Introduction

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

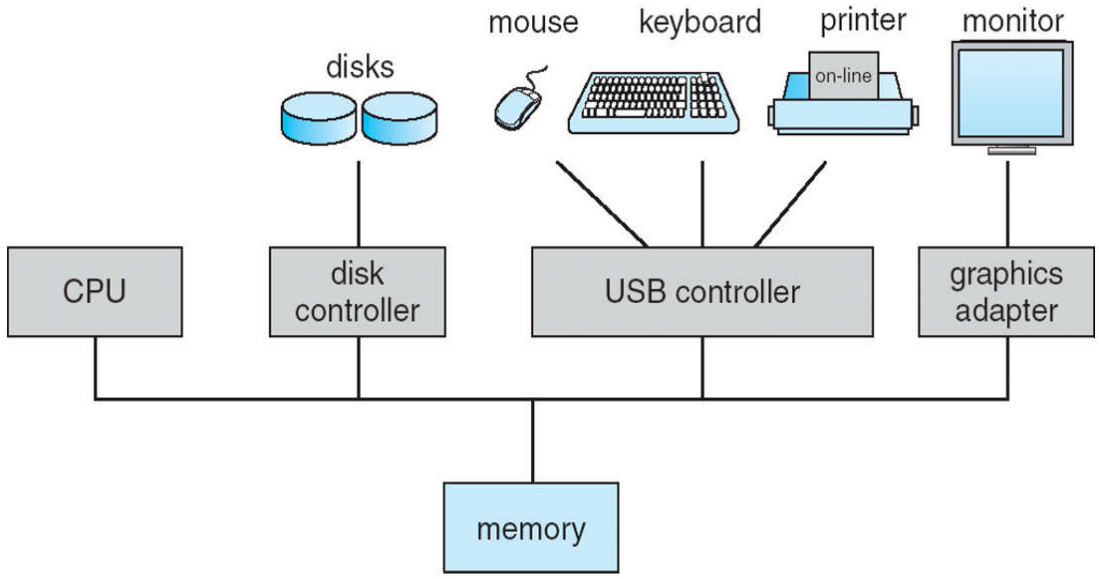
Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

Content

- What Operating Systems Do
- Computer System Components
- Views of Operating Systems
- Evolution of Operating Systems

What Operating Systems Do

- A modern computer consists of:
  - One or more processors
  - Main memory
  - Mass storage
  - Various input/output devices
    - Printers, mice, keyboards, monitors, etc.
    - Network adapter, etc.



Computer Hardware Organization

What Operating Systems Do

- An Operating System is a program that acts as an *intermediary* or *interface* between a user of a computer and the computer hardware.
  - convenient *abstraction* of complex hardware devices
  - *protection* of access to shared resources
    - *security* and *authentication*
  - *communication* amongst logical entities
- An Operating System goals at:
  - control/execute user or application programs
  - make the computer system convenient to use
  - ease the solving of user problems
  - use the computer hardware in an efficient manner

What Operating Systems Do

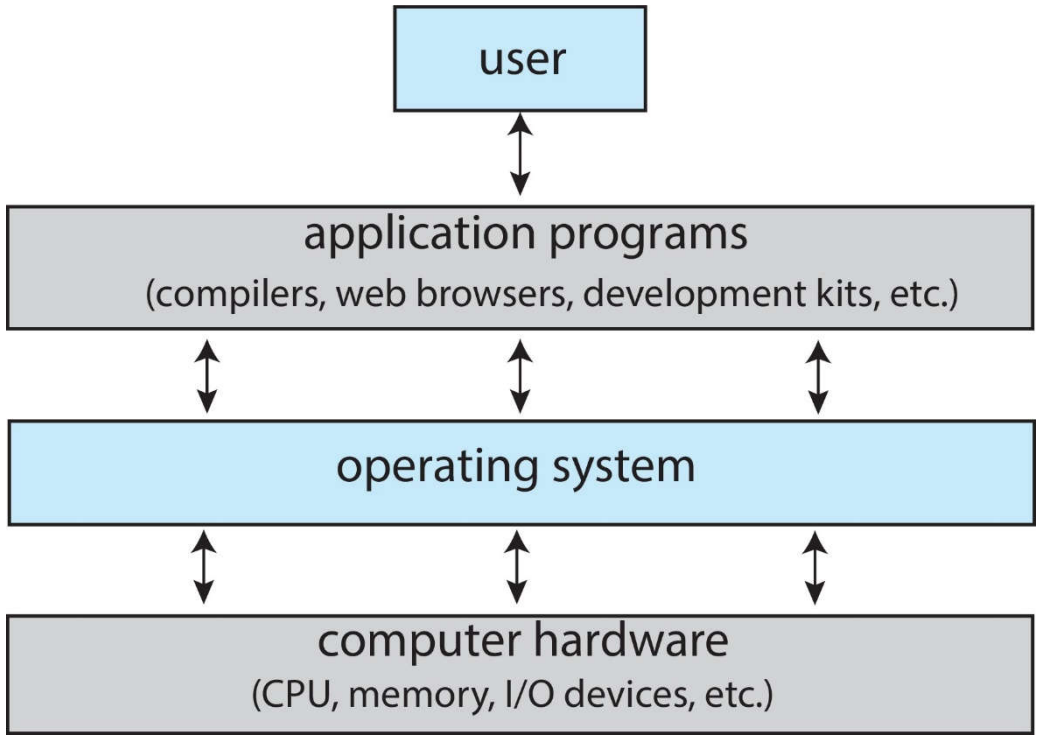
- What an Operating System needs to do depends on the point of view.
  - Users want convenience, ease of use and good performance.
    - They don’t care about resource utilization (资源利用率).
    - A shared computer such as mainframe or minicomputer must keep all users happy.
  - Users of dedicated (专用) systems such as workstations have dedicated resources but frequently use shared resources from servers.
  - Handheld computers are resource poor, optimized for usability and battery life.
  - Some computers have little or no user interface, such as embedded computers in devices and automobiles.

Computer System Components

- Hardware
  - provides basic computing resources
    - CPU, Memory, I/O devices, Communication.
- Operating System
  - controls and coordinates use of the hardware among various application programs for various users
- Application Programs (Application Systems)
  - ways in which the system resources are used to solve computing problems of the users
    - Word processors, Compilers, Web browsers, Database systems, Video games.
- Users
  - People, Machines, Other computers.

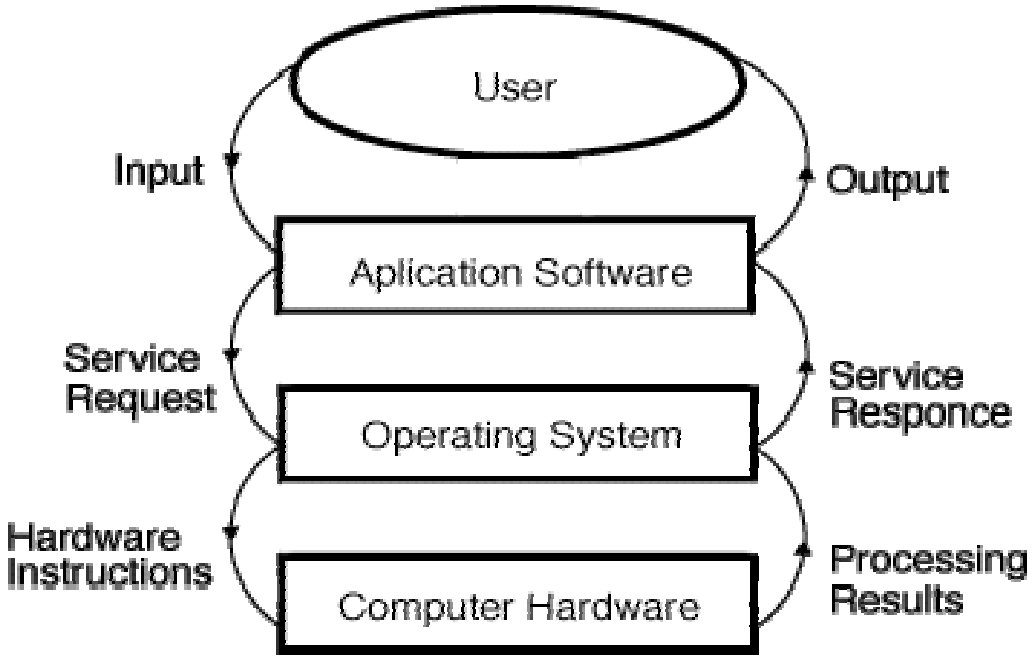
Computer System Components

- Abstract view of the components of a computer system.



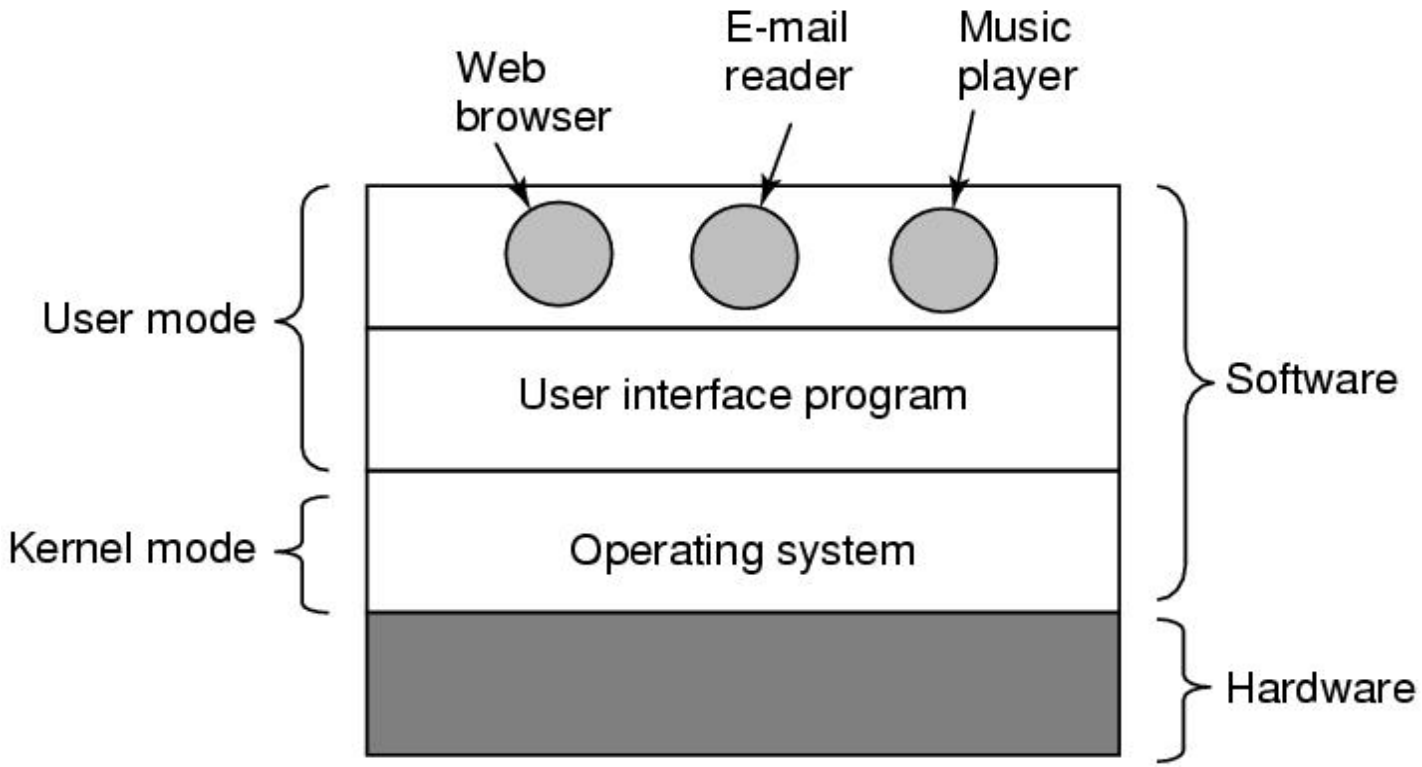
Computer System Components

- A Dynamic View.



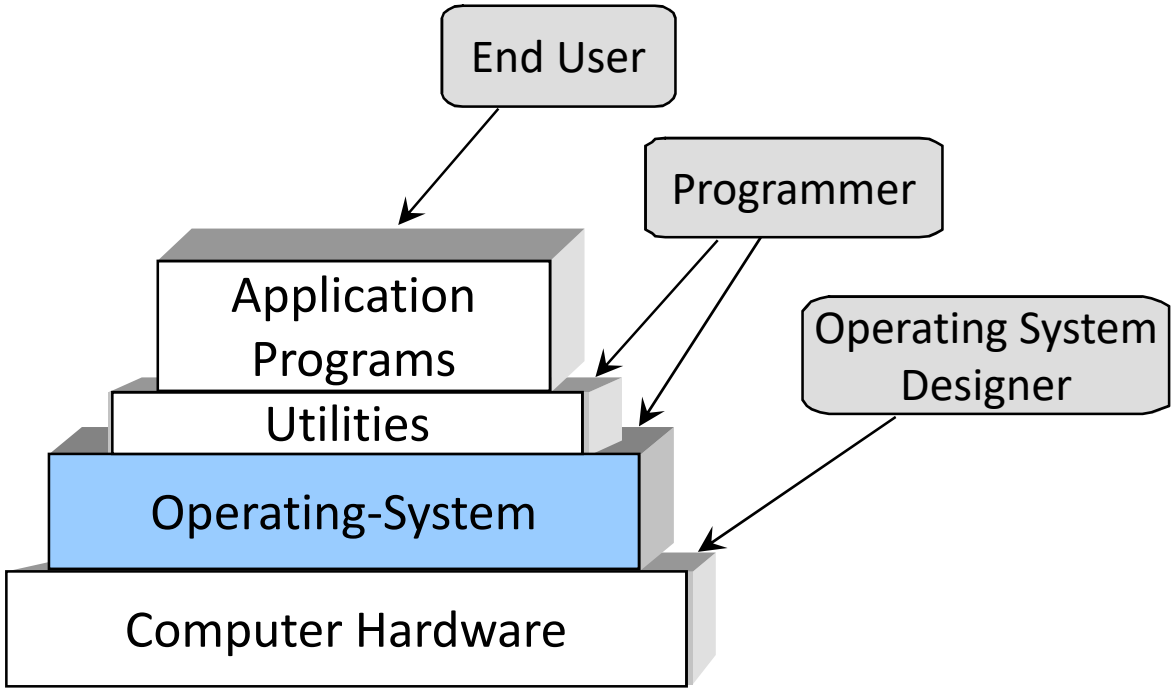
Computer System Components

- Schematic diagram of a modern computer system.



