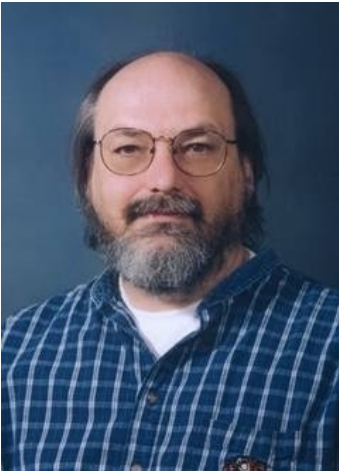# Introduction to UNIX

History of Unix

Kenneth Thompson

…
1966 -- Joins Bell Labs Computing Research Department, working on the Multics project
1969 -- <u>Develops UNIX* operating system</u>
1970 -- Writes B language, precursor to Dennis Ritchie's C language
1971 -- Moves UNIX from the PDP-7 to the PDP-11
1973 -- Rewrites UNIX in Dennis Ritchie's C language
…

Dennis M. Ritchie

…
1968 -- Joins the Bell Labs team working on Multics, a joint effort of Bell Labs, MIT and GE to develop a general computer operating system
1972 -- <u>Creates C language</u>
…

History of Unix

- Ken Tompson develops it on PDP-7 in Bell Lab (1969)
- Dennis Ritchie joins and writes it in C
- Sixth Edition (1976)
  - First version allowing other institutions to use
  - After then UNIX was divided in AT&T and BSD series
- AT&T UNIX
  - System V, Release 3 (1987): employs STREAMS
  - System V, Release 4 (1989): C/Korn shell, Job control, Symbolic link
- BSD UNIX
  - BSD 4 (1980): Job control, Reliable signal
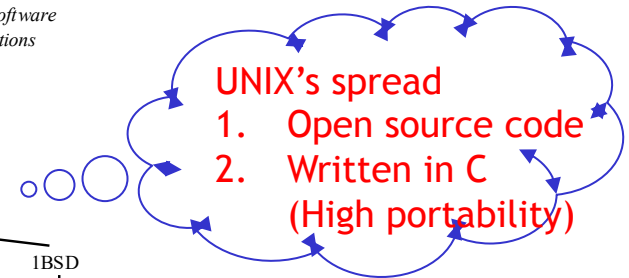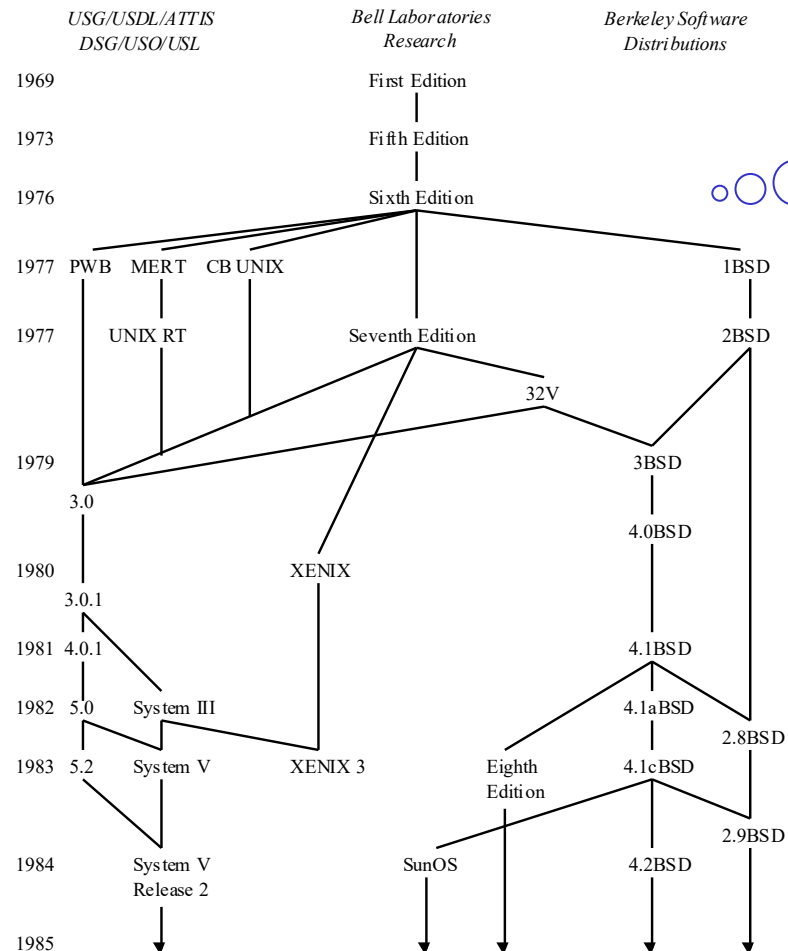 - BSD 4.4 (1993)

