北京交通大学

Chapter 1&2

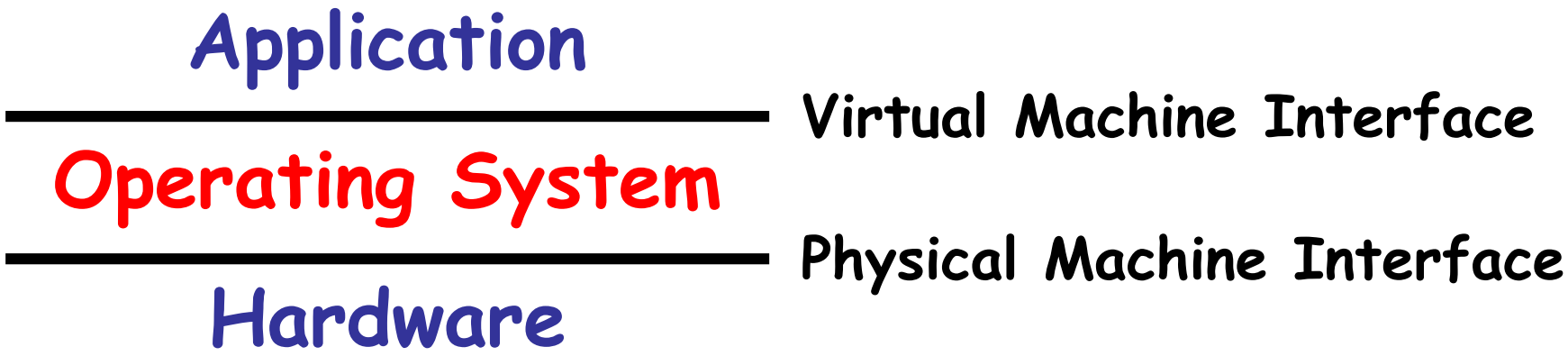
Introduction to OS

Outline

- What is an Operating System,… Really?
- Operating System *Definition*
- What are the pieces of the OS? Operating Systems *Components*
- What does the OS do? Operating System *Services*
- Operating System Design and Implementation
- *History* and Evolution of OS

How do we tame complexity?

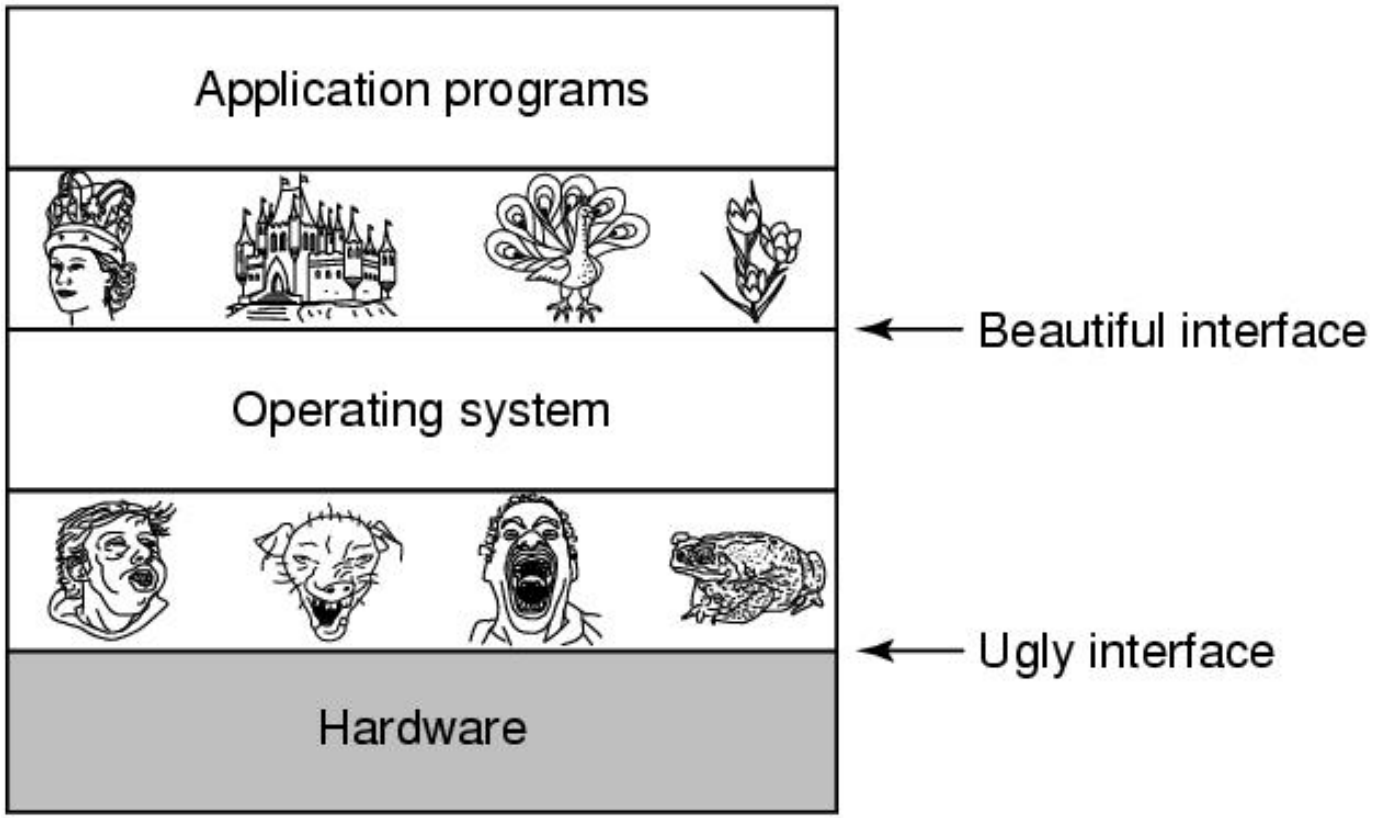
OS Tool: Virtual Machine Abstraction



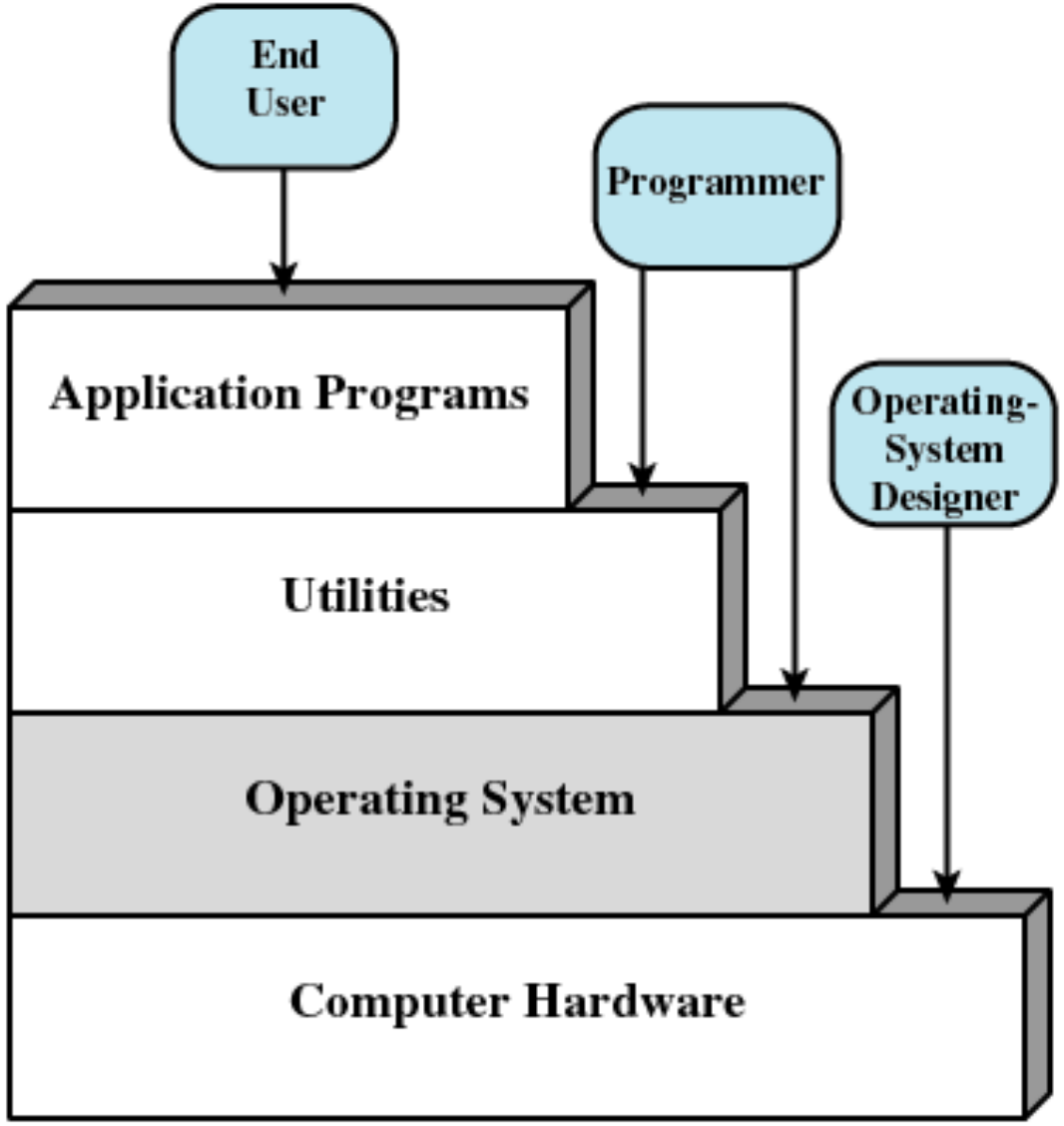
- **Software Engineering Problem:**
  - Turn hardware/software quirks ⇒ what programmers want/need

Operating System as an Extended Machine

- Operating system turns **ugly** hardware into **beautiful** abstractions.

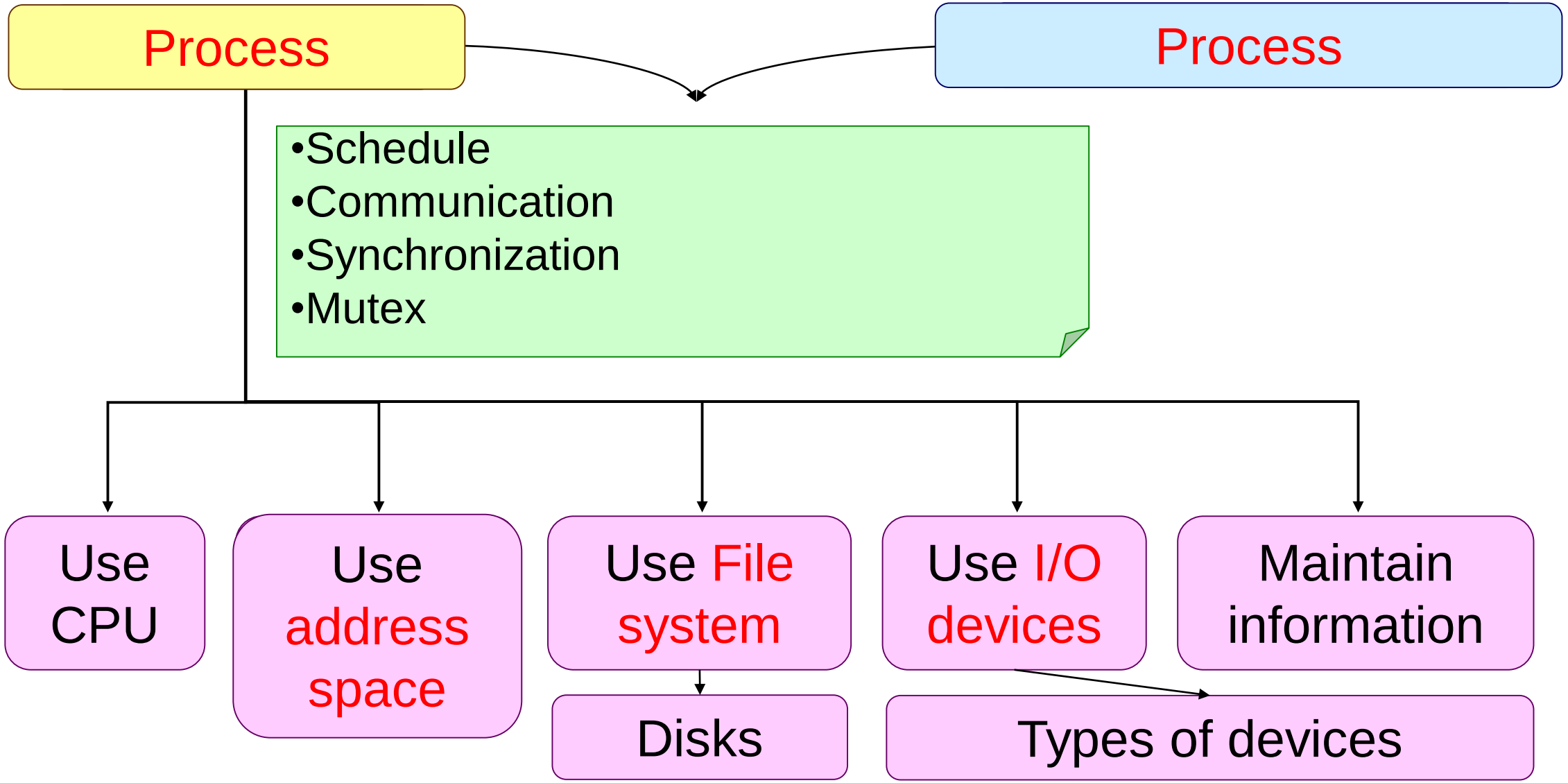


Layers of Computer System



- An OS implements a virtual machine that is (hopefully) easier and safer to program and use than the raw hardware.