Computer System Components

- Layers of a Computer System.



Views of Operating Systems

- Provide consistent abstractions to apps, even on diverse hardware
  - File systems, Window Systems, Communications, …
  - Processes, threads
  - VMs, containers
  - Naming systems
- Manage resources shared among multiple applications
  - Memory, CPU, storage, …
- Achieved by specific algorithms and techniques
  - Scheduling
  - Concurrency
  - Transactions
  - Security
- Across immense scale – from 1's to Billions

Views of Operating Systems

- Services provided by an Operating System
  - Facilities for program creation
    - editors, compilers, linkers, debuggers, etc.
  - Program execution
    - loading in memory, I/O and file initialization.
  - Access to I/O and files
    - deals with the specifics of I/O and file formats.
  - System access
    - resolves conflicts for resource contention (解决资源冲突)
    - protection in access to resources and data

Views of Operating Systems

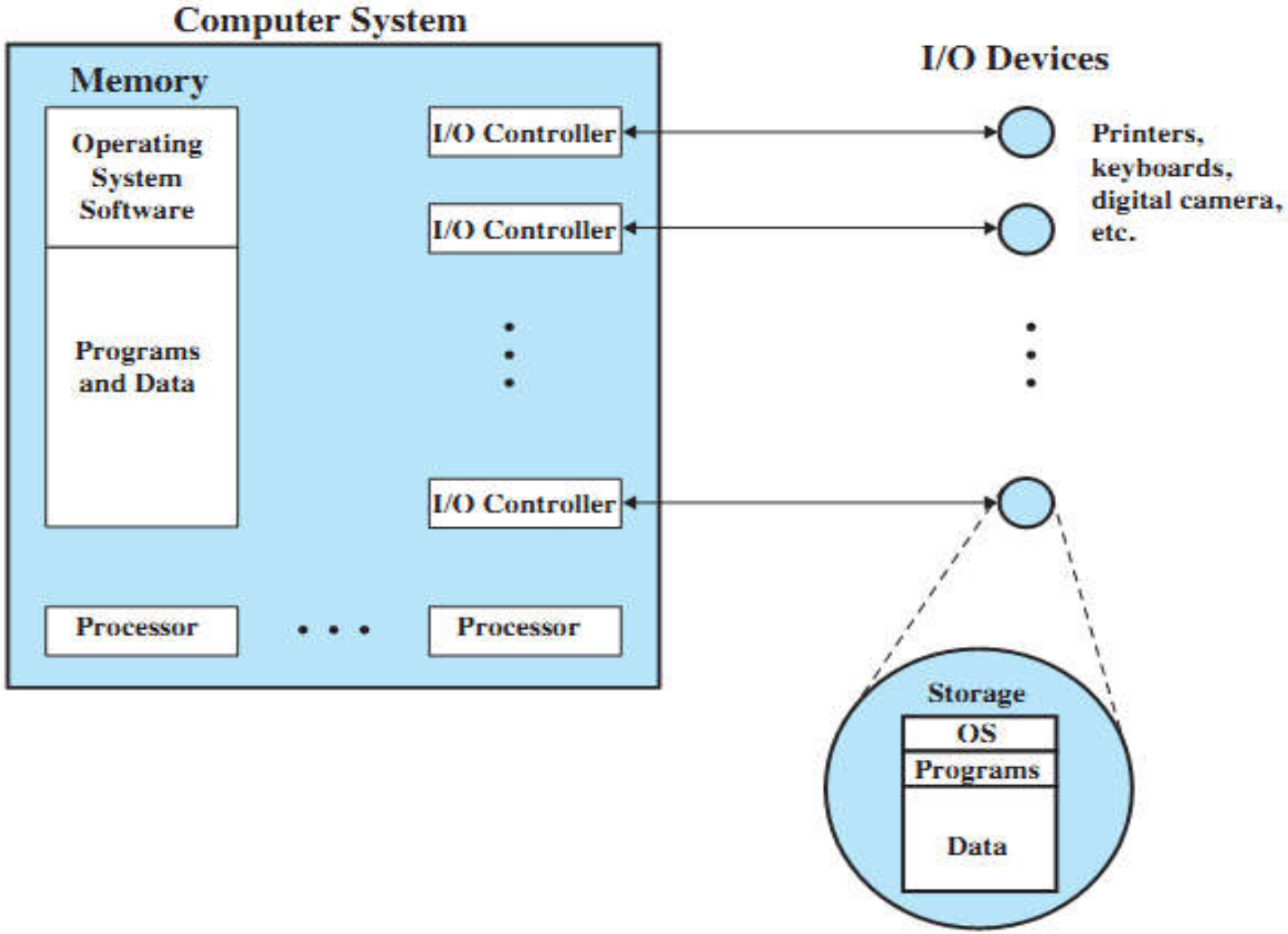
- There are three classical views of Operating Systems
  - Resource Manager
    - manages and allocates resources
      - sort of a bottom-up view
  - Control Program
    - controls the execution of user programs and operations of I/O devices
      - sort of a black box view
  - Command Executer
    - provides an environment for running user commands
      - sort of a top-down view
- One modern view of Operating Systems
  - Virtual Machine

Views of Operating Systems

- OS as a Resource Manager
  - manages and protects multiple computer resources
    - CPU, Processes, Internal/External memory, Tasks, Applications, Users, Communication channels, etc.
  - handles and allocates resources to multiple users or multiple programs running at the same time and space
    - e.g., processor time, memory, I/O devices
  - decides between conflicting requests (发生冲突的资源请求) for efficient and fair resource use
    - e.g., maximize throughput, minimize response time
  - Some OS named after the Resource Manager view
    - DEC RSX – Resource Sharing eXecutive
    - MIT Multics – MULTiplexed Information & Computing Services
    - IBM MFT/MVT – Multiple Fixed/Variable Tasks
    - IBM MVS – Multiple Virtual Storage
    - DEC VMS – Virtual Memory System
    - MVS TSO – Time Sharing Option
    - CTSS – Compatible Time Sharing System

Views of Operating Systems

- OS as a Resource Manager



Views of Operating Systems

- OS as a Control Program
  - manages all the components of a complex computer system in an integrated manner
  - controls the execution of user programs and I/O devices to prevent errors and improper use of computer resources
  - looks over and protects the computer
    - Monitor, Supervisor, Executer, Controller, Master, Coordinator, …
  - Some OS named after the Control Program view
    - Unisys MCP – Master Control Program
    - DR CP/M – Control Program/Microcomputer
    - IBM VM/CP – VM Control Program
    - IBM AIX – Advanced Interactive eXecutive
    - DEC RSX – Resource Sharing eXecutive

Views of Operating Systems

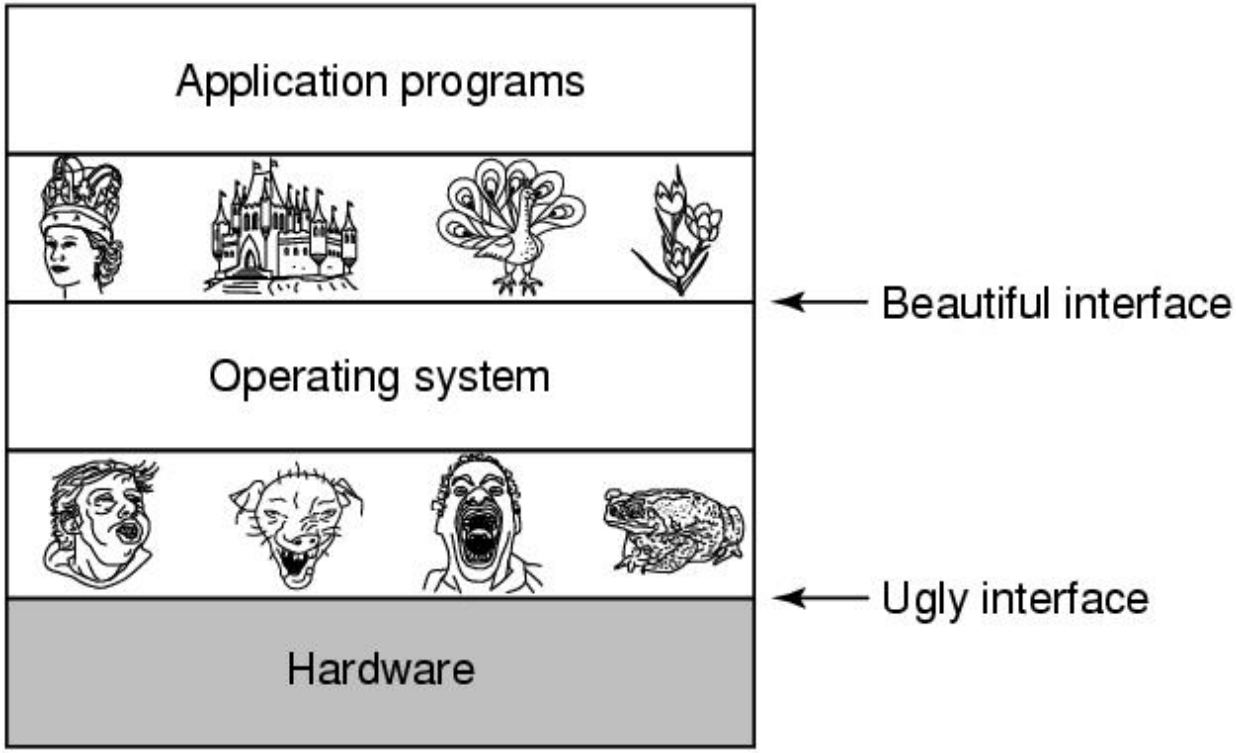
- OS as a Command Executer
  - interfaces between the users and machine
  - supplies services/utilities to users
  - provides the users with a convenient CLI (Command Line/Language Interface, or Command Interpreter), also called a Shell in UNIX, for entering the user commands
  - some OS named after the Command Executer view
    - IBM AIX – Advanced Interactive Executive
    - IBM VM/CMS – Conversational Monitor System

Views of Operating Systems

- OS as a Virtual Machine
  - acts as an interface between the user and hardware that hides the details of the hardware
    - hardware like I/O devices.
  - constructs higher-level (virtual) resources out of lower-level (physical) resources
    - e.g., files
  - OS is defined as a collection of software enhancements, executed on the bare hardware, culminating in (达到) a high-level virtual machine that serves as an advanced programming environment.
    - virtual machine = software enhancement
      = extended machine
      = abstract machine.

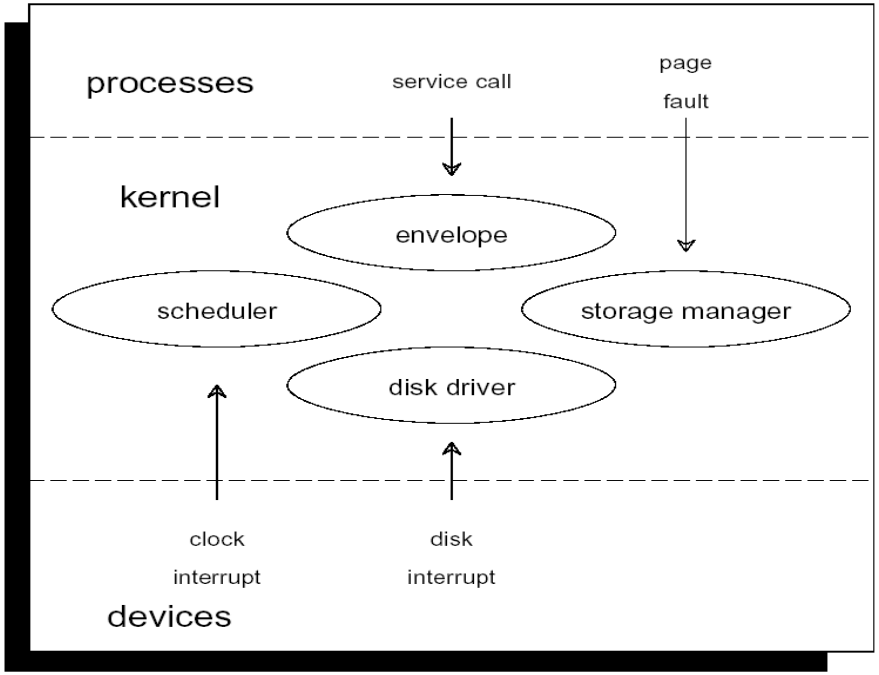
Views of Operating Systems

- OS as a Virtual Machine



Definition of Operating System

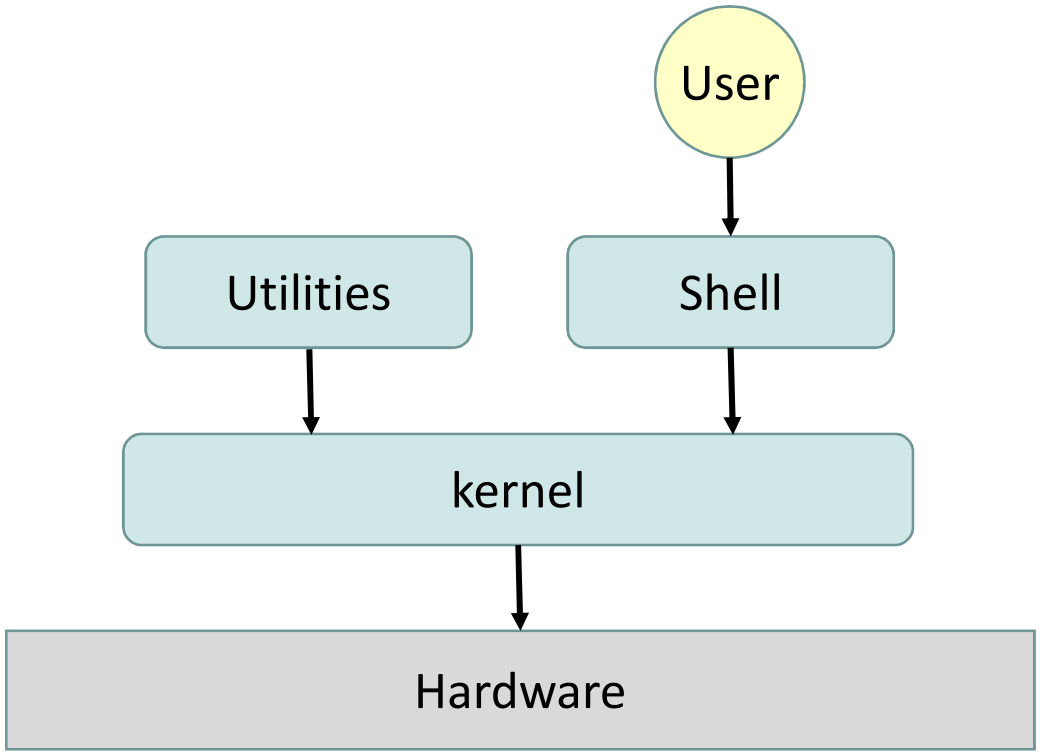
- There is no universally accepted definition for Operating System.
  - "Everything a vendor ships when you order an operating system" is good approximation but varies widely.
  - "The one program running at all times on the computer" is only the *Kernel*.
    - Everything else is either a system program (utilities, ships with the operating system) or an application program.