History of Unix

Figure 1.1 The UNIX system family tree, 1969-1985
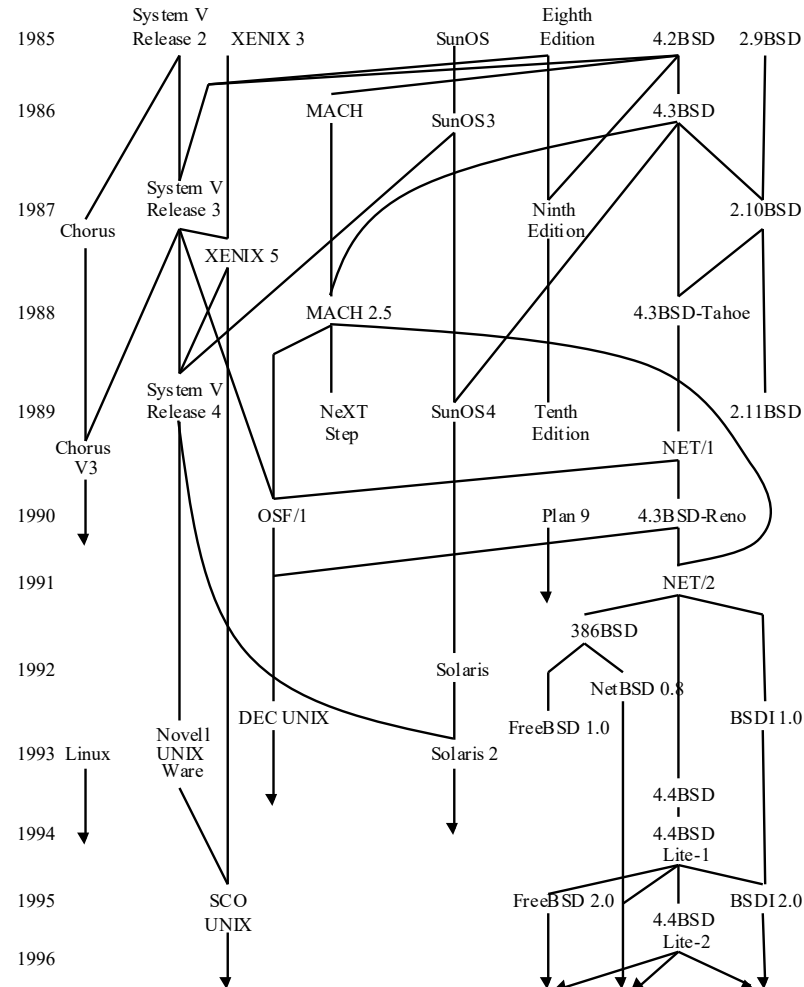
4

History of Unix

Figure 1.2 The UNIX system family tree, 1986-1996

UNIX Standardization

- ANSI C (1989)
  - American National Standards Institute
  - Establish C language grammar, library and header files
  - For portability to various operating systems
- POSIX (1988)
  - Portable Operating System Interface for Computer Environments
  - Define OS-providing services
  - 1003.1: Interface standard (called POSIX.1)
  - 1003.2, 1003.7, and so on.

Unix Characteristics

- **What is Good with Unix?**
  - Open System
  - Small is beautiful philosophy
    - file: just stream of bytes
    - Data, Device, Socket, Process, … can be treated as a file.
  - Portability
    - high-level language
    - client-server model, clustering, …
  - True Parallelism
    - Multitasking, Multiprocessor, …

Unix Characteristics

- **What is Wrong with Unix?**
    - Too many variants
        - dumping ground
    - Not small and simple any more
        - uncontrolled growth
    - Lack of GUI
        - not now (MIT's X)

What is Linux?