Run Programs on a Computer



What is an operating system?

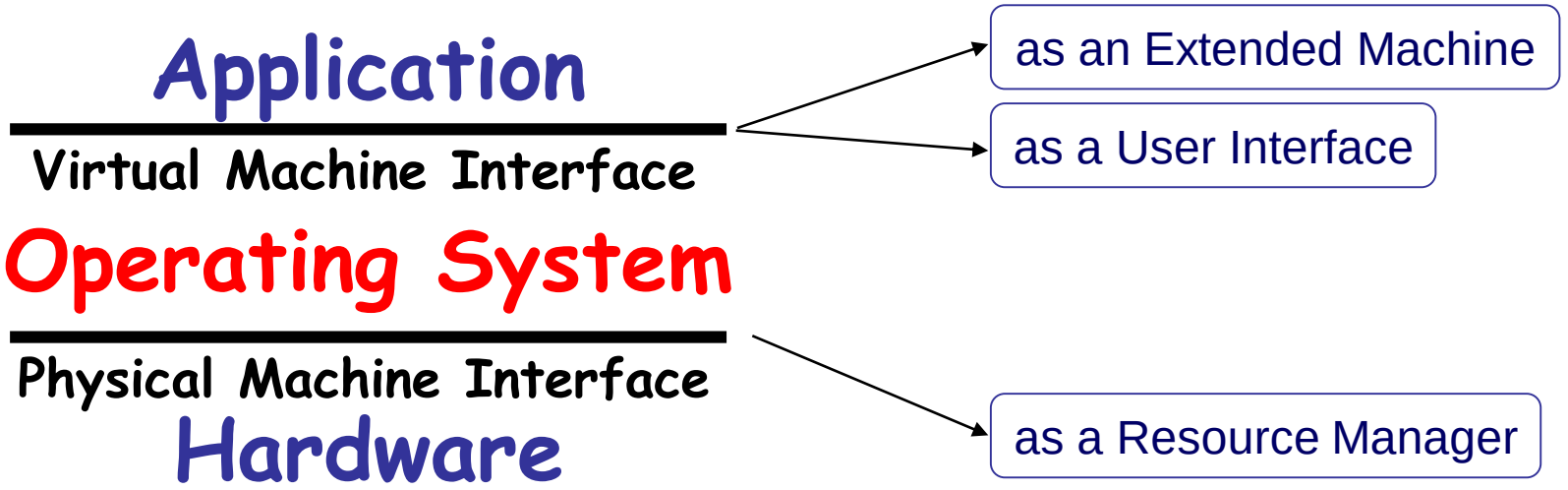
What is an OS?

- No universally accepted definition
- Silberschatz and Galvin:

  “An OS is Similar to a government”
- The question: does a government do anything useful by itself?

What is an OS?

- Operating system is a kind of system software to *manage hardware resources*, *control program running*, *improve man-machine interface* and *provide support for application software*



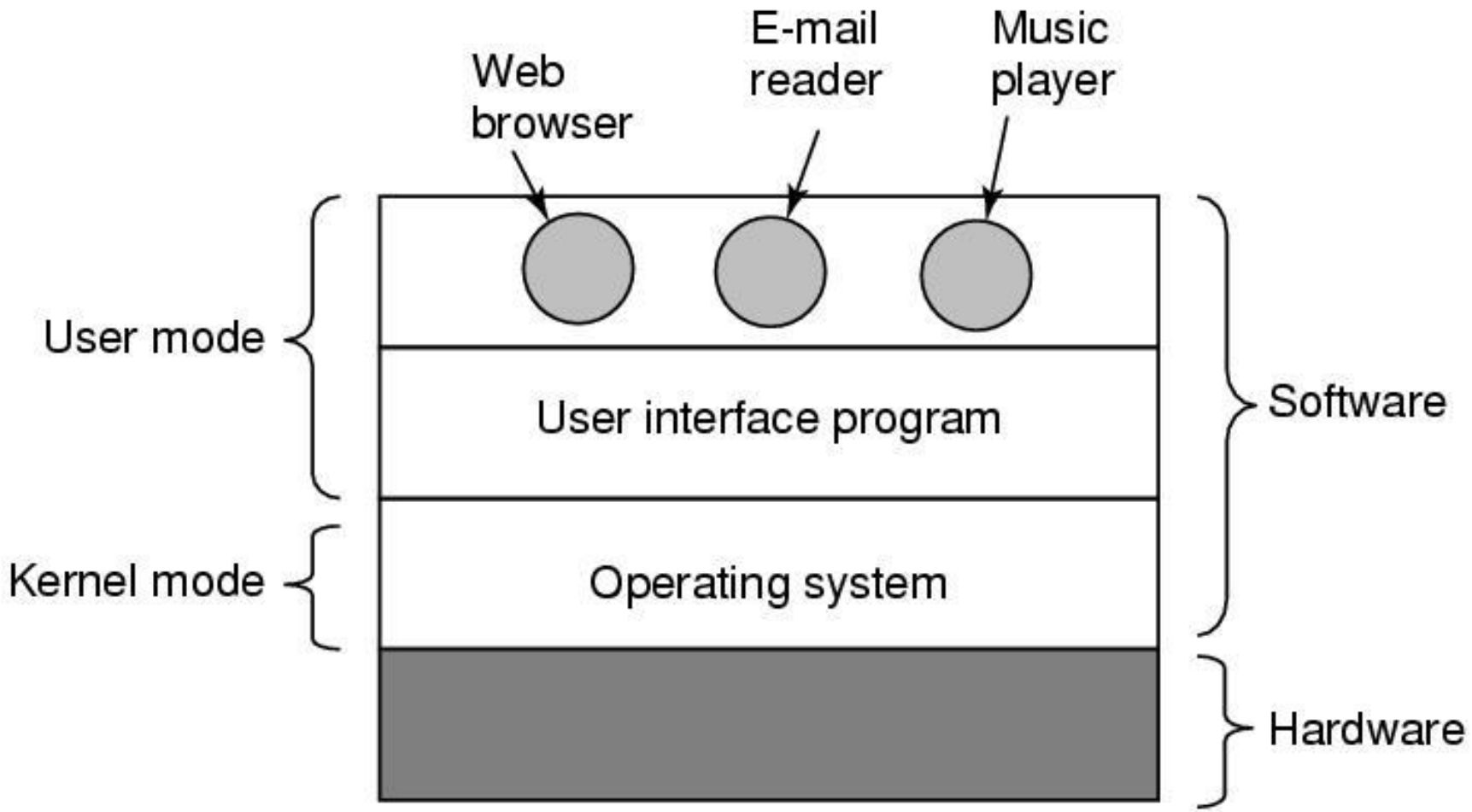
What is an OS?

- The Operating System as an extended machine and a user interface
  - Hides the messy details which must be performed
  - Presents user with a virtual machine, easier to use
  - Executes user programs and make user problems solving easier
- The Operating System as a resource manager
  - Each program gets time with the **resource**
  - Each program gets space on the **resource**
  - Decides between conflicting requests for efficient and fair resource use
  - Makes the computer system convenient to use

Operating System

- Operating surgeon
- Operating person
- Operating system

Where the operating system fits



Kernel

- Shell – Command line interface
- GUI – Graphical user interface
- **Kernel** – The interior of the operating system
  - "The one program running at all times on the computer" is the **kernel.**
  - Everything else is either a system program (ships with the operating system) or an application program

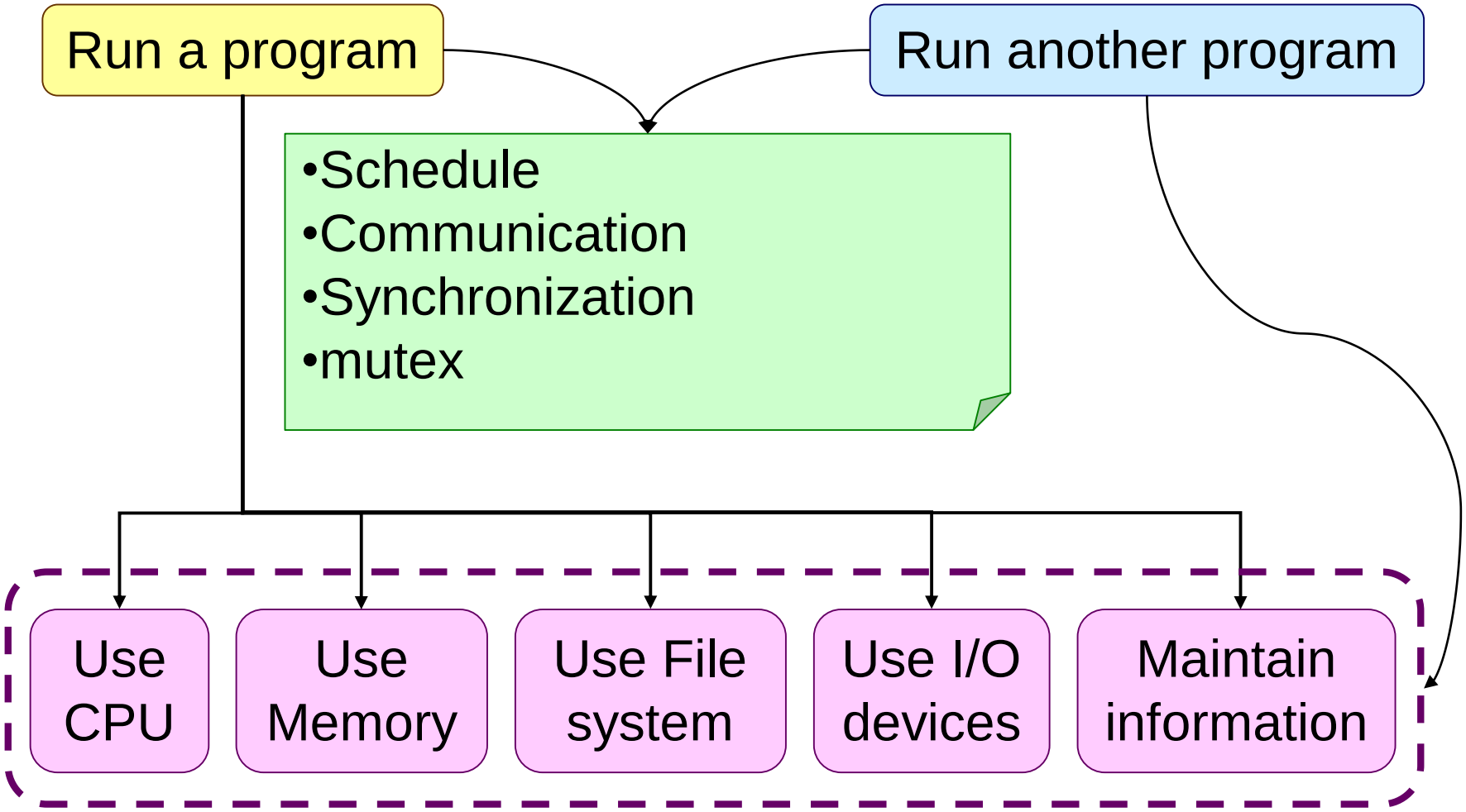
Computer Startup

- **bootstrap program (启动程序)** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

BIOS -> MBR -> 引导加载程序 -> 内核 -> init process -> login

What are the pieces of the OS?

Run Programs on a Computer



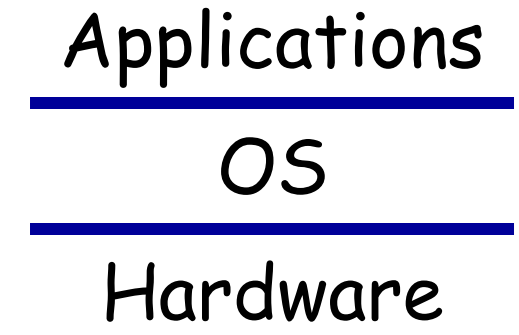
Operating Systems Components

- Process Management
- Main-Memory Management
- Secondary-Storage Management
- File Management
- I/O System management
- User Interfaces
- Networking
- Protection System