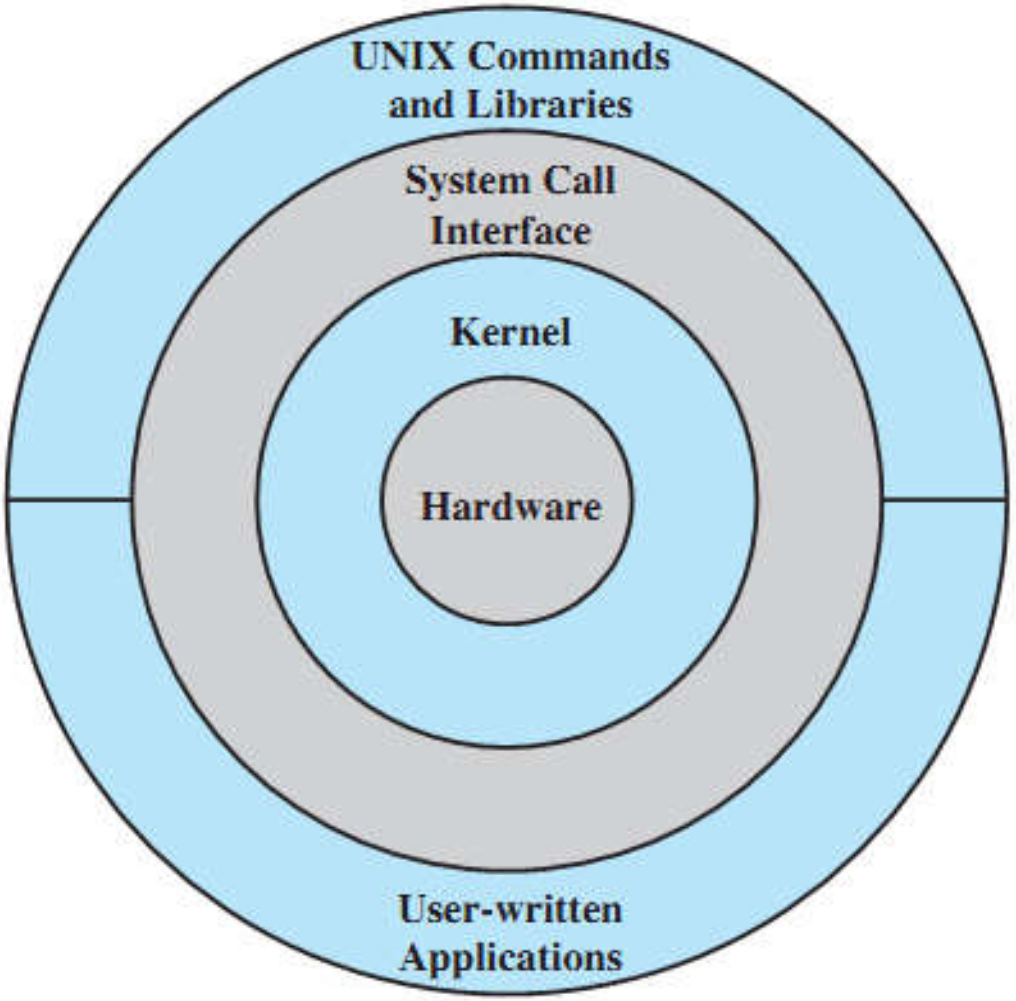
Definition of Operating System

- The Command Line Interface (CLI) allows direct command entry by the user.
  - sometimes implemented in kernel, sometimes by systems program
  - sometimes multiple flavors implemented – shells
  - primarily fetches a command from user and executes it
  - sometimes commands built-in, sometimes just names of programs; if the latter, adding new features doesn't require shell modification.
- The shell used to be in the kernel but now is a (first among equals) utility outside of it.
  - Easy to change/debug
  - Many of them
    - sh, bsh, csh, ksh, tcsh, wsh, bash.
  - Possible to switch between them
    - chsh.

UNIX Shell and Utilities

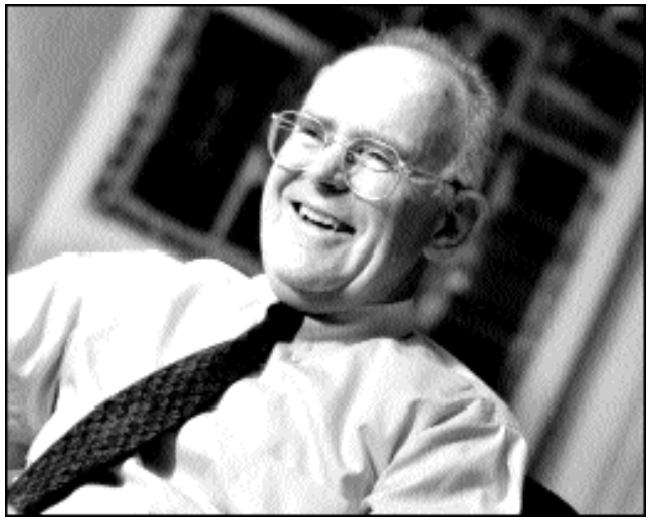


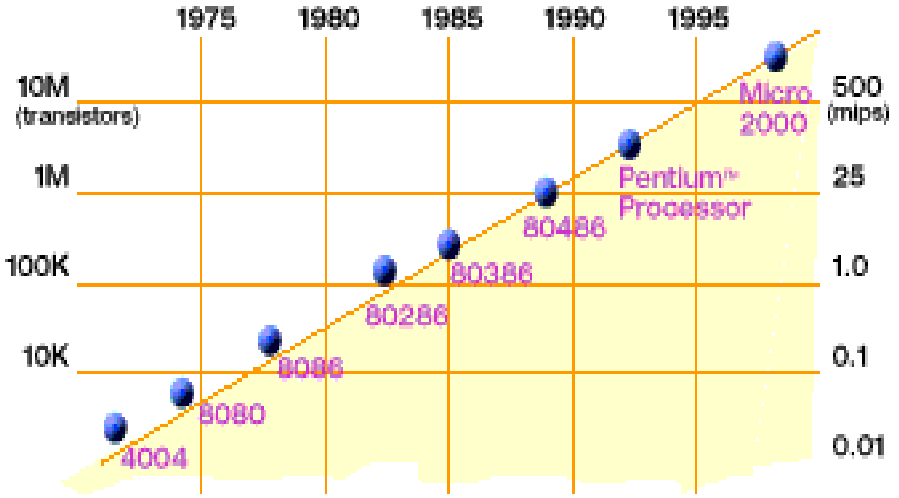
General UNIX Architecture



Moore's Law

- *Gordon Moore* (co-founder of Intel) predicted in 1965 that the transistor *density* of semiconductor chips would *double roughly every 18 months*.
  - Microprocessors have become smaller, denser, and more powerful.
- *Moore*'s Law has (officially) ended -- Feb 2016.
  - No longer getting 2 x transistors/chip every 18 months …
    - or even every 24 months
  - Vendors moving to 3D stacked chips (堆叠芯片)



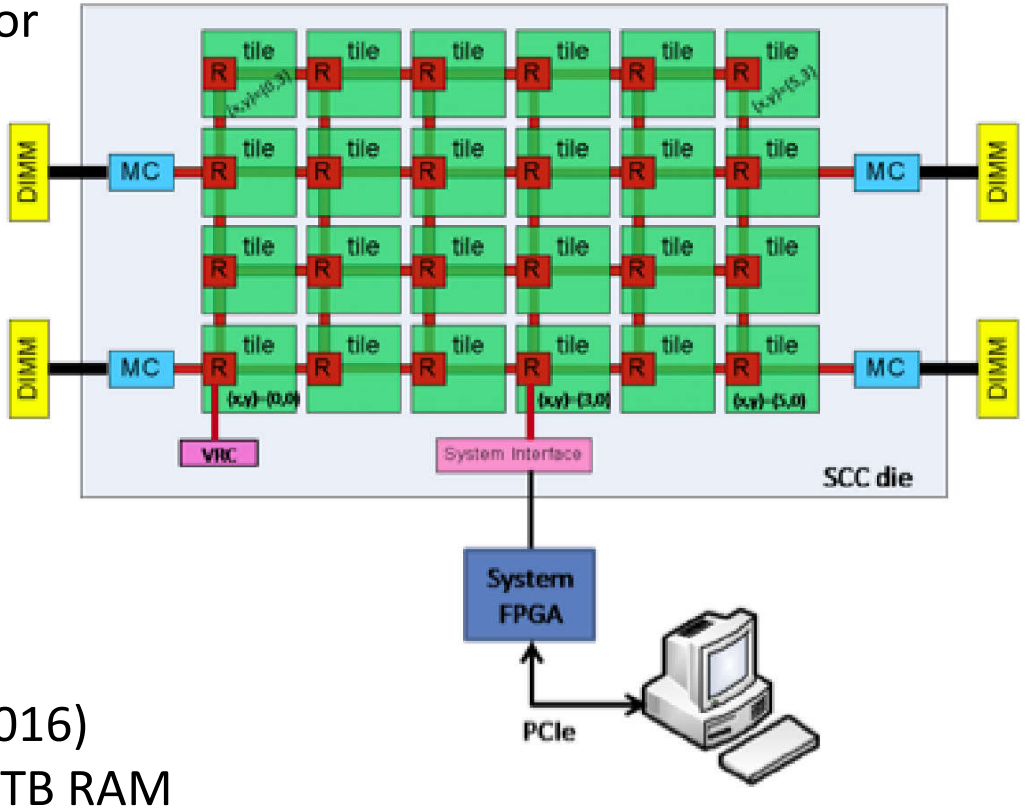


Many-core Chips

- "Many-Core" refers to many processors per chip
  - 64? 128? Hard to say exact boundary
- Intel 80-core multicore chip (in Feb 2007)
  - 80 simple cores
  - Two FP-engines / core
  - Mesh-like network
  - 100 million transistors
- How to program with many-core chips?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - Use 76 for virus checking?
- Parallelism must be exploited at all levels.

Many-core Chips

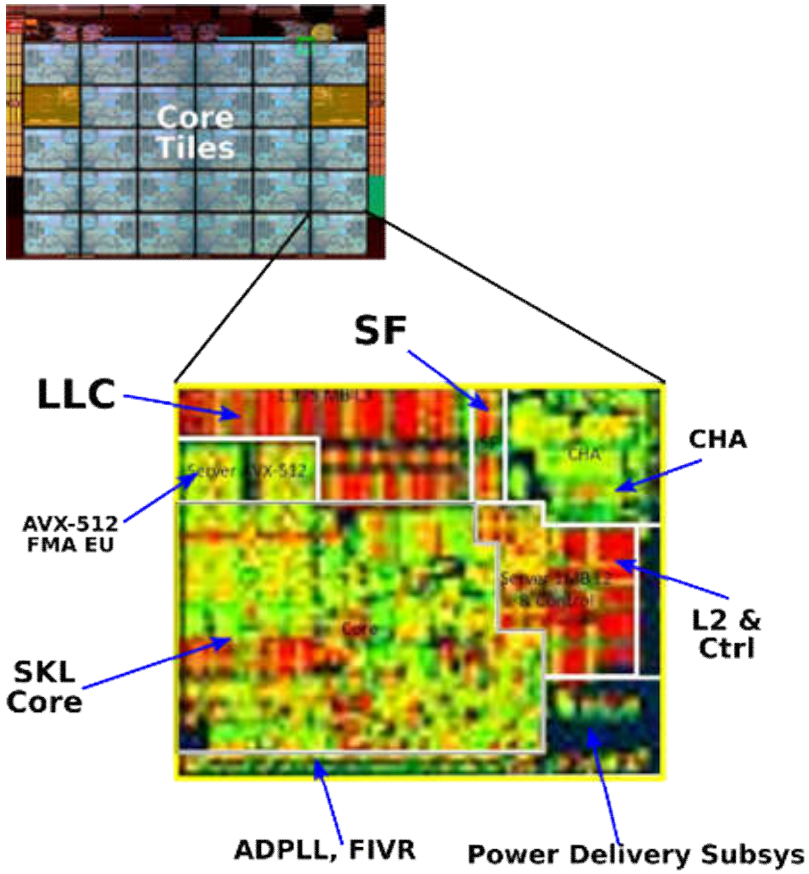
- Intel Single-Chip Cloud Computer (SCC in August 2010)
  - 24 “tiles” with two cores per tile (48 cores)
  - 24-router mesh network
  - 4 DDR3 memory controllers
  - hardware support for message-passing