- Linux
  - UNIX-like OS
  - Powerful and stable as other commercial OS
    - Kernel is developed by many hackers
    - Tested and improved by many people over the world

  - COPYLEFT(open source)
    - Follows GNU General Public License (GPL)
      - Source code as well as execution file
      - Anyone can involve modification of source code
      - Open not only modified program but also modified source code

Linux History

■ history

- 1991 : Linus Tovalds (Graduate student)
  - version 0.01 based on Minix
  - Paging, timer interrupt, device driver, file system, …
- 1992 : Linux distro (= Kernel + application programs)
- 1993 : stabilize kernel
- 1994 : version 1.0
- 1996 : version 2.0
- 1999 : version 2.2
- 2001 : version 2.4
- 2003 : version 2.6
- 2012 : version 3.0, 3.2
- 2019 : version 5.0

Kernel Distro Website : http://www.kernel.org

Linus Benedict Torvalds

Why Linux?

- Why Linux?
  - Powerful (see next page)
  - Unix based (POSIX 1003.1 standard support)
    - ➔ Can use powerful S/W in Unix
  - open source code
    - ➔ 'experimenting' with the system is possible.
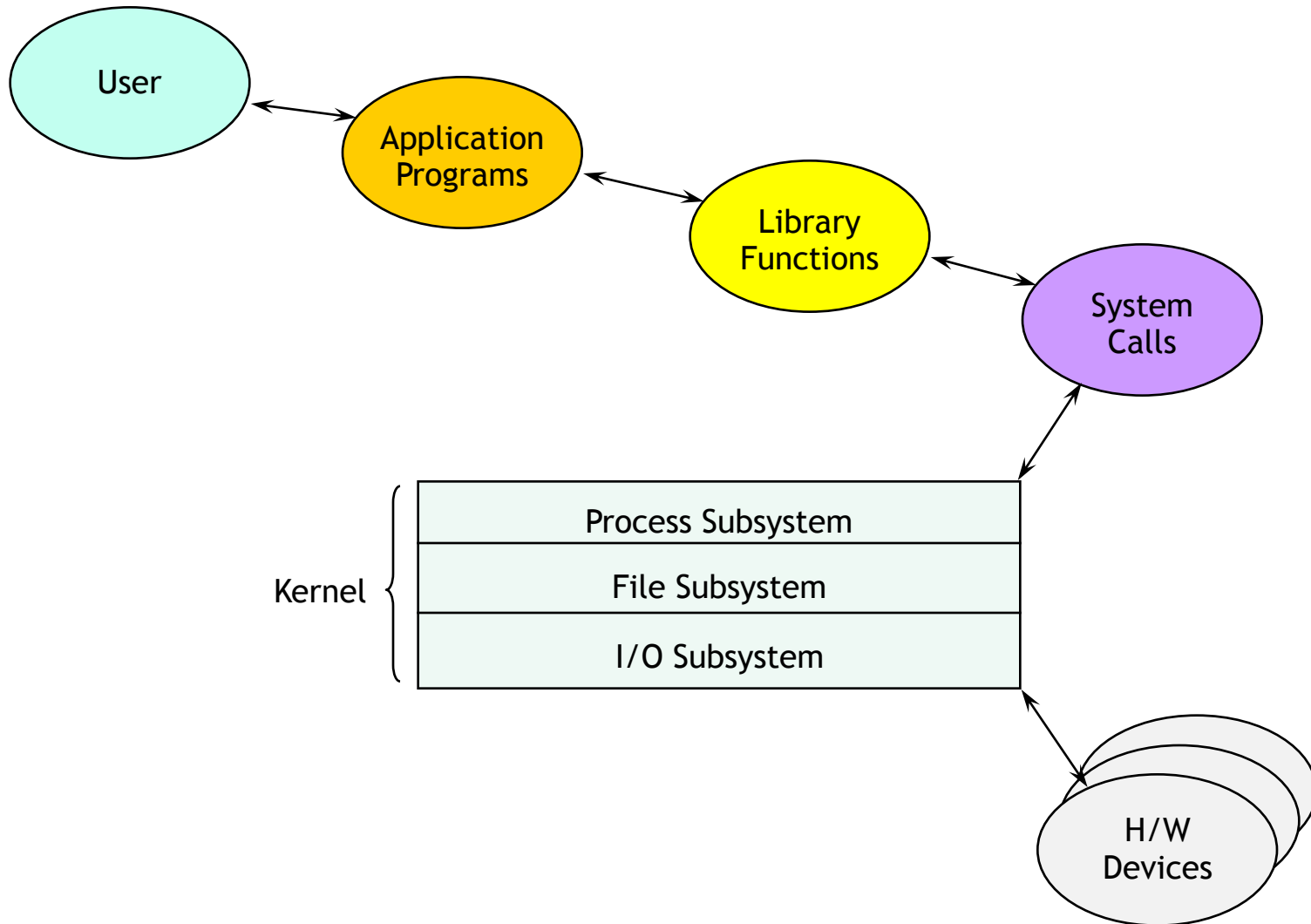 - Good documentations
    - FAQ, HOWTO

- ➔ We use Linux in lab.