Part 2

Part 3

Part 4

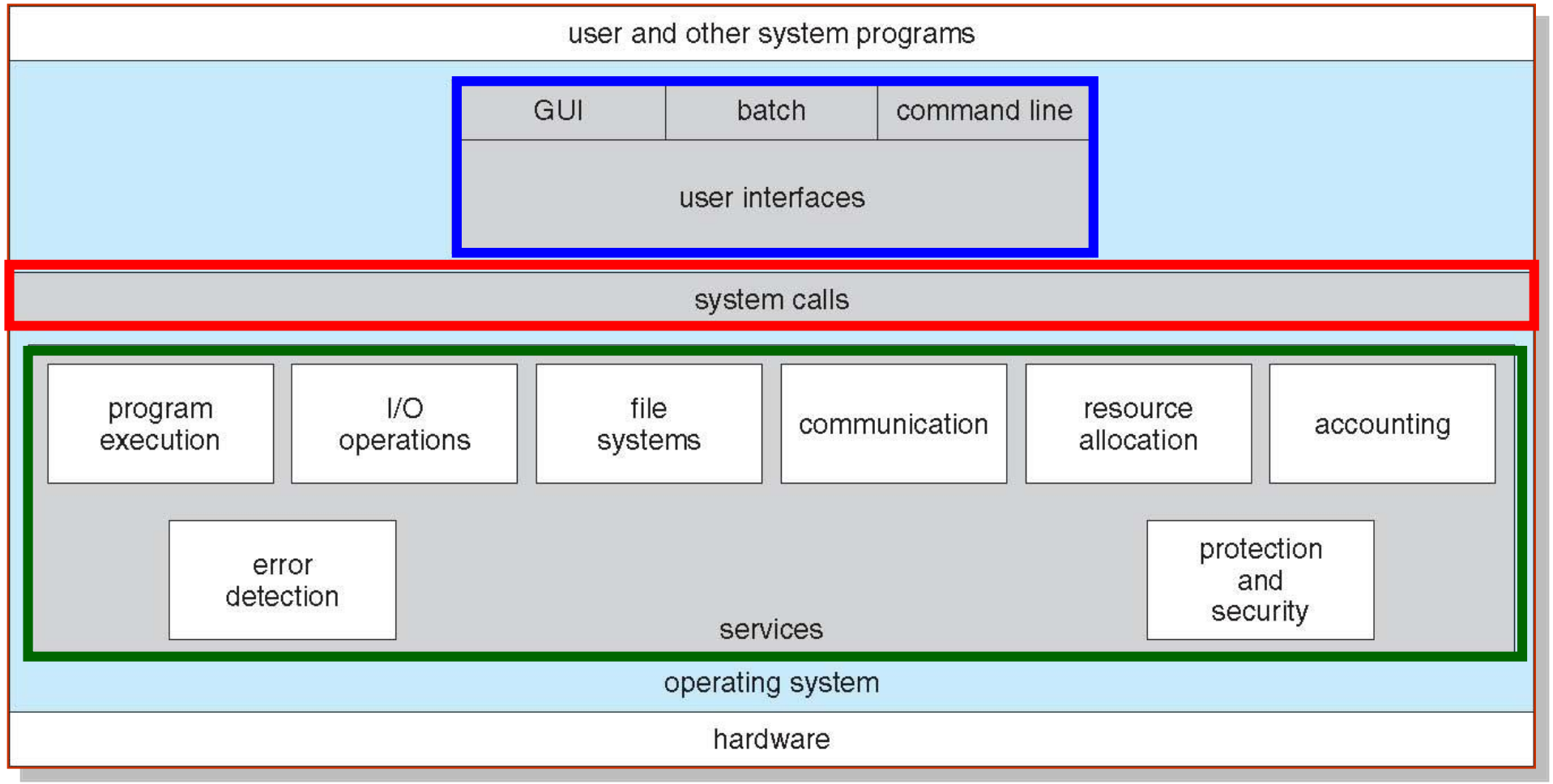


What does the OS do?

Operating System Services

- User interface
- Program execution
  - How do you execute concurrent sequences of instructions?
- I/O operations
  - Standardized interfaces to extremely diverse devices
- File system manipulation
  - How do you read/write/preserve files?
- Communications
- Error detection & recovery
- Resource allocation
- Protection
- ……

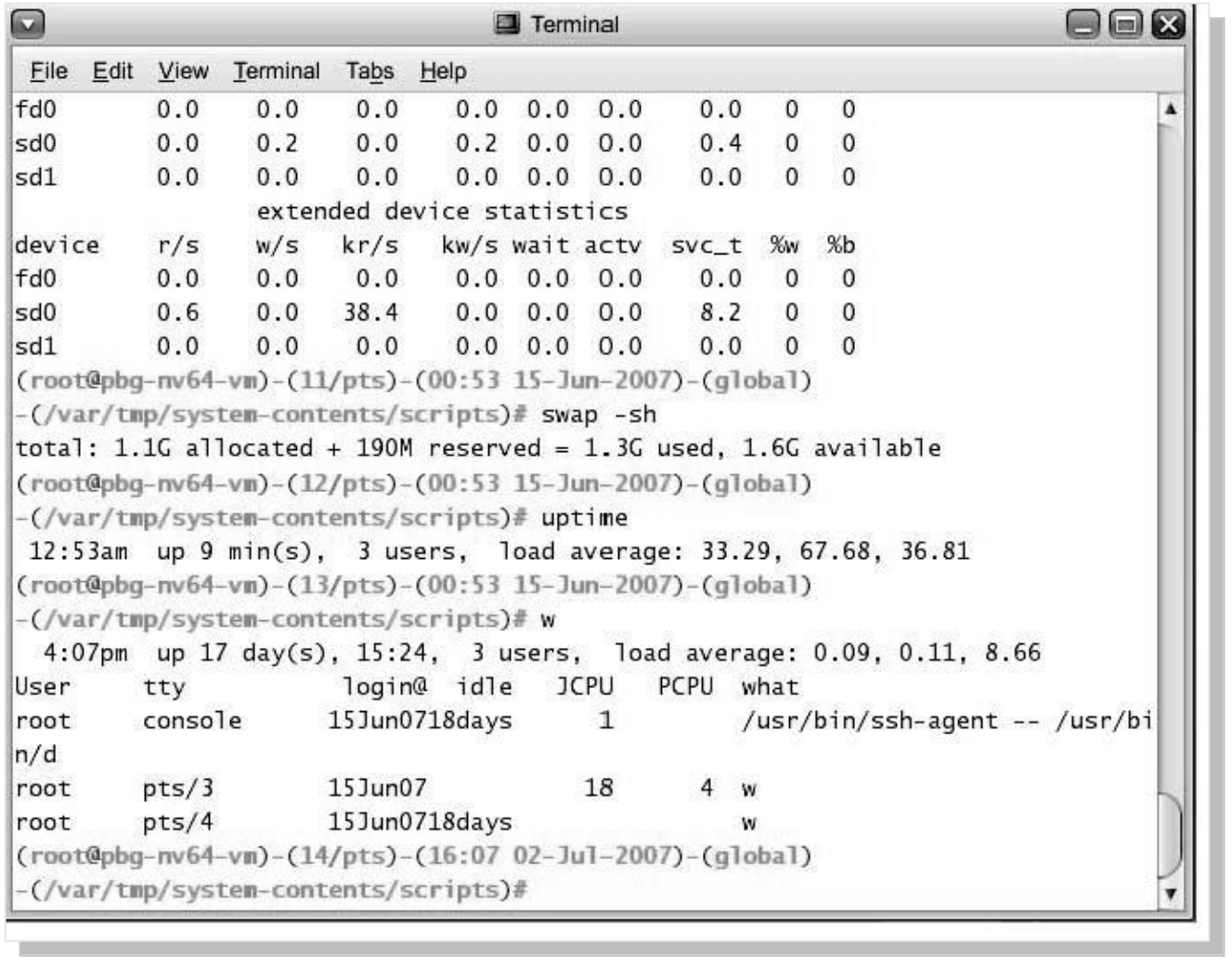
A View of Operating System Services



User Operating System Interface - CLI

- Command Line Interface (CLI) or command interpreter allows direct command entry
  - Primarily fetches a command from user and executes it
- Example
  - UNIX, DOS

CLI

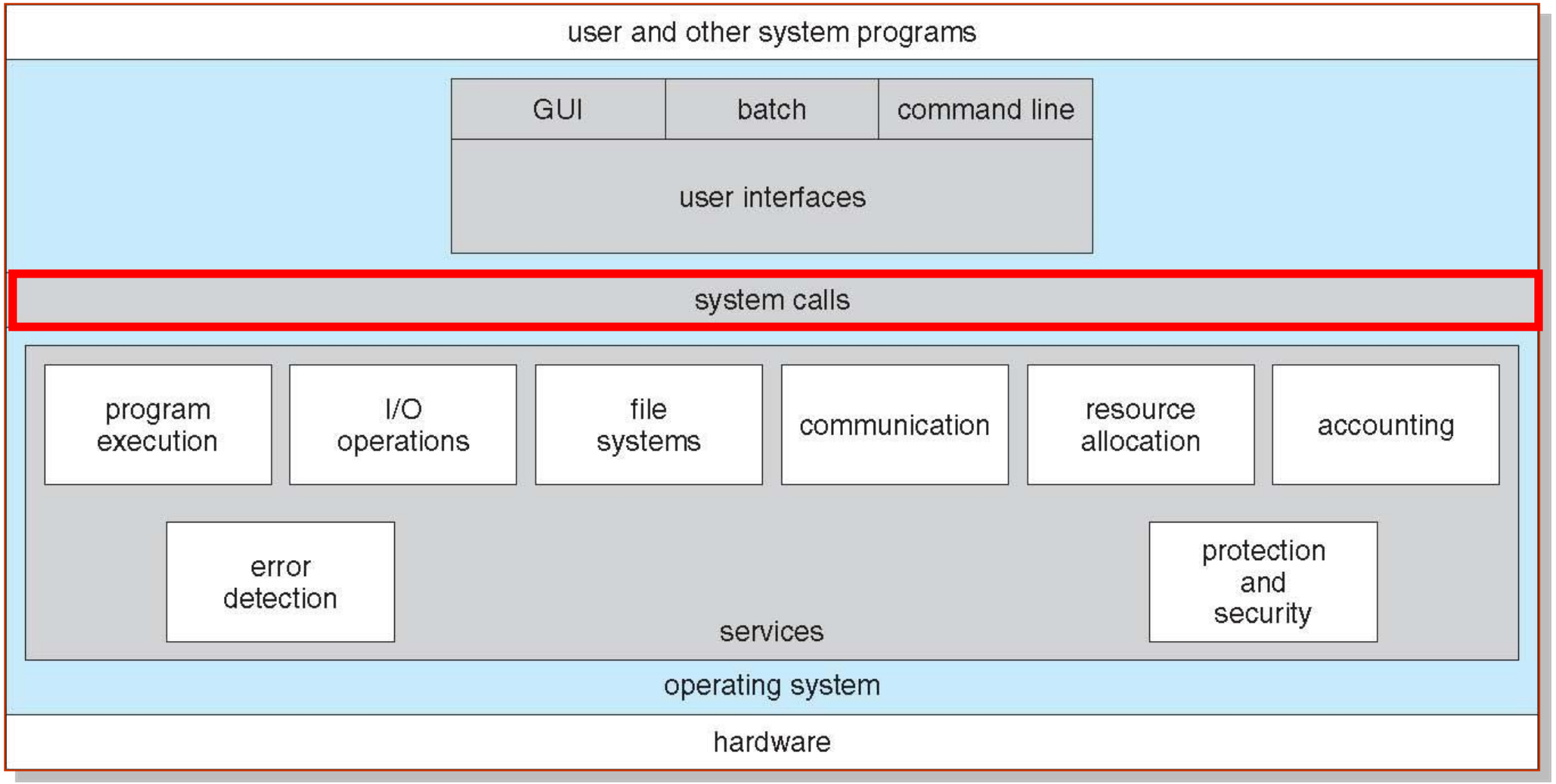

Terminal
File Edit View Terminal Tabs Help
fd0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
sd0 0.0 0.2 0.0 0.2 0.0 0.0 0.4 0 0
sd1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
extended device statistics
device r/s w/s kr/s kw/s wait actv svc_t %w %b
fd0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
sd0 0.6 0.0 38.4 0.0 0.0 0.0 8.2 0 0
sd1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0 0
(root@pbg-nv64-vm)-(11/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vm)-(12/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vm)-(13/pts)-(00:53 15-Jun-2007)-(global)
-(/var/tmp/system-contents/scripts)# w
4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User tty login@ idle JCPU PCPU what
root console 15Jun0718days 1 /usr/bin/ssh-agent -- /usr/bi
n/d
root pts/3 15Jun07 18 4 w
root pts/4 15Jun0718days w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#

User Operating System Interface - GUI

- User-friendly desktop interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions
  - Invented at Xerox PARC (1970s)
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI “command” shell

What are System Calls?

