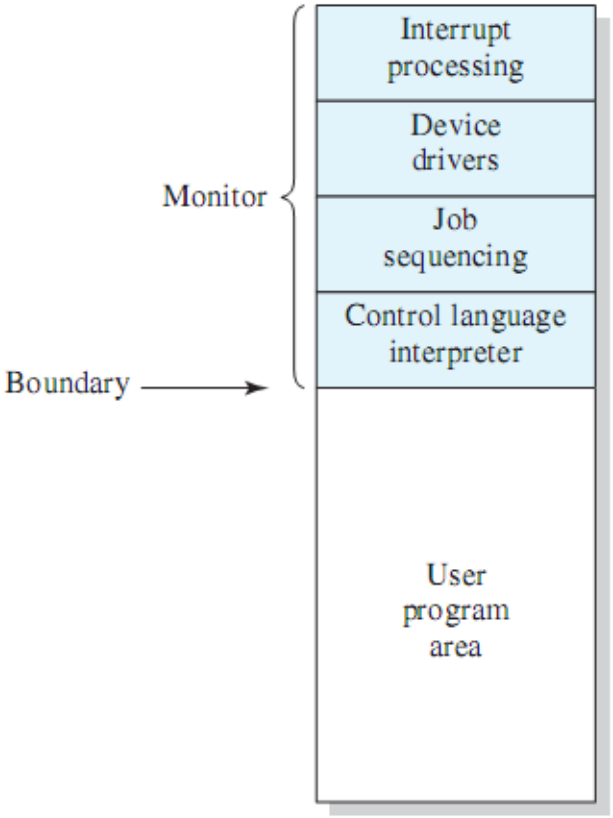
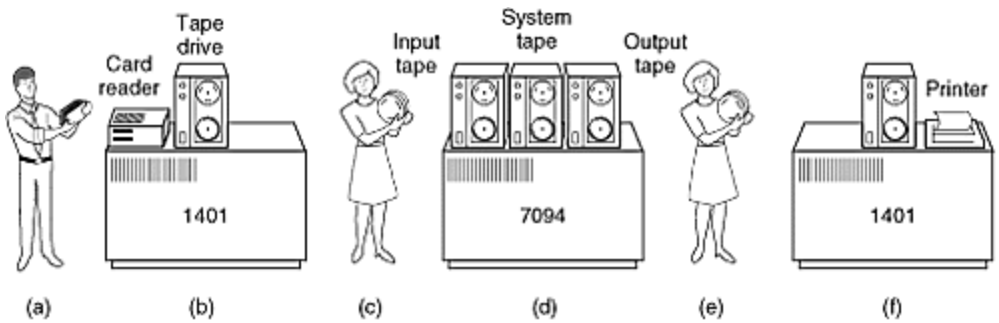


Figure 2.3 **Memory Layout for a Resident Monitor**

1950's – Batch Processing



IBM 7090

Pros:

- CPU kept busy, less idle time
- Monitor could provide I/O services

Cons:

- No longer interactive – longer turnaround time
- Debugging more difficult
- CPU still idle for I/O-bound jobs
- Buggy jobs could require operator intervention

Multiprogrammed Batch Systems

- CPU is often idle
  - Even with automatic job sequencing.
  - I/O devices are slow compared to processor

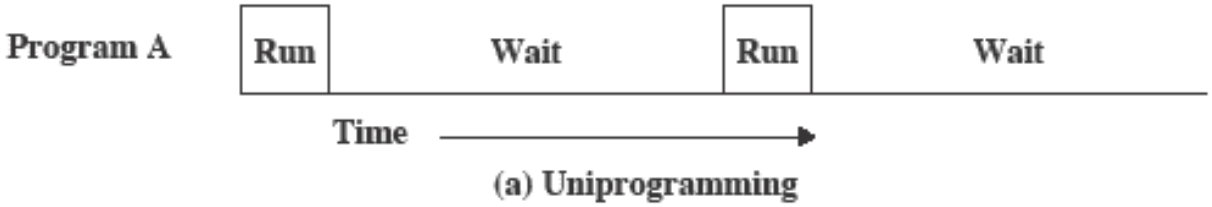
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