Why Linux?

- **Merits**
  - Multi-Tasking
  - Multi-User access
  - POSIX 1003.1 standard support
  - Support various file systems
    - EXT2FS, EXT3FS, JFS, ReiserFS, NTFS, …
  - Support various network protocols
    - TCP/IP, SLIP, PPP, …
  - Support various architecture
    - 80*86, SPARC, ARM, PPC,…
  - Multi-processor
  - …

Unix/Linux Structure

User

Application
Programs

Library
Functions

System
Calls

Kernel {

| Process Subsystem |
| File Subsystem |
| I/O Subsystem |

H/W
Devices

Unix/Linux Structure

- **Application program**
  - e.g. ls, mkdir, chmod, vi, sh
- **Library**
  - Library function eventually calls system call
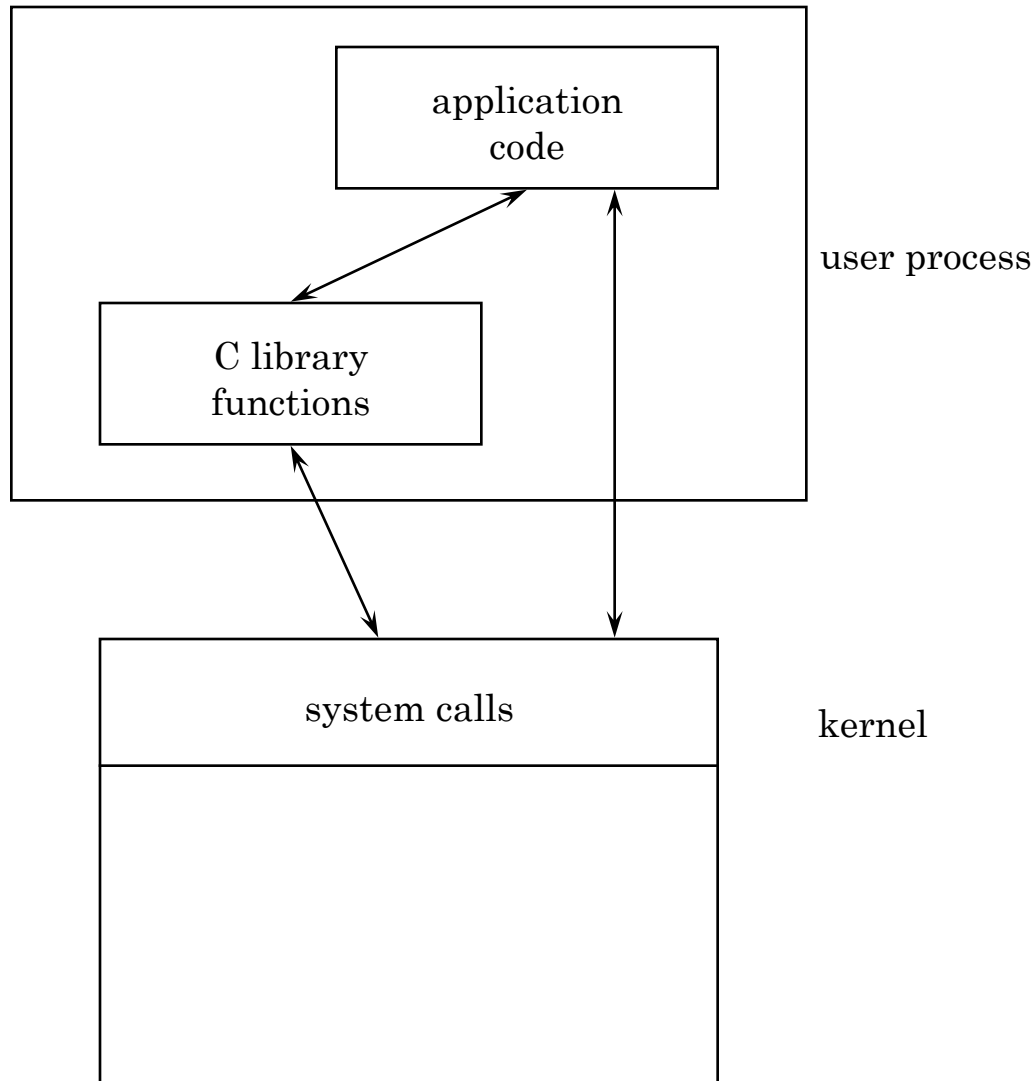 - e.g. printf() ➜ write()