- Amazon X1 instances (2016)
  - 128 virtual cores, 2 TB RAM

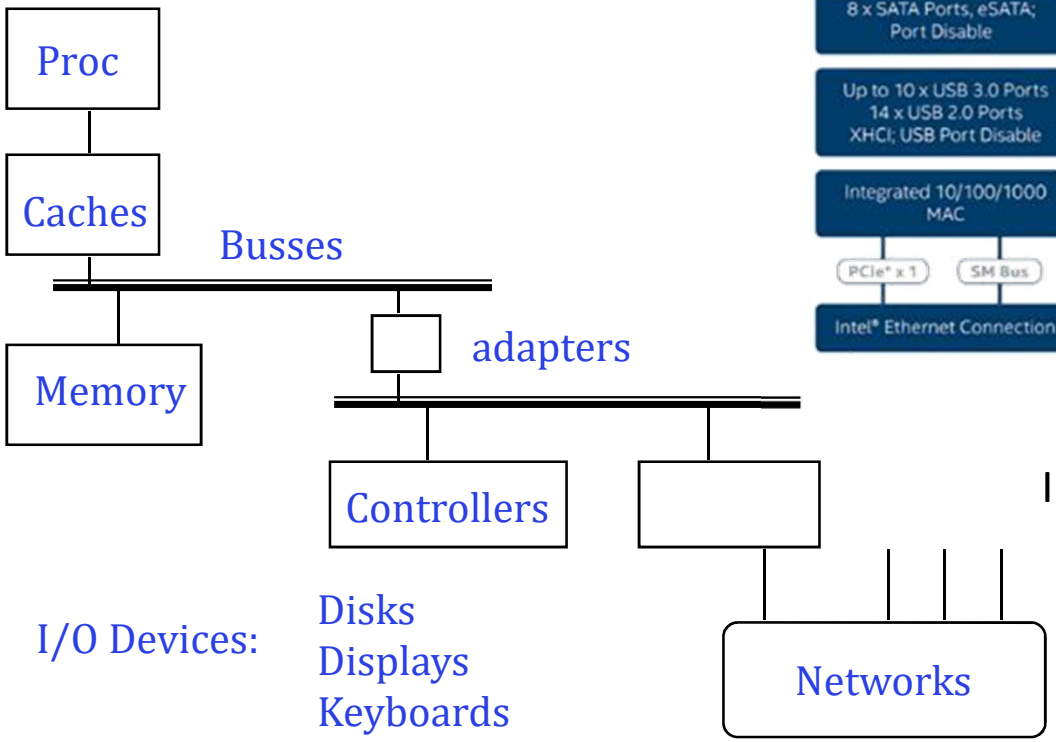
The World Is Parallel: Intel SkyLake (2017)

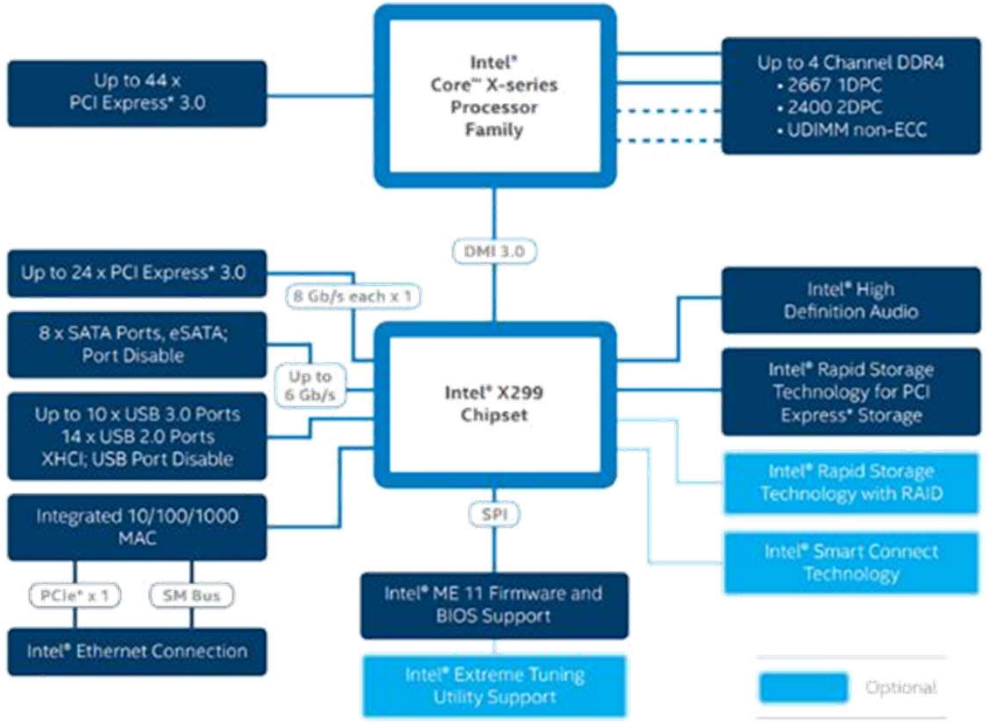
- Up to 28 Cores, 58 Threads
  - 694 mm² die size (estimated)
- Many different instructions
  - Security, Graphics
- Caches on chip:
  - L2: 28 MiB
  - Shared L3: 38.5 MiB (non-inclusive)
  - Directory-based cache coherence
- Network:
  - On-chip Mesh Interconnect
  - Fast off-chip network directly supports 8-chips connected
- DRAM/chips
  - Up to 1.5 TiB
  - DDR4 memory



The World Is Parallel: Intel SkyLake (2017)

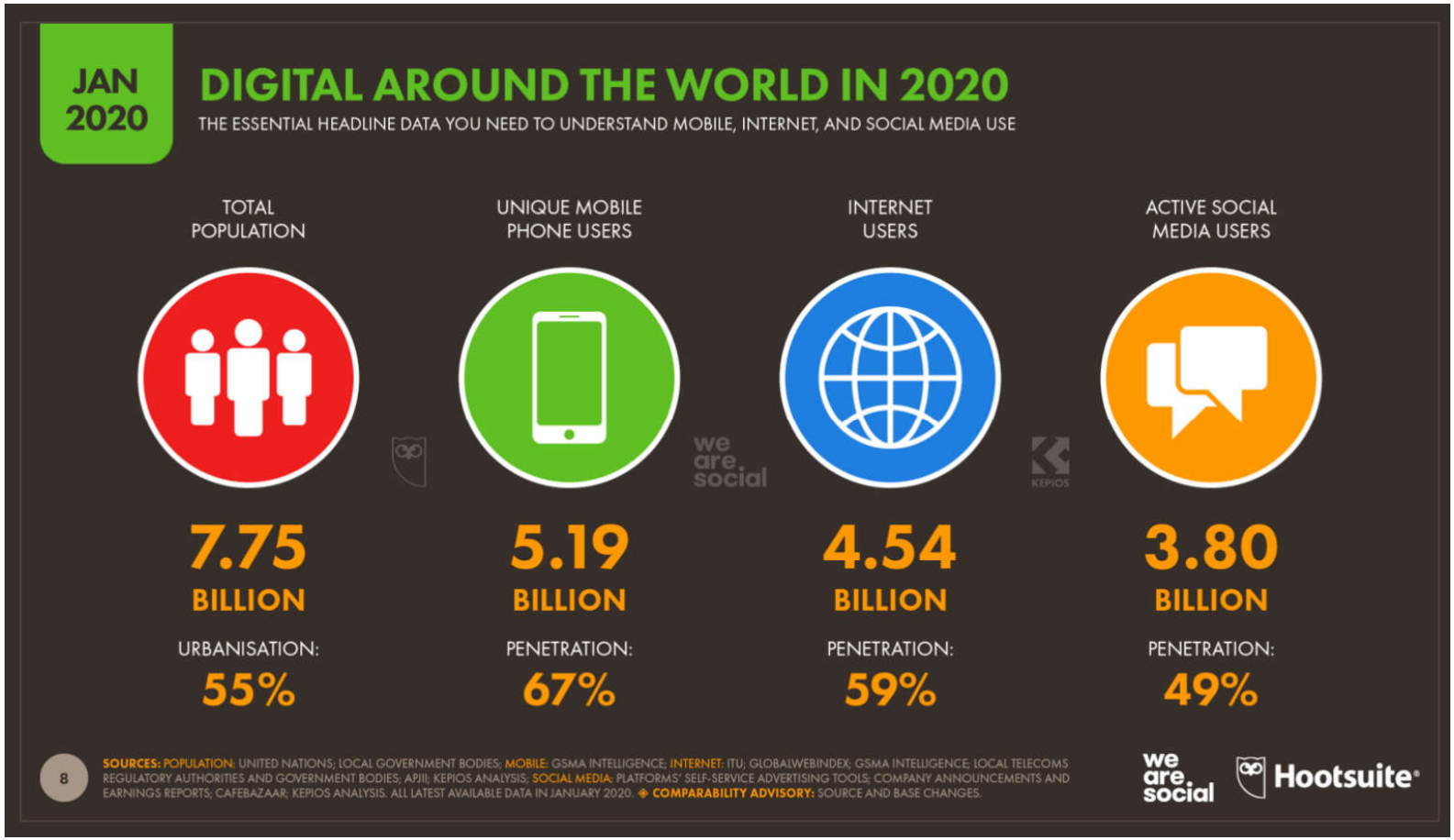
- Hardware functionality comes with great complexity.



