操作系统

第2章 操作系统概述 Operating System Overview

孙承杰
哈工大计算学部

E-mail： sunchengjie@hit.edu.cn
2025年秋季学期

Learning Objectives

- Summarize the key functions of an operating system
- Discuss the evolution of operating systems for early simple batch systems to modern complex systems.
- Give a brief explanation of the major achievements
- discuss virtual machines and virtualization
- Discuss Windows, UNIX and Linux

Outline

- Operating System Objectives and Functions
- Evolution of Operating Systems
- Major Achievements
- Virtual Machines and Virtualization
- Traditional Operating System
  - Windows
  - Unix
  - Linux

Definition of Operating systems

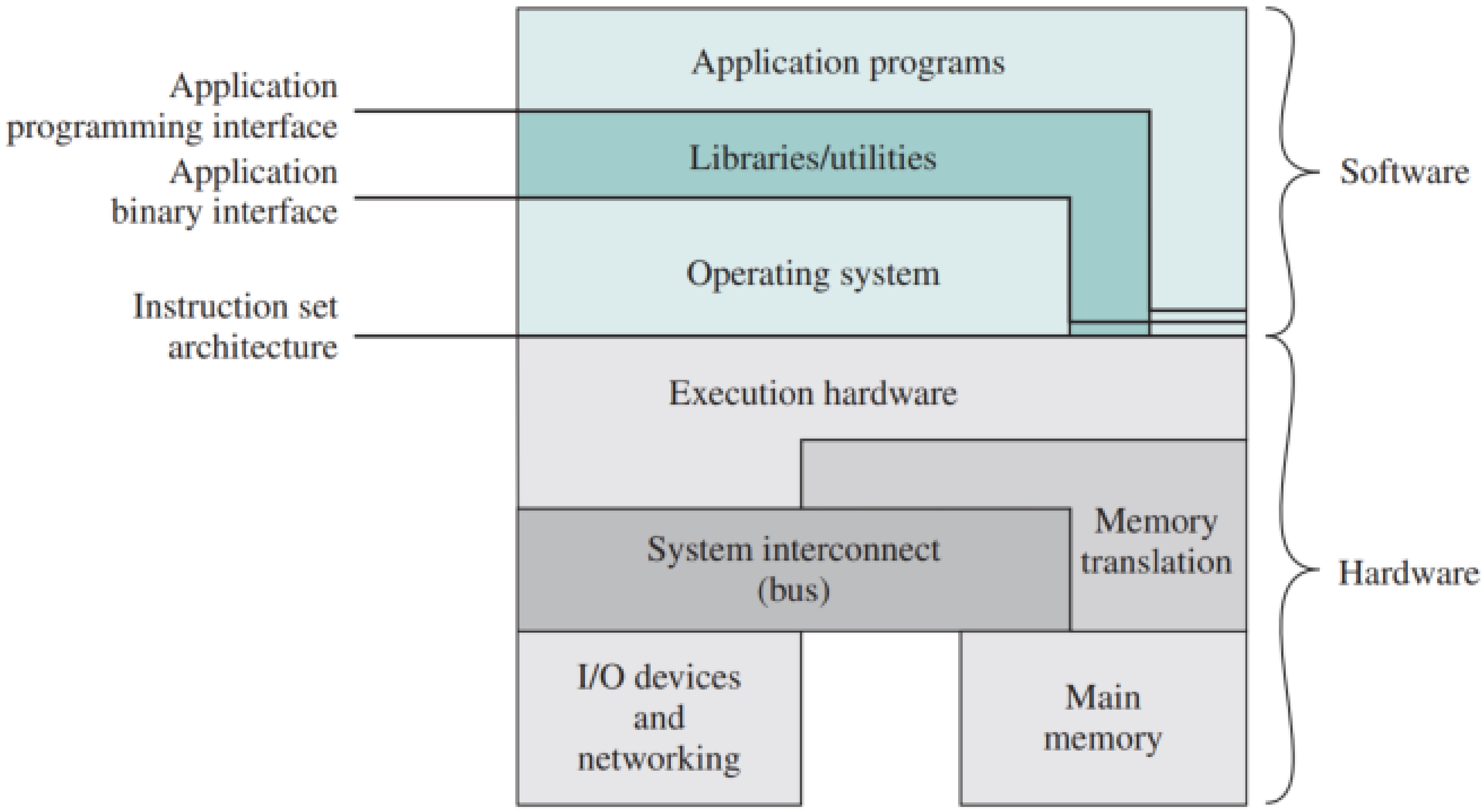
Operating systems are those programs that interface the machine with the applications programs. The main function of these systems is to dynamically allocate the shared system resources to the executing programs. As such, research in this area is clearly concerned with the management and scheduling of memory, processes, and other devices. But the interface with adjacent levels continues to shift with time. Functions that were originally part of the operating system have migrated to the hardware. On the other side, programmed functions extraneous to the problems being solved by the application programs are included in the operating system.

-- WHAT CAN BE AUTOMATED?: THE COMPUTER SCIENCE AND ENGINEERING RESEARCH STUDY,
MIT Press, 1980

OS Objectives and Functions

- A program that controls the execution of application programs
- An interface between applications and hardware
- Main objectives of an OS
  - Convenience
  - Efficiency
  - Ability to evolve （扩展能力）

As a User/Computer Interface



Computer Hardware and Software Structure

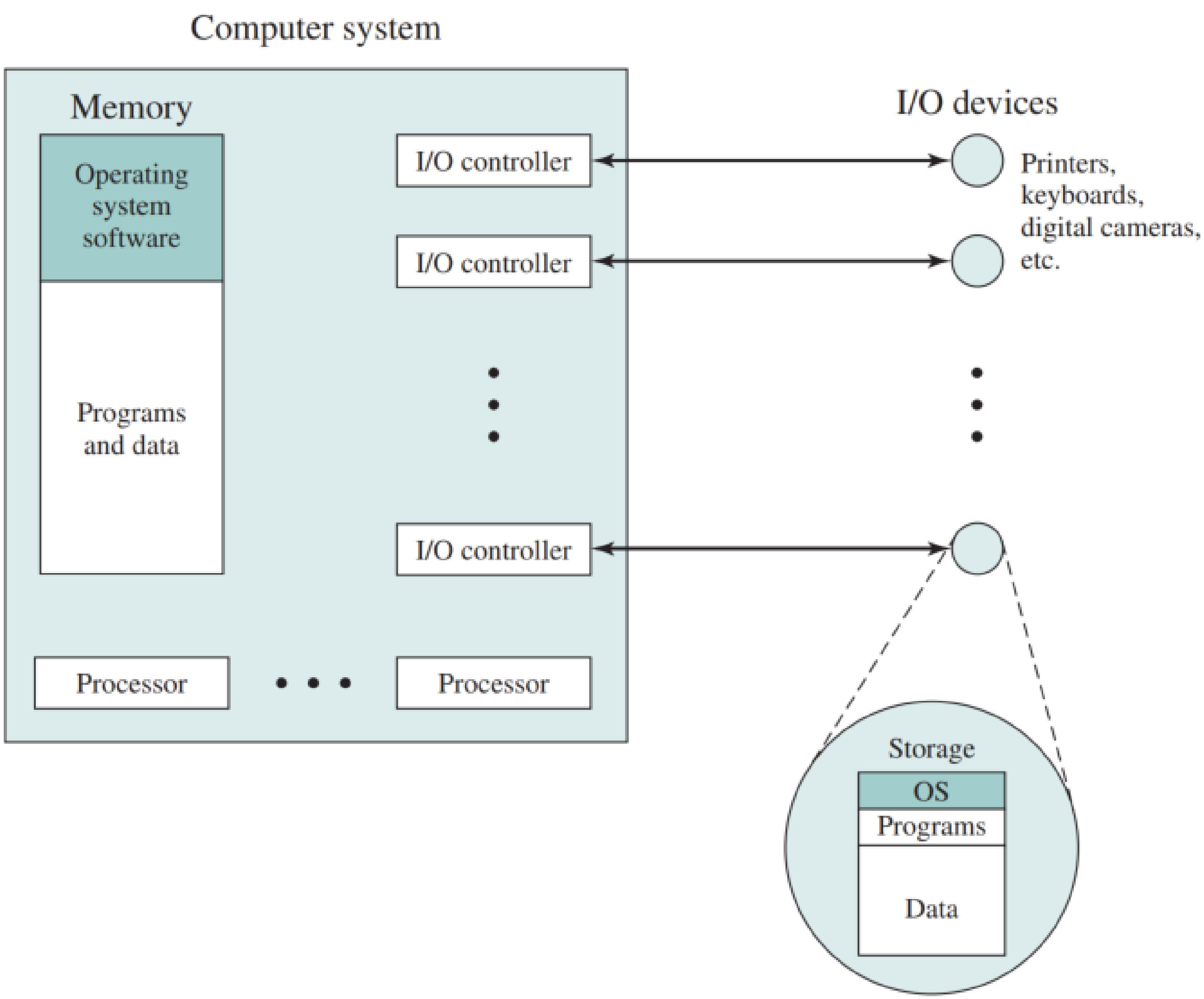
OS Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting
- Three key interfaces in a typical computer system
  - Instruction set architecture (ISA)
  - Application binary interface (ABI)
  - Application programming interface (API)

As Resource Manager

- A computer is a set of resources for the movement, storage, and processing of data
- The OS is responsible for managing these resources

As Resource Manager



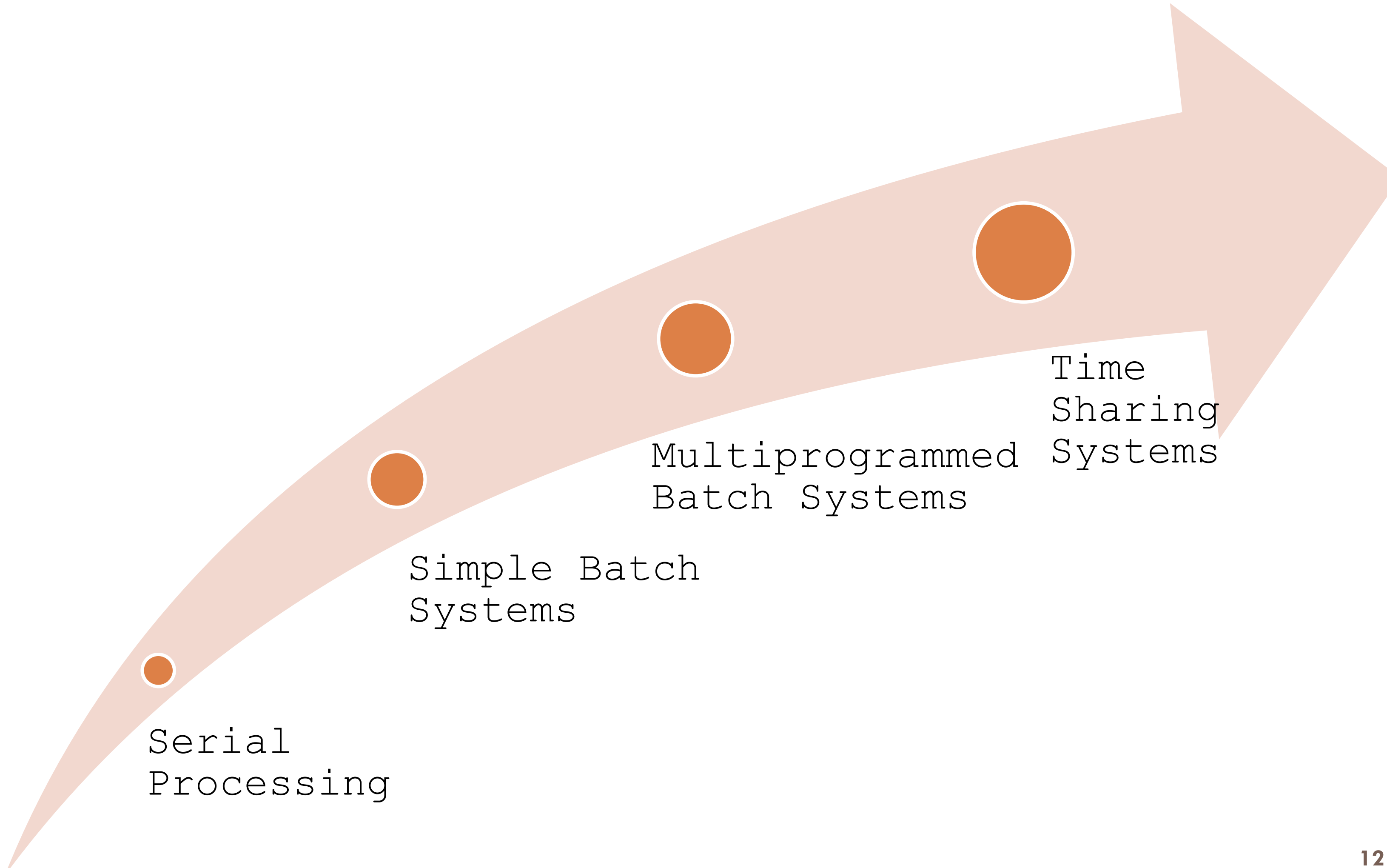
OS as a control mechanism

- Unusual in two respects
  - Functions in the same way as ordinary computer software
    - Program, or suite of programs, executed by the processor
  - Frequently relinquishes control and must depend on the processor to allow it to regain control