A View of Operating System Services



System Calls

- System calls provide the interface between a running program and the operating system
- Exact type and amount of information vary according to OS and call

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

System Calls

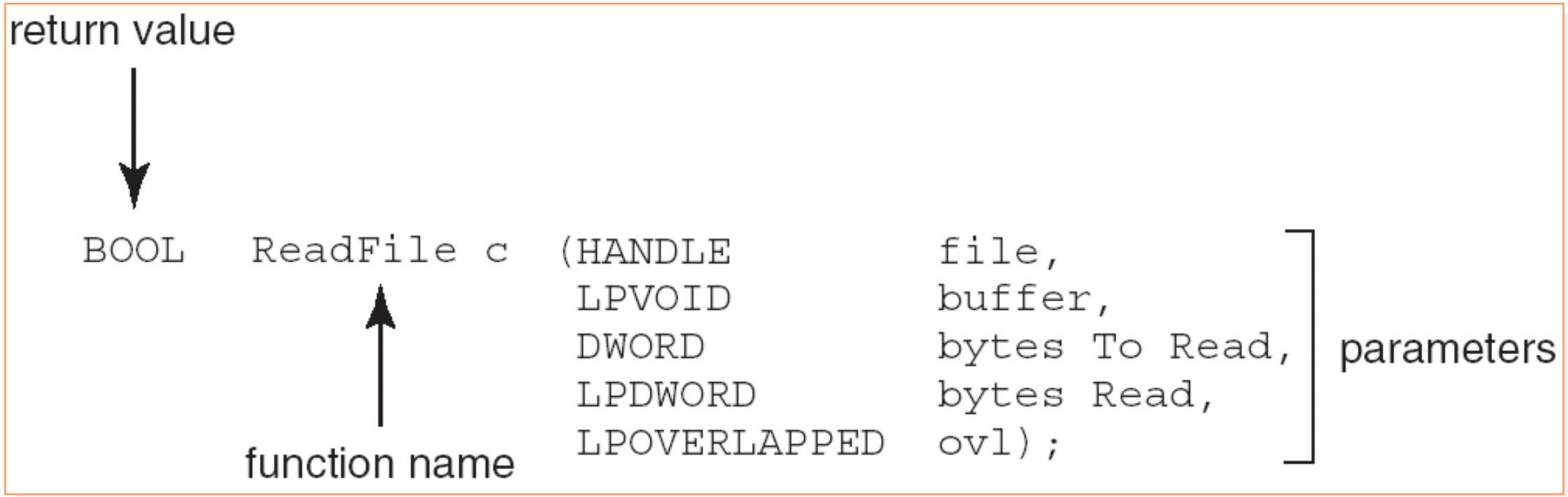
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use

System Calls

- Three most common APIs are
  - Win32 API for Windows
  - POSIX API (Portable Operating System Interface) for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)

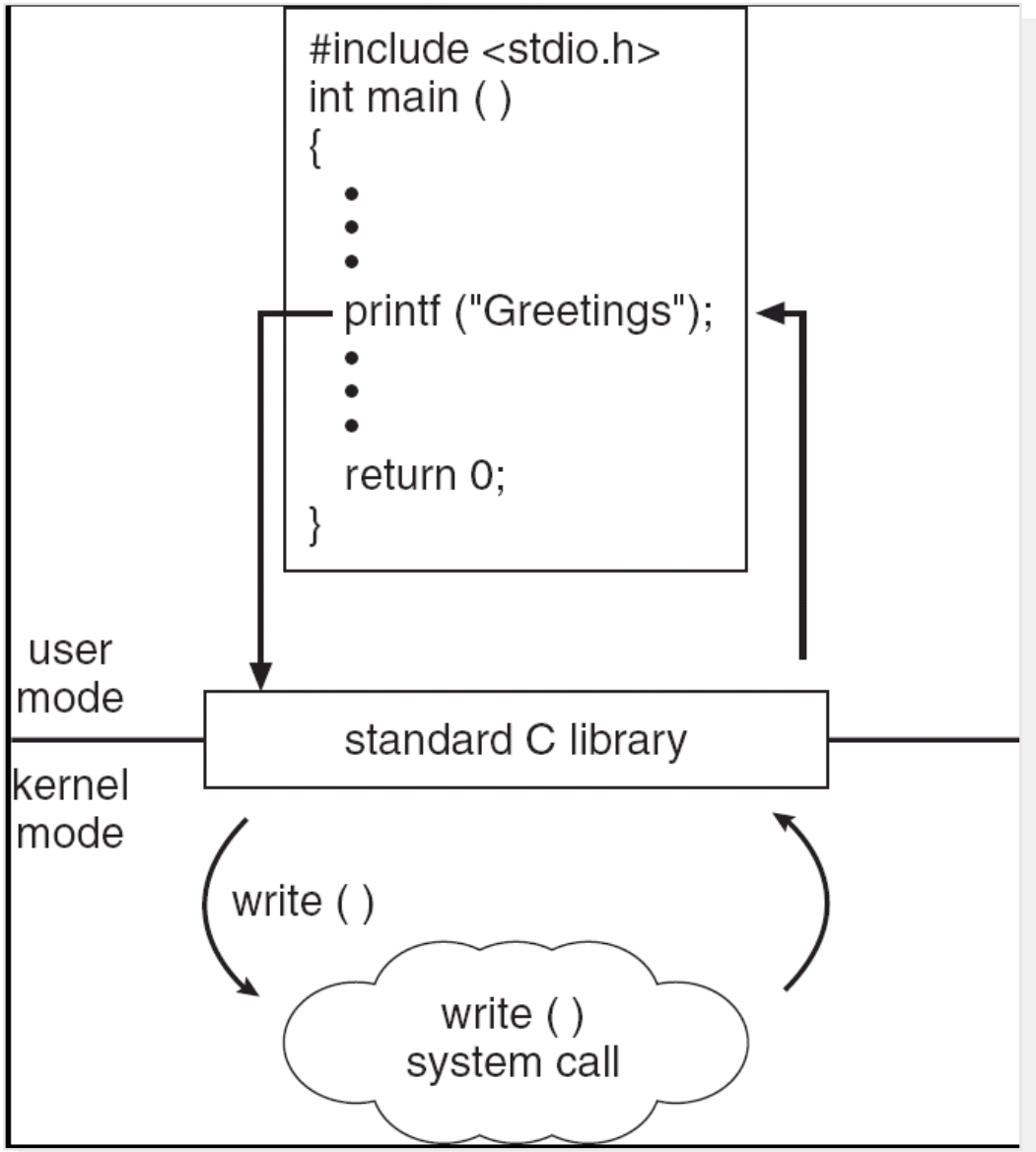
Example of Standard API

- Consider the *ReadFile()* function in the Win32 API—a function for reading from a file



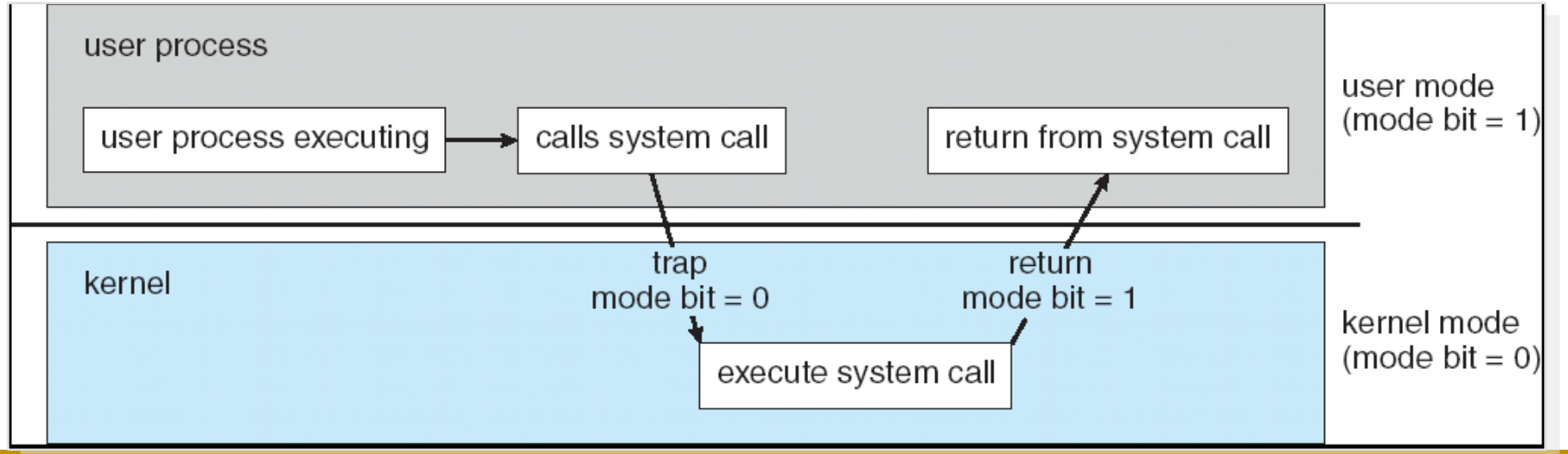
Standard C Library Example

- C program invokes *printf()* statement, which calls *write()* system call



Dual Mode Operation

- Hardware provides at least two modes:
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode: Normal programs executed
- Some instructions/ops prohibited in user mode:
  - Example: cannot modify page tables in user mode
    - Attempt to modify ⇒ Exception generated
- Transitions from user mode to kernel mode:
  - System Calls, Interrupts, Other **exceptions**



Trap

- To obtain services from the operating system, a user program must make a system call, which traps into the kernel and invokes the operating system.
- The **Trap** instruction switches from user mode to kernel mode and starts the operating system.
- When the work has been completed, control is returned to the user program at the instruction following the system call.

Operating System Design and Implementation

OS -- Design and Implementation

- Design and Implementation of OS is not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications

Operating System Design Goals

- Affected by choice of hardware, type of system
- User goals and System goals
  - User goals: OS should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals: OS should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- The two goals are contradictory sometimes
  - In the past, the secondary goal is more important.

System Implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.
  - is more compact.
  - is easier to understand and debug.
  - is far easier to port (move to some other hardware)

History and Evolution of OS

History and Evolution of OS

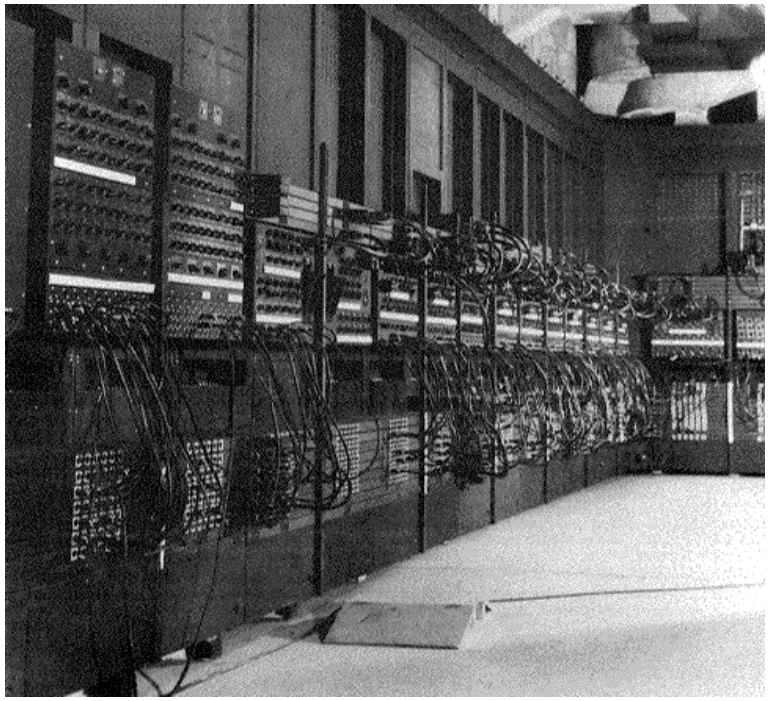
- History of Operating Systems
  - Really a history of resource-driven choices
- Evolution of an Operating System
  - Hardware upgrades plus new types of hardware
  - New services
  - Fixes
    - I/O Devices Slow – Multiprogramming
    - Interactive jobs - Time Sharing

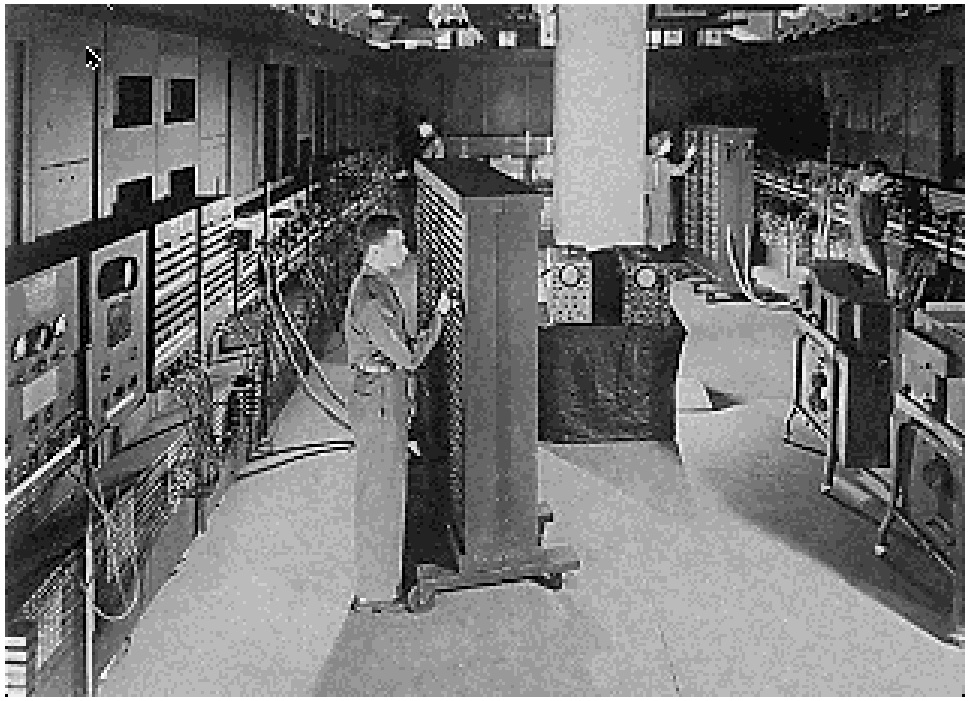
Moore's law effects

- Transportation in over 200 years:
  - 2 orders of magnitude from horseback @10mph to Concorde @1000mph
  - Computers do this every decade!
- What does this mean for us?
  - Techniques have to vary over time to adapt to changing tradeoffs

Dawn of time ENIAC: (1945-1955)