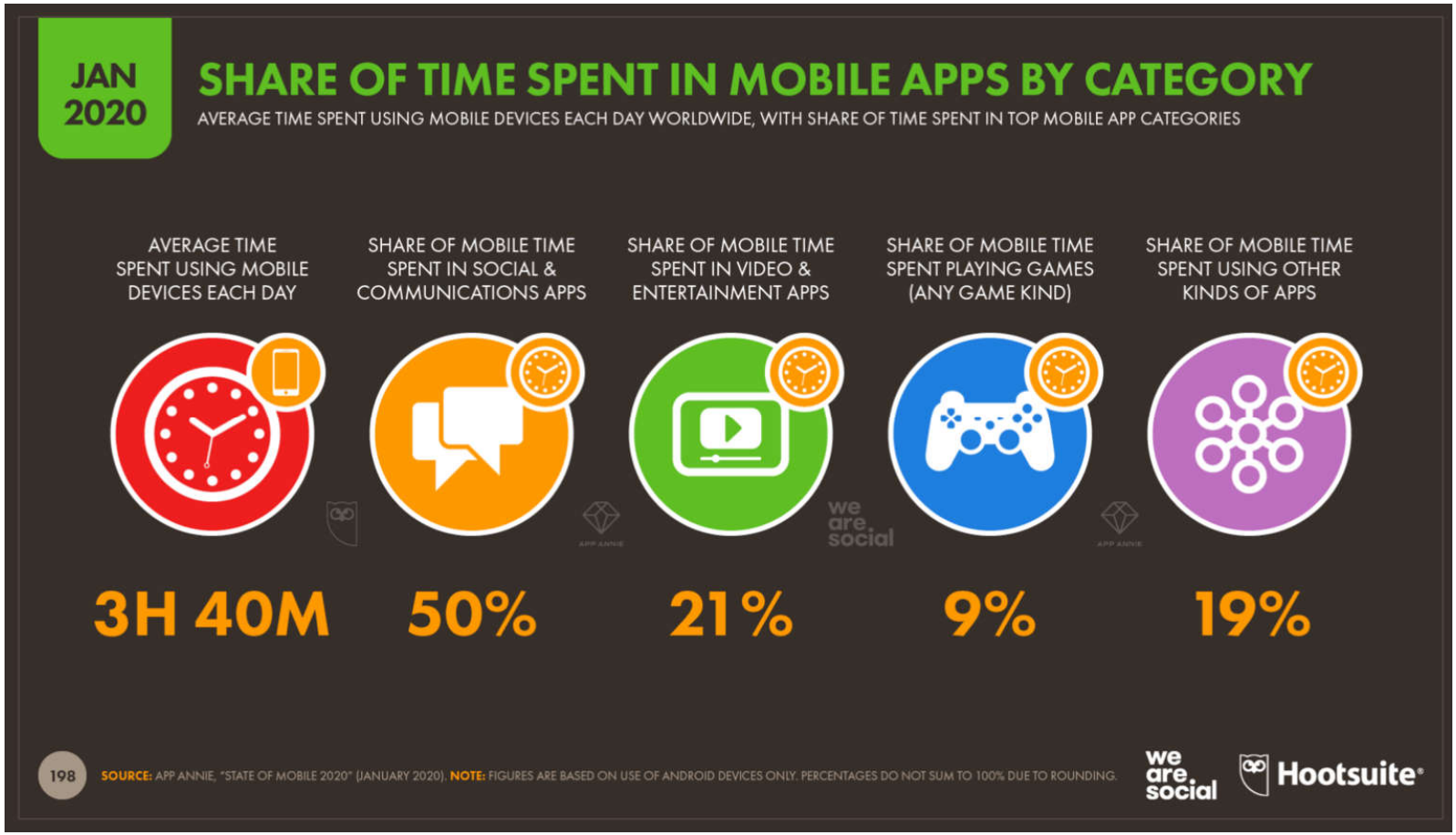


Intel Skylake-X I/O Configuration

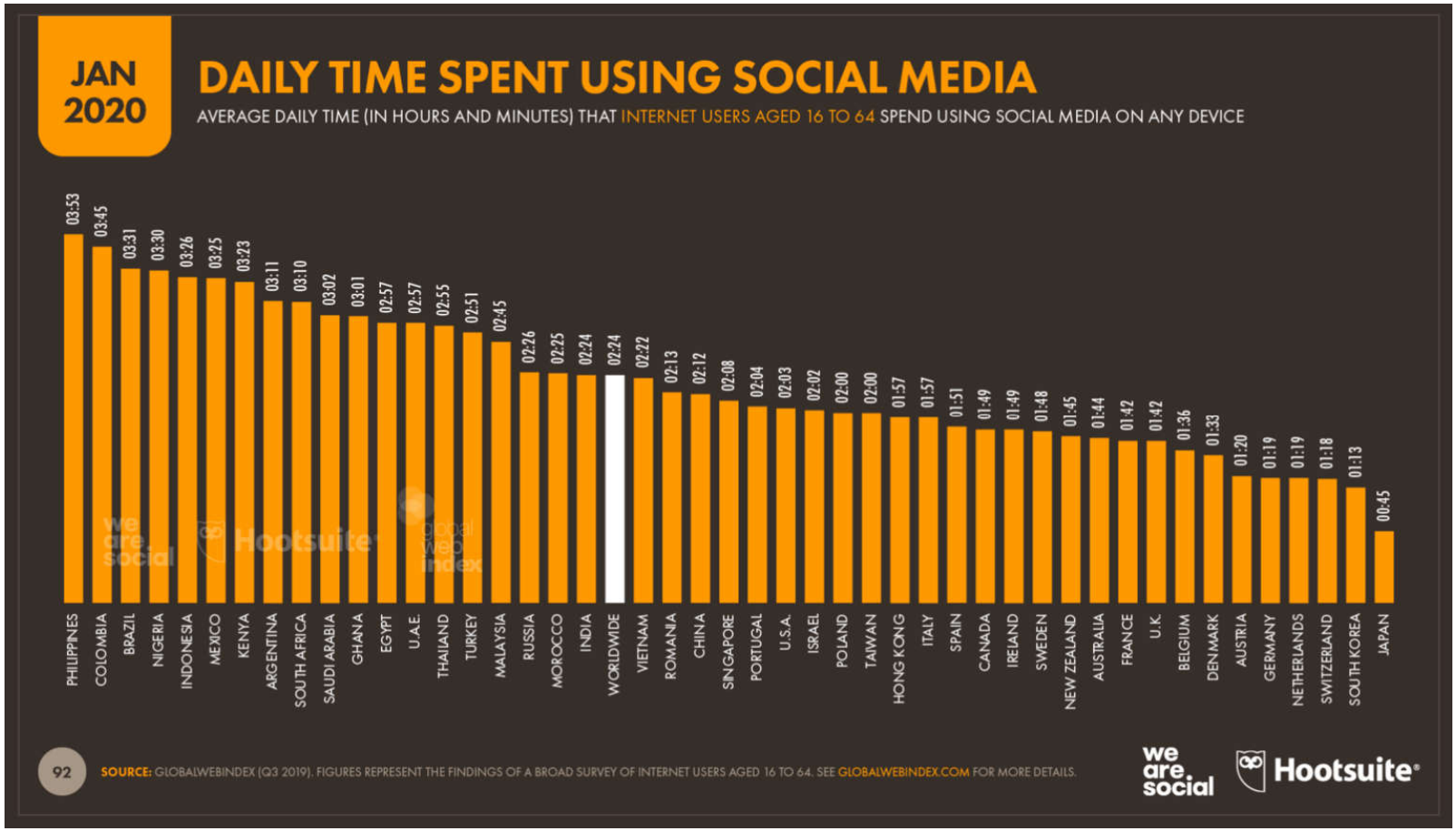
Society Connected



Society Connected



Society Connected

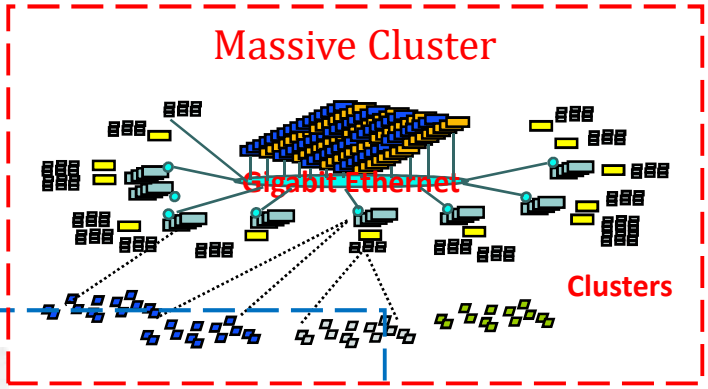


Society Connected

- MISR-2018, ITU 2019

Societal Scale Information Systems

- The world is a large distributed system
  - Microprocessors in everything
  - Vast infrastructure behind them



Internet Connectivity

MEMS for Sensor Nets

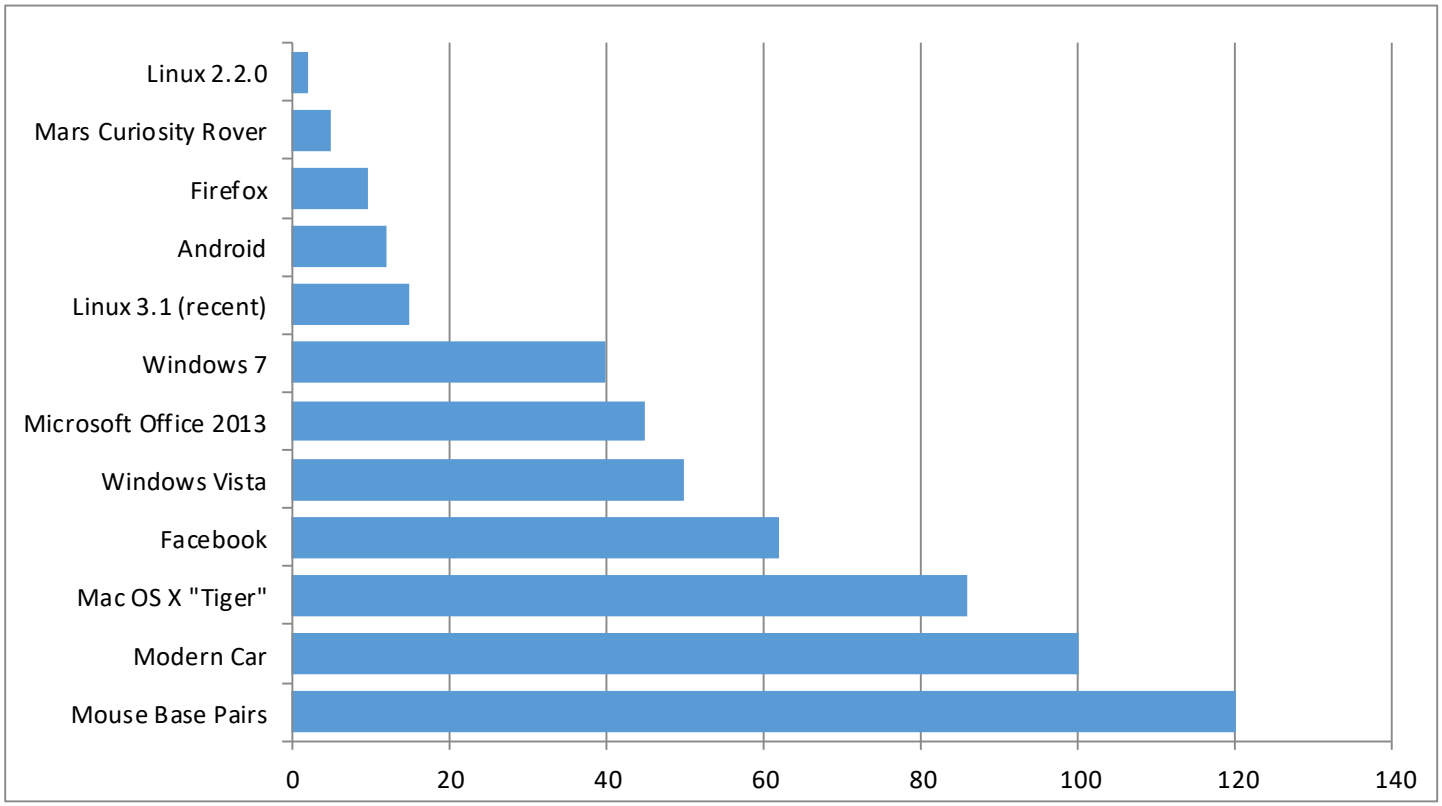
Databases
Information Collection
Remote Storage
Online Games
Commerce
...

Challenge of Complexity

- Applications consisting of
  - … a variety of software modules that …
  - … run on a variety of devices (machines) that
    - … implement different hardware architectures
    - … run competing applications
    - … fail in unexpected ways
    - … can be under a variety of attacks

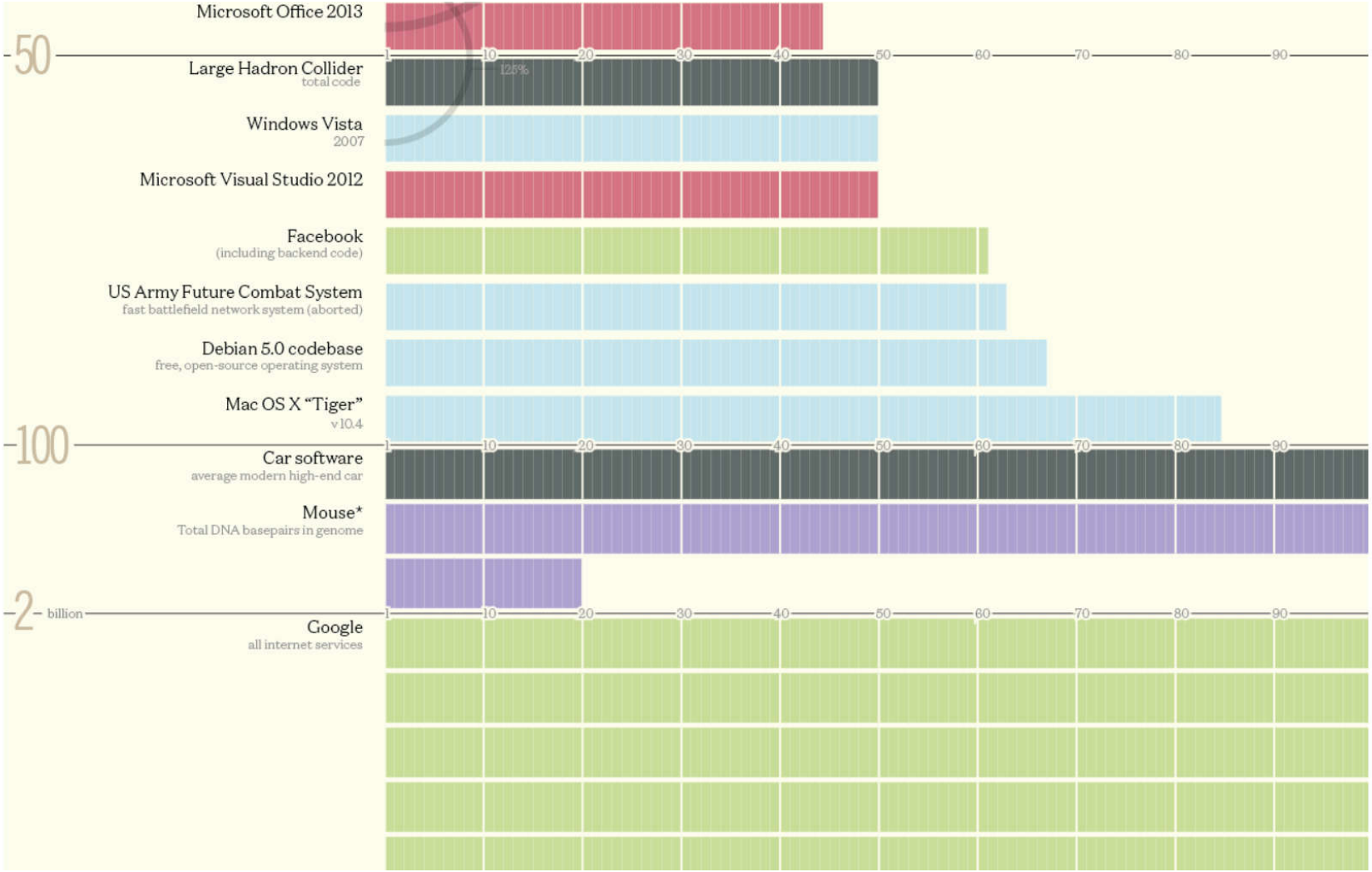
Challenge of Complexity

- Increasing Software Complexity
  - It is not feasible to test software for all possible environments and combinations of components and devices. The question is not whether there are bugs but how well are they managed.



Millions of lines of software source codes

Challenge of Complexity



Millions of lines of software source codes

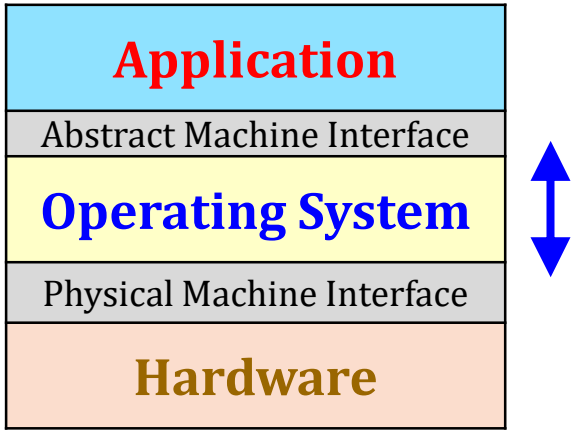
Challenge of Complexity

- Every piece of computer hardware different
  - Different CPU
    - Pentium, ARM, PowerPC, ColdFire, MIPS
  - Different amounts of memory, disk, …
  - Different types of devices
    - Mice, keyboards, sensors, cameras, fingerprint readers, touch screen.
  - Different networking environment
    - Cable, DSL, Wireless, …
- Questions:
  - Does the programmer need to write a single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does a faulty program crash everything?