Evolution of Operating Systems

- OS will evolve over time for a number of reasons
  - Hardware upgrades
  - New types of hardware
  - New services
  - Fixes

Evolution of Operating Systems



Serial Processing

Earliest Computers

- No operating system
  - programmers interacted directly with the computer hardware
  - Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in "series"

Problems

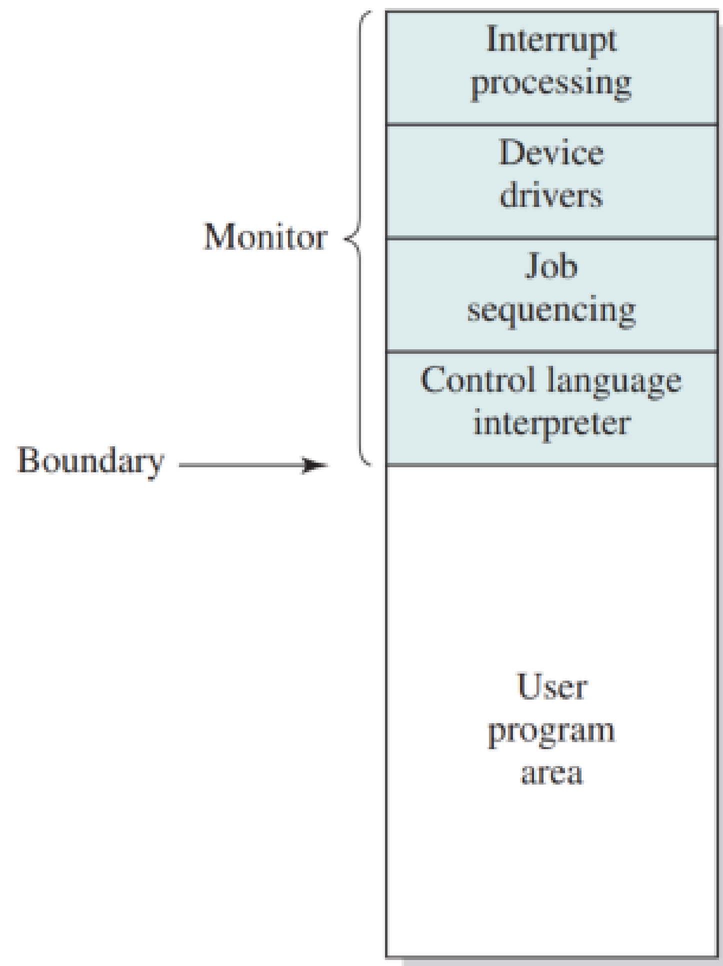
- Scheduling
  - most installations used a hardcopy sign-up sheet to reserve computer time
    - time allocations could run short or long, resulting in wasted computer time
- Setup time
  - A single program called a **job**
  - a considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - important to maximize processor utilization
- Monitor
  - user no longer has direct access to processor
  - job is submitted to computer operator who batches them
  - together and places them on an input device
  - program branches back to the monitor when finished
- To understand how this scheme works
  - monitor point of view
  - processor point of view

Monitor Point of View

- 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



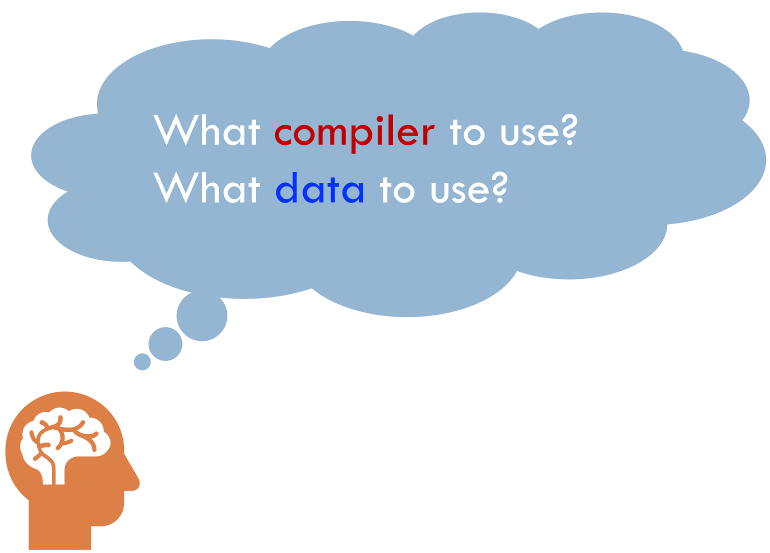
Memory Layout for a Resident Monitor

Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "control is passed to a job" means processor is fetching and executing instructions in a user program
- "control is returned to the monitor" means that the processor is fetching and executing instructions from the monitor program

Job Control Language (JCL)

- JCL is a special type of programming language used to provide instructions to the monitor
  - alternately seize and relinquish control
  - other hardware features are also desirable




```
$JOB
$FTN
•
•        FORTRAN instructions
•
$LOAD
$RUN
•
•        Data
•
$END
```

Desirable Hardware Features

Memory protection for monitor

- while the user program is executing, it must not alter the memory area containing the monitor

Timer

- prevents a job from monopolizing the system

Privileged instructions

- can only be executed by the monitor

Interrupts

- gives OS more flexibility in controlling user programs

Modes of Operation

User Mode

- user program executes in user mode
- certain areas of memory are protected from user access
- certain instructions may not be executed

Kernel Mode

- monitor executes in kernel mode
- privileged instructions may be executed
- protected areas of memory may be accessed

Considerations of memory protection and privileged instructions lead to the concept of modes of operation

Simple Batch System Overhead

- ☑ Despite overhead, the simple batch system improves utilization of the computer

- ☒ some main memory is now given over to the monitor
- ☒ some processor time is consumed by the monitor

Processor time alternates between execution of user programs and execution of the monitor

Multiprogrammed Batch Systems

- Processor 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

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

System Utilization Example

Multiprogramming

Multiprogramming

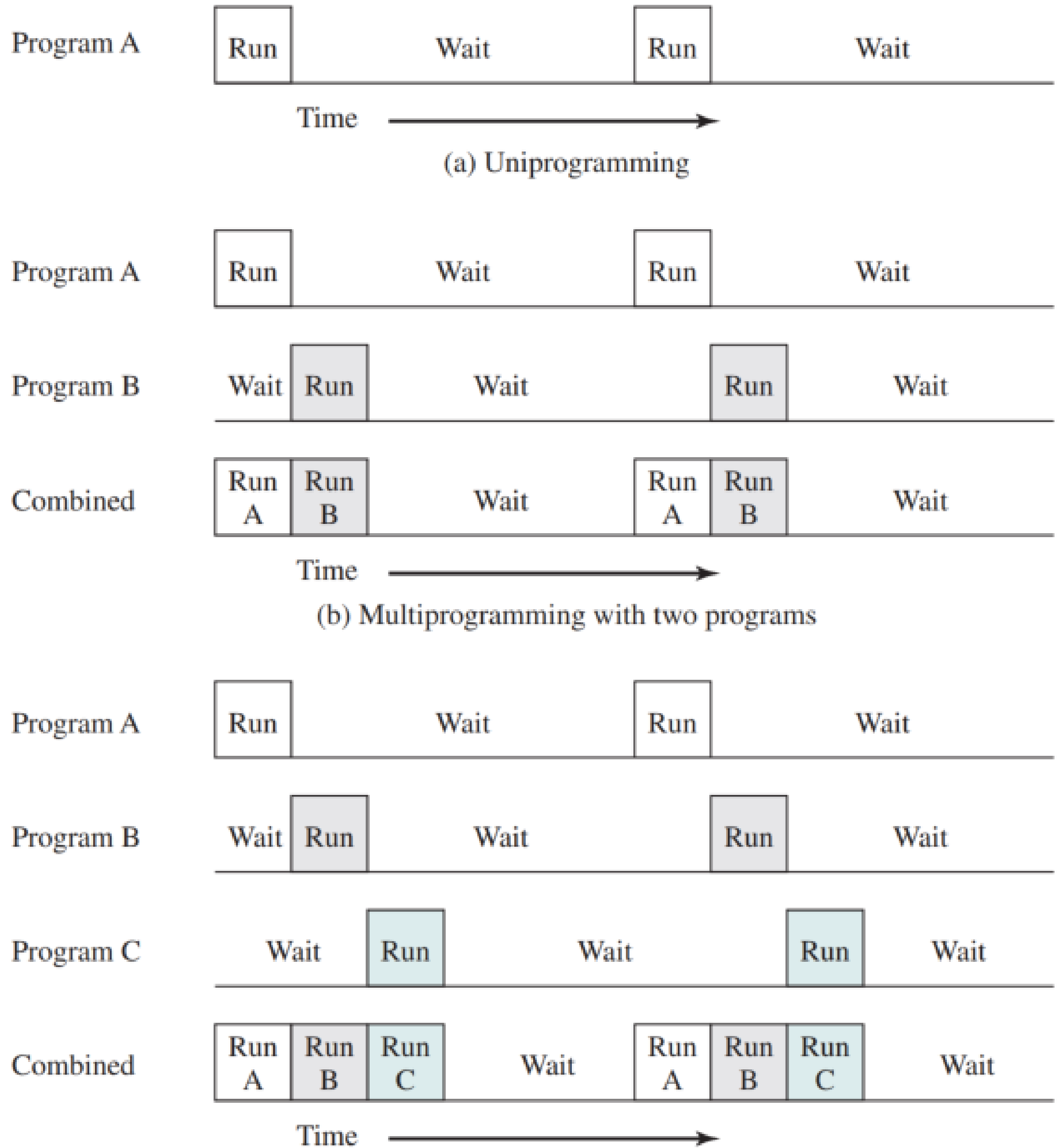
- a.k.a. multitasking
- memory is expanded to hold three, four, or more programs
- switch among all of them
- central theme of modern operating systems