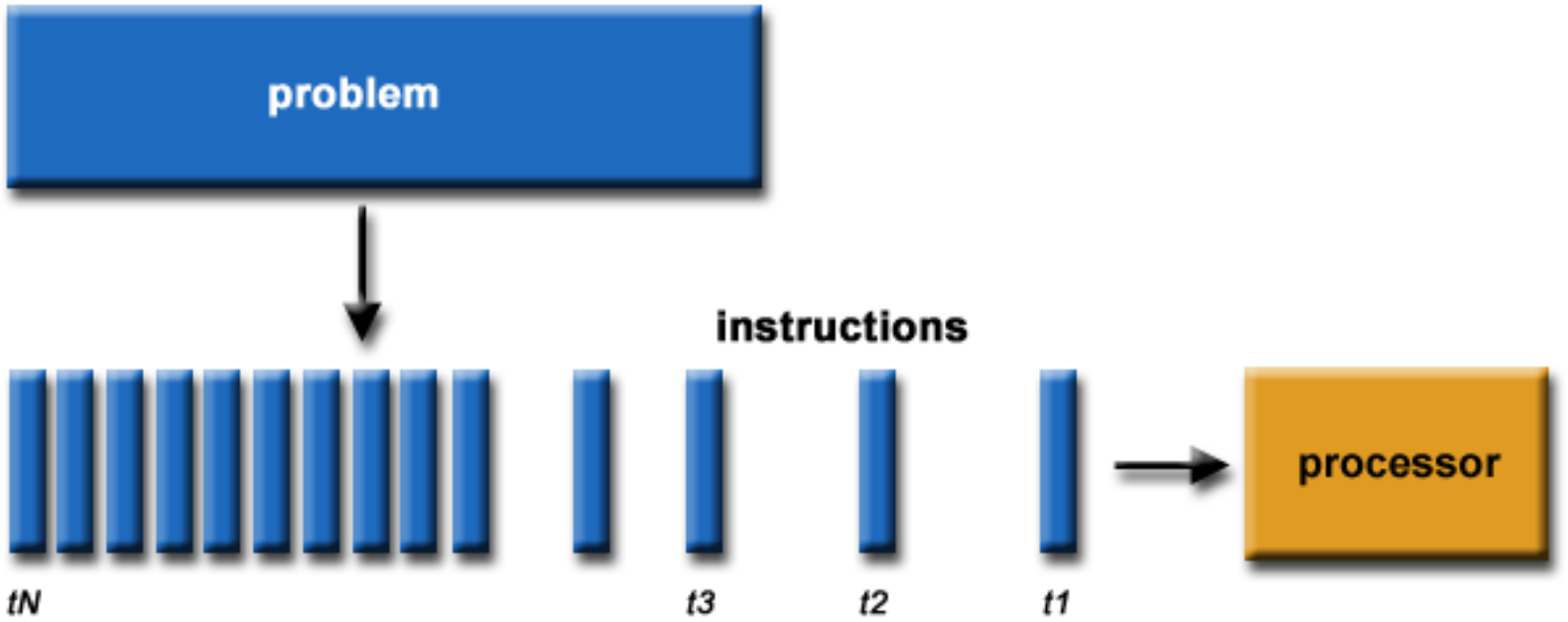
- “The machine designed by Drs. Eckert and Mauchly was a monstrosity. When it was finished, the ENIAC filled an entire room, weighed thirty tons, and consumed two hundred kilowatts of power.” http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML





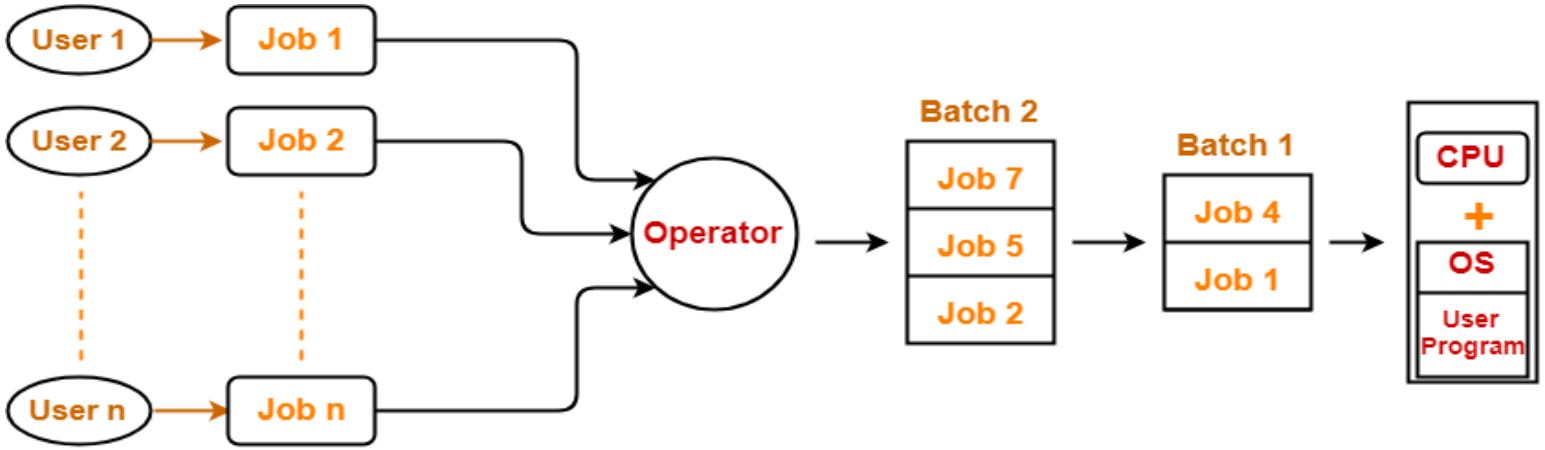
Evolution of Operating Systems

- Serial Processing
  - No operating system
  - Source Code ⇒ Compiler ⇒ Object Code ⇒ Hardware



History Phase 1 (1948-1970)

- Hardware Expensive, Humans Cheap
  - When computers cost millions of $'s, optimize for more efficient use of the hardware!
  - Early Batch System
    - Lack of interaction between user and computer



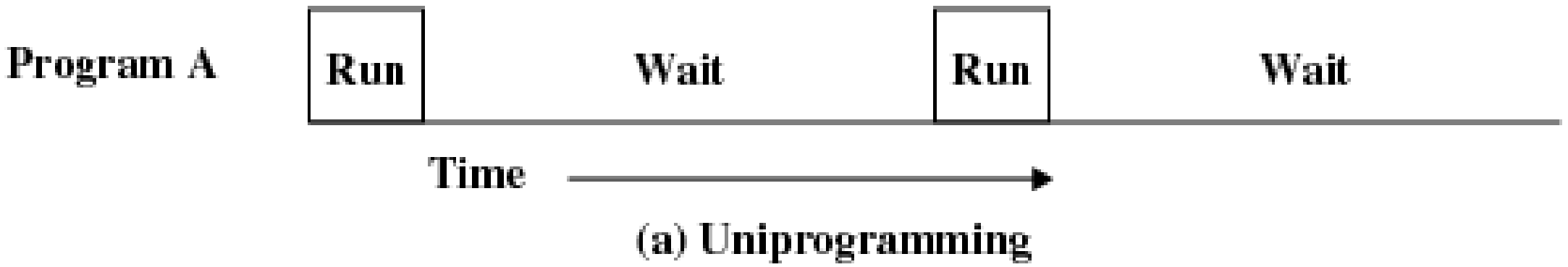
Batch Operating System

History Phase 1½ (late 60s/early 70s)

- Interrupts
- Multiprogramming: several programs run simultaneously
  - Need memory protection between programs and/or OS
- Complexity gets out of hand:
  - Multics: announced in 1963, ran in 1969
    - 1777 people “contributed to Multics”
    - Turning award lecture from Fernando Corbató (key researcher): “On building systems that will fail”
  - OS 360: released with 1000 known bugs
- OS finally becomes an important science

Uniprogramming

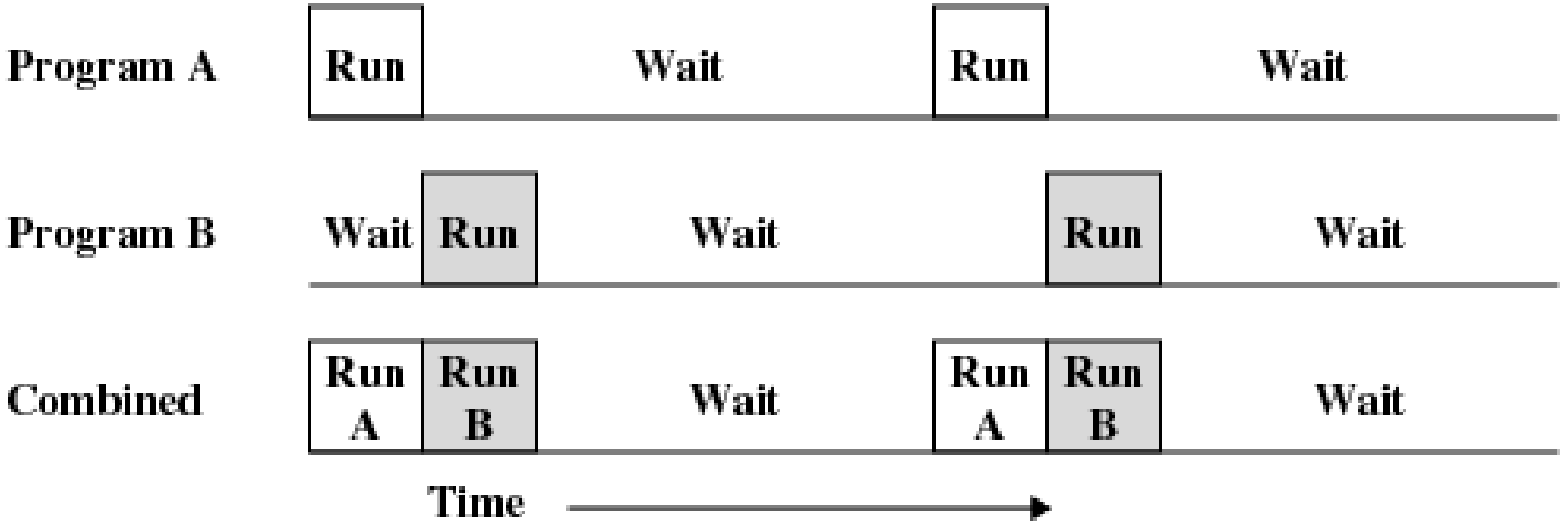
- Processor must wait for I/O instruction to complete before preceding