Example: Some Mars Rover ("Pathfinder") Requirements

- Pathfinder hardware limitations/complexity:
  - 20Mhz processor, 128MB of DRAM, VxWorks OS
  - cameras, scientific instruments, batteries, solar panels, and locomotion equipment
  - Many independent processes work together
- Can't hit reset button very easily!
  - Must reboot itself if necessary
  - Must always be able to receive commands from Earth
- Individual Programs must not interfere
  - Suppose the MUT (Martian Universal Translator Module) buggy
  - Better not crash antenna positioning software!
- Further, all software may crash occasionally
  - Automatic restart with diagnostics sent to Earth
  - Periodic checkpoint of results saved?
- Certain functions time critical:
  - Need to stop before hitting something
  - Must track orbit of Earth for communication

OS Abstracts Underlying Hardware



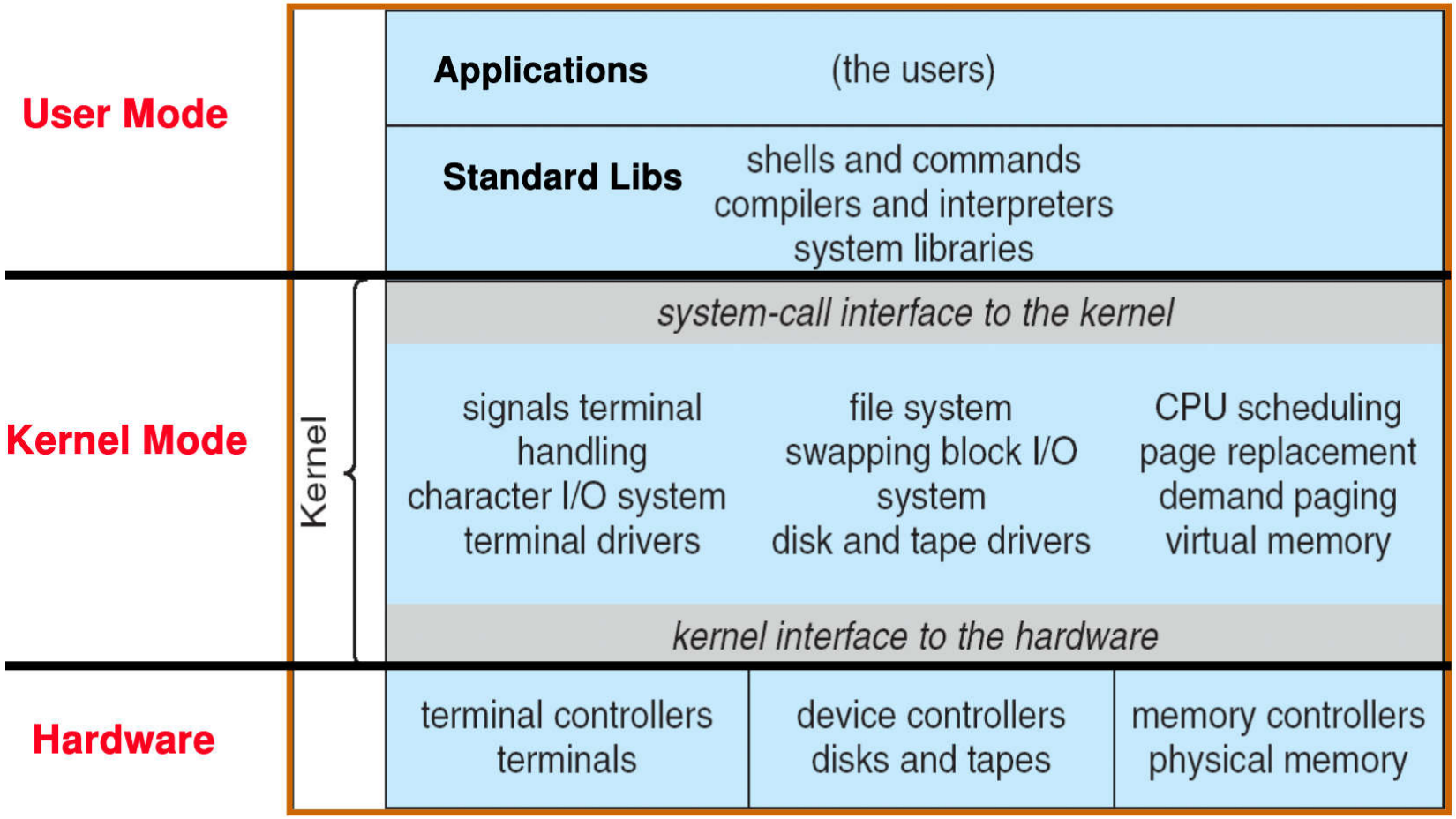
- OS Abstraction
  - Processor → Thread
  - Machines → Processes
  - Networks → Sockets
  - Memory → Address Space
  - Disks, SSDs, … → Files
- OS Goals
  - Remove software/hardware quirks (fight complexity)
  - Optimize for convenience, utilization, reliability, …
    - to help the programmer
  - Protect processes and the Kernel
    - run multiple applications and keep them from interfering with or crashing the operating system or each other
- For any OS area (e.g. file systems, virtual memory, networking, scheduling), considering:
  - What *hardware interface* to handle? (physical reality)
  - What *software interface* to provide? (nicer abstraction)

Protecting Processes

- Use two features offered by hardware to protect processes
  - Dual Mode Operation
    - Hardware provides at least two modes:
      - Kernel Mode (or "supervisor" / "protected" mode)
      - User Mode
    - Processes run in user or kernel mode. Certain operations are prohibited when running in user mode.
    - carefully controlled transitions between user mode and kernel mode
      - System calls, interrupts, exceptions.
  - Address Translation
    - Each process has a distinct and isolated address space.
    - Hardware translates from virtual to physical addresses.
    - Policy
      - No program can read or write memory of another program or of the OS.

Protecting Processes

- UNIX Structure



Virtual Machines

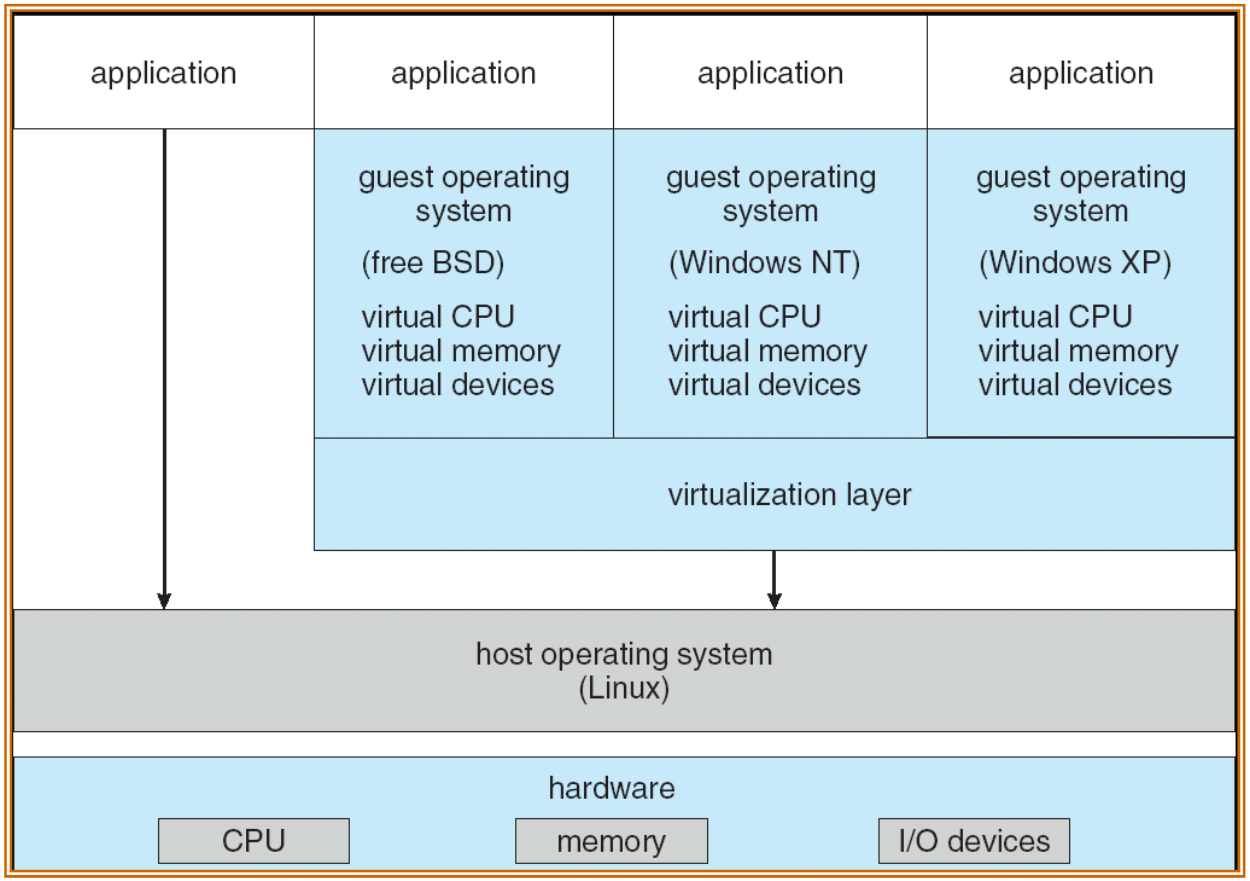
- A virtual machine is a program virtualizing every detail of a hardware configuration so perfectly that we can run an operating system (and many applications) on top of it.
  - E.g., VMWare Fusion, Virtual Box, Parallels Desktop, Xen, Vagrant.
  - long history (60's in IBM OS development)
  - provides isolation
  - complete insulation from change
  - the norm in the Cloud
    - Server Consolidation (SCON, 服务器整合)
- Two types of Virtual Machines (VMs)
  - Process VM: supports the execution of a single program; this functionality typically provided by OS
  - System VM: supports the execution of an entire OS and its applications (e.g., VMWare Fusion, Virtual box, Parallels Desktop, Xen)

Virtual Machines

- Process Virtual Machines
  - Programming simplicity
    - Each process thinks it has all memory/CPU time.
    - Each process thinks it owns all devices.
    - Different devices appear to have same high level interface.
    - Device interfaces are more powerful than raw hardware
      - Bitmapped display $\Rightarrow$ windowing system
      - Ethernet card $\Rightarrow$ reliable, ordered, networking (TCP/IP)
  - Fault isolation
    - processes unable to directly impact other processes
    - bugs cannot crash whole machine
  - Protection and Portability
    - E.g., Java interface safe and stable across many platforms

Virtual Machines

- System Virtual Machines
  - Useful for OS development
    - A crashing OS is restricted to one VM
    - aiding program testing on the other OS



What is an Operating System, … Really

- Most Likely:
  - Memory Management
  - I/O Management
  - CPU Scheduling
- What about?
  - File System?
  - Multitasking/multiprogramming?
  - User Interface?
  - Multimedia Support?
  - Communications?
    - Does Email belong in OS?
  - Internet Browser?

What is an Operating System, … Really

- Operating systems provide a virtual machine abstraction to handle diverse hardware
  - Operating systems simplify application development by providing standard services
- Operating systems coordinate resources and protect users from each other
  - Operating systems can provide an array of fault containment, fault tolerance, and fault recovery

Evolution of operating systems

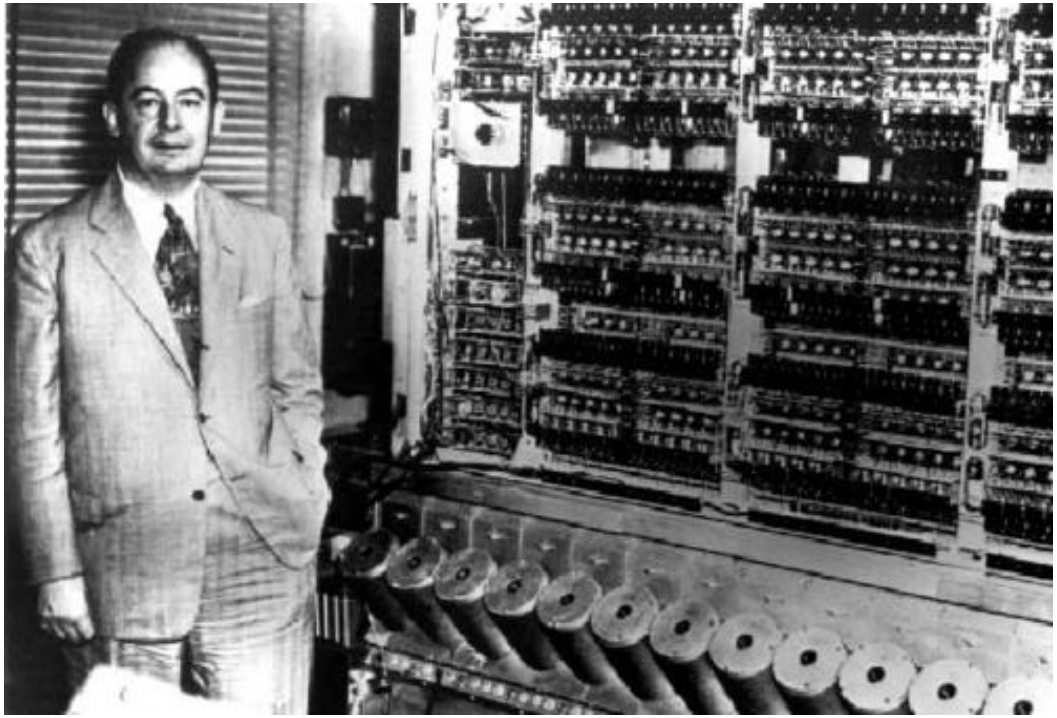
- The evolution of operating systems is directly dependent to the development of computer systems and how users use them.
- A quick tour of computing systems through the past years in the timeline.
  - Early Systems (1950)
  - Simple Batch Systems (1960)
  - Multiprogrammed Batch Systems (1970) 多道批处理系统
  - Time-Sharing and Real-Time Systems (1970)
  - Personal/Desktop Computers (1980)
  - Multiprocessor Systems (1980)
  - Networked/Distributed Systems (1980)
  - Web-based Systems (1990)

Early Systems (1950)

- Some Important Time Points
  - 1945: ENIAC, Moore School of Engineering, University of Pennsylvania
  - 1949: EDSAC and EDVAC
  - 1949: BINAC - a successor to the ENIAC
  - 1951: UNIVAC by Remington
  - 1952: IBM 701
  - 1956: The interrupt
  - 1954-1957: FORTRAN language