Uniprogramming

- The processor spends a certain amount of time executing, until it reaches an I/O instruction
- it must then wait until that I/O instruction concludes before proceeding

memory management?
which one to run? scheduling

Multiprogramming Example



(a) Uniprogramming

(b) Multiprogramming with two programs

(c) Multiprogramming with three programs

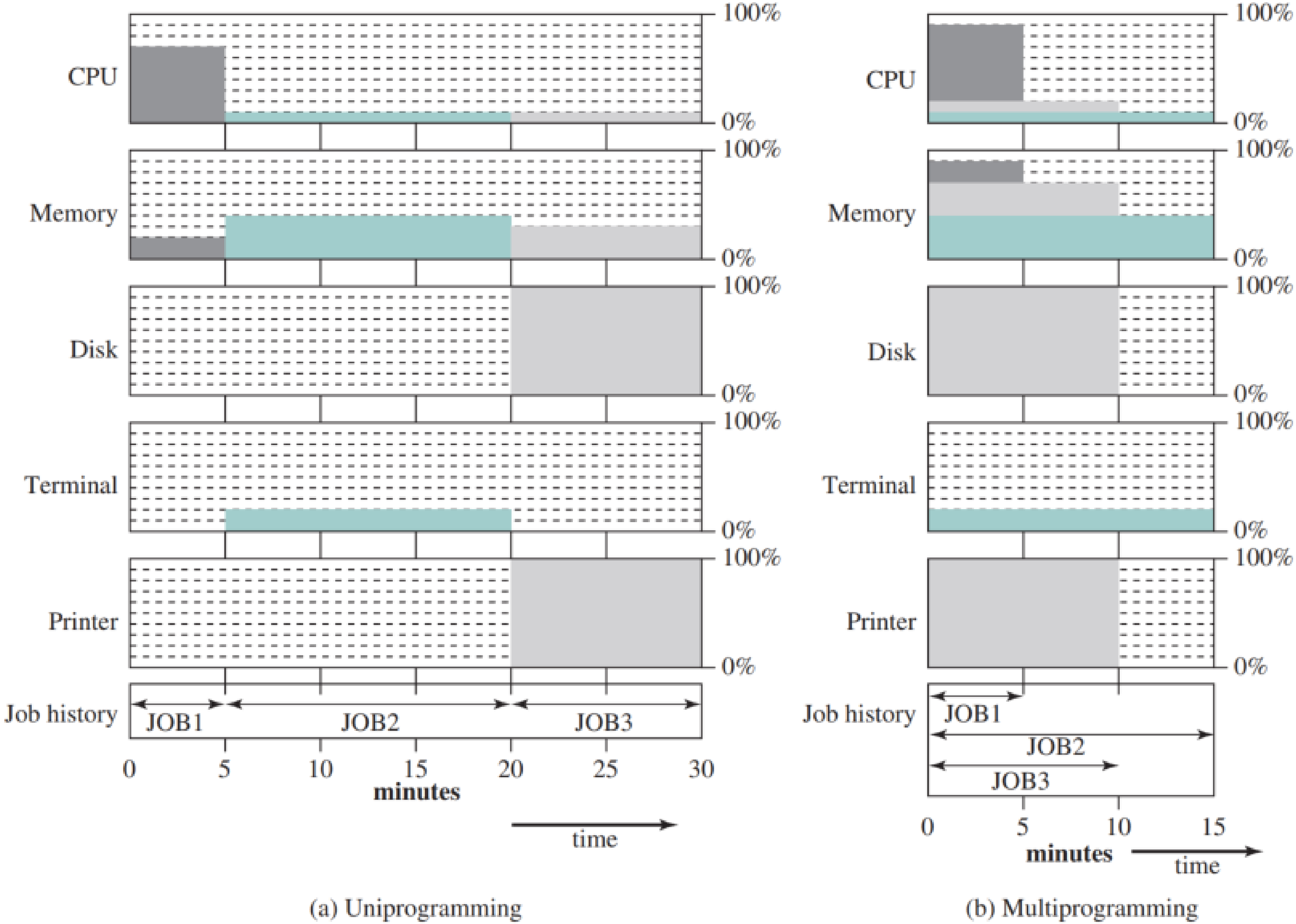
Multiprogramming Example

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Table 2.2 Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min



Utilization Histograms

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals

Table 2.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Compatible Time-Sharing Systems

CTSS

- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC--1961
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

Time Slicing

- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each interrupt OS regained control and could assign processor to another user
- At regular time intervals the current user would be preempted and another user loaded in
- Old user programs and data were written out to disk
- Old user program code and data were restored in main memory when that program was next given a turn

CTSS Operation

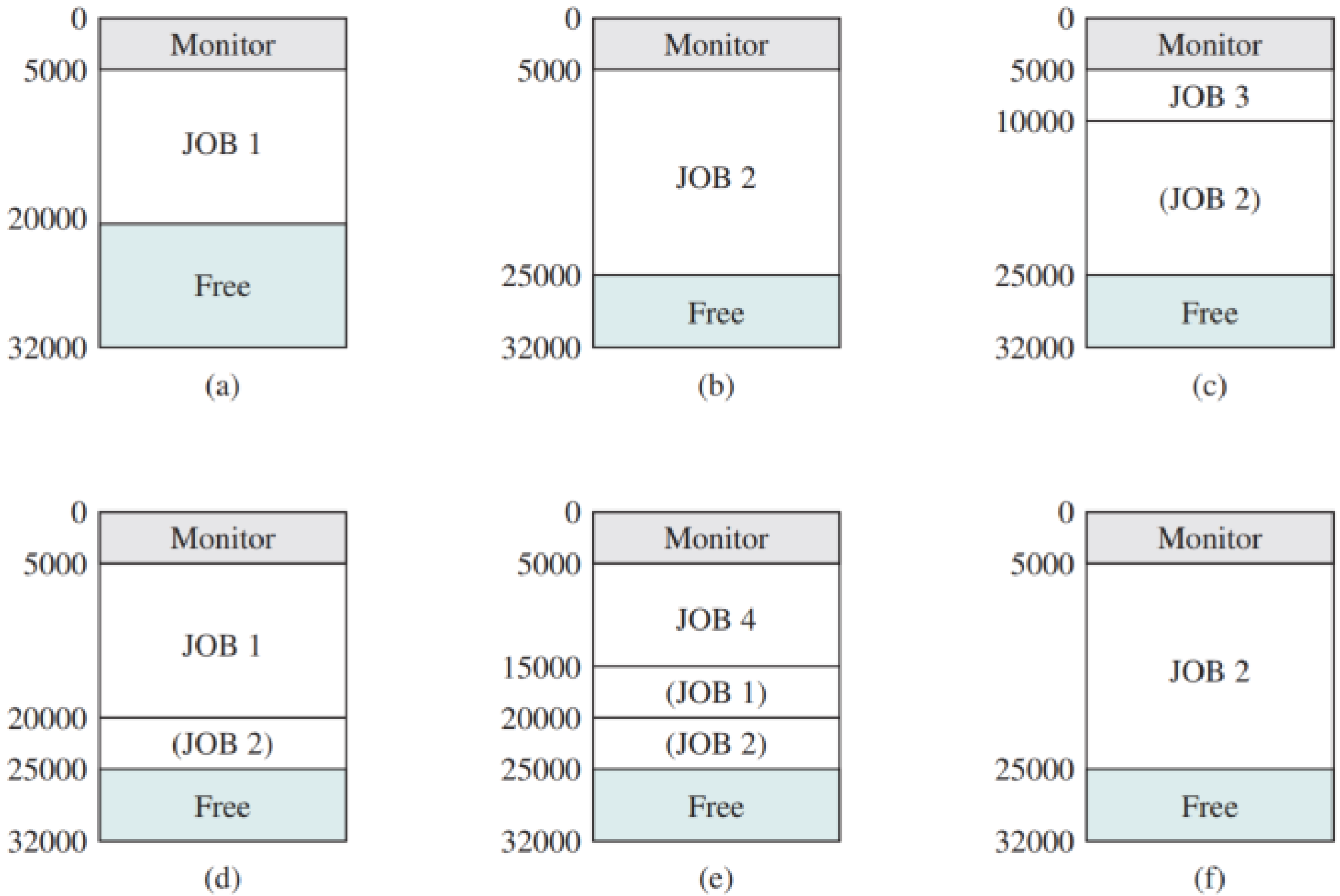


Figure 2.7 **CTSS Operation**

Major Advances

- Operating Systems are among the most complex pieces of software ever developed
- Major advances in development include:
  - Processes（进程）
  - Memory management
  - Information protection and security
  - Scheduling and resource management

Process

- Fundamental to the structure of operating systems
  - first used by the designers of Multics in the 1960s
  - a somewhat more general term than job
- Definition for the term process
  - A program in execution
  - An instance of a program running on a computer
  - The entity that can be assigned to and executed on a processor
  - A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Development of the Process

- Three major lines created problems in timing and synchronization that contributed to the development of the concept of the process

multiprogramming batch operation

- processor is switched among the various programs residing in main memory

time sharing

- be responsive to the individual user but be able to support many users simultaneously

real-time transaction systems

- a number of users are entering queries or updates against a database

Problems

- The design of the system software to coordinate these various activities (interrupts) turned out to be remarkably difficult
  - impossible to analyze all of the possible combinations of sequences of events
  - absence of some systematic means of coordination and cooperation among activities
  - programmers resorted to ad hoc methods based on their understanding
  - These efforts were vulnerable to subtle programming error whose effects could be observed only when certain relatively rare sequences of actions occurred

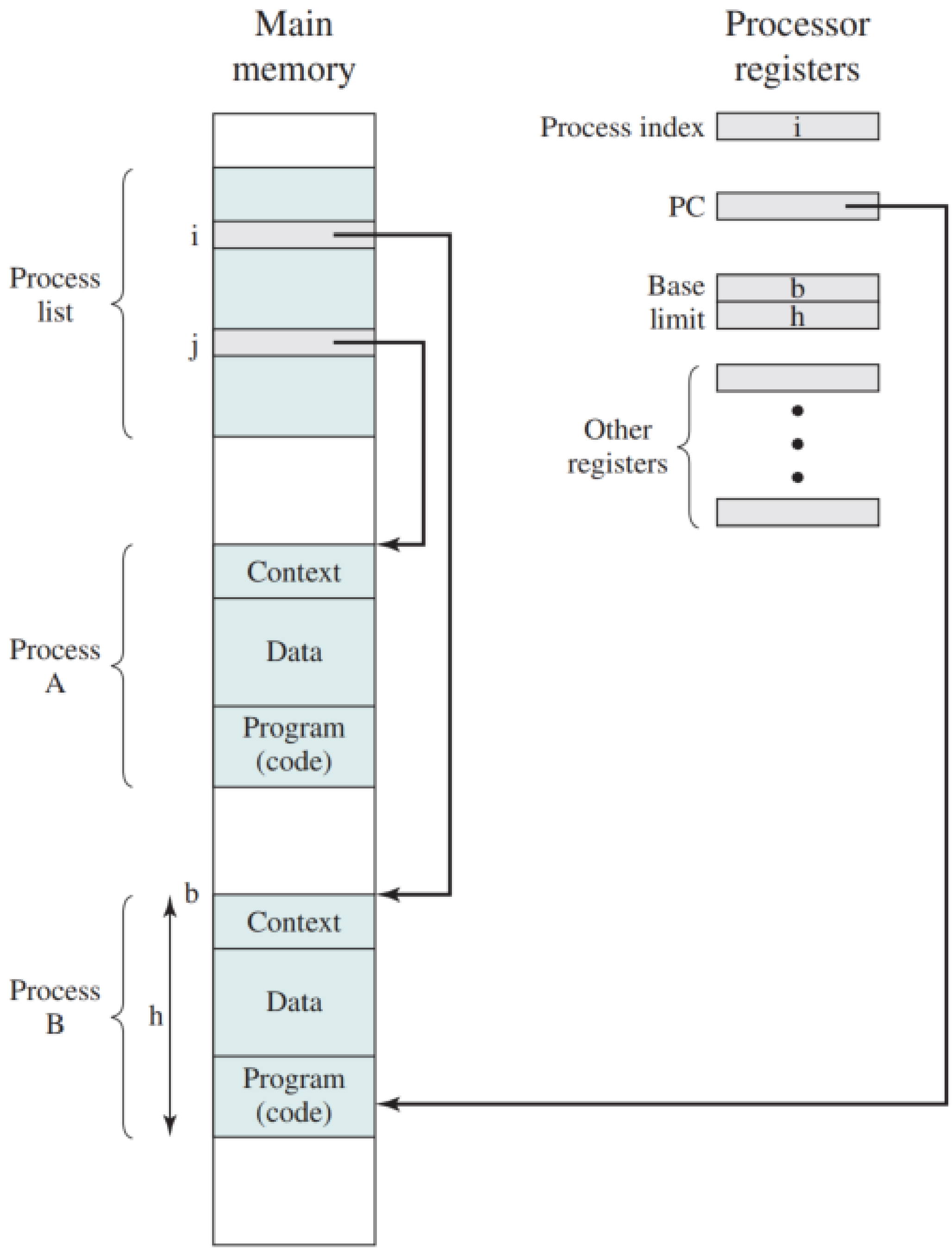
Causes of Errors

- Improper synchronization
  - a program must wait until the data are available in a buffer
  - improper design of the signaling mechanism can result in loss or duplication
- Failed mutual exclusion
  - more than one user or program attempts to make use of a shared resource at the same time
  - only one routine at a time allowed to perform an update against the file
- Nondeterminate program operation
  - program execution is interleaved by the processor when memory is shared
  - the order in which programs are scheduled may affect their outcome
- Deadlocks
  - two or more programs to be hung up waiting for each other
  - may depend on the chance timing of resource allocation and release