(a) Uniprogramming

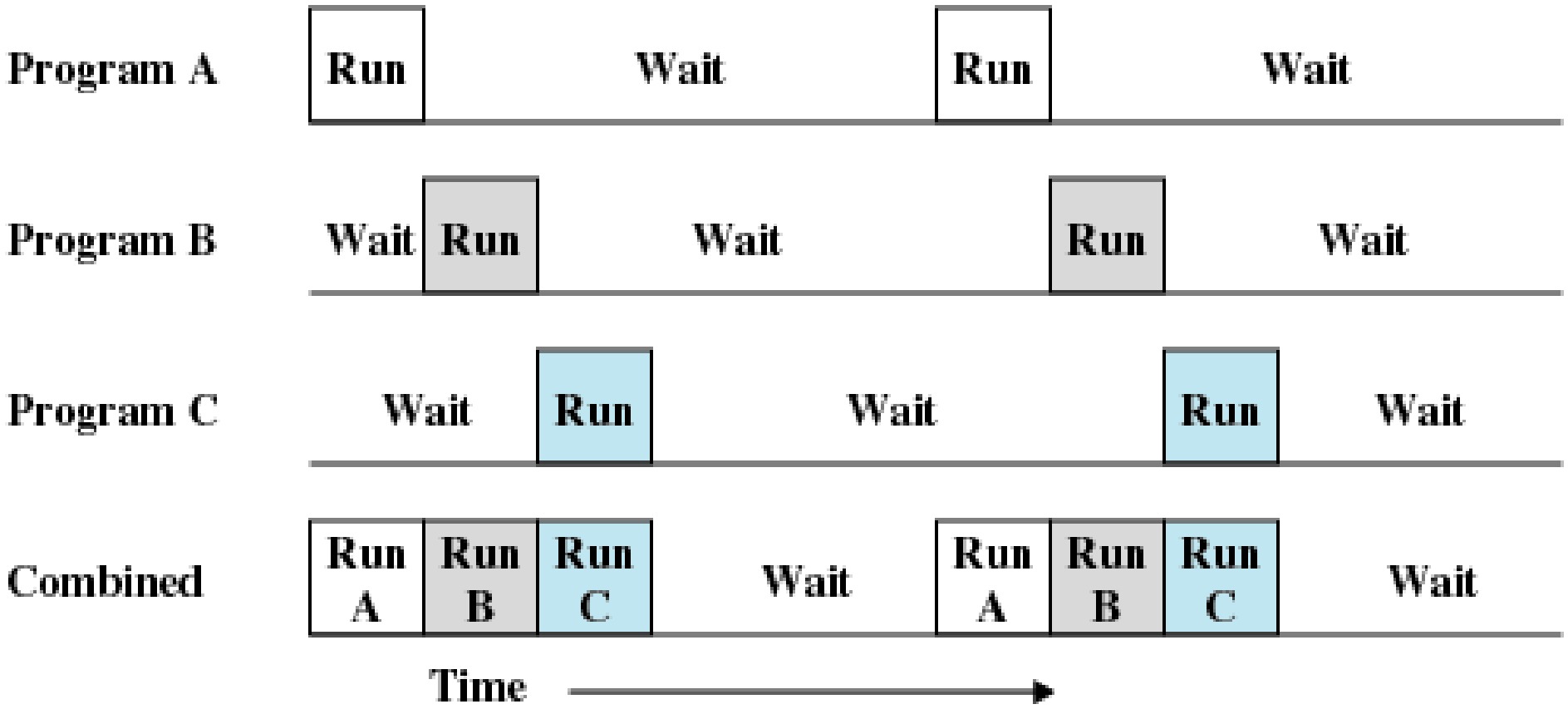
Multiprogramming

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



(b) Multiprogramming with two programs

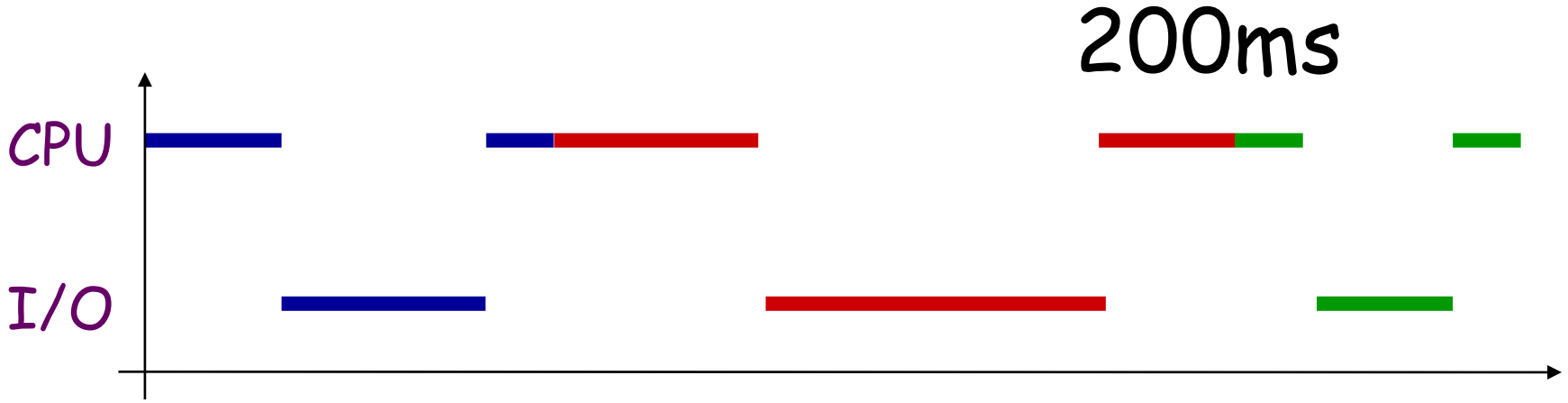
Multiprogramming



(c) Multiprogramming with three programs

Multiprogramming

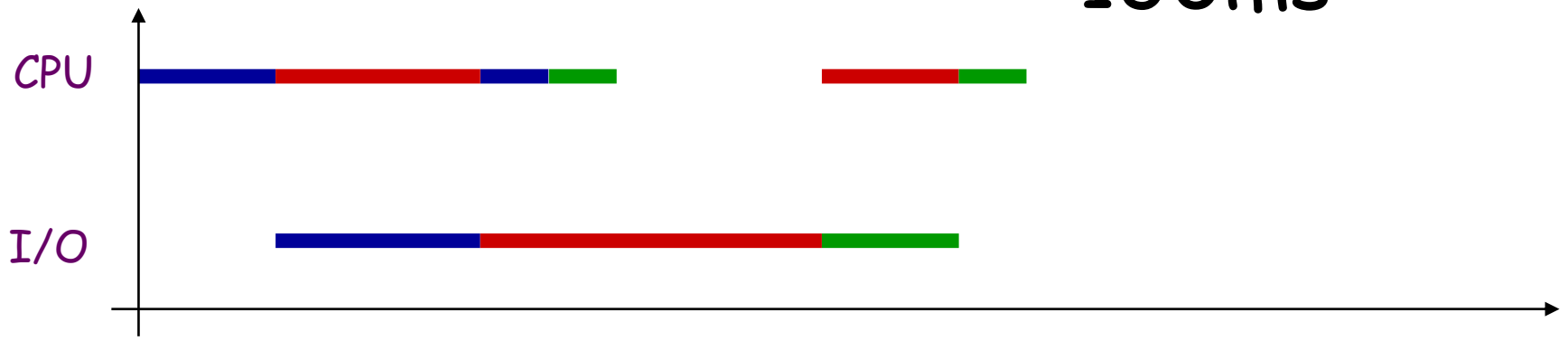
Program	Operation		
	CPU	I/O	CPU
A	20ms	30ms	10ms
B	30ms	50ms	20ms
C	10ms	20ms	10ms



Multiprogramming

Program	Operation		
	CPU	I/O	CPU
A	20ms	30ms	10ms
B	30ms	50ms	20ms
C	10ms	20ms	10ms

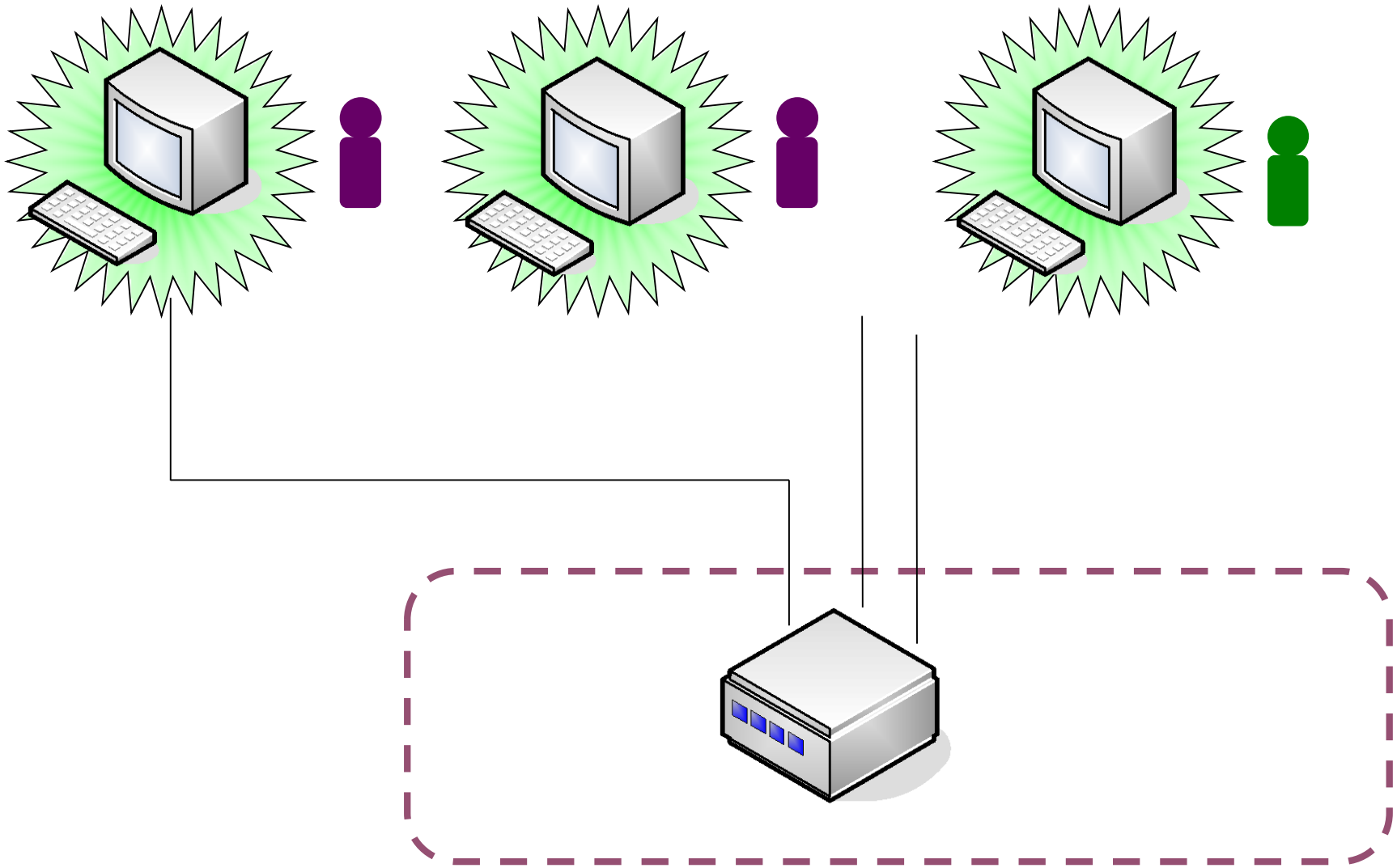
130ms



History Phase 2 (1970-1985)

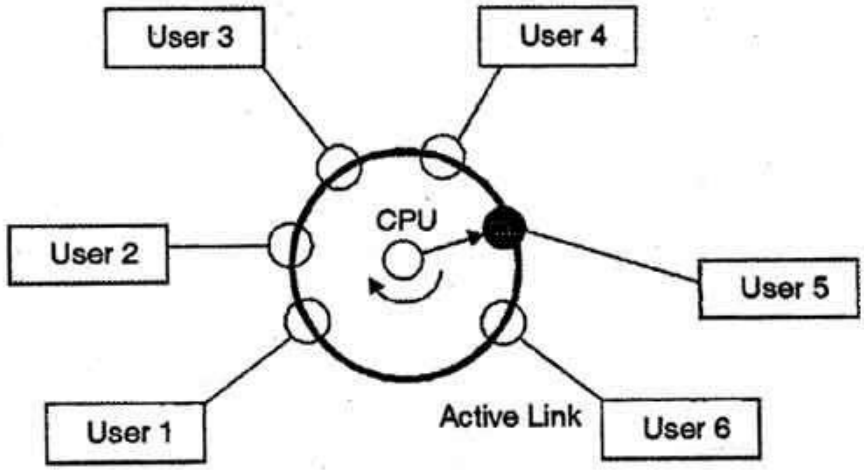
- Hardware Cheaper, Humans Expensive
- Computers available for tens of thousands of dollars instead of millions
- OS Technology maturing/stabilizing
- Interactive timesharing:
  - Use cheap terminals (~$1000) to let multiple users interact with the system at the same time
  - Sacrifice CPU time to get better response time
  - Users do debugging, editing, and email online

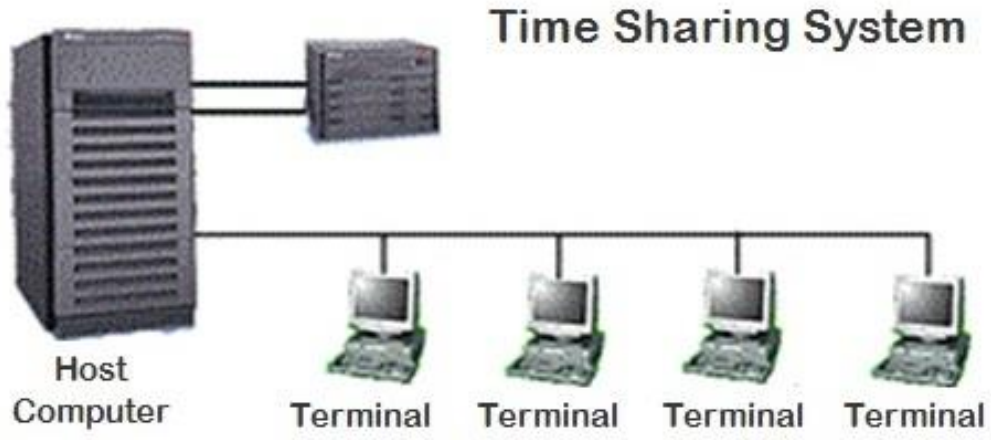
Time Sharing



What is Time Sharing?

- Time Sharing
  - Handle multiple interactive jobs
  - Processor's time is shared among multiple users
  - Multiple users simultaneously access the system through terminals





Multiprogramming and Timesharing

- **Multiprogramming** is needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**

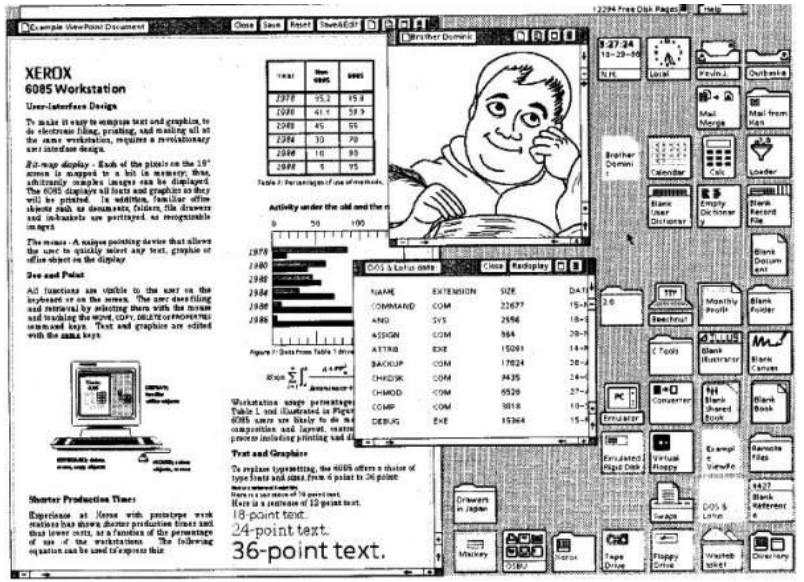
History Phase 3 (1981-)

- Hardware Very Cheap, Humans Very Expensive
  - Computer costs $1K, Programmer costs $100K/year
  - If you can make someone 1% more efficient by giving them a computer, it's worth it!
  - Use computers to make people more efficient
- Personal computing:
  - Computers cheap, so give everyone a PC
- Limited Hardware Resources Initially:
  - OS becomes a subroutine library
  - One application at a time (MSDOS, …)