Early Systems (1950)

- *John von Neumann* and ENIAC

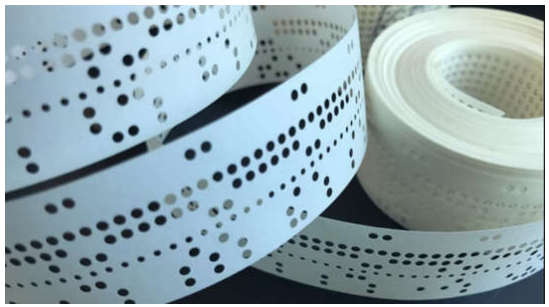


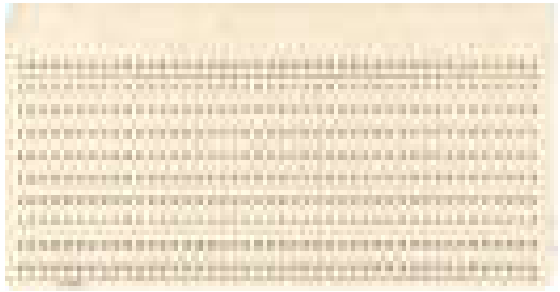
Early Systems (1950)

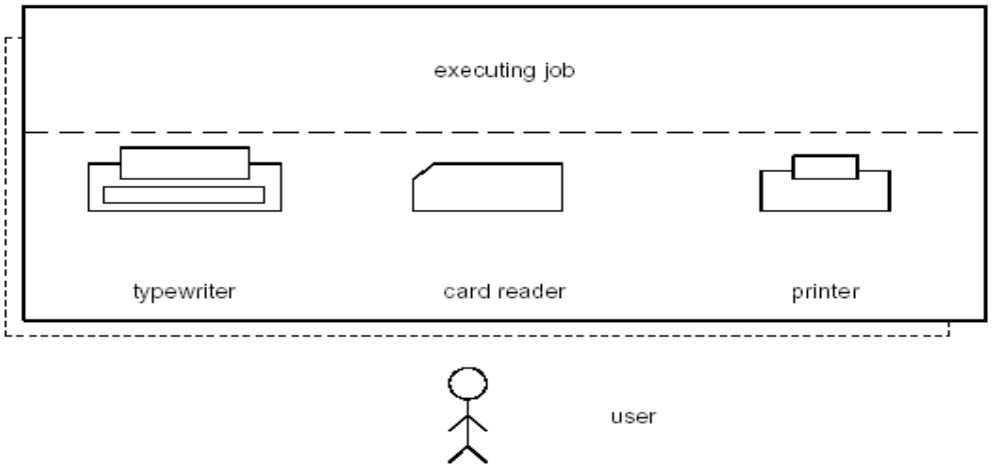
- Operating Systems by the Late 1950s
  - By the late 1950s Operating systems were well improved and started supporting following usages:
    - Single stream batch processing
    - Using common, standardized, input/output routines for device access
    - Program transition capabilities to reduce the overhead of starting a new job
    - Error recovery to clean up after a job terminated abnormally
    - Job control languages that allowed users to specify the job definition and resource requirements

Early Systems (1950)

- Structures
  - Single user system
  - Programmer/User as operator (lie an open shop)
  - Large machines running from console
  - Paper Tape or Punched Cards







- **Early Systems (1950)**
  - Characteristics
    - Early software
      - Assemblers
      - Libraries of common subroutines (I/O, Floating-point, etc.)
      - Device drivers
      - Compilers
      - Linkers.
    - Need significant amount of setup time
    - Extremely slow I/O devices
    - Very low CPU utilization
    - High secure

Operating Systems in 1960s

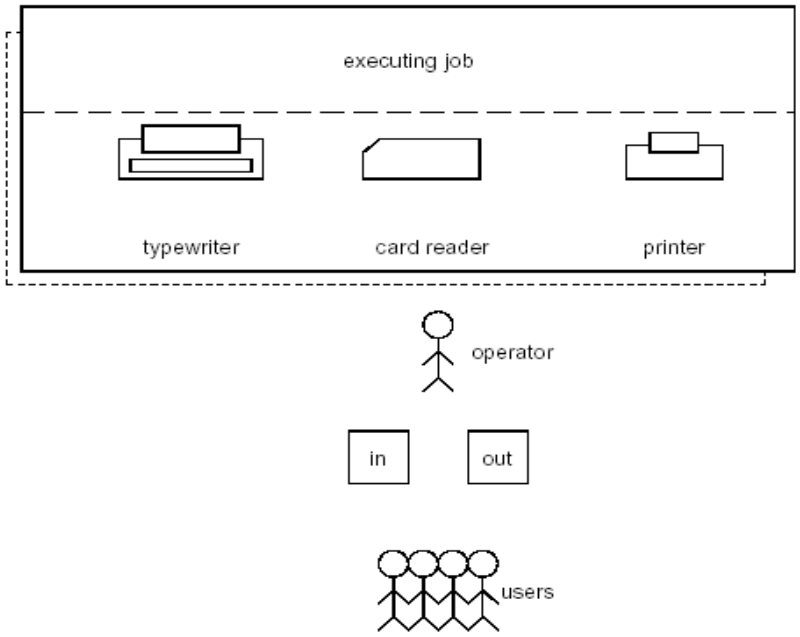
- Some Important Time Points
  - 1960s: Disks become mainstream
  - 1961: The dawn of minicomputers
  - 1962: Compatible Time-Sharing System (CTSS) from MIT
  - 1963: Burroughs Master Control Program (MCP) for the B5000 system
  - 1964: IBM System/360
  - 1964 and onward: Multics
  - 1966: Minicomputers get cheaper, more powerful, and really useful
  - 1967-1968: The mouse
  - 1969: The UNIX Time-Sharing System from Bell Telephone Laboratories

Operating Systems in 1960s

- Simple Batch Systems
  - use of high-level languages, magnetic tapes
  - Jobs are batched together by type of languages.
  - An operator was hired to perform the repetitive tasks of loading jobs, starting the computer, and collecting the output (an operator-driven shop).
  - It was not feasible for users to inspect memory or patch programs directly.

Operating Systems in 1960s

- Simple Batch Systems
  - use of high-level languages, magnetic tapes.
  - Jobs are batched together by type of languages.
  - An operator was hired to perform the repetitive tasks of loading jobs, starting the computer, and collecting the output (an operator-driven shop).
  - It was not feasible for users to inspect memory or patch programs directly.



Operating Systems in 1960s

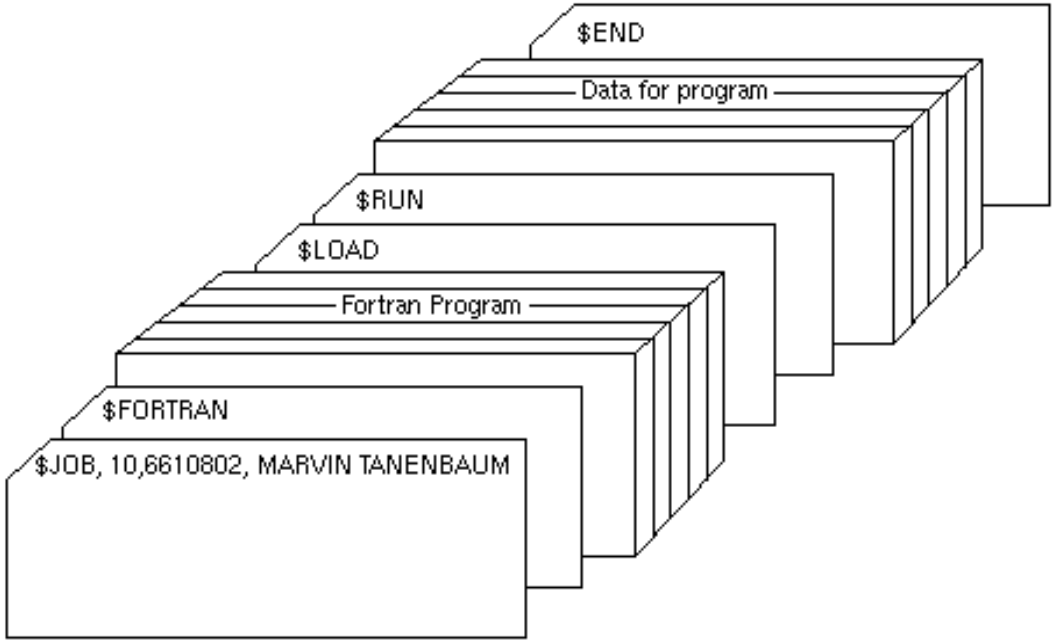
- Simple Batch Systems
  - Operations
    - The user submits a job (written on cards or tape) to a computer operator.
    - The computer operator place a batch of several jobs on an input device.
    - A special program, the monitor, manages the execution of each program in the batch.
    - Monitor utilities are loaded when needed.
    - Resident Monitor (常驻监控程序) is always in main memory and available for execution.

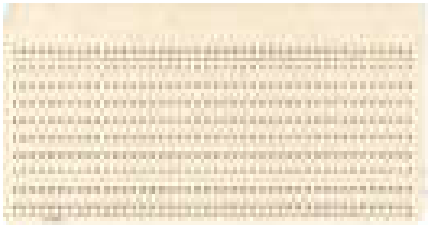
Operating Systems in 1960s

- Simple Batch Systems
  - Ideas of Simple Batch Systems
    - reduce setup time by batching similar jobs
    - alternate execution between user program and the monitor program
    - rely on available hardware to effectively alternate execution from various parts of memory
    - use Automatic Job Sequencing – automatically transfer control from one job when it finishes to another one
  - Problems
    - How does the monitor know about the nature of the job (e.g., Fortran versus Assembly) or which program to execute?
    - How does the monitor distinguish:
      (a) job from job?
      (b) data from program?
  - Solution
    - introduce Job Control Language (JCL) and control cards

Operating Systems in 1960s

- Simple Batch Systems
  - Example: card deck of a job





Operating Systems in 1960s

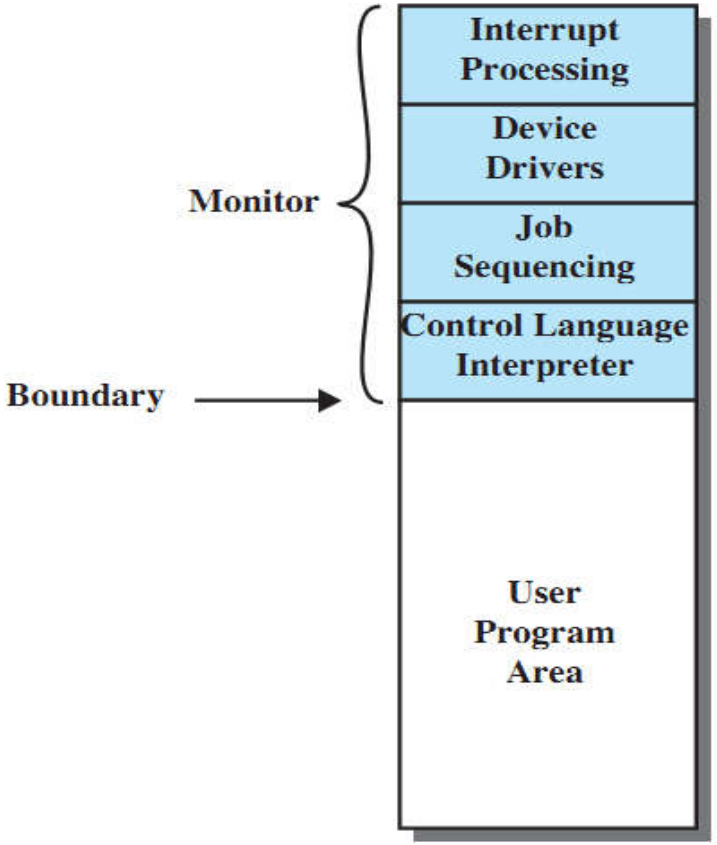
- Simple Batch Systems
  - Job Control Language (JCL)
    - JCL is the language that provides instructions to the monitor:
      - what compiler to use
      - what data to use
      - Example of job format:
        - $FTN loads the compiler and transfers control to it.
        - $LOAD loads the object code (in place of compiler).
        - $RUN transfers control to user program.
    - Each read instruction (in user program) causes one line of input to be read.
    - Causes (OS) input routine to be invoked:
      - checks for not reading a JCL line
      - skip to the next JCL line at completion of user program

Operating Systems in 1960s

- Simple Batch Systems
  - Resident Monitor
    - *Resident Monitor* (Job Sequencer) is the first rudimentary Operating System (常驻监控程序是第一种初具雏形的操作系统).
      - Initial control is in monitor.
      - The next program is loaded and transfers control to it
      - When job completes, the control is transferred back to monitor.
      - Control is automatically transferred control from one job to another, no idle time between programs

Operating Systems in 1960s

- Simple Batch Systems
  - Resident Monitor
    - *Resident Monitor* (Job Sequencer) is the first rudimentary Operating System (常驻监控程序是第一种初具雏形的操作系统).