Components of a Process

- **an executable program**
- **the associated data needed by the program**
  - variables, work space, buffers, etc.
- **the execution context -- process state**
  - internal data
    - OS is able to supervise and control the process
  - includes the contents of the various process registers
  - includes information
    - the priority of the process
    - whether the process is waiting for the completion of a I/O event

Typical Process Implementation



Memory Management

- **five principal storage management responsibilities**
  - process isolation（进程隔离）
  - automatic allocation and management （自动分配和管理）
  - support of modular programming（支持模块化程序设计）
  - protection and access control（保护和访问控制）
  - long-term storage（长期存储）

Virtual Memory

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available

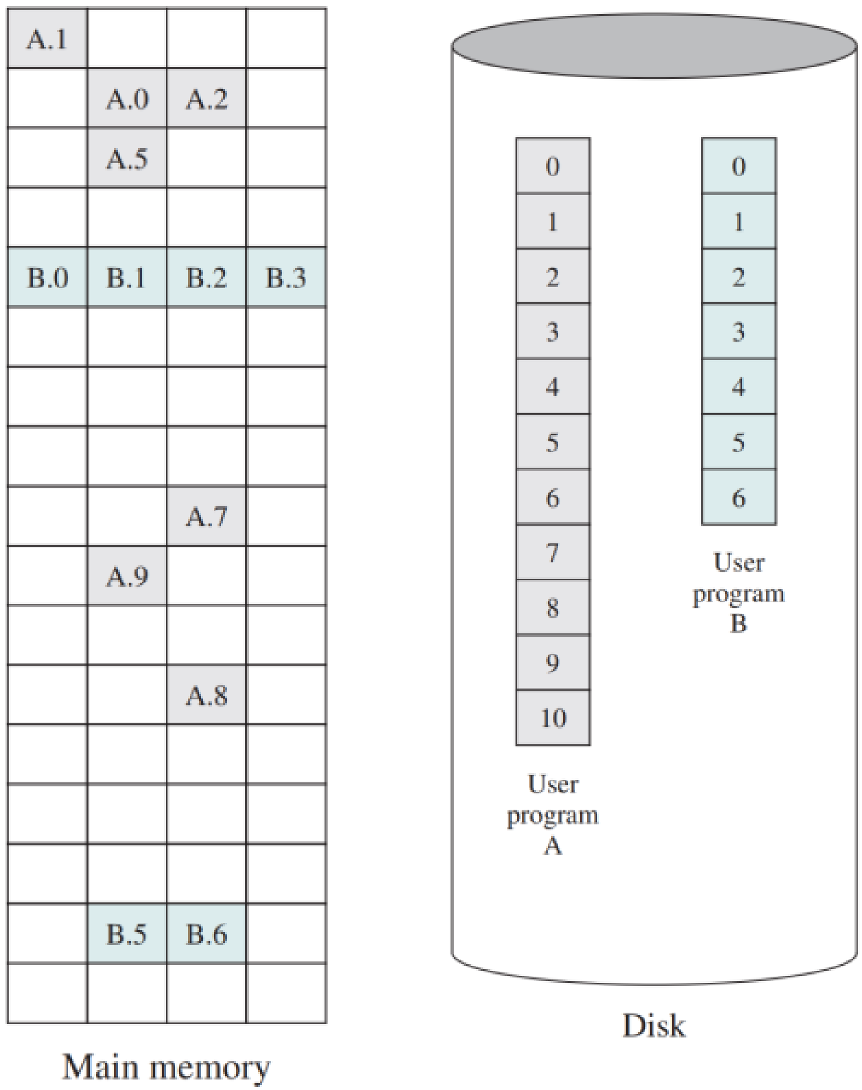
- Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently

Paging

- Allows processes to be comprised of a number of fixed-size blocks, called pages
- Program references a word by means of a virtual address
  - consists of a page number and an offset within the page
  - each page may be located anywhere in main memory
- Provides for a dynamic mapping between the virtual address used in the program and a real (or physical) address in main memory

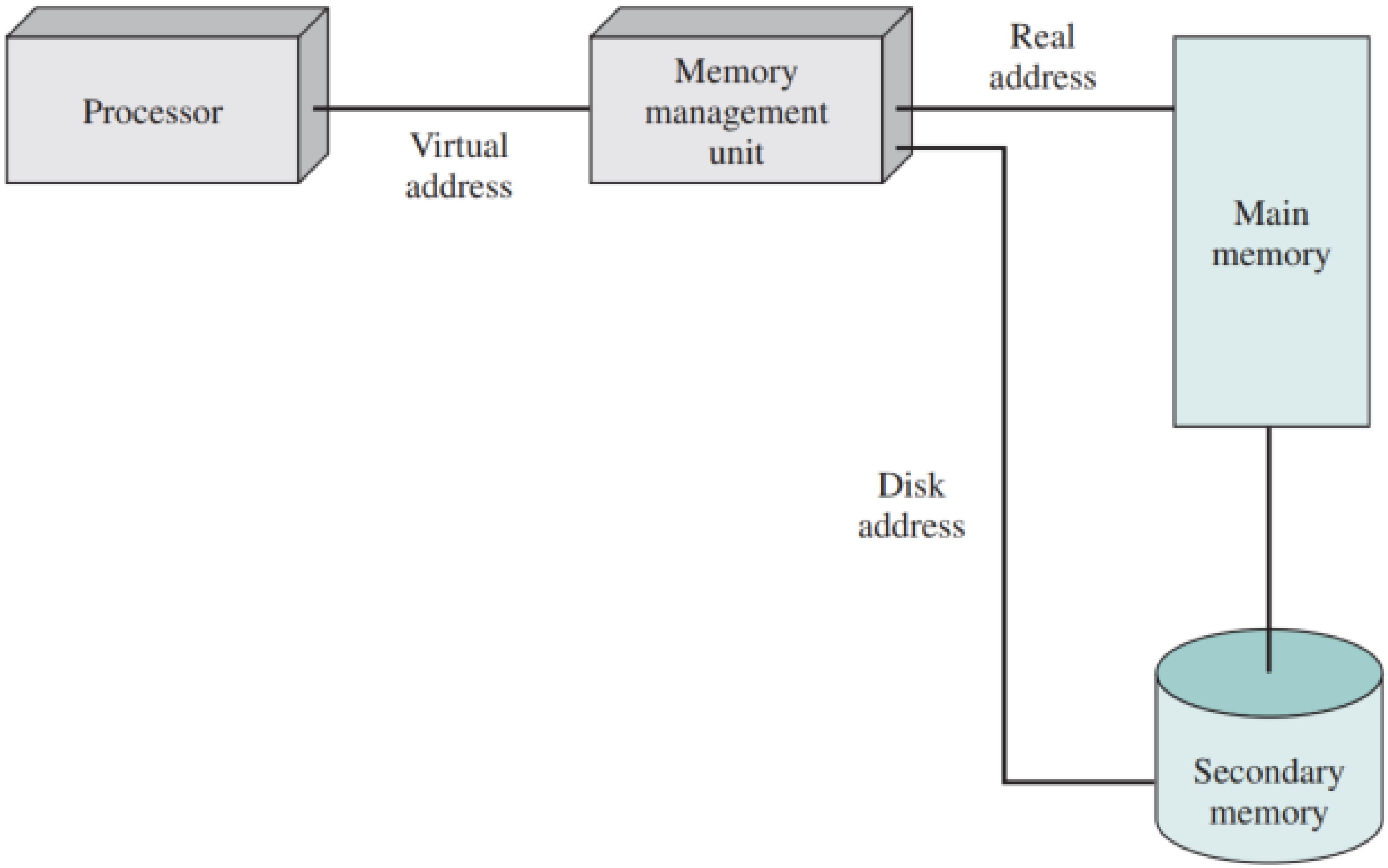
Virtual Memory Concepts

Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.



Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages of all programs plus the OS are on disk, as are files.

Virtual Memory Addressing



Information Protection and Security

- Availability（可用性）
  - Concerned with protecting the system against interruption
- Confidentiality（保密性）
  - Assures that users cannot read data for which access is unauthorized
- Data integrity（数据完整性）
  - Protection of data from unauthorized modification
- Authenticity（认证）
  - Concerned with the proper verification of the identity of users and the validity of messages or data

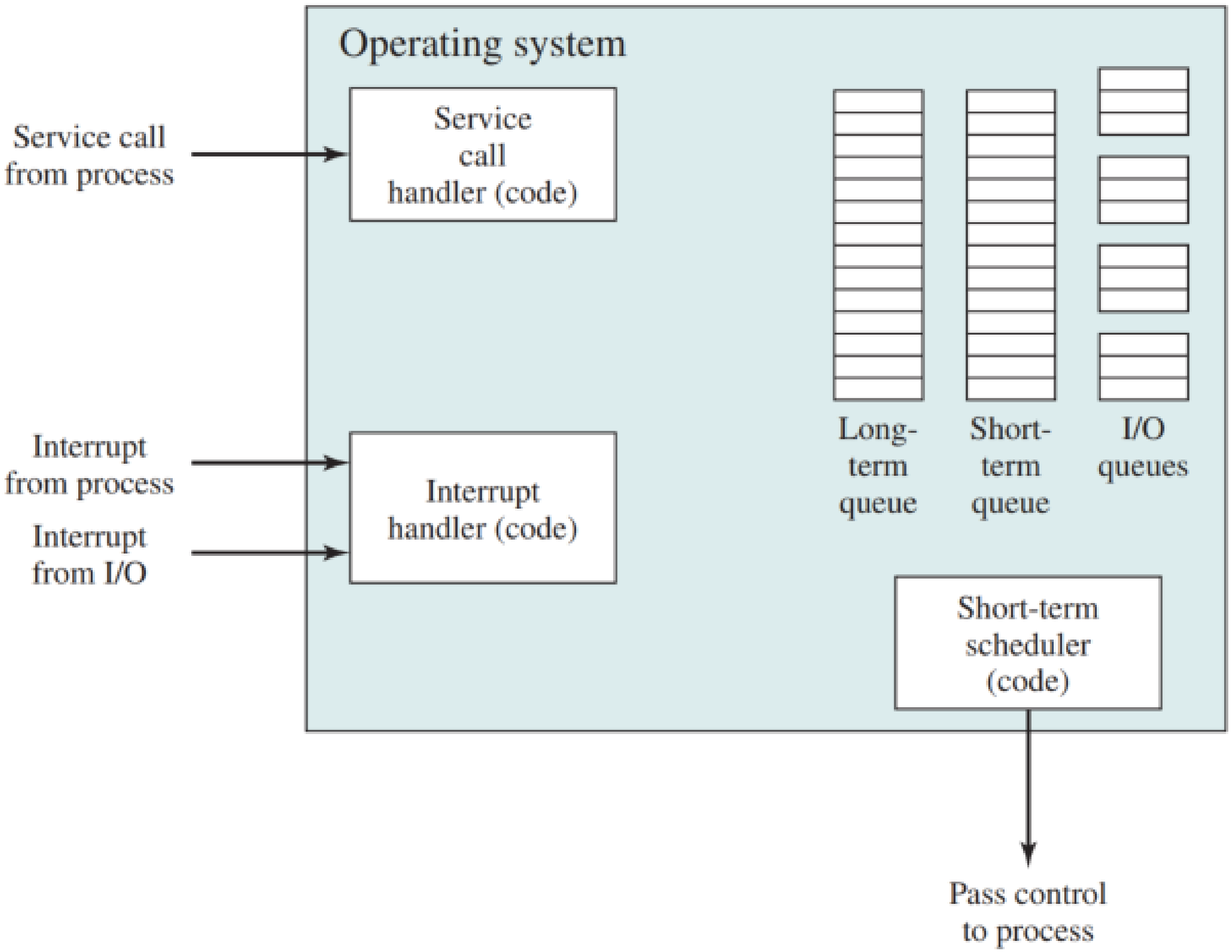
Scheduling and Resource Management

- Fairness

- Differential responsiveness
  - discriminate among different jobs with different service
  - make allocation and scheduling decisions to meet the total set of requirements and make these decisions dynamically