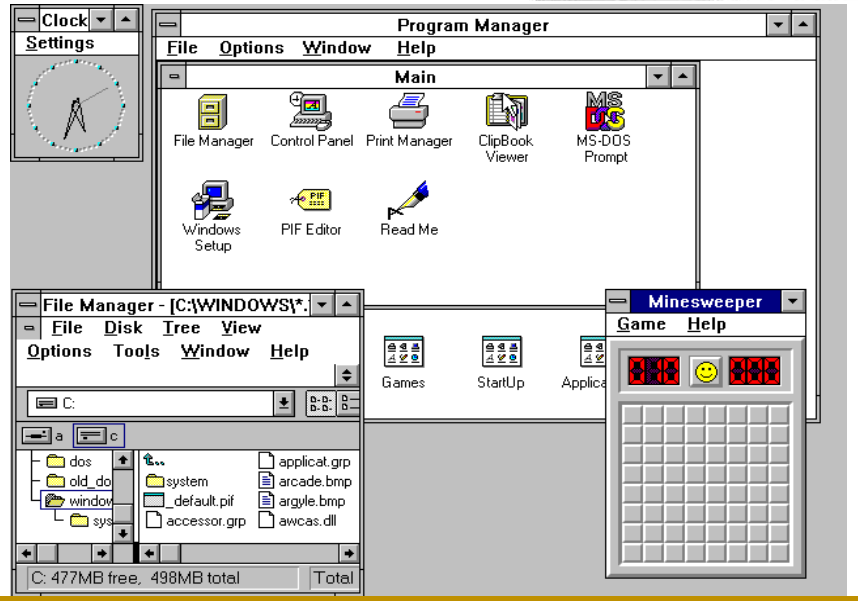
History Phase 3 (con't)

- Graphical User Interfaces
- Xerox Star: 1981
- Apple Lisa/Machintosh: 1984
  - "Look and Feel" suit 1988
- Microsoft Windows:
  - Win 1.0 (1985)
  - Win 3.1 (1990)
  - Win 95 (1995)
  - Win NT (1993)
  - Win 2000 (2000)
  - Win XP (2001)
  - Win 10

Xerox Star

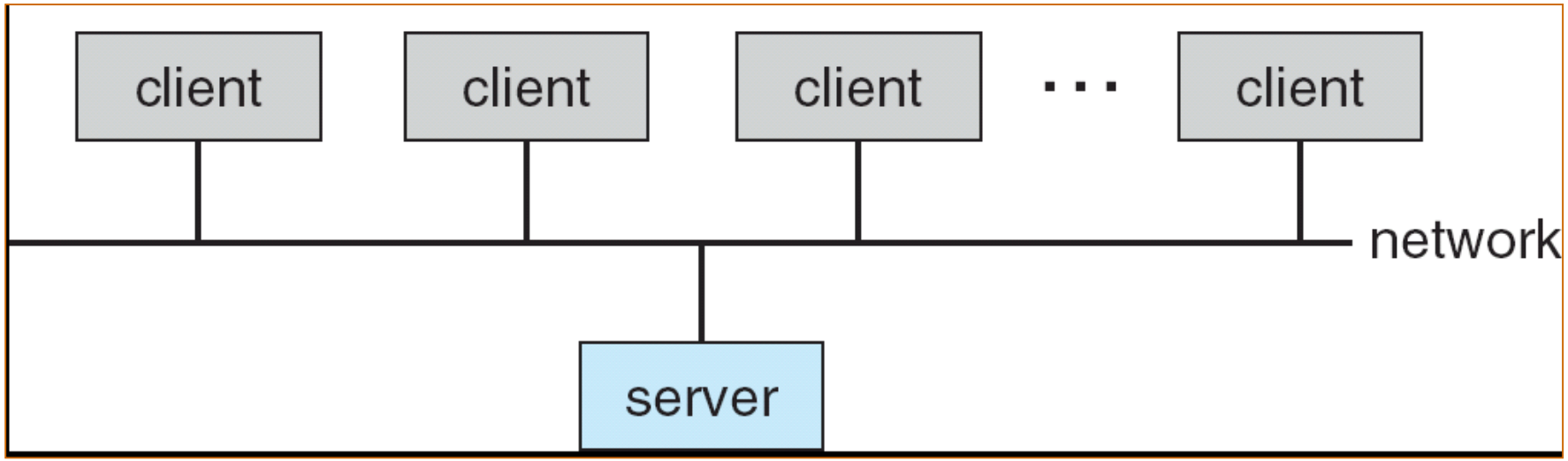


Windows 3.1



History Phase 4 (1989—):

- Distributed Systems
- Networking
  - Different machines share resources
- Services
  - Computing
  - File Storage



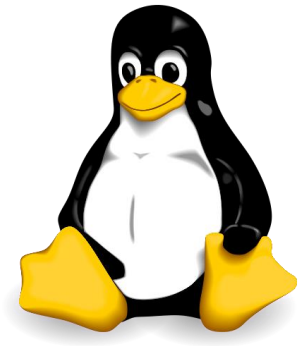
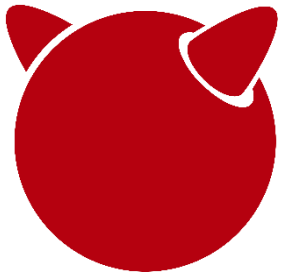
History Phase 5 (1995-): Mobile Systems

- Ubiquitous Mobile Devices
  - Laptops, PDAs, phones
  - Small, portable, and inexpensive
  - Limited capabilities (memory, CPU, power, Small display screens, etc.…)



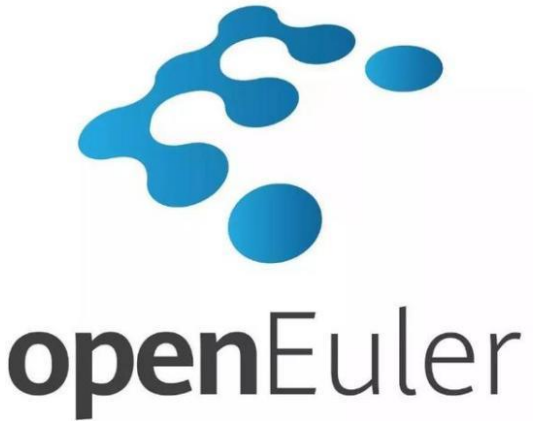

Open-Source Operating Systems

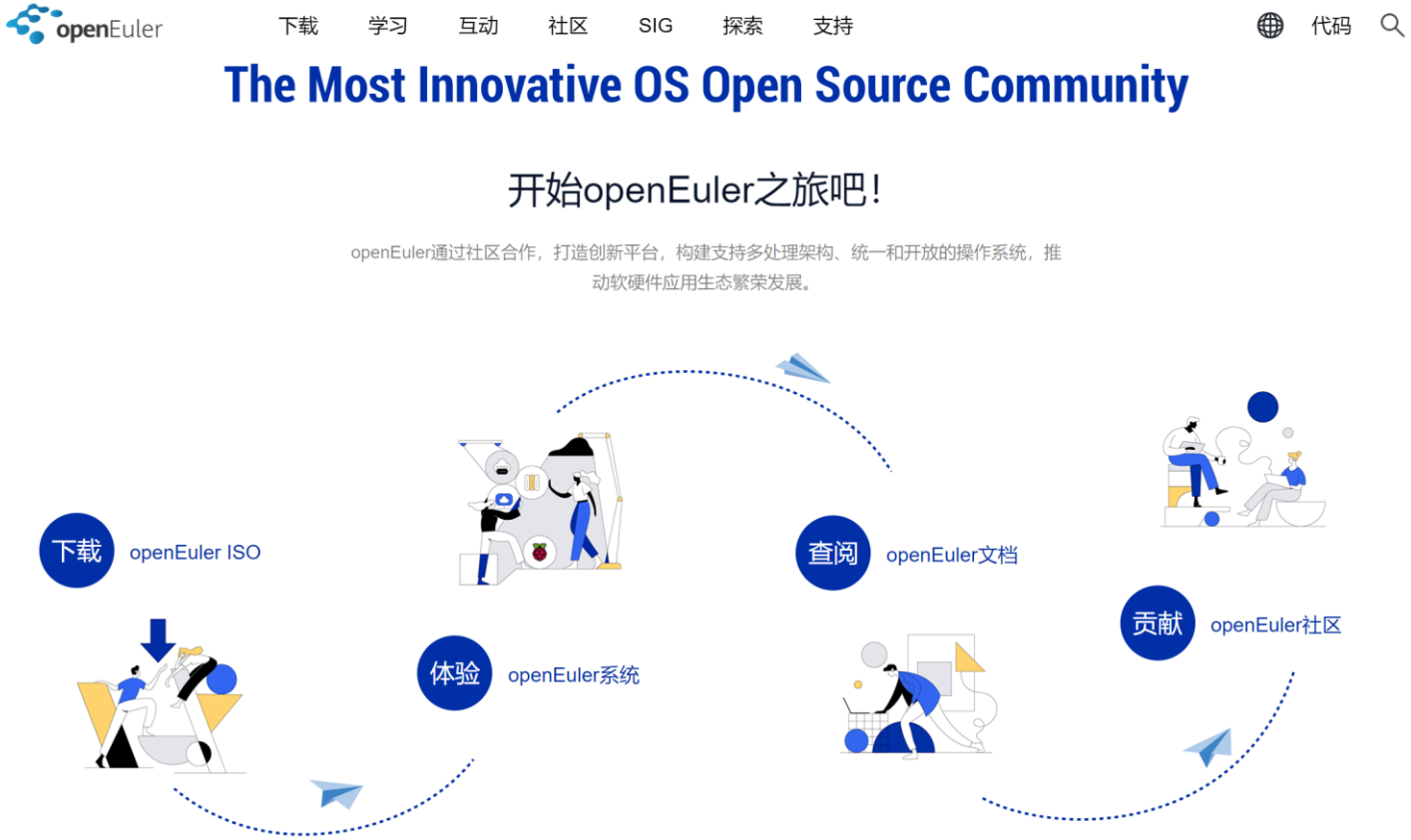
- Started by Free Software Foundation (FSF), which has “copyleft” GNU GPL (GNU General Public License)
- Examples include GNU/Linux, BSD UNIX (including core of Mac OS X), and Sun Solaris

Open-Source Operating Systems

- https://www.openeuler.org/zh/





Our Operating Systems

- 麒麟操作系统：Kylin OS

  www.ubuntukylin.com/

- 统信操作系统：UOS

  www.uniontech.com/

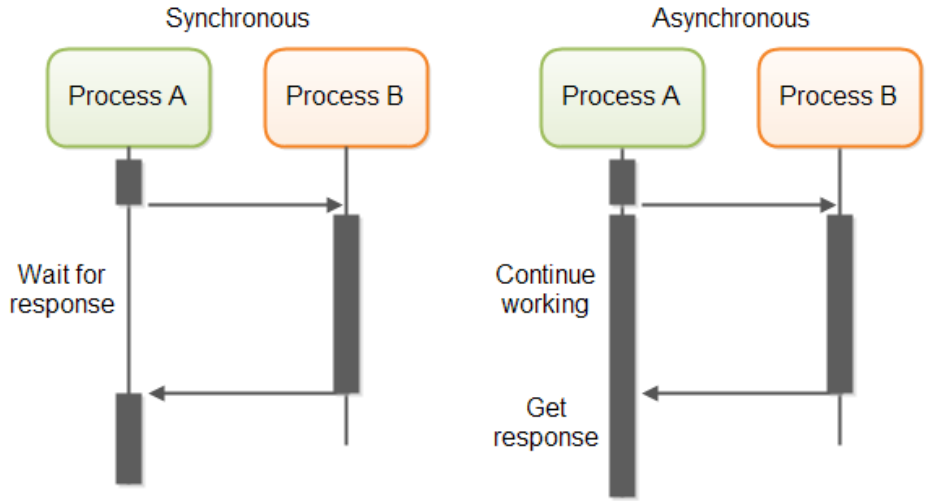
Characteristics (features) of an OS

Characteristics(features) of an OS

- Concurrency
  - Multiple events occurring at the same time period.
- Sharing
  - Multiple processes to share limited computer system resources.
  - Exclusive sharing / At the same time sharing
  - Difficult to achieve optimal allocation of resources

Characteristics(features) of an OS

- Virtualization
  - A physical entity maps to a number of the corresponding logical entity - timeshare or space sharing.
- Asynchronization
  - The completion time of the service is uncertain and may fail



OS Instances

What OSes are there?