- **System call**
  - Interface between Application & Operating system
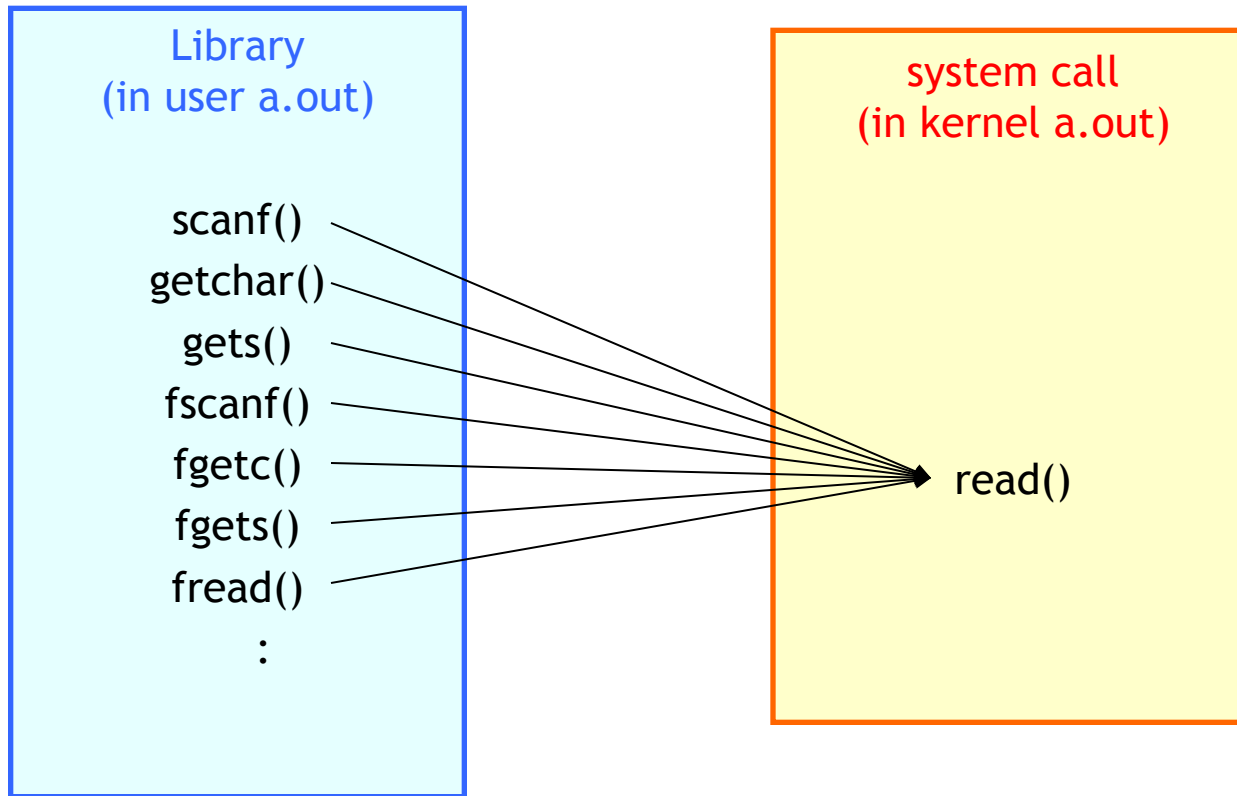 - System call is called then Kernel code is executed
- **Kernel**
  - Manages efficient usage of system resource
  - process/memory/file/IO management
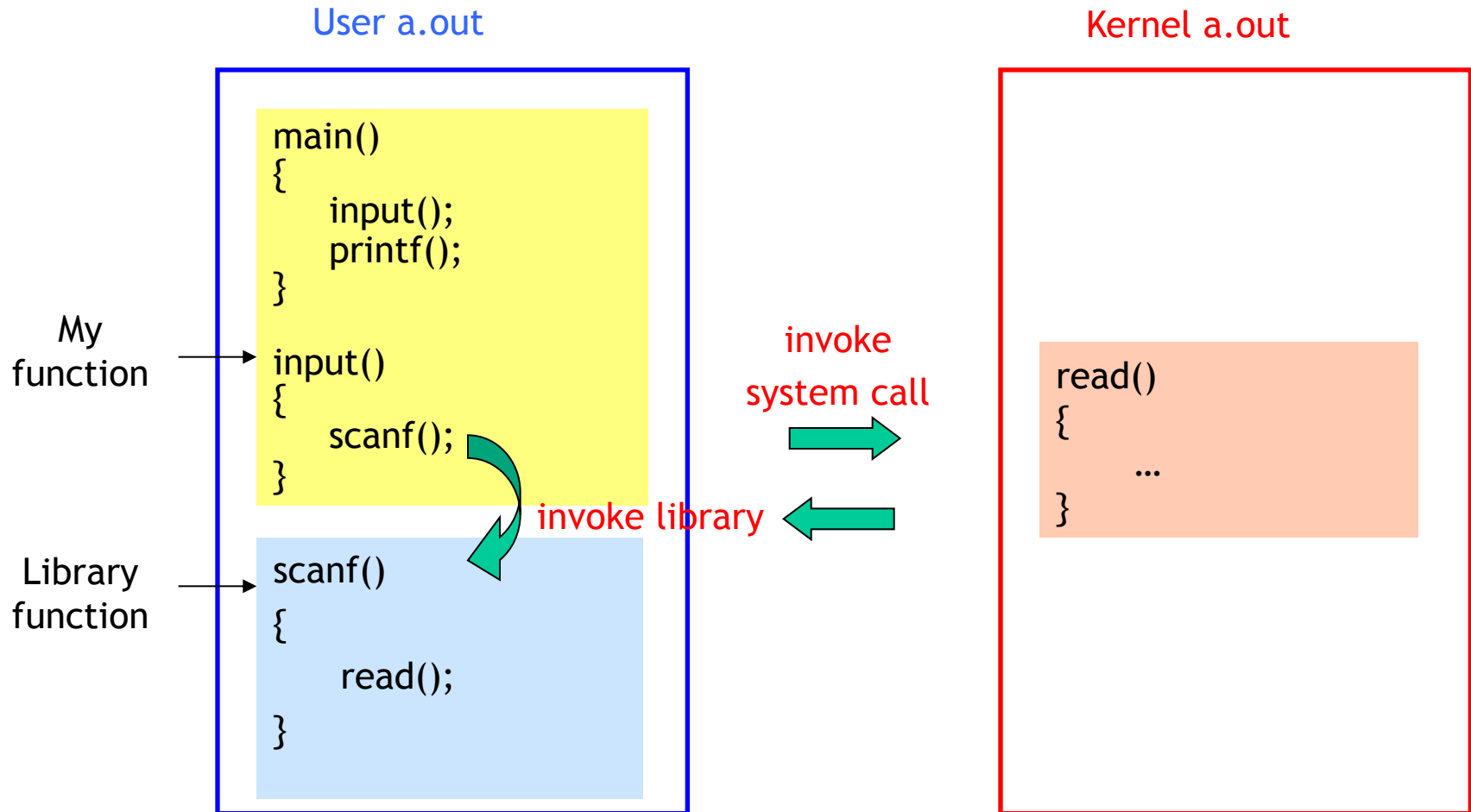
Library vs. system call