- Efficiency
  - these criteria conflict and finding the right balance
    - maximize throughput
    - minimize response time
    - accommodate as many users as possible

Key Elements for Multiprogramming



Developments Leading To Modern Operating Systems

- Microkernel architecture
  - A microkernel architecture assigns only a few essential functions to the kernel, including address space management, interprocess communication (IPC), and basic scheduling.
- Multithreading
  - Multithreading is a technique in which a process, executing an application, is divided into threads that can run concurrently
- Symmetric multiprocessing
- Distributed operating systems
- Object-oriented design

Fault Tolerance

- Fault tolerance refers to the ability of a system or component to continue normal operation despite the presence of hardware or software faults.
  - involves some degree of redundancy
  - intended to increase the reliability of a system
  - increased fault tolerance comes with a cost

Operating System Mechanisms

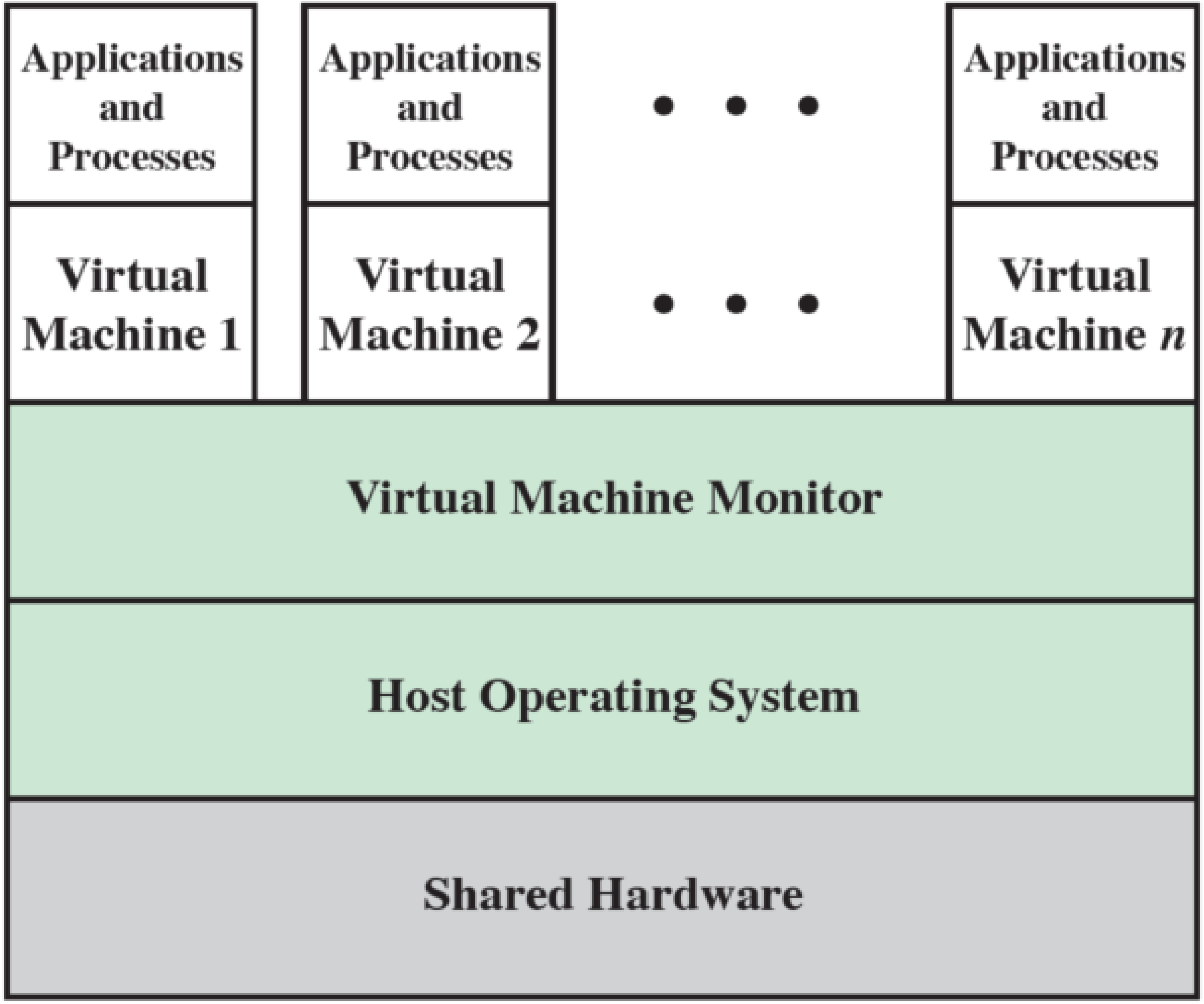
- A number of techniques can be incorporated into OS software to support fault tolerance.
  - Process isolation（进程隔离）
  - Concurrency controls（并发控制）
  - Virtual machines（虚拟机）
  - Checkpoints and rollbacks（检测点和回滚机制）

Virtual Machines and Virtualization

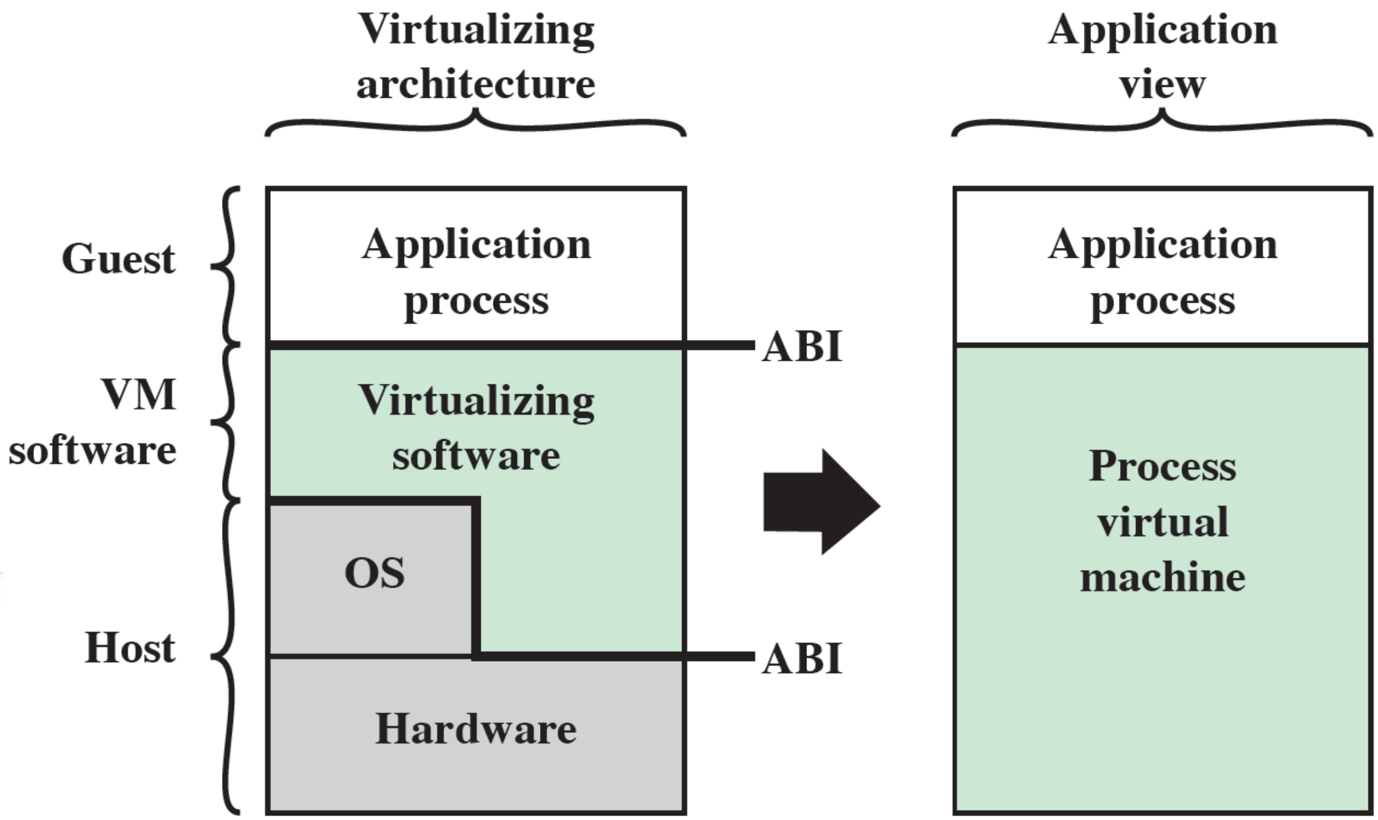
Virtualization

- enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS
- a machine can host numerous applications, including those that run on different operating systems, on a single platform
- host operating system can support a number of virtual machines (VM)
- each has the characteristics of a particular OS and, in some versions of virtualization, the characteristics of a particular hardware platform