- Microsoft OS: MS DOS, MS Windows 3.x, 95, 98, ME, NT, 2000, XP, Vista 7, 8, 10
- UNIX: BSD, SRV4, OSF1, SCO UNIX, IRIX, AIX, Solaris, Linux
- Embedded OS: WinCE, Symbian, uClinux, Palm OS
- Mobile OS: Android, iOS. Harmony OS

Source: http://www.osdata.com/kind/summary.htm

Operating Systems

History of Windows

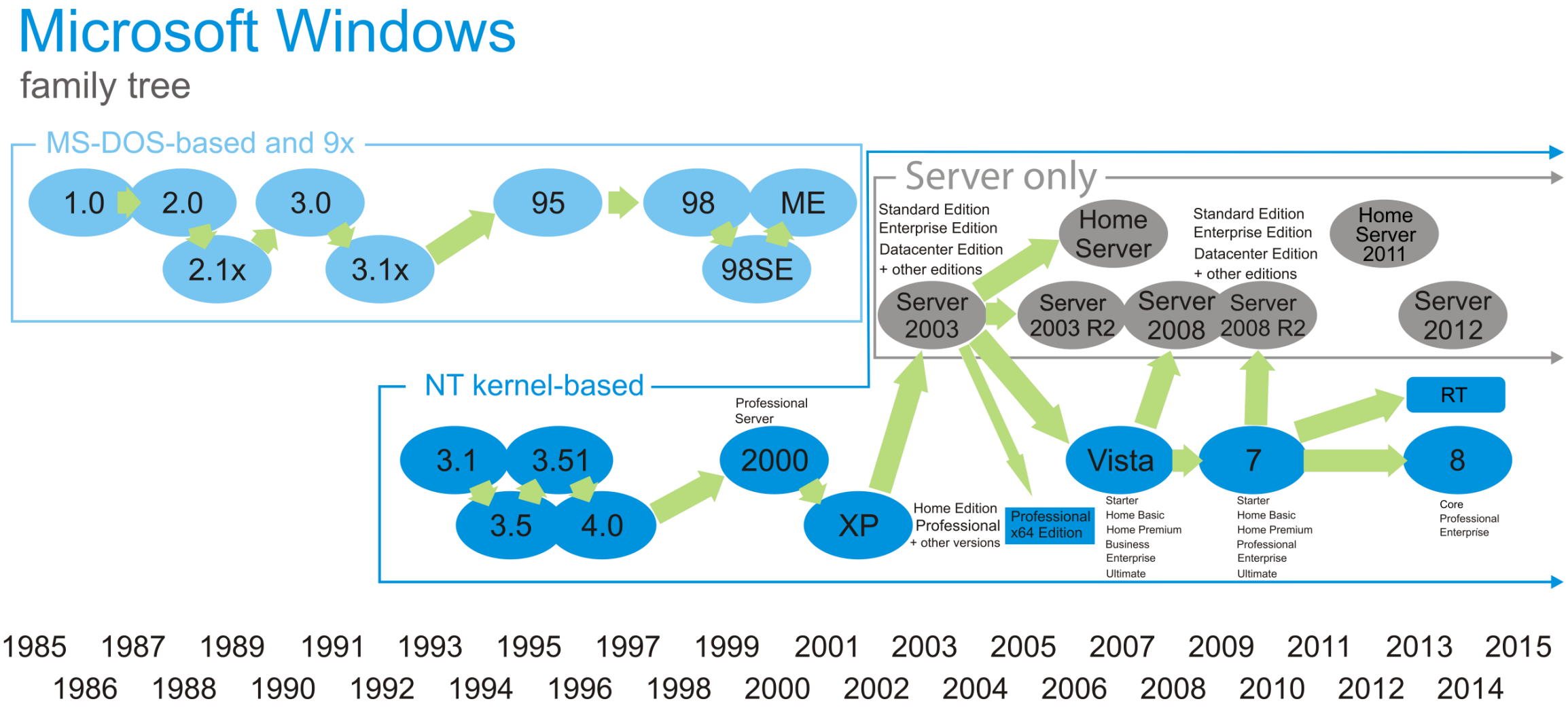
Year	MS-DOS	MS-DOS based	NT-kernel based
1981	MS-DOS 1.0		
1983	MS-DOS 2.0		
1984	MS-DOS 3.0		
1990		Windows 3.0	
1991	MS-DOS 5.0		
1992		Windows 3.1	
1993			Windows NT 3.1
1995	MS-DOS 7.0	Windows 95	
1996			Windows NT 4.0
1998		Windows 98	
2000	MS-DOS 8.0	Windows Me	Windows 2000
2001			Windows XP
2006			Windows Vista

2009: Windows 7

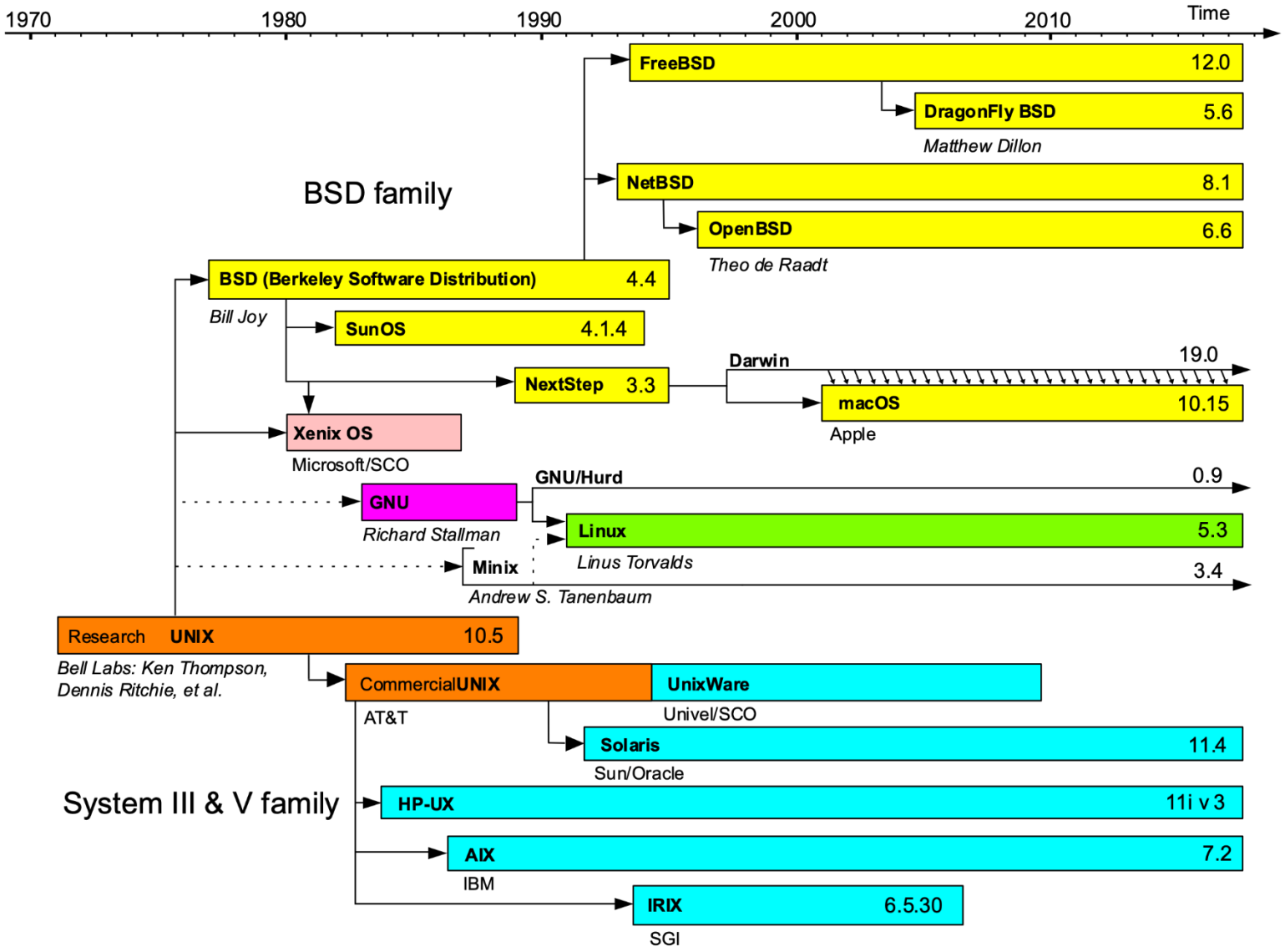
2012: Windows 8

2015: Windows 10

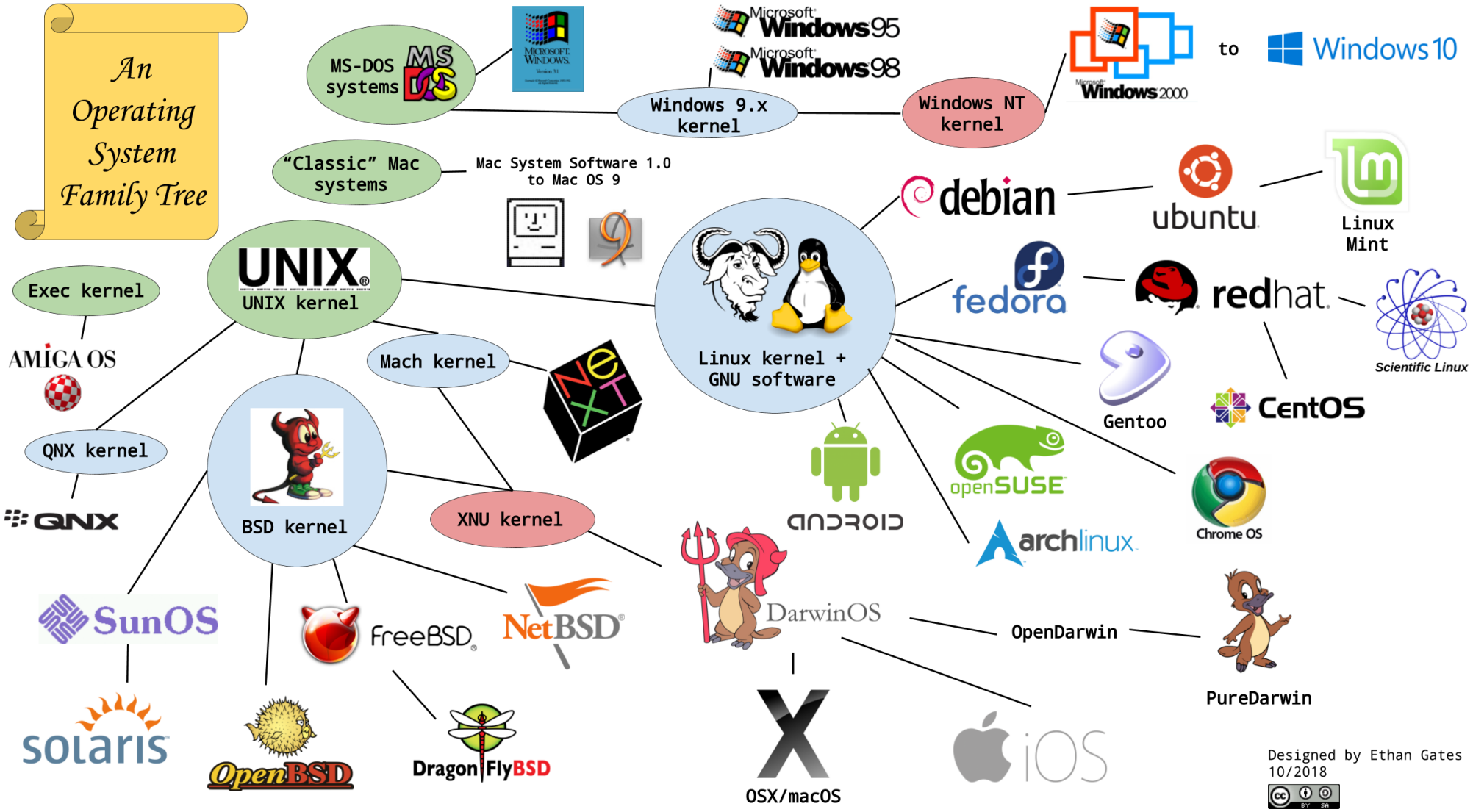
Microsoft Windows Family Tree



Unix Family Tree



Operating System Family Tree



Why to study operating system?

Why to study OS?

- OS is a software running on most machines
  - PC, Laptop, smartphone, watch …
- OS is a highly complicated software
  - Windows: 50M lines of source code
  - Linux: 25M lines of source code

- It contains many important techniques and principles
  - Resource allocation, performance tunning…
- Studying OS internals makes you a **capable** programmer
  - Know how it works and how it works better

Why to study OS?

- OS is the fundation of ALL software
  - Better user-space software
    - Invoke proper kernel API and consistent with kernel design
    - Know what can and cannot be done
  - Better performance
    - Resource allocation
    - Caching
- OS concepts benefit your whole life
  - OS concepts are re-usable when implementing other software
  - Lessons learned from OS can be applied to complex software systems, such as Hadoop and OpenStack

Why to study OS?

- Existing OSes are very good, I probably will not develop an OS
  - You probability land a super interesting job
- Do the existing OSes solve all the problems?
  - Multi-core/parallel
  - Non-Volatile storage
  - Security
  - Energy
  - Mobility/AIot
  - Correct/verification

Why are we studying OS?

- A lot of organizations are doing OS research
  - TOP universities
    - MIT, Stanford, Berkeley, …
  - Computing industries
    - Abroad: Microsoft, Google, Apple, IBM, HP, Amazon, VMware, …
    - Domestic: Huawei, Kylin, Uniontech, Alibaba, Tencent, …
  - Academic research association
    - SOSP, OSDI, HotOS
    - ACM SIGOPS
    - USENIX ATC
    - ……

Why to study OS?

- Non-textbook answers
  - For profit
    - Interview = coding + system design
    - Great system == Great product == Great Company

SYSTEM DESIGN INTERVIEW QUESTIONS AND ANSWERS

Google

ANDROID

Chrome OS

Fuchsia by Google

Wear OS by Google

How to learn operating system?

How to learn OS?

- Grasp the key operating system issues
  - OS manages concurrency
    - Concurrency leads to interesting programming challenges
  - OS manages the raw hardware
    - CPU, memory, disk, ……
    - Time dependent behavior, illegal behavior, hardware failure
  - OS must be efficient, low power consumption, secure and reliable
    - OS should provide reasonable resources to the application timely
    - An OS error means a machine error
    - OS must be more stable than user programs

How to learn OS?

- Learning OS requires systematic thinking
  - OS is not just trivial scheduling algorithms
    - Most disk scheduling algorithm have been implemented by hardware
    - Process scheduling is a minor topic
  - Concurrency is a small part of OS
    - There is no pipe and philosopher problem in the kernel
    - The kernel mechanism needs to consider the application and hardware
  - Resource tradeoff
    - Time and space – predictability and fairness of performance
  - Collaboration between software and hardware
    - How to make interrupts, exceptions, and context switches really work

北京交通大学

Thank you !

Q & A