Operating Systems in 1960s

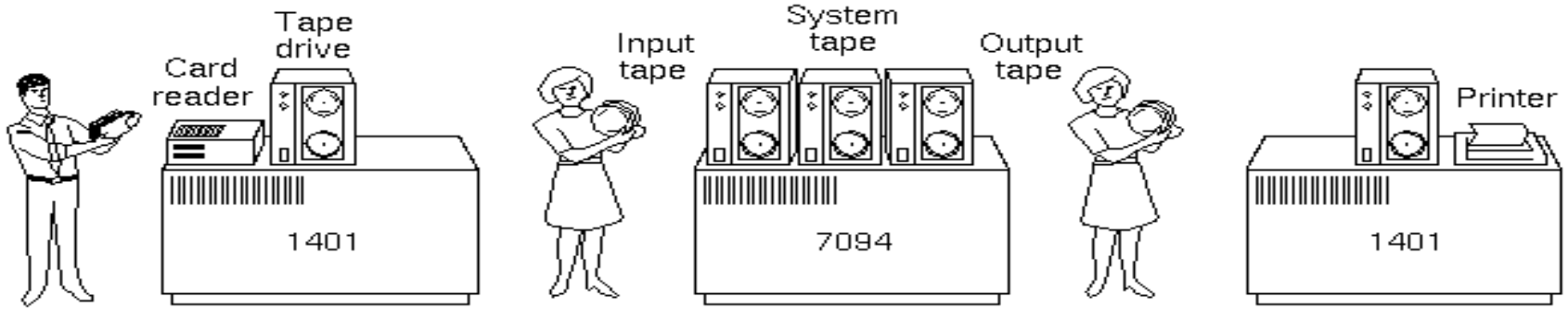
- Simple Batch Systems
  - Parts of Resident Monitor:
    - Control Language Interpreter
      - responsible for reading and carrying out instructions on the cards
    - Loader
      - loads systems programs and applications programs into memory
    - Device drivers
      - know special characteristics and properties for each of the system's I/O devices

Operating Systems in 1960s

- Simple Batch Systems
  - Desirable Hardware Features
    - Memory protection
      - do not allow the memory area containing the monitor to be altered by a user program
    - Privileged instructions
      - can be executed only by the resident monitor
      - A trap occurs if a program tries these instructions.
    - Interrupts
      - provide flexibility for relinquishing control to and regaining control from user programs
      - Timer interrupts prevent a job from monopolizing the system.

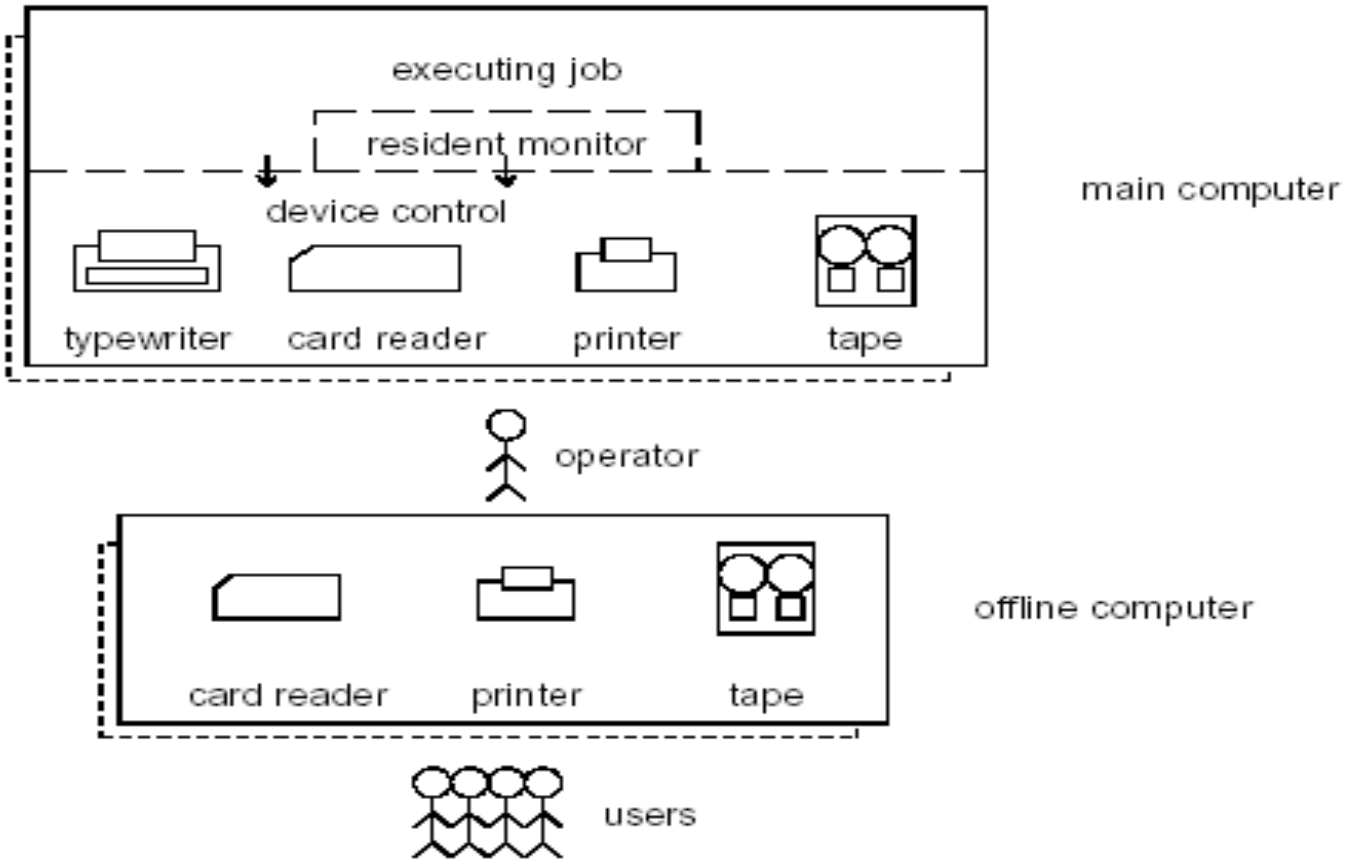
Operating Systems in 1960s

- Simple Batch Systems
  - Offline Operation
    - Problem:
      - Card Readers and Printers are slow (compared to Tape).
      - I/O and CPU could not overlap.
    - Solution: Offline Operation (Satellite Computers)
      - speed up computation by loading jobs into memory from tapes while card reading and line printing is done off-line using smaller machines



Operating Systems in 1960s

- Simple Batch Systems
  - Offline Operation
    - Main and Offline Computer

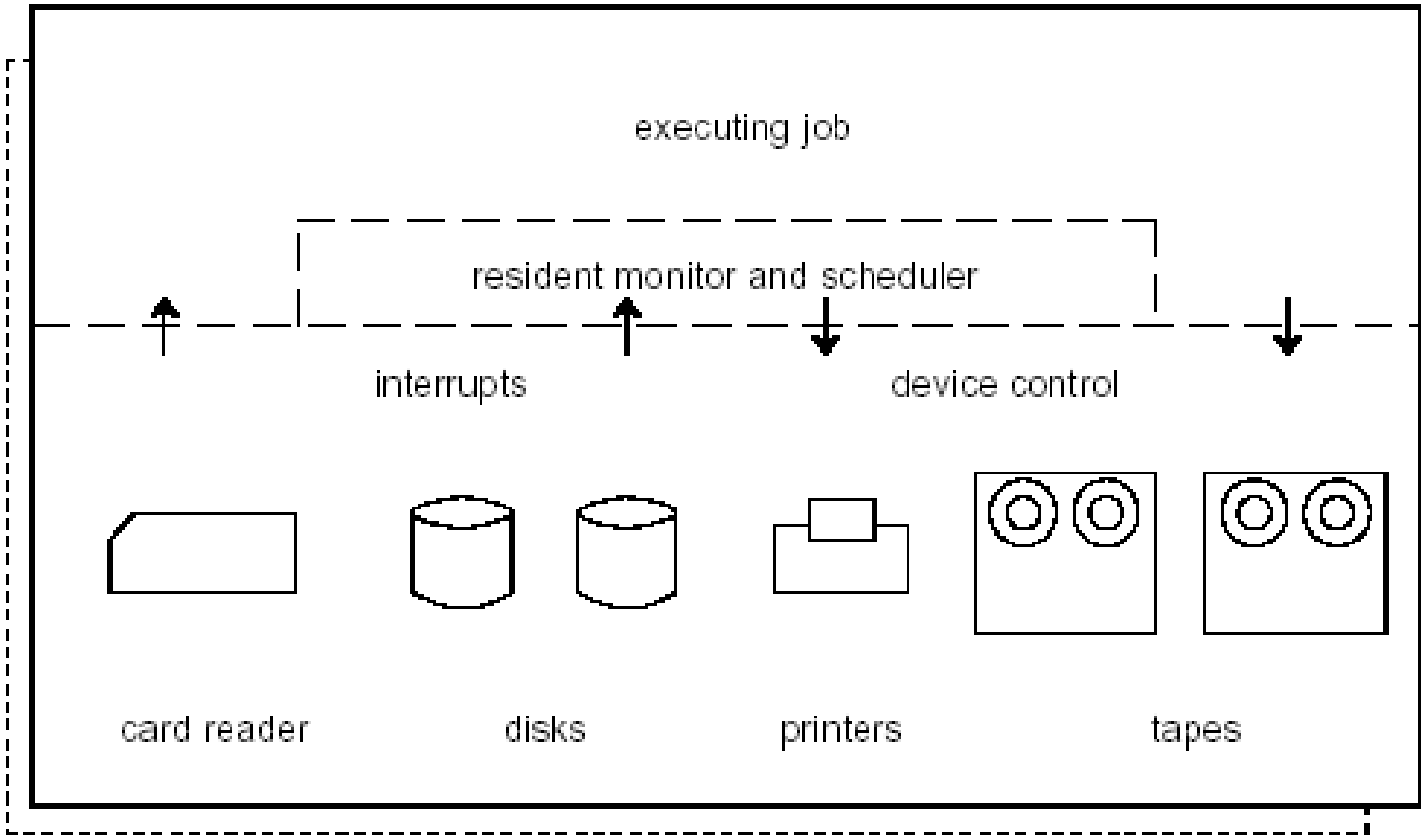


Operating Systems in 1960s

- Simple Batch Systems
  - SPOOLing
    - Problem:
      - Card reader, Line printer and Tape drives are slow (compared to Disk).
      - I/O and CPU could not overlap.
    - Solution: SPOOLing
      - Overlap I/O of one job with the computation of another job (using double buffering, DMA, etc.)
      - Technique is called SPOOLing: Simultaneous Peripheral Operation On Line (外围设备联机并行操作, 假脱机).

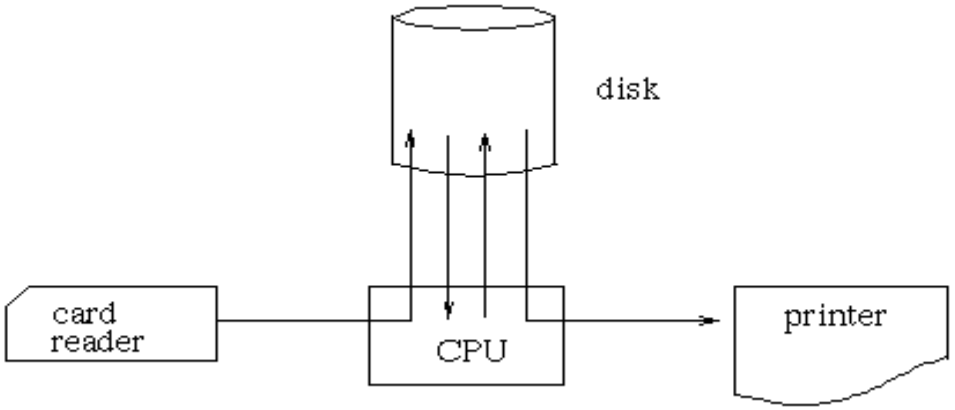
Operating Systems in 1960s

- Simple Batch Systems
  - SPOOLing
    - SPOOLing System Components



Operating Systems in 1960s

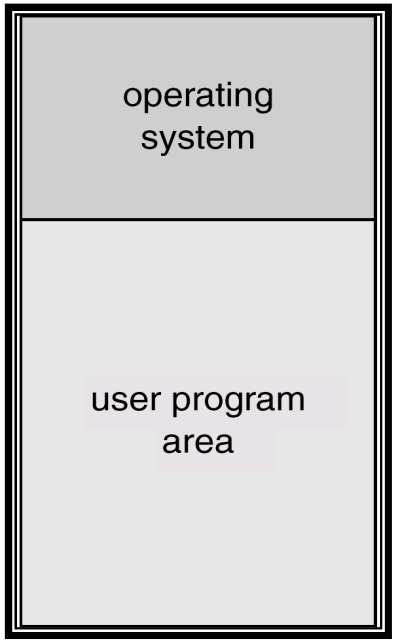
- Simple Batch Systems
  - SPOOLing
    - While executing one job, the OS:
      - reads next job from card reader into a storage area on the disk (Job pool)
      - Outputs printout of previous job from disk to printer

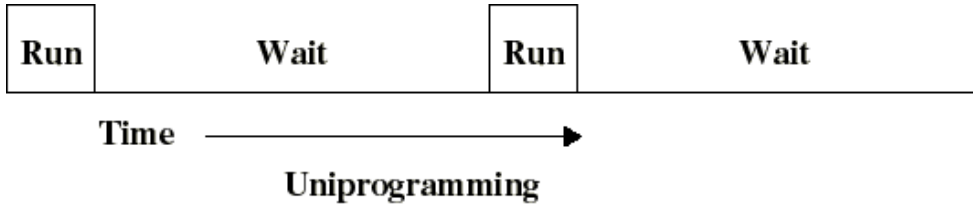


    - Job pool
      - data structure that allows the OS to select which job to run next in order to increase CPU utilization

Operating Systems in 1960s

- Simple Batch Systems
  - In a uni-programming system
    - I/O operations are exceedingly slow (compared to instruction execution).
    - A program containing even a very small number of I/O operations, will spend most of its time waiting for them.
    - Hence: poor CPU usage when only one program is present in memory.





The memory layout of a uni-programming system

Accomplishments after 1970

- Some Important Time Points
  - 1971: Intel announces the microprocessor
  - 1972: IBM comes out with VM: the Virtual Machine Operating System
  - 1973: UNIX 4th Edition is published
  - 1973: Ethernet
  - 1974: The Personal Computer Age begins
  - 1974: *Bill Gates* and *Paul Allen* wrote BASIC for the Altair
  - 1976: Apple II
  - 1981: IBM introduces the IBM PC on Aug. 12
  - 1983: Microsoft begins work on MS-Windows
  - 1984: Apple Macintosh comes out
  - 1990: Microsoft Windows 3.0 comes out
  - 1991: GNU/Linux
  - 1992: The first Windows virus comes out
  - 1993: Windows NT
  - 2007: iOS
  - 2008: Android OS