A typical architecture of VM

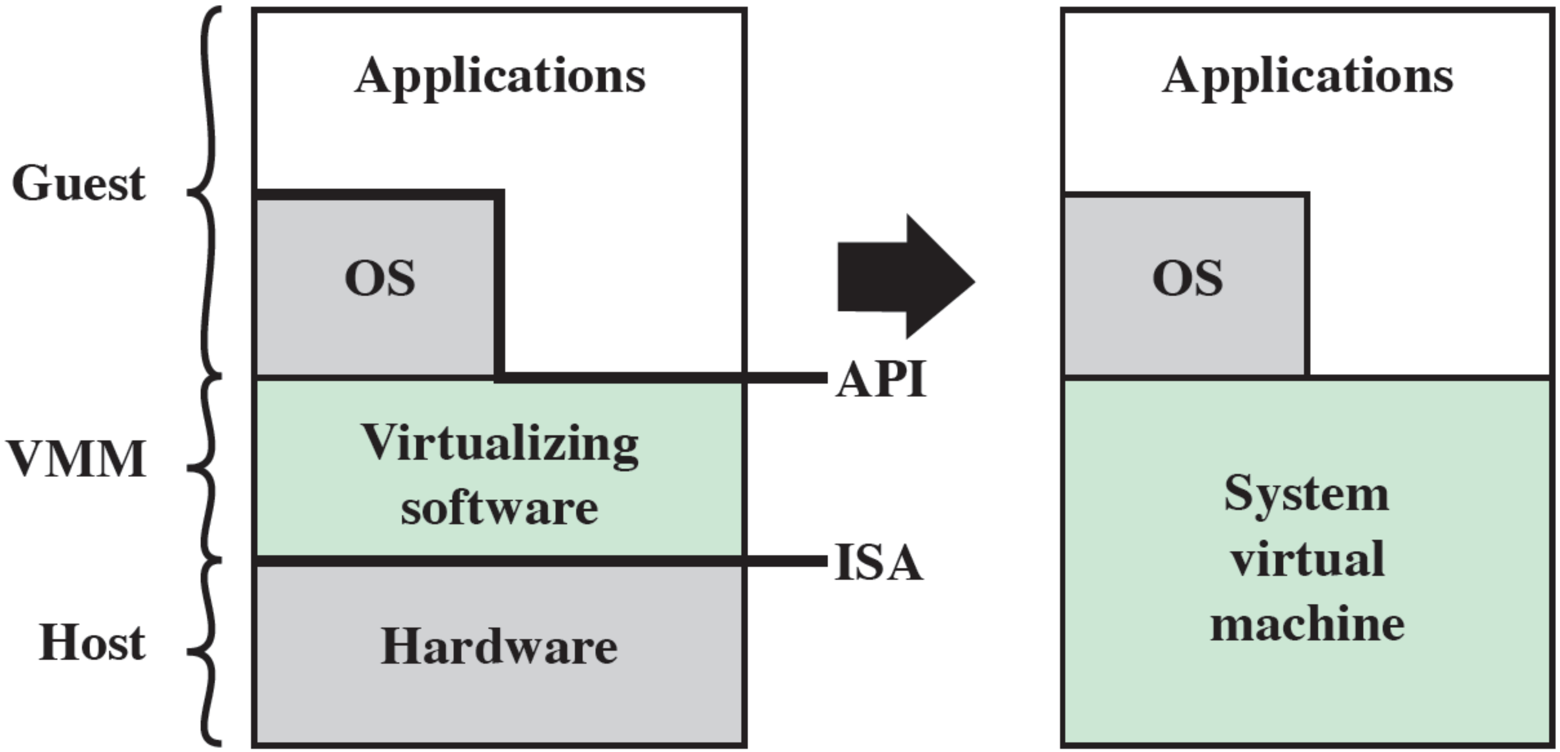


Process and System Virtual Machines



(a) Process VM

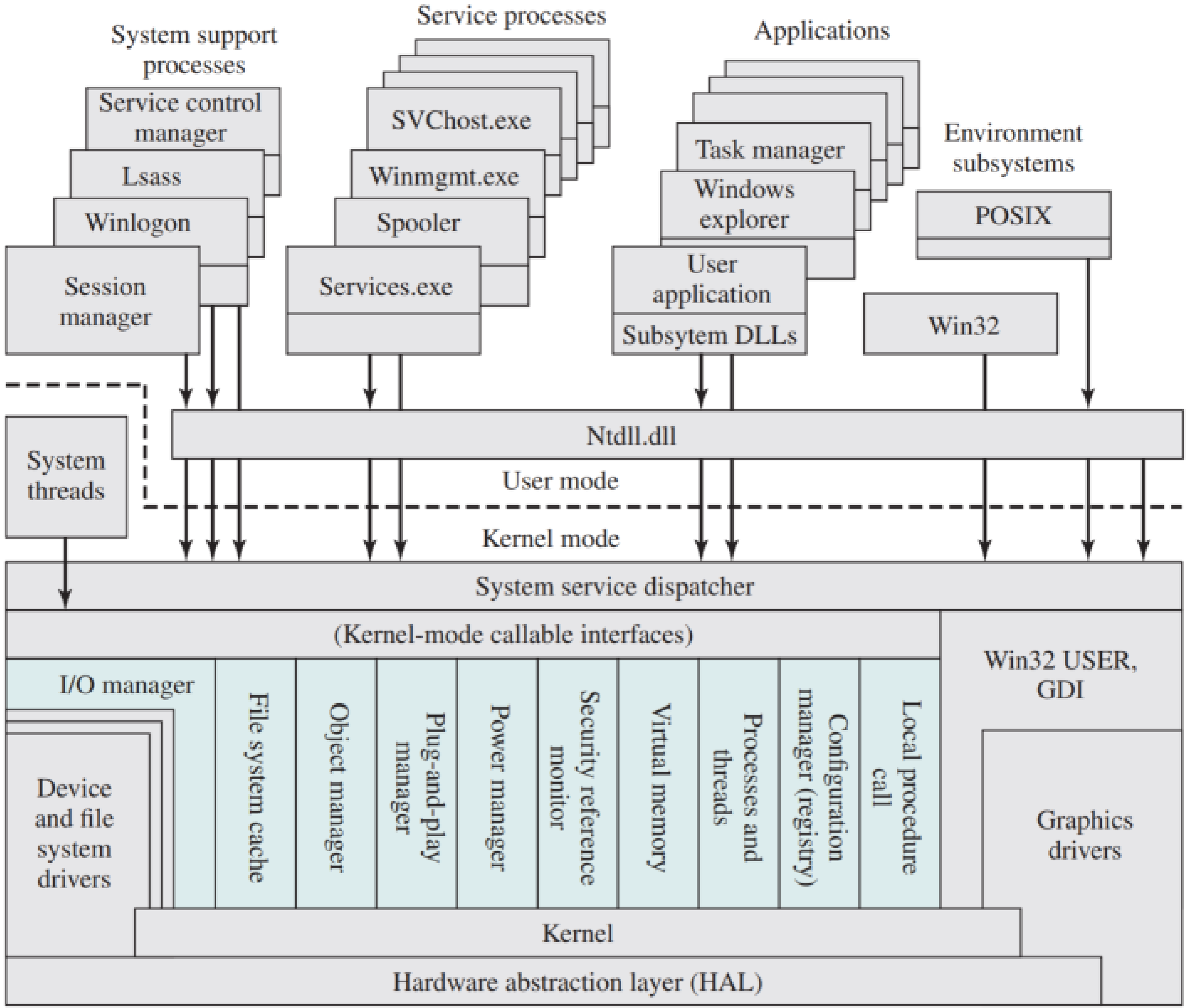
Process and System Virtual Machines



(b) System VM

Microsoft Windows Overview

- MS-DOS 1.0 released in 1981
  - 4000 lines of assembly language code
  - ran in 8 Kbytes of memory
  - used Intel 8086 microprocessor
- Windows 3.0 shipped in 1990
  - 16-bit
  - GUI Interface
- Windows 95
  - 32-bit version
  - led to the development of Windows 98 and Windows Me
- Windows NT (3.1) in 1993
  - 32-bit OS with the ability to support older DOS and Windows applications
- Windows 2000
  - included services and functions to support distributed processing
  - Active Directory
  - plug-and-play
- Windows XP released in 2001
- Windows Vista shipped in 2007
- Windows Server released in 2008
- Windows 7 shipped in 2009
- Windows Azure
  - targets cloud computing