application
code

user process

C library
functions

system calls

kernel

Library vs. system call

■ Example

Library vs. system call

User a.out

Kernel a.out

```
main()
{
    input();
    printf();
}

input()
{
    scanf();
}
```

My function

Library function

```
scanf()
{
    read();
}
```

invoke system call

invoke library

```
read()
{
    ...
}
```

Program execution

user
mode

kernel
mode

user
mode

kernel
mode

input()   read()   printf()   write()

User defined
function call

Library
function call

System
call

Return
from
kernel

System
call

Program
begins

Program
ends
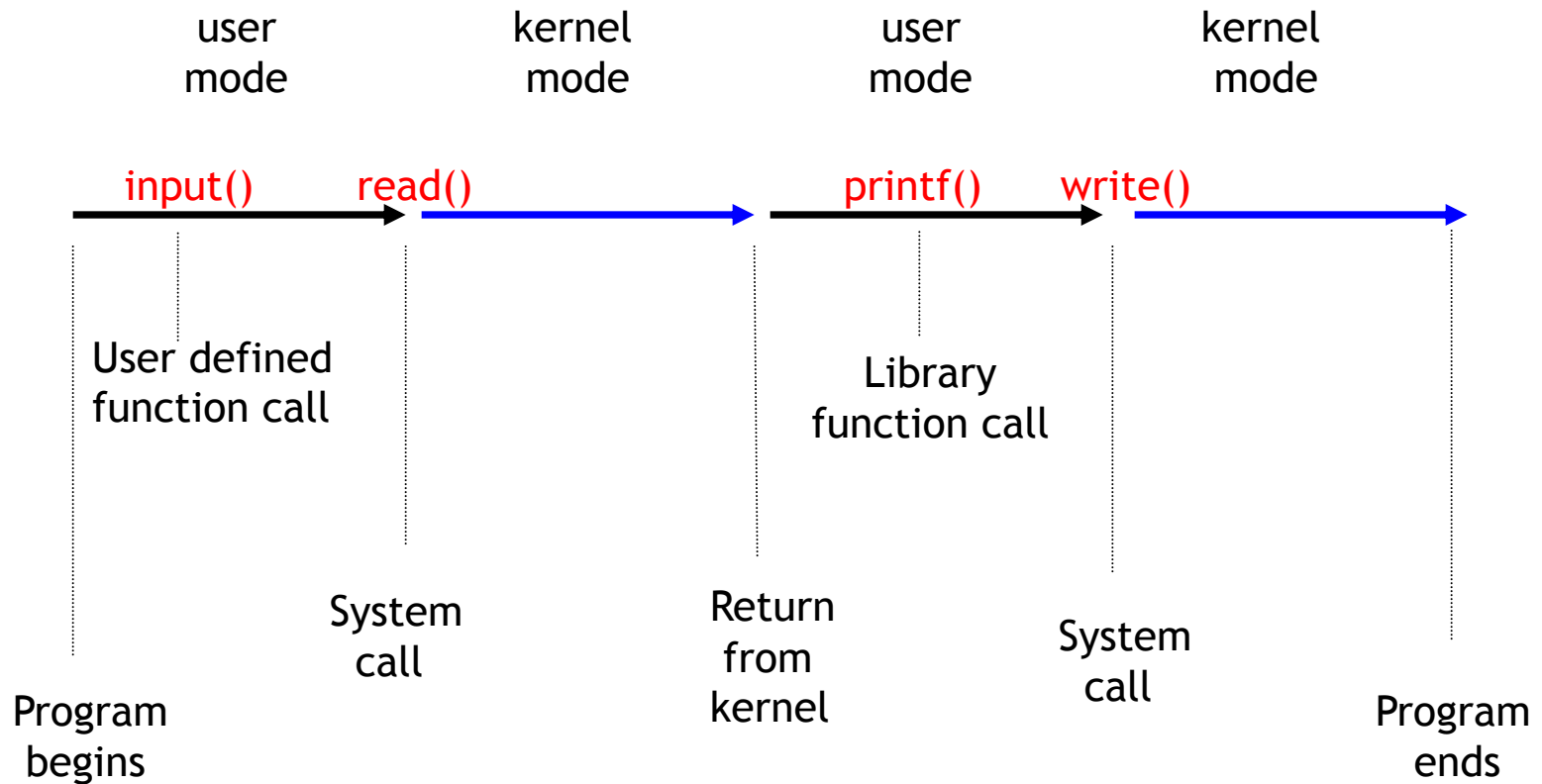
Kernel mode & User mode

- **kernel mode**
    - privileged mode
    - no restriction is imposed on the kernel of the system
    - may use all the instructions of the processor
    - manipulate the whole of the memory
    - talk directly to the peripheral controllers

Kernel mode & User mode

- user mode
  - normal execution mode for a process
  - has no privileges
    - certain instructions are forbidden
    - only allowed to zones allocated to it
    - cannot interact with the physical machine
  - process carries out operations in its own environment, without interfering with other processes
  - process may be interrupted at any moment