Percent CPU Utilization $= \frac{1}{31} = 0.032 = 3.2\%$

Figure 2.4 System Utilization Example

Uniprogramming

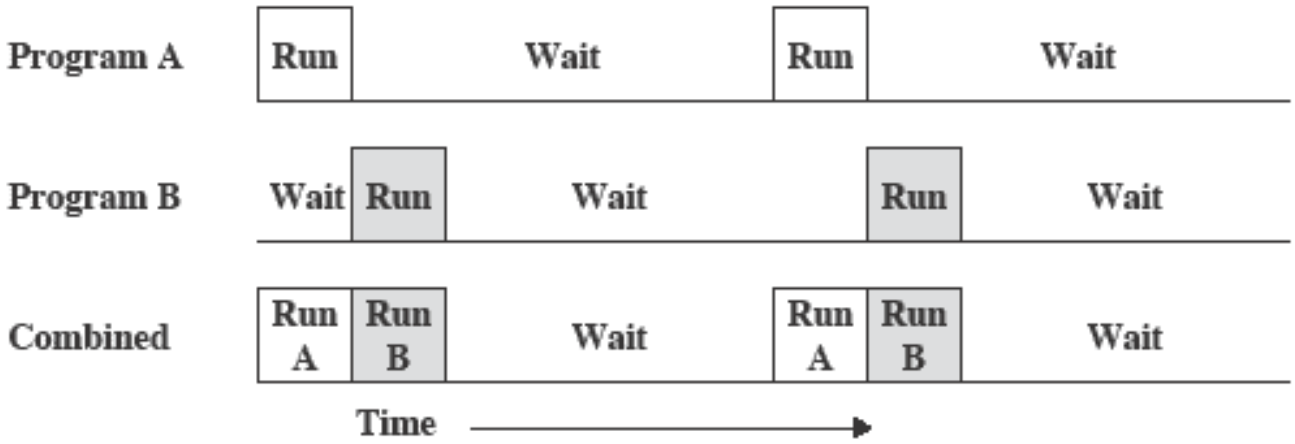
- Processor must wait for I/O instruction to complete before preceding



(a) Uniprogramming

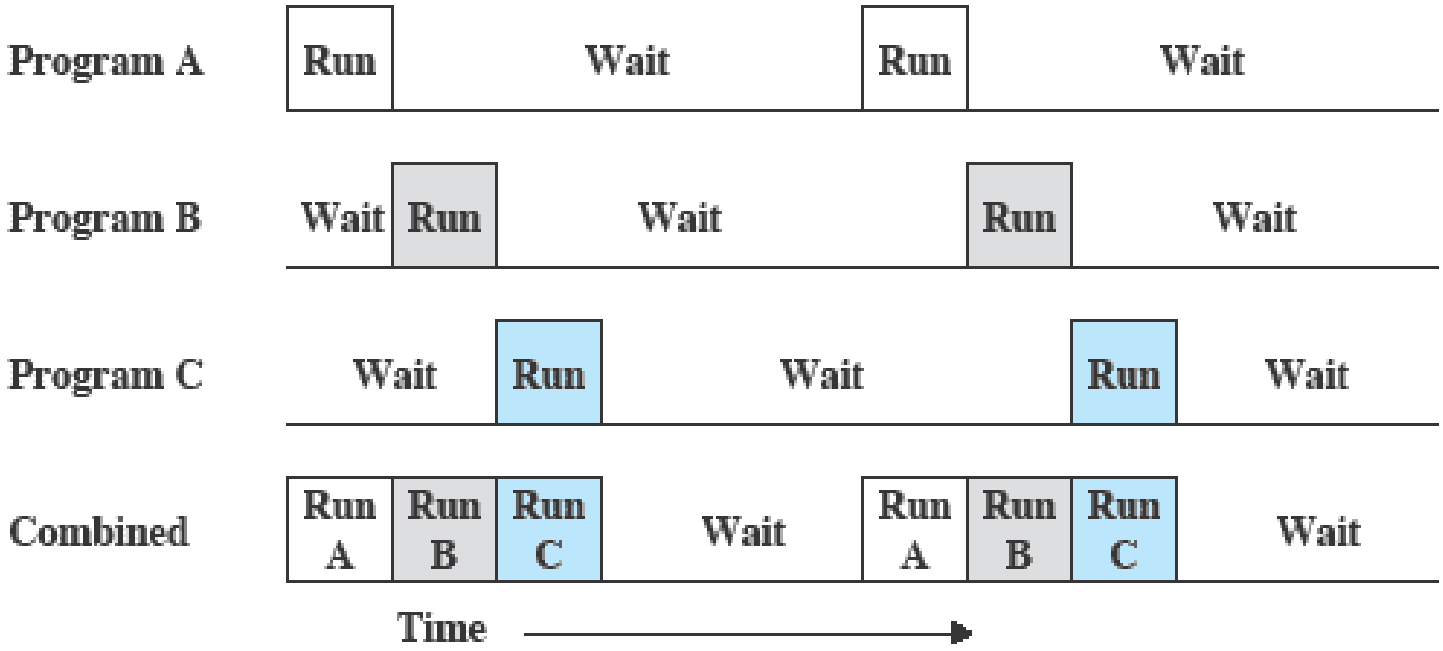
Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs

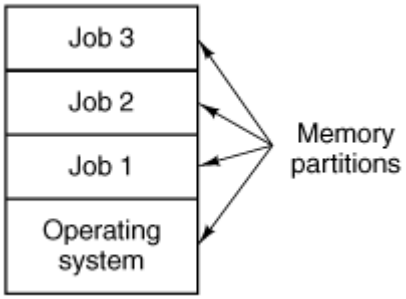
Multiprogramming



(c) Multiprogramming with three programs

1960's – Multiprogramming (time-sharing)

- CPU and I/O devices are multiplexed (shared) between a number of jobs
  - While one job is waiting for I/O another can use the CPU
  - SPOOLing: Simultaneous Peripheral Operation OnLine
    - 1st and simplest multiprogramming system
- Monitor (resembles O/S)
  - Starts job, spools operations, I/O, switch jobs, protection between memory



1960's – Multiprogramming (time-sharing)

IBM System 360

Pros:

- Paging and swapping (RAM)
- Interactiveness
- Output available at completion
- CPU kept busy, less idle time

Cons:

- H/W more complex
- O/S complexity?

1970's - Minicomputers and Microprocessors

- Trend toward many small personal computers or workstations, rather than a single mainframe.
  - Advancement of Integrated circuits
- Timesharing
  - Each user has a terminal and shares a single machine (Unix)

1980's – Personal Computers & Networking

- Microcomputers = PC (size and $)
- MS-DOS, GUI, Apple, Windows
- Networking: Decentralization of computing required communication
  - Not cost-effective for every user to have printer, full copy of software, etc.
  - Rise of cheap, local area networks (Ethernet), and access to wide area networks (Arpanet)

1980's – Personal Computers & Networking

- OS issues:
  - Communication protocols, client/server paradigm
  - Data security, encryption, protection
  - Reliability, consistency, availability of distributed data
  - Heterogeneity
  - Reducing Complexity
- Ex: Byte Ordering

1990's – Global Computing

- Dawn of the Internet
  - Global computing system
- Powerful CPUs cheap! Multicore systems
- High speed links
- Standard protocols (HTTP, FTP, HTML, XML, etc)
- OS Issues:
  - Communication costs dominate
    - CPU/RAM/disk speed mismatch
    - Send data to program vs. sending program to data
  - QoS (Quality of Service) guarantees
  - Security

2000's – Embedded and Ubiquitous Computing

- Mobile and wearable computers

- Networked household devices
- Absorption of telephony, entertainment functions into computing systems

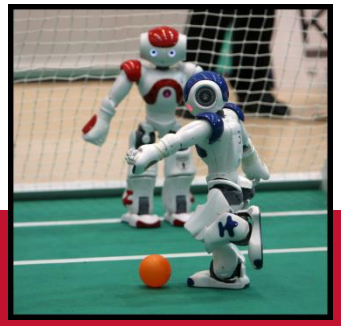
- OS issues:
  - Security, privacy
  - Mobility, ad-hoc networks, power management
  - Reliability, service guarantees

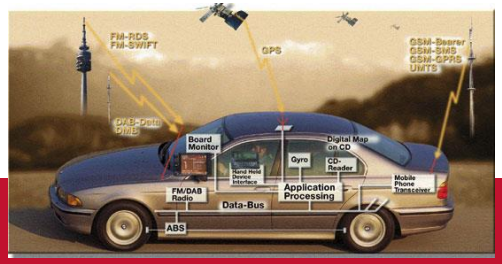
2000's – Embedded and Ubiquitous Computing

- Real-time computing
  - Guaranteed upper bound on task completion

- Dedicated computers/Embedded systems
  - Application specific, designed to complete particula tasks

- Distributed systems
  - Redundant resources, transparent to user





Multi-core

- New hotness in CPU design. Not going away.
  - Why?
- Being able to program with threads and concurrent algorithms will be a crucial job skill going forward
  - Don't leave SBU without mastering these skills
  - We will do some advanced multi-threaded programming in the labs

OS History Summary

- OS's began with big expensive computers used interactively by one user at a time.
- Batch systems sequences jobs to keep computer busier. Interactivity sacrificed.
- Multiprogramming developed to make more efficient use of expensive hardware and restore interactivity.
- Cheap CPU/memory/storage make communication the dominant cost.
- Multiprogramming still central for handling concurrent interaction with environment.