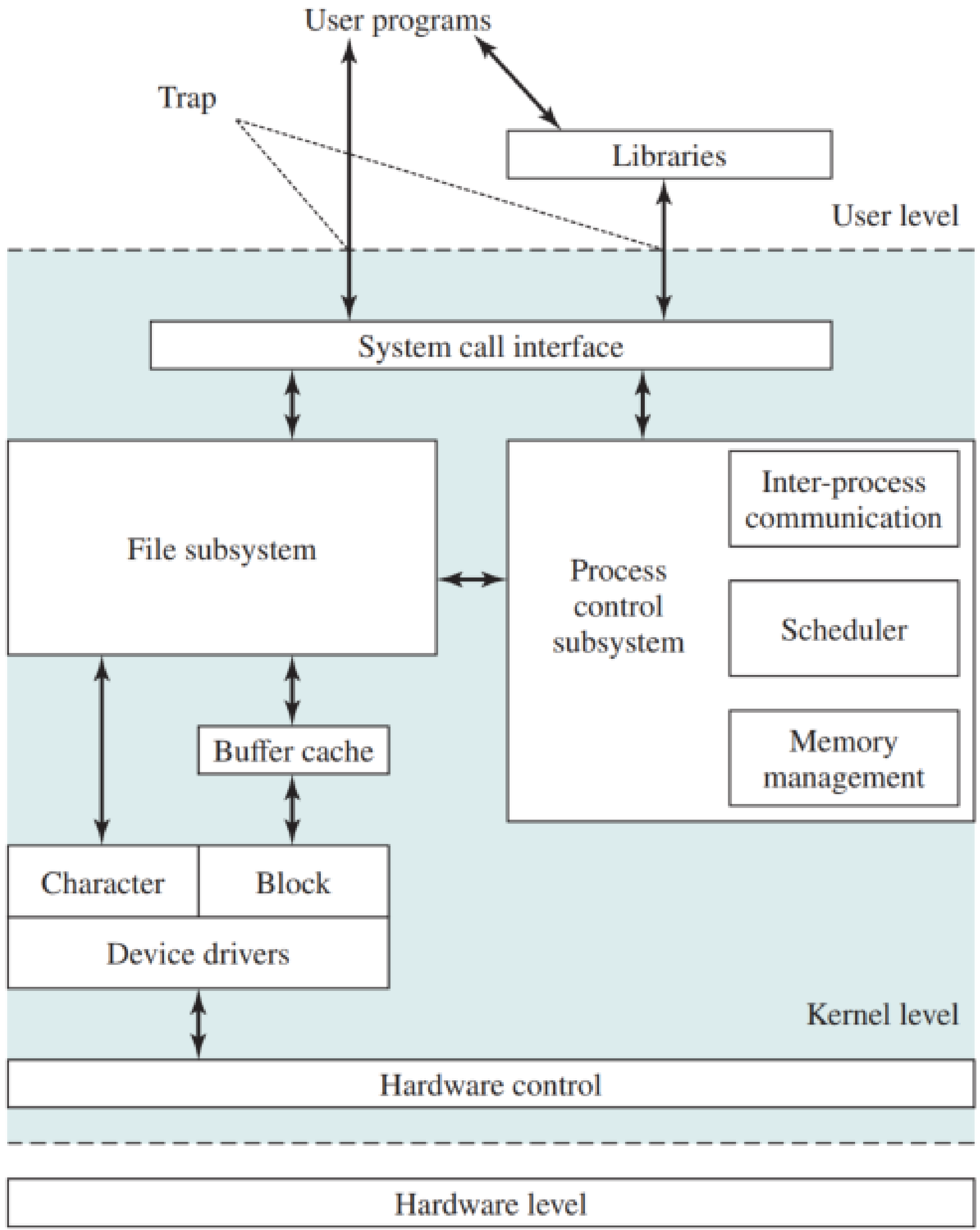


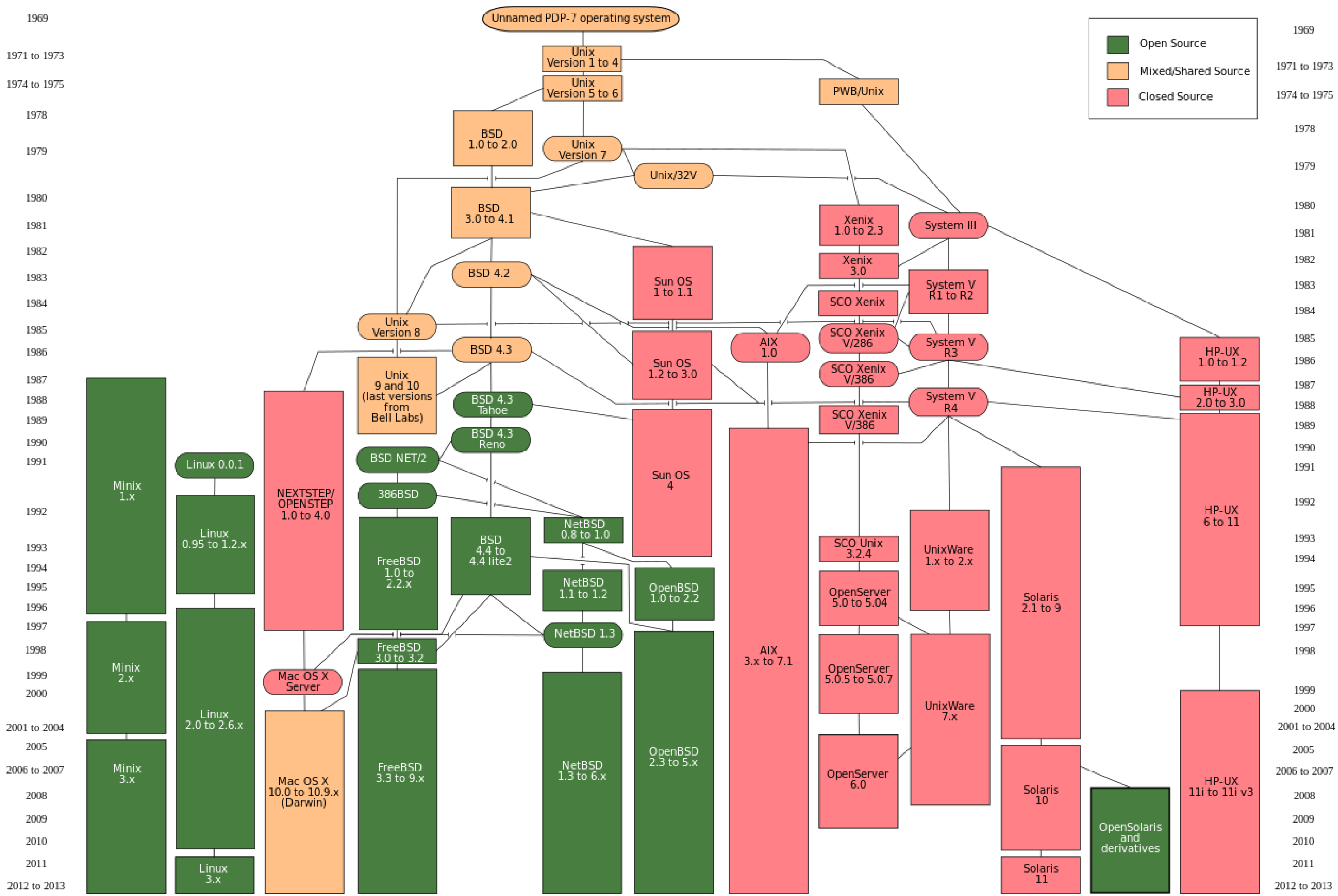
Windows Internals Architecture

Traditional UNIX Systems

- Were developed at Bell Labs and became operational on a PDP-7 in 1970
  - Incorporated many ideas from Multics
- PDP-11 was a milestone because it first showed that UNIX would be an OS for all computers
- Next milestone was rewriting UNIX in the programming language C
  - demonstrated the advantages of using a high-level language for system code
- Was described in a technical journal for the first time in 1974
- First widely available version outside Bell Labs was Version 6 in 1976
- Version 7, released in 1978 is the ancestor of most modern UNIX systems
- Most important of the non-AT&T systems was UNIX BSD (Berkeley Software Distribution)

Traditional UNIX Architecture





Evolution of Unix and Unix-like systems

Thompson wrote the first version of the Unix operating system for PDP-7 in a month. The PDP-7 he used had only 4K of 18-bit words.

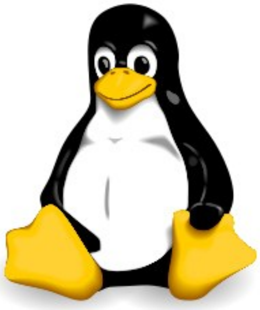
In 1983 Thompson and Ritchie received the ACM A. M. Turing Award. The Turing Award selection committee wrote

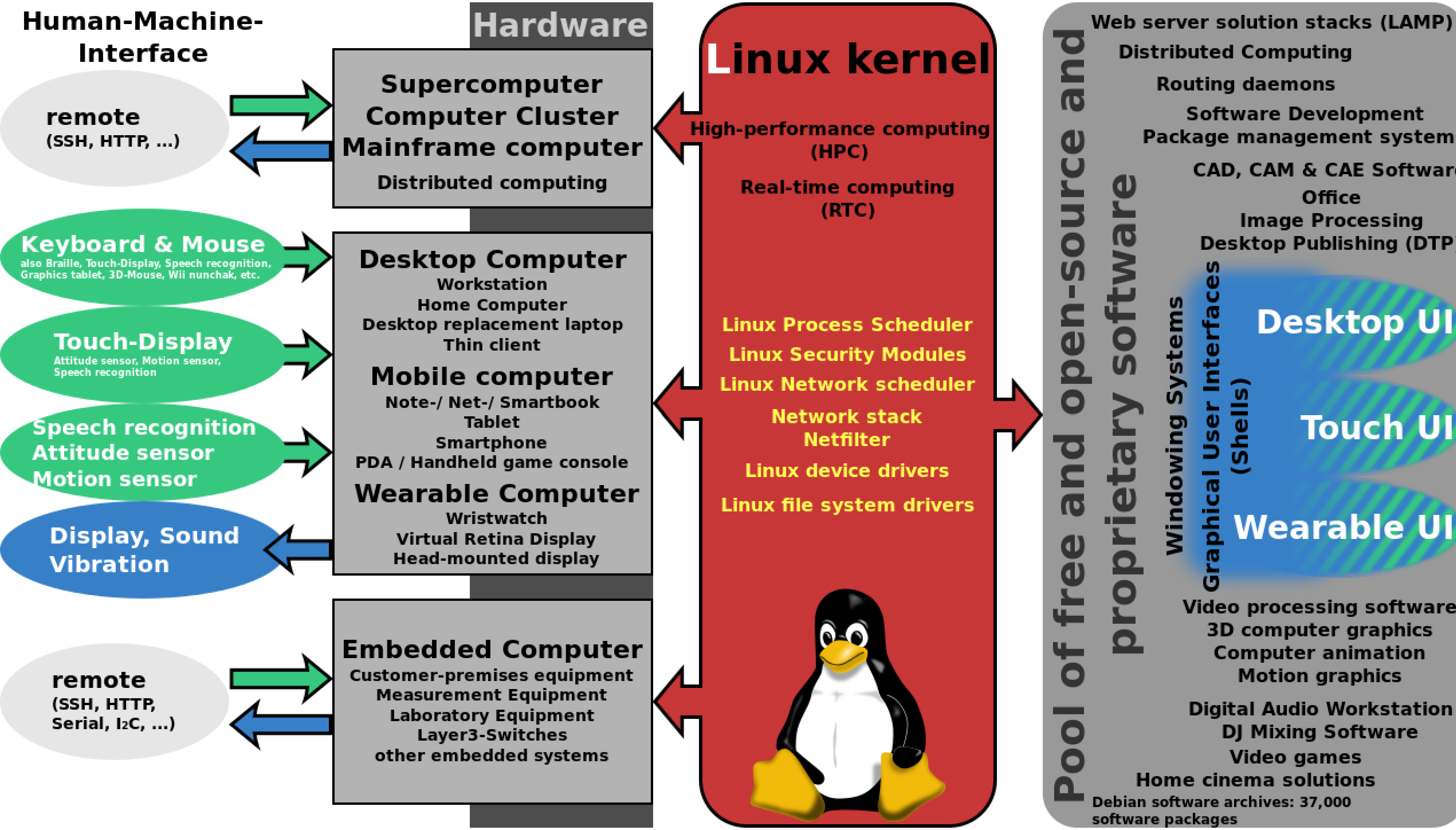
> *The success of the UNIX system stems from its tasteful selection of a few key ideas and their elegant implementation. The model of the Unix system has led a generation of software designers to new ways of thinking about programming. The genius of the Unix system is its framework, which enables programmers to stand on the work of others.*

LINUX Overview

- Started out as a UNIX variant for the IBM PC
- Linus Torvalds, a Finnish student of CS, wrote the initial version
- Linux was first posted on the Internet in 1991
- Today it is a full-featured UNIX system that runs on several platforms
- Is free and the source code is available
- Key to success has been the availability of free software packages
- Highly modular and easily configured

Linus Torvalds, principal author of the Linux kernel





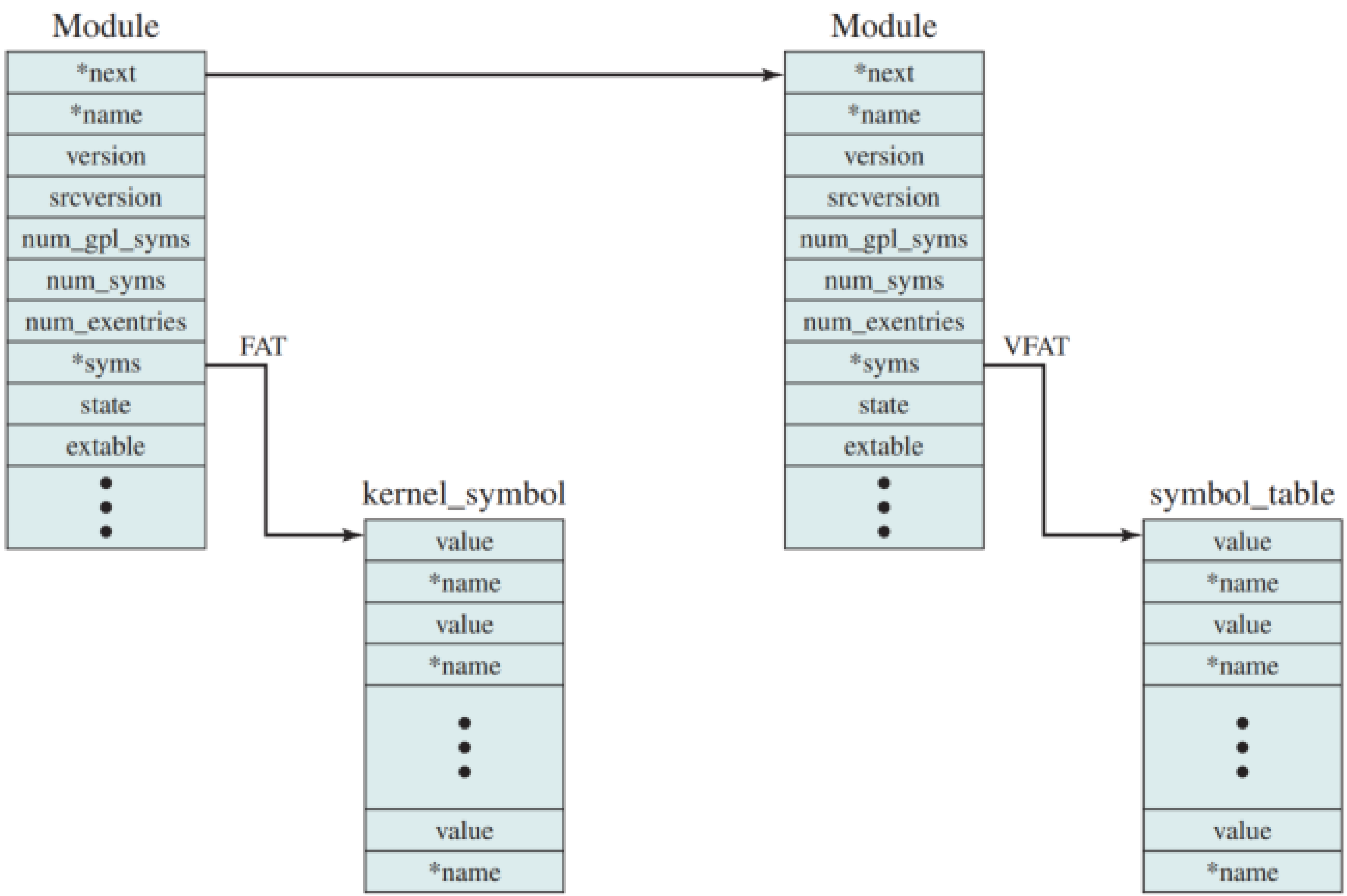
Linux is ubiquitously found on various types of hardware

Modular Monolithic Kernel

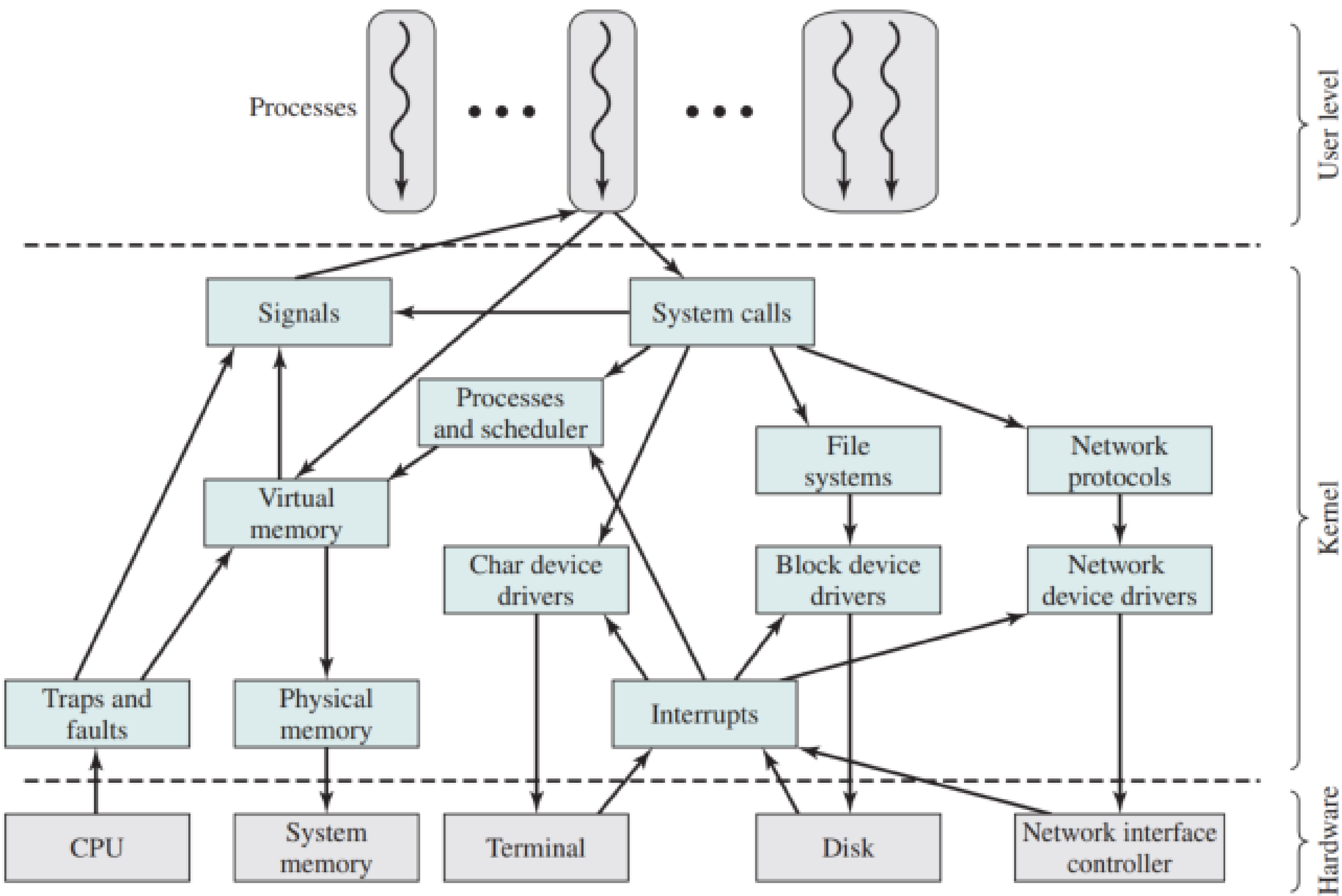
- **Monolithic Kernel**
  - Includes virtually all of the OS functionality in one large block of code that runs as a single process with a single address space
  - All the functional components of the kernel have access to all of its internal data structures and routines
- **Linux is structured as a collection of modules**

- **Loadable Modules**
  - Relatively independent blocks
  - A module is an object file whose code can be linked to and unlinked from the kernel at runtime
  - A module is executed in kernel mode on behalf of the current process
  - two important characteristics
    - Dynamic linking（动态链接）
    - Stackable modules（可堆叠模块）

Linux Kernel Modules

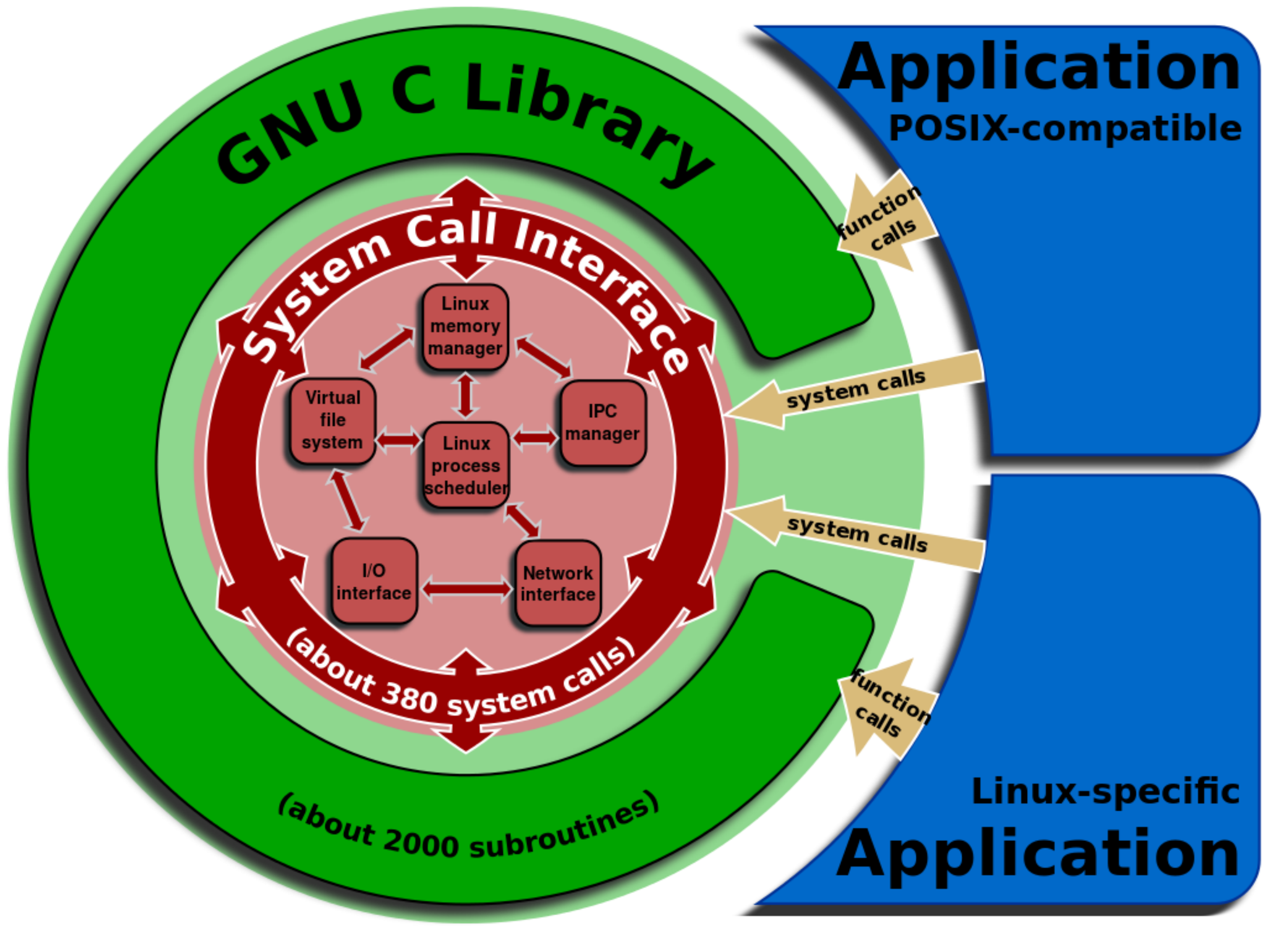


Linux Kernel Components

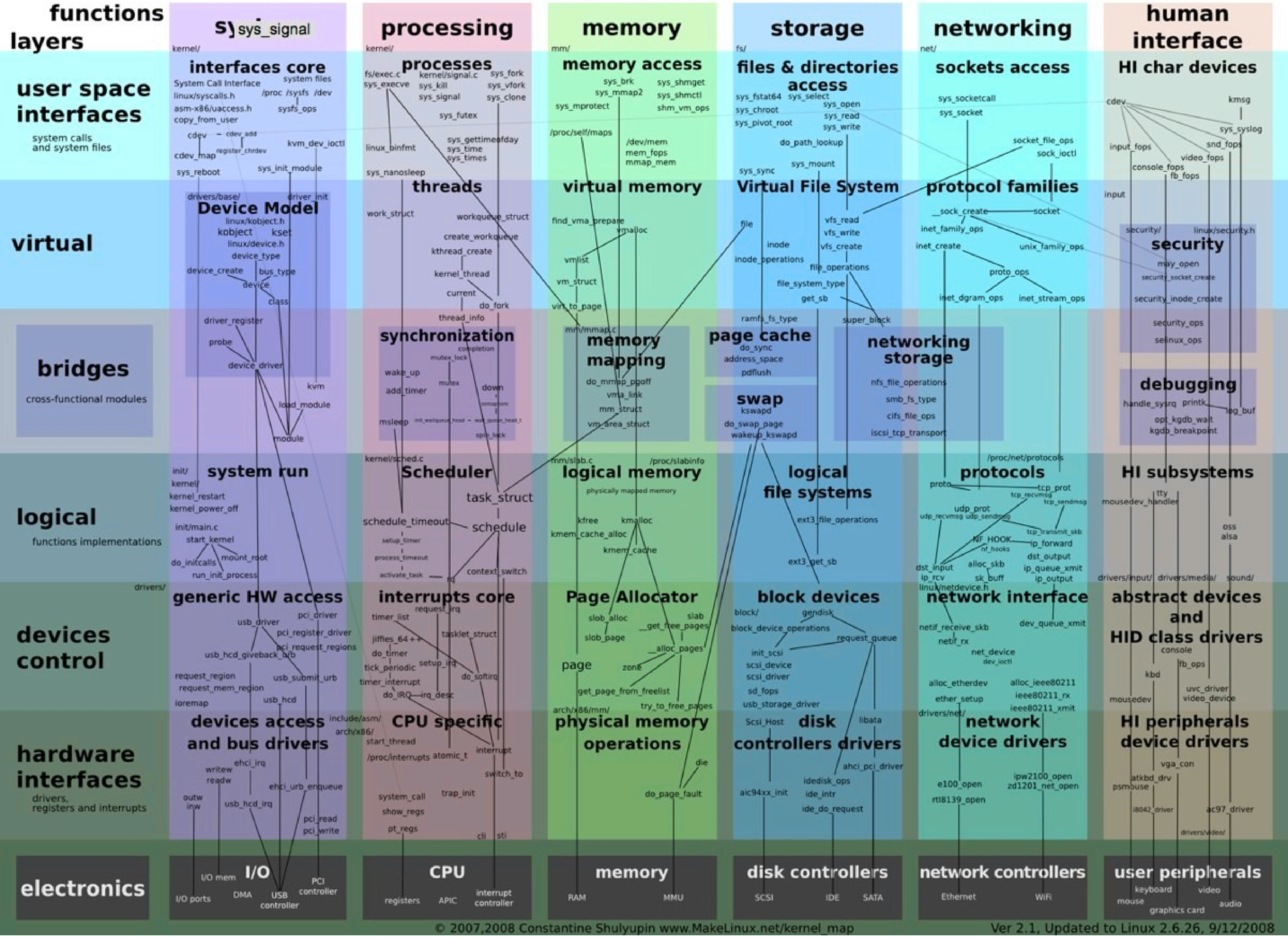


In computing, a **system call** is how a program requests a service from an operating system's kernel. This may include hardware related services (e.g. accessing the hard disk), creating and executing new processes, and communicating with integral kernel services (like scheduling). **System calls** provide **an essential interface** between **a process and the operating system**

System calls



Linux kernel map



Summary

- **Objectives and functions**
  - Convenience, efficiency, ability to evolve
  - User/computer interface
  - Resource manager
- **Evolution**
  - Serial processing
  - Simple batch systems
  - Multiprogrammed batch systems
  - Time sharing systems
- **Major achievements**
  - Processes
  - Memory management
  - Information protection and Security
  - Scheduling and resource management
- **Virtual machines and virtualization**
- **Traditional operating system**
  - Windows, Unix